Scalable Vector Graphics (SVG)

Untersuchung des XML-Standards für zweidimensionale Vektorgraphiken und Erstellung eines SVG-Tutorials

Diplomarbeit von Iris Fibinger

Fachhochschule Karlsruhe

Fachbereich Sozialwissenschaften

Studiengang Technische Redaktion

März 2001

Danke

meiner Familie, die mir permanent menschlichen Halt gibt
– ganz besonders meinen beiden Nichten, die demnächst das Licht der Welt erblicken werden und die dadurch meinem Leben einen völlig neuen Aspekt geben,

dem Team des „XML-Labors“ für die menschliche Unterstützung
– besonders Martin Rechner für’s Korrekturlesen,

Herrn Prof. Muthig, unserem Studiengangsleiter, der mit seiner Aufmerksamkeit nicht nur mir so manches Mal das studentische Leben „rettete“ und schliesslich

den beiden Professorinnen Sissi Closs und Cosima Schmauch für die Betreuung von Diplomarbeit und Projekt.

Hiermit bestätige ich, dass ich die Diplomarbeit selbständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe.

Karlsruhe, März 2001

Iris Fibinger

Inhalt

Vorwort 1

1 Einführung 2
1.1 Die Idee zum Thema 2
1.2 Das Projekt „XML + Directory-Server“ 2
1.3 Voraussetzungen für die Diplomarbeit 3
1.3.1 Räumlichkeiten 3
1.3.2 Hardware 3
1.3.3 Software 3

2 Scalable Vector Graphics (SVG) 4
2.1 Überblick 4
2.1.1 XML und seine Instanzen 4
2.1.2 Was ist SVG? 5
2.1.3 Zur Geschichte von SVG 6
2.1.4 Die Recommendation vom W3C 6
2.1.5 Reaktionen auf SVG 6
2.1.6 SVG-unterstützende Software 6
2.1.6.1 Aktive (verarbeitende) Programme 6
2.1.6.1.1 Adobe Illustrator® 7
2.1.6.1.2 CorelDraw® 10
2.1.6.1.3 Jasc WebDraw™ 11
2.1.6.1.4 Sphinx™ open 12
2.1.6.1.5 SVGMaker (Druckertreiber) 13
2.1.6.1.6 Weitere Programme 14
2.1.6.2 Passive Programme (Viewer) 15
2.1.6.2.1 Adobe SVG-Viewer 15
2.1.6.2.2 IBM SVG-View 15
2.1.6.2.3 Csiro SVG Toolkit 16
2.2 SVG-Graphiken erstellen 16
2.2.1 Grundlagen 17
2.2.1.1 SVG 17
2.2.1.1.1 Das Tag <svg> 17
2.2.1.1.2 Das Grundgerüst eines SVG-Dokuments 17
2.2.1.2 Kommentare 18
2.2.1.2.1 Strukturierungselemente in SVG 18
2.2.1.2.2 System- und Ressourcen-Einstellungen für die Wiedergabe von Graphiken ermitteln19
2.2.1.2.3 Symbole und Pictogramme 21
2.2.1.2.4 Einbinden von (Raster-)Bildern in ein SVG-Dokument 22
2.2.1.3 Koordinatensysteme und Einheiten 22
2.2.1.3.1 Der Ursprung des SVG-Koordinatensystems 22
2.2.1.3.2 Einheiten 23
2.2.1.4 Gesteuerte Ansichten 23
2.2.1.4.1 Das Attribut ‚viewBox‘ 23
2.2.1.4.2 Das Attribut ‚preserveAspectRatio‘ 25
2.2.2 Graphik 26
2.2.2.1 Formatierungen 26
2.2.2.1.1 Cascading Stylesheets (CSS) 26

2.2.2.1.2 Das Attribut ‚style' 28
2.2.2.1.3 Formatierungen über spezielle Attribute 28
2.2.2.1.4 Entities 28
2.2.2.1.5 Die eXtensible Stylesheet Language (XSL) 28
2.2.2.2 Farben 29
2.2.2.2.1 Farben festlegen 30
2.2.2.2.2 Farbprofile des International Color Consortiums (ICC) .30
2.2.2.2.3 Füllungen und Linien 31
2.2.2.2.4 Füllungsattribute 31
2.2.2.2.5 Linienattribute 32
2.2.2.3 Vektoren 35
2.2.2.3.1 Grundformen 35
2.2.2.3.2 Beliebige Pfade mit dem Tag <path> 40
2.2.2.3.3 Pfeile 43
2.2.2.4 Texte 45
2.2.2.4.1 Texte auszeichnen 45
2.2.2.4.2 Texte ausrichten 48
2.2.2.4.3 Texte formatieren 49
2.2.2.4.4 Textpfade 51
2.2.2.4.5 Fonts 52
2.2.2.5 Graphische Effekte 55
2.2.2.5.1 Koordinatensysteme der Effekte 55
2.2.2.5.2 Verläufe 55
2.2.2.5.3 Füllmuster 59
2.2.2.5.4 Masken (‚clipping' und ‚masking') 61
2.2.2.5.5 Filter 65
2.2.2.6 Transformationen 68
2.2.3 Multimedia 75
2.2.3.1 Animationen 75
2.2.3.2 Andere Medien einbinden 80
2.2.4 Linking und Scripting 81
2.2.4.1 Linking 81
2.2.4.2 Scripting 82
2.2.4.2.1 Das Document Object Model (DOM) 82
2.2.4.2.2 Die script-gesteuerte Interaktion 82
2.2.4.2.3 Die script-gesteuerte Animation 84
2.2.5 SVG und andere Standards 87
2.2.5.1 Andere Sprachkonzepte in SVG integrieren 87
2.2.5.1.1 Namensräume (namespaces) 87
2.2.5.1.2 Das Tag <foreignObject> 87
2.2.5.2 SVG-Graphiken in andere Sprachkonzepte integrieren 89
2.2.5.2.1 SVG-Graphiken in HTML einbinden 89
2.2.5.2.2 SVG-GraphikenüberFormattingObjects(fo)inexterneDokumente transformieren90

3 Das SVG-(Online-)Tutorial 91
3.1 Konzept 91
3.1.1 Zielgruppe 91
3.1.2 Funktionalität des Tutorials 91
3.1.3 Aufbau des Tutorials 92
3.1.4 Gewählte Sprache für's Tutorial 94
3.1.5 Äussere Gestaltung des Tutorials 94

3.1.5.1 Das Layout 95
3.1.5.2 Die Farbwahl 96
3.1.5.3 Die Schrift 96
3.2 Dynamische Umsetzung 96

4 SVG im Einsatz 98
4.1 Graphik 98
4.1.1 Allgemeine Graphik 98
4.1.2 Kartographie und Geoinformationssysteme 98
4.1.3 Technische Zeichnungen 98
4.2 Multimediale Präsentationen 99
4.3 Dokumenation 99

5 Eigene Einschätzungen zu SVG 100

Literatur 101

Index 103

Anhang

Wer sagt was zu SVG? 106

Nützliche Links zu SVG 110

Vorwort

War es früher Gang und Gebe, dass der Geograph neben seinen Forschungsarbeiten zur Beschreibung der Erde das Land vermass und anhand der gewonnenen Daten eine Karte zeichnete, so können wir heute von solch einer Vielseitigkeit nur träumen. Aus einem Beruf wurden mehrere. Und das auch deswegen, weil wir uns vermehrt mit den Mitteln auseinandersetzen müssen, anstelle des eigentlichen Zwecks.

Diesem Problem begegnen wir ganz besonders in der Informationstechnologie, in der wir fast täglich mit neuen Standards, Programmiersprachen oder sonstigen Willkürlichkeiten der Software-Hersteller rechnen müssen. Datei-Formate überfluten uns derart, dass wir mit den Kenntnissen darüber fast nicht mehr hinterher kommen. „Kannst Du mir mal den Unterschied zwischen einem JPEG, einem PNG und einem GIF erklären?“ Diese Frage wurde mir bereits mehrfach gestellt.

Es ist wohl unser aller Anliegen, die Komplexität der heutigen Zeit wenigstens an den Stellen zu vereinfachen, an denen es Sinn macht. Es geht nicht darum, dass wir uns zurückentwickeln, sondern, im Gegenteil, wir sollen uns vorwärts bewegen, dahin, dass wir wieder mehr Zeit für’s Wesentliche haben, das Inhaltliche, das Kreative.

Einen ersten Schritt in diese Richtung macht das World Wide Web Consortium (W3C), das versucht, dem Ideen- und Standardenberg des Web-Bereichs ein einheitliches Fundament zu geben: die e**X**tensible **M**arkup **L**anguage (XML). Sie ist die Wurzel aus der sich immer neue Sprachen zur Lösung alter Probleme entwickeln. Und das Schöne: Sie sind verhältnismässig einfach zu erlernen, und funktionieren alle nach dem gleichen Prinzip.

Mit einem dieser XML-Standards durfte ich mich im Rahmen dieser Diplomarbeit beschäftigen: der Sprache zur Beschreibung vektororientierter Graphiken – **S**calable **V**ector **G**raphics (SVG). Und nach „getaner Arbeit“ kann ich sagen: SVG verdient Ihre Aufmerksamkeit.

1 Einführung

1.1 Die Idee zum Thema

Ist es wirklich die Aufgabe einer angehenden technischen Redakteurin, sich mit einem Graphikstandard auseinanderzusetzen? Unter Berücksichtigung meines zuvor absolvierten Studiums, der Kartographie, kann ich diese Frage eindeutig mit „Ja“ beantworten. Natürlich interessiert es mich, was in der graphischen Welt vor sich geht und noch mehr, wenn es mit XML zu tun hat.

Dass XML auch zur Erstellung von Graphiken taugt, konnte ich zunächst gar nicht glauben. Doch genau darüber kamen meine Professorin Sissi Closs und ich ins Gespräch und die Idee für das Diplomarbeitsthema war geboren: Die Untersuchung des XML-Standards für zweidimensionale Vektorgraphiken **S**calable **V**ector **G**raphics (SVG) im Rahmen des Forschungsprojekts „XML + Directory-Server“ am Institut für Innovation und Transfer (IIT). (Es stand noch eine zweite Idee im Raum, nämlich die Erstellung einer mehr oder weniger automatisierten Kommentierung von DTDs , diese erübrigte sich jedoch aufgrund ausreichend vorhandener Tools.)

Bei meinen ersten „Forschungsschritten“ bemerkte ich nun, dass der Standard zwar Zukunft hat, dass aber ausser der überaus technisch gehaltenen Empfehlung des W3C‘s und einem sicherlich schön gemachten, aber auch relativ oberflächlichen Tutorial des Graphik-Software-Herstellers Adobe nicht viel Literatur zu SVG existierte. Schon gar nicht auf Deutsch. Die Überlegung, über diesen Standard etwas muttersprachliches zu schreiben, lag also auf der Hand. Dass es gerade ein Online-Tutorial sein sollte, lag wiederum in der Natur der Sache: SVG ist in erster Linie ein Online-Standard. Es war den Versuch wert, diesen direkt für den Aufbau einer Online-Dokumentation zu nutzen. Ausserdem lehrte es mich meine bereits erlangte berufliche Erfahrung, dass sich Probleme mit einem Programm oder einer Programmiersprache nicht simulieren lassen. Auf sie muss man in der Praxis stossen.

1.2 Das Projekt „XML + Directory-Server“

Das Projekt „XML + Directory-Server“ (von September 1999 bis Ende Februar 2001) wurde vom Softwarelabor Karlsruhe am IIT durchgeführt. Unterstützung bekam es dabei vom Bundesministerium für Bildung und Forschung. Das Team, unter den betreuenden Händen der beiden Professorinnen Cosima Schmauch und Sissi Closs, setzte sich vorwiegend aus (z. T. ehemaligen) Studenten der Fachbereiche Wirtschaftsinformatik und Sozialwissenschaften (mit dem Studiengang „Technische Redaktion“) zusammen.

Eines der Hauptziele dieses Projekts war zunächst die Untersuchung darüber, ob sich (LDAP-)Directory Server für die Verwaltung von XML-Dokumenten eignen und welche Vor- bzw. Nachteile sich daraus im Vergleich zu anderen Datenbanken-Management-Systemen ergeben. Dazu wurden auch im Rahmen

einer anderen Diplomarbeit zahlreiche Performance Tests durchgeführt.

Ein weiterer Aspekt war die Erstellung einer Document Type Definition (DTD) für die Fachhochschule, deren gesamter Verwaltungsdaten-Bestand darüber für vielseitige Anwendungen, wie Print, Web oder sonstige Ausgabemedien abrufbar gemacht werden sollte.

Schliesslich galt es, XML, den Metastandard selbst, genauer unter die Lupe zu nehmen, samt eines Teils der mittlerweile zahlreich gewordenen Instanz-Sprachen: der e**X**tensible **S**tylesheet **L**anguage (XSL[T] und Formatting Objects), Xlink, XPath, XPointer, XMLQuery und SVG.

Die gesammelten Ergebnisse wurden auf einer eigenen Website, die übrigens selbst aus XML-Dokumenten mit Hilfe der Transformierungssprache XSL(T) in HTML generiert wurde, zusammengestellt (http://www.fbwi.fh-karlsruhe.de/lin02/xmldirectory).

1.3 Voraussetzungen für die Diplomarbeit

1.3.1 Räumlichkeiten

Während der Diplomarbeit hatte ich einen HIWI-Vertrag beim IIT, so dass ich die dortigen Räumlichkeiten nutzen konnte. Der Vorteil lag in erster Linie in der Zusammenarbeit mit dem Team des Projekts, das mir in jedem Fall von menschlicher Seite eine Stütze war.

1.3.2 Hardware

Von Seiten des IIT stand mir zunächst ein PC (500 MHz) mit Pentium Intell III-Prozessor zur Verfügung. Ausserdem testete ich einiges auf meinem eigenen Apple Macintosh (G3/350) aus.

Zu den Ausgabe-Geräten zählten zwei 19''-Bildschirme und diverse Drucker.

1.3.3 Software

Für die Erstellung der Ausarbeitung verwendete ich:

- Microsoft Word 97,
- Adobe FrameMaker + SGML 5.5 und
- Adobe Photoshop 5.5.

Auf einen Grossteil der verwendeten Software für den praktischen Teil – das Tutorial – gehe ich verstärkt im Kapitel *‚SVG-unterstützende Software‘* ein. In der Hauptsache griff ich hierfür auf das Graphik-Programm Adobe Illustrator 9.0 zurück.

Seite 6

2 Scalable Vector Graphics (SVG)

2.1 Überblick

2.1.1 XML und seine Instanzen

XML ist ein Web-Standard, mit dem wir endlich einmal über den Horizont des „momentanen Bedarfs“ hinausblicken können. Es handelt sich dabei um eine Metasprache zur Definition von semantischen Tags für verschiedenste Instanz-Sprachen, die untereinander „kompatibel“ sein sollen. Mit Hilfe dieser Tags strukturieren wir reine Text-Dokumente (ASCII-Text), die frei sind von jeglicher Formatierung. Letztere kann nachträglich von bzw. mit speziell darauf abgestimmter Software vorgenommen werden. Das beste Beispiel hierfür ist das bekanntlich rein text-basierte HTML. Öffnen wir ein solches Dokument im Browser, erscheinen manche Texte fett, andere werden in Tabellen angeordnet usw. Die Software erkennt anhand der Tags was sie wie anzeigen soll.

Für standardisierte XML-Instanzen empfiehlt es sich, die Tag-Definitionen in der sogenannten **D**ocument **T**ype **D**efinition (DTD) vorzunehmen, damit später bei der Erstellung von Instanz-Dokumenten eine einheitliche Struktur besteht. Zum Überprüfen, ob ein solches Dokument auch wirklich auf der Grammatik der DTD basiert (Validierung), werden Parser eingesetzt, die heute schon in den meisten Browsern implementiert sind.

Wichtige Instanzen

Einige Instanz-Sprachen erhalten immer mehr Bedeutung und lassen sich auch kaum noch umgehen. Dazu gehört zunächst die e**X**tensible **S**tylesheet **L**anguage (XSL). Sie wird uns zukünftig wohl einiges an Arbeit abnehmen, da wir damit beispielsweise bei der Erstellung einer Website auf die Auszeichnung immer wiederkehrender Inhalte verzichten können. Mit XSL können wir sie hineingenerieren bzw. transformieren (weswegen der Abkürzung „XSL“ ganz gerne das Anhängsel „T“ für „Transformation“ beigefügt wird). Ausserdem hat XSL noch eine andere Komponente: Formatting Objects. Darunter verstehen wir einen ganzen Katalog von Formatierungselementen, im Sinne der **C**ascading **S**tyle**s**heets (CSS), mit denen wir die äussere Gestaltung (das Layout) beispielsweise von Websites beeinflussen können.

Mit den Standards XLink, XPath und XPointer bekommen wir, im Vergleich zu HTML, neue Möglichkeiten der Verlinkung. Damit wird nicht mehr nur eine Stelle innerhalb des selben Dokuments (Anker) oder eine andere Web-Site angesprochen, es können alle möglichen Stellen in allen möglichen Dokumenten aufgerufen bzw. referenziert werden. Wir können beispielsweise (wie mit dem Online-Hilfe-Programm Microsoft RoboHelp) einzelne „Topics“ in einem Dokument ablegen und von einem anderen aus gezielt abrufen.

In diesem Zusammenhang möchte ich auch den Unterschied der beiden Adressierungsstufen **U**niform **R**essource **L**ocator (URL) und **U**niform **R**essource **I**dentifier (URI) erklären, die uns im Kapitel *‚SVG-Dokumente erstellen‘* hin und wieder begegnen werden. Über die URL lokalisieren wir im Moment noch HTML- sowie XML-Dokumente im Browser. Dabei liegt die Position eines Dokuments im Vordergrund. Diese müssen wir genau angeben, damit das Dokument gefunden und angezeigt werden kann, z. B. *http://www.beispiel.de/verzeichnis/beispiel.xml*. In der XML-Spezifikation wurden darüberhinaus URIs berücksichtigt, mit denen wir noch einen Schritt weiter gehen, da über sie sämtliche Ressourcen, z. B. die Inhalte eines Dokuments, angesprochen werden können. Leider befinden sich URIs noch in einer Art Versuchsstadium, d. h. sie werden wohl noch von keiner Software in vollem Umfang unterstützt (E. R. HAROLD 2000, S. 36f).

2.1.2 Was ist SVG?

Auch SVG ist eine XML-Instanz. Hier beschreiben einfache Text-Dateien aufwendige Vektor-Graphiken einschliesslich deren Animations- und Interaktionsmöglichkeiten. Das wiederum stellt gerade für’s Web eine kleine Revolution dar, zumal damit typische Online-Formate wie Shockwave oder PDF und selbst HTML unter einen „Hut“ gebracht werden können.

Mit SVG können wir wie mit jedem guten Graphikprogramm Zeichnungen erstellen, diese mit Texten kombinieren, transformieren und, was die Fähigkeiten der meisten derartigen Programme (mit Ausnahme von Macromedia Flash und z. T. Adobe Illustrator) übersteigt, Filter anwenden, die wir normalerweise nur aus der digitalen (Raster-)Bildverarbeitung kennen. Im Kapitel *‚SVG-Graphiken erstellen‘* werde ich auf die Gestaltungsmöglichkeiten noch näher eingehen.

Seite 26

Ein ganz grosser Vorteil aller Markup Languages ist die strukturierte Ablage der einzelnen Daten. D. h. wir können gezielt auf einzelne Elemente und Eigenschaften zugreifen, diese abändern oder ersetzen. Für SVG ist das deshalb ein wesentlicher Aspekt, da in der Graphik bereits einfachste Inhalte einen relativ komplexen Grundaufbau haben. Wer sich schon einmal mit der Seitenbeschreibungssprache PostScript auseinandergesetzt hat, weiss, wieviel Seiten Sourcecode schon ein schlichtes Rechteck hervorbringen kann. Und wer versucht hat, in diesem Code Änderungen vorzunehmen, der ist vermutlich daran verzweifelt. Mit SVG haben wir eine semantische Struktur, d. h., die Bezeichnungen der Elemente und Attribute sind aussagekräftig gehalten. Das Tag `<rect>` steht für ein Rechteck, seine beiden Attribute `‚width‘` und `‚height‘` für dessen Ausmasse. Wir finden uns also schnell zurecht und müssen dabei auf den überaus hohen Informationsgehalt von PostScript nicht verzichten. In der Tat haben beide Sprachen viel miteinander gemeinsam. SVG wurde quasi auf der Basis von PostScript entwickelt, was wir später, im Kapitel *‚Beliebige Pfade mit dem Tag <path>‘*, sehen werden.

Seite 40

2.1.3 Zur Geschichte von SVG

Erste Konzepte für die Sprache wurden bereits 1998 erarbeitet: Im April reichten Adobe, IBM, Netscape und Sun beim W3C eine Note über die **P**recision **G**raphics **M**arkup **L**anguage (PGML) ein. Einen Monat später lag ein Vorschlag der Firmen HP, Microsoft, Macromedia und Visio zur **V**ector **M**arkup **L**anguage (VML) auf dem Tisch des Konstortiums. Schliesslich entstand als Folgeprodukt aus beidem im Oktober 1998 SVG.

2.1.4 Die Recommendation vom W3C

Die vom W3C ins Leben gerufene SVG-Working Group, zu denen zahlreiche führende Firmen der Computer-Industrie gehören, (u. a. Adobe, Apple, Autodesk, BitFlash, Corel, HP, IBM, ILOG, INSO, Macromedia, Microsoft, Netscape, OASIS, Open Text, Quark, RAL [CCLRC], Sun, Visio, Xerox), arbeiten an der Plaunung dieses Standards.

Im Moment liegt SVG in der Candidate Recommendation vor (Stand: 2. November 2000), d. h. einem Status, in dem sich noch einiges ändern kann.

Die über 500 DIN A4-Seiten der Empfehlung, die bereits auf die Komplexität des Standards schliessen lassen, umfassen, neben den technischen Angaben zum Aufbau von SVG-Strukturen, die DTD und spezielle SVGDOM-Interfaces, die der Implementierung in Programmiersprachen wie Java, JavaScript oder ECMAScript dienen.

Seite 82

2.1.5 Reaktionen auf SVG

Die grundsätzlichen Meinungen zu SVG besonders seitens der Software-Hersteller sind sehr positiv. Und das obwohl sich die Recommendation, wie wir gerade vernommen haben, in einem nicht endgültigen Status befindet.

Dennoch: Glaubt man einigen Einschätzungen, wird sich gerade der sicherlich grösste Konkurrent Macromedia Flash, der mittlerweile in der 5. Version erschienen ist, gegenüber SVG noch eine Weile lang behaupten. Einige Reaktionen habe ich in Form von Presse-Mitteilungen und Website-Ausschnitten im Anhang aufgeführt (*‚Wer sagt was zu SVG?‘*).

Seite 103

2.1.6 SVG-unterstützende Software

Die generell sehr positiven Reaktionen der Software-Hersteller spiegeln sich in der Anzahl der Produkte wieder. Dabei stelle ich eine recht breitgefächerte Auswahl fest, sowohl, was die Preispalette angeht, von Freeware bis hin zu kommerziellen Programmen, als auch in punkto Plattformen und Branchenspezifik. Es findet sich bereits heute schon für fast jeden Bedarf das Richtige.

2.1.6.1 Aktive (verarbeitende) Programme

Zu den aktiven Programmen zähle ich ebenso Programme, die SVG-Dokumente erzeugen (exportieren), wie auch solche, in die wir SVG-Dateien einlesen

(importieren) können. Das ist eigentlich auch der einzige Bereich, in dem wir auf kostenpflichtige Software treffen, jedoch ist die bereits vorhandene Freeware ebenfalls sehr „produktiv".

Zur Analyse der aktiven Programme habe ich folgende Fragen berücksichtigt:

- Auf welchen Plattformen läuft das Programm, PlugIn etc.?
- Auf welche Anwendergruppe ist das Tool zugeschnitten?
- Wie arbeiten wir mit dem Tool (WYSIWYG-, Text-Editor etc.)?
- Ist das Programm kommerziell, Free- oder Shareware?
- Hat das Tool SVG-Im- *und* Exportmöglichkeiten?
- Wie strukturiert legt das Tool die SVG-Daten ab?
- Welche Tool-Unterstützung haben wir z. B. für Verlinkung und Scripts?
- Welche Features bietet das Programm darüberhinaus (Schnittstellen zu anderen Programmen, sonstige Im- und Exportmöglichkeiten etc.)?

2.1.6.1.1 Adobe Illustrator®

(Mit diesem Programm habe ich das SVG-Tutorial erstellt, weswegen ich hier auch auf die meisten Bugs bzw. Vorteile gestossen bin. Das erklärt, weshalb ich auf dieses Programm ausführlicher eingehen werde als auf die nachfolgenden.)

Mit Illustrator kommt ganz bestimmt die Haupt-Zielgruppe von SVG zu ihren Gunsten, die Graphiker (inclusive Kartographen). Dieses Programm ist neben Macromedia Freehand eines der am weitest verbreiteten Zeichenprogramme im professionellen Bereich. Sein Hauptvorteil gegenüber dem Konkurrenten liegt wohl in der hybriden Funktionalität: Es kann sowohl Vektoren verarbeiten wie auch Photoshop-Filter auf Rasterdaten anwenden. (Generell ist die Zusammenarbeit mit dem aus gleichem Hause stammenden Bildverarbeitungsprogramm ausgesprochen gut.)

Mit der Version 9.0 wird nun ein SVG-Export-PlugIn ausgeliefert (funktioniert übrigens auch mit Version 8.0), durch das das Programm einen echten Pluspunkt erhält. Dennoch, gerade an dieser Funktionalität gibt es noch einiges zu verbessern.

Nachteile

Zunächst fällt die unstrukturierte Ablage von Stylesheets, Vektoren und Texten auf.

- Zwar können wir, besser sogar als mit Freehand, eine hierarchische Ebenenstruktur aufbauen, deren Benennungen auch in die ID der SVG-Elemente bzw. Gruppen übernommen werden, aber die Namen der Formatvorlagen (in Illustrator „Stile" genannt) werden nicht als CSS-Bezeichner eingetragen, was vielleicht auch darauf zurückzuführen ist, dass wir in Illustrator keine Textformatvorlagen anlegen können (nur reine Graphik-Stile). So erhalten wir die unschönen Klassen-Bezeichner „.st0" hochgezählt bis

„.st[n]“. Die Properties selbst werden offensichtlich nach bestimmten programm-internen Routinen in immer gleichen Schemen abgelegt. Generell werden zahlreiche Property-Definitionen eingebunden, die eigentlich der Standardeinstellung entsprechen. Nachfolgend haben wir ein Beispiel, in dem vorwiegend solche Standardeinstellungen definiert wurden:

```
.st8{fill-rule:nonzero;clip-rule:nonzero;fill:#FFFFFF;
     stroke:#000000;stroke-miterlimit:4;}
```

Kurzum, auf die Ablage von Stylesheets haben wir keinen Einfluss.

- Nicht viel anders sieht es mit Vektoren aus. Egal, ob wir in Illustrator ein Rechteck, einen Kreis oder ein Polygon zeichnen, beim SVG-Export gehen die Informationen darüber verloren. Wir erhalten ausschliesslich allgemeine Pfade (siehe Kapitel *‚Beliebige Pfade mit dem Tag <path>‘*). Diese blähen den Sourcecode auf, da zur Beschreibung derart ausgezeichneter Vektoren viel mehr Einzel-Informationen benötigt werden. Nachfolgend sehen wir die Illustrator-Konvertierung eines einfachen Rechtecks:

Seite 40

```
<path class="st17" d="M150.333,84.278H0V4h150.333v80.278z"/>
```

In der Pfadbeschreibung (Attribut ‚d‘) haben wir ausserdem den inkonsequenten Aufruf von relativen (Kleinbuchstaben) und absoluten Positionsangaben (Grossbuchstaben) der Vektorknoten.

- Bei Texten werden Sonderzeichen in Entities (Unicode) konvertiert. Diese erhalten auch noch eine separate Auzeichnung, was nicht optimal ist, da für jeden ausgezeichneten Text die Koordinaten-Positionen explizit angegeben werden müssen. Das wiederum erschwert die Nachbearbeitung von Texten (siehe Kapitel *‚Texte auszeichnen‘*).

Seite 45

(Übrigens, ID-Bezeichnungen, die von der Ebenen-Bennenung aus Illustrator übernommen wurden, behalten nach dem Export ihre Sonderzeichen.)

- Eingebettete Fonts werden von Illustrator als Binär-Daten in den Sourcecode hineingeschrieben, was von validierenden Parsern wohl kaum verstanden werden dürfte. Hier wäre es besser, wenn SVG-Fonts erzeugt würden (siehe Kapitel *‚Fonts‘*).

Seite 52

- Darüberhinaus ist es nachteilig, dass Filtereffekte, die wir in Illustrator auch auf Vektorgraphiken anwenden können, beim Export in separate Bilddaten konvertiert werden und nicht, wie man es erwarten könnte, als SVG-Filtereffekte (siehe Kapitel *‚Filter‘*).

Seite 65

- Schliesslich haben wir keine Tool-Unterstützung in Bezug auf Hyperlinks.

Vorteile

Es gibt aber auch einige positive Aspekte: Dazu gehören vor allem diverse Einstellungsmöglichkeiten, die uns vor dem Export zur Verfügung stehen. So können wir angeben,

- ob Zeichensätze

- ➔ mit dem Systemzeichensatz des Users referenziert oder
- ➔ in extern abgelegte, stark komprimierte Fonts (Compact Embedded Font-Format) konvertiert werden sollen (siehe Kapitel ‚*Fonts*').

Seite 52

- mit welchem Zeichenstandard das spätere SVG-Dokument verbunden wird:
  - ➔ dem ISO-8859-1-Standard (Latin-1),
  - ➔ dem UTF-8-Standard oder
  - ➔ dem UTF-16-Standard.
- in welcher Form Stylesheets abgelegt werden sollen:

Seite 26

  - ➔ als Entity-Referenzen (im Doctype des SVG-Dokuments),
  - ➔ als Stilelemente (CSS) oder
  - ➔ als Stilattribute (Attributsauszeichnung).
- Ausserdem haben wir die Möglichkeit, externe Scripts (*.js) einschliesslich Event-Handler mit Zeichenobjekten zu verbinden.
- Für den Ex- und Import von geocodierten GIS- oder CAD-Daten ist für Illustrator eine optionale Schnittstelle erhältlich: MaPublisher von Avenza.

Ich fasse zusammen:

Adobe Illustrator® 9.0 (für MacOS und Windows, Windows NT)

Kommerzielles Graphikprogramm mit WYSIWYG-Funktionalität

[-] CSS-Bezeichner werden nicht-semantisch abgelegt.
[-] CSS-Properties werden sogar mit Standardeinstellungen abgelegt.
[-] Grundformen (Rechtecke, Kreise etc.) werden nicht als solche konvertiert.
[-] Vektorknoten erhalten inkonsequenter Weise absolute und relative Positionsangaben.
[-] Sonderzeichen von Texten werden als Entity abgelegt und separat ausgezeichnet.
[-] Eingebettete Fonts werden in Binär-Daten umgewandelt.
[-] Eingebettete Bilder werden in Binär-Daten umgewandelt.
[-] Illustrator-Effekte werden als Bilder anstatt als SVG-Filter abgelegt.
[-] Nur der Export von SVG-Daten ist möglich (kein Import).
[-] Links können nicht zugewiesen werden.

[+] Ebenen-Bezeichnungen werden in die ID der Gruppe übernommen.
[+] Es gibt diverse Auswahlmöglichkeiten beim Export in Bezug auf Fonts, Bilder, Zeichen-Standards und Stil-Ablagen.
[+] Scripts können zugewiesen werden.
[+] Es gibt eine optionale Schnittstelle zum Ex- und Import von GIS- und CAD-Daten: Avenza MaPublisher.

Fazit: Es gibt momentan bessere Programme, aber ich denke, da es sich bei Adobe doch um einen etablierten Software-Hersteller handelt, wird sich da auch noch einiges tun. Gerade die zuletzt genannte MaPublisher-Schnittstelle stellt in jedem Fall einen wirklich grossen Vorteil dar.

2.1.6.1.2 CorelDraw®

Das Zeichenprogramm, das wir im Gesamtpaket mit diversen anderen Software-Produkten, u. a. für Bildbearbeitung und Charts, erhalten, spricht ebenfalls in der Hauptsache Graphiker an. Des relativ günstigen Preises wegen findet es jedoch gerade bei Home-Anwendern grossen Absatz.

Für die Versionen 9 und 10 erhalten wir nun auch einen SVG-Im- und Export-Beta-Filter (getestet wurde er für Version 9).

Nachteile

- Generell ist es nachteilig, dass wir für Formatvorlagen (Styles) keine Namen vergeben können. Somit haben wir hier ebenfalls keinen Einfluss auf die Bennenung der CSS-Bezeichner, auch wenn diese immerhin schon in drei Klassen aufgeteilt werden: ‚`.fil`' für Füllungsattribute, ‚`.str`' für Linienattribute und ‚`.fnt`' für Textattribute.
- Der grösste Nachteil, der mir beim Test auffiel, war die Verwendung eigenkreierter Attribute beim Exportieren von eingetteten SVG-Fonts (siehe Kapitel ‚*Fonts*'). Diese werden von validierenden Parsern natürlich nicht unterstützt.

Seite 52

- Es gibt zwar in CorelDraw die Möglichkeit, mit VisualBasic (for Applications) die Funktionalität des Programms zu erweitern und darüberhinaus mit einer Corel-eigenen Sprache Scripts mit Graphiken zu verknüpfen, aber auf SVG hat das keine Auswirkung. Folglich haben wir für den Standard keine Scripting-Unterstützung.

Vorteile

- Wenn auch die Umsetzung fragwürdig ist (s. o.), so kann ich wenigstens den Ansatz positiv bewerten, beim Einbetten auf SVG-Fonts zurückzugreifen.
- Im Ggs. zu Illustrator fällt die sehr strukturierte Ablage der Vektoren auf: Grundformen (Rechtecke, Kreise usw.) werden auch als solche behandelt.
- Wir können festlegen, ob Styles
  - ➔ als CSS in den Sourcecode eingebunden,
  - ➔ in ein externes CSS hineingeschrieben oder
  - ➔ in die Attributsauszeichnung übernommen

  werden sollen.
- Wir können für die Konvertierung von Bildern mehrere Formate angeben:
  - ➔ JPEG,
  - ➔ GIF oder
  - ➔ PNG.
- Schliesslich haben wir die Möglichkeit über eine Eingabemaske Hyperlinks zu setzen.

Ich fasse zusammen:

CorelDraw® 9 / 10 (Beta-Filter für Windows und Windows NT)

Kommerzielles Graphikprogramm mit WYSIWYG-Funktionalität

[-] CSS-Bezeichner werden nicht-semantisch abgelegt.
[-] SVG-Fonts werden mit eigen-definierten Attributen abgelegt.
[-] Es gibt keine Scripting-Möglichkeiten in von SVG unterstützten Programmiersprachen.

[+] Es gibt diverse Auswahlmöglichkeiten beim Export in Bezug auf Fonts, Bilder und Stil-Ablagen.
[+] Grundformen (Rechtecken, Kreisen etc.) werden auch als solche abgelegt.
[+] Ebenenbezeichnungen werden in die ID der Elemente übernommen.
[+] Tool-unterstützte Verlinkung ist möglich.
[+] Im- und Export von SVG-Daten ist möglich.
[+] Das Programm selbst bietet diverse Im- und Exportmöglichkeiten, herausheben möchte ich an der Stelle besonders AutoCAD (.dxf)

Fazit: Wenn ich für die Erstellung des Tutorials heute noch einmal wählen könnte, würde ich auf dieses Programm zurückgreifen.

2.1.6.1.3 Jasc WebDraw™

WebDraw (früher Trajectory Pro) aus dem Hause Jasc ist ein ausschliesslich auf den Export von SVG-Graphiken ausgerichtete Tool. Die Funktionalitäten sprechen wieder einmal in erster Linie Graphiker an.

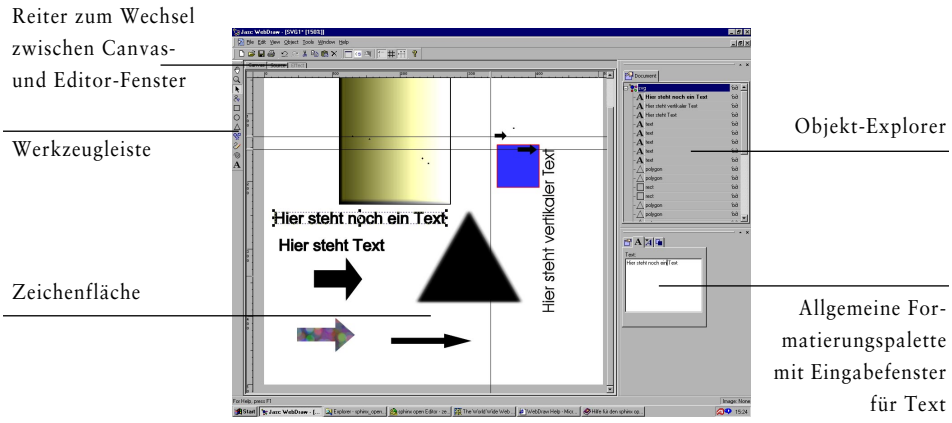


Abb. 2-1: Die WebDraw-Oberfläche

Da es sich um eine Freeware handelt, verblassen die Nachteile geradezu gegenüber den Vorteilen. Hier bedarf es auch keiner weiteren Erläuterungen mehr.

Jasc WebDraw™ Preview Release 4 (für Windows / Windows NT)

Graphikprogramm (Freeware) mit WYSIWYG-Funktionalität und Editor

[-] Styles werden ausschliesslich in die Attributsauszeichnung eingebunden.
[-] Die Texteingabe ist etwas schwerfällig und muss über ein Eingabefenster vorgenommen werden.
[-] Im Objekt-Explorer können einzelne Layer nicht vertauscht werden.
[-] Die Online-Hilfe des Programms ist etwas dürftig.

[+] Es besteht die Wahl zwischen WYSIWYG- und Text-Editor.
[+] Grundformen (Rechtecke, Kreise etc.) werden auch als solche abgelegt.
[+] Zahlreiche vordefinierte SVG-Filter lassen sich auf Graphiken und Texte anwenden.
[+] Im- und Export von SVG-Graphiken ist möglich.
[+] Tool-unterstützte Verlinkung über ein Eingabefenster ist möglich.
[+] Die Graphiken können als JPEG oder BMP exportiert werden.

Fazit: Für eine Freeware haben wir hier schon erstaunlich hohe Funktionalität (fast die eines kommerziellen Graphikprogramms). Wenn es auch für wirklich anspruchsvolle Arbeiten nicht ausreicht.

2.1.6.1.4 Sphinx™ open

Mit dem Programm der übrigens deutschen Firma in-GMBH kommen gerade auch technische Zeichner (Import von AutoCAD [.dxf]) zu ihren Gunsten. Es ist auch als Tool für die graphische Online-Überwachung von Datenbeständen, Prozessabläufen und Systemen in Firmen der Gebäudetechnik, der Verkehrsleittechnik, der Energieversorgung etc. vorgesehen.

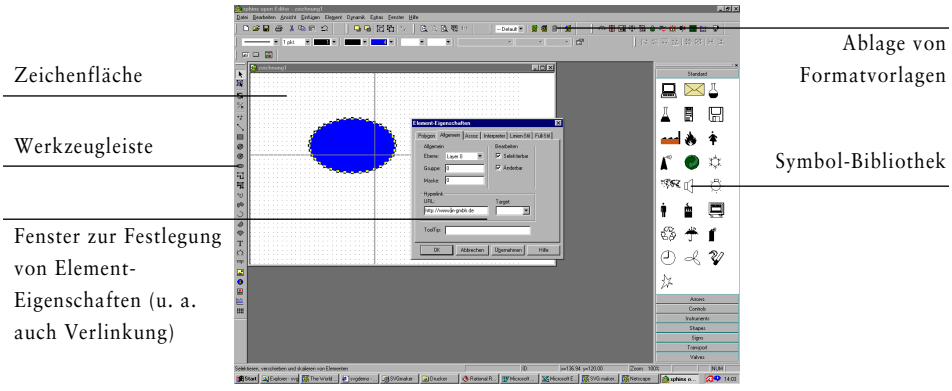


Abb. 2-2: Die Sphinx open-Oberfläche

Besonders fällt hier die breite Plattform-Unterstützung auf:

in-GMBH Sphinx™ open (Demo-)Version 6.0 beta
(für UNIX, Windows NT, Java-Umgebung)

Visualisierungstool mit WYSIWYG-Funktionalität
(Demo-Version ist Freeware)

[-] Styles werden ausschliesslich in die Attributsauszeichnung eingebunden.

[+] Grundformen (Rechtecke, Kreise etc.) werden auch als solche abgelegt.
[+] Animationen lassen sich über ein Auswahlfenster hinzufügen.
[+] Elemente lassen sich CAD-ähnlich strukturieren.
[+] Tool-unterstützte Verlinkung über ein Eingabefenster ist möglich.
[+] Die Software bietet eine Symbol-Bibliothek an.
[+] Das Programm bietet diverse Exportmöglichkeiten: PS, EPS, PNG, HTM und SVG.
[+] Dateien lassen sich als sogenannte ‚Sphinx open SVG-Edition' (.gsv) speichern.
[+] Das Programm bietet bereits eine recht gute Online-Hilfe.

Fazit: Sphinx open ist eigentlich schon für anspruchsvollere Visualisierungen mit automatisierten Animationen gemacht. Leider konnte ich nur die Demo-Version testen, denn, die Vollversion soll noch einige zusätzliche Features wie die Unterstützung von ActiveX und JavaBeans umfassen.

2.1.6.1.5 SVGMaker (Druckertreiber)

Der grundsätzliche Vorteil dieses Druckertreibers der Firma Software Mechanics liegt darin, dass wir damit aus jedem Windows-Programm SVG-Dateien in „Print-Files drucken" können. Leider gibt es auch ganz erhebliche Nachteile:

Software Mechanics SVGMaker Version 1.00.1130
(für Windows 2000 / Windows NT)

Druckertreiber
(Demo-Version ist Freeware – hier wird allerdings ein Demo-Banner „aufgedruckt")

[-] Alle Schriften werden in Pfade konvertiert!!!!!
[-] Der Treiber legt alles in unstrukurierten Pfaden ab.

[+] Es lassen sich aus jedem Programm SVG-Dateien erzeugen (Output-Formate: *.svg und *.svgz [komprimiertes SVG-Format]).

Fazit: Der Ansatz ist sehr gut. Damit wäre fast jedes Problem in Sachen „SVG-Erstellung" gelöst. Die Umsetzung hingegen ist nur etwas für Anspruchslose.

Nachfolgend sehen wir eine mit dem SVGMaker (Demo) konvertierte UML-Graphik aus Rational Rose:

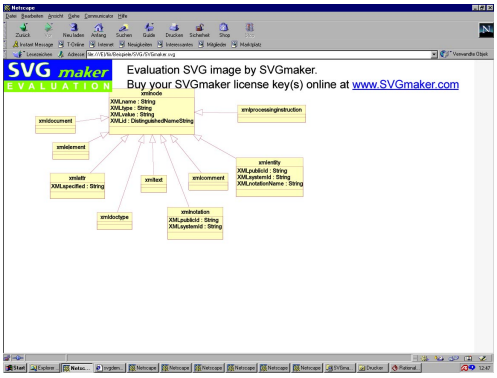


Abb. 2-3: Beispiel: Über SVGMaker konvertierte Rational Rose-Datei

2.1.6.1.6 Weitere Programme

Es gibt noch zahlreiche weitere Programme, die ich auch zum Teil getetstet habe, deren Ausarbeitung aber an der Stelle zu weit gehen würde. Daher beschränke ich mich auf die Auflistung einiger Features:

pstoedit (GeoCities)

Hierbei handelt es sich um ein kleines Batch-Prgrämmchen für Windows 9.x und Windows NT, das die Konvertierung von PostScript- und PDF-Dateien in SVG (und andere Formate) ermöglicht (vgl. SVGMaker).

PICT to SVG (von Chuck Houpt)

Mit dem Mac-Prgrämmchen lassen sich PICT-Path-Dokumente durch „Draufziehen“ auf das Programm-Icon in SVG-Dateien konvertieren.

SVG2PDF (Digital Applications Inc.)

Dieses Batch-Programm zeichnet sich zunächst durch die relativ breite Plattform-Unterstützung aus: Windows 9.x, 2000, NT und MacOS. Vermutlich funktioniert es auch auf allen Plattformen gleich gut (nicht getestet: Windows 2000). Hiermit werden SVG-Dateien in PDF-Dateien konvertiert.

Catwalk (SchemaSoft)

Da von diesem Programm bisher nur ein Prototyp vorgestellt wurde, konnte ich es selbst nicht testen. Es verspricht aber, interessant zu werden: Catwalk ist ein XSL(T)-Generator, mit dem sich intuitiv (ohne Programmierkenntnisse) XSL-Stylesheets und -Transformationen erstellen lassen. Dabei liegt das Hauptaugenmerk in der Erstellung von XSL(T)-Dokumenten, die während des Parsens beliebige XML-Daten in SVG-Graphiken transformieren.

SVGObjects für WebObjects (Apple)

Ebenfalls nicht getestet, aber interessant ist SVGObjects für Apple’s Application Server WebObjects. Dabei handelt es sich um ein Set aus Elementen zum serverseitigen Generieren von SVG-Graphiken sowie rudimentären Animatio-

nen und PDF-Dateien.

2.1.6.2 Passive Programme (Viewer)

2.1.6.2.1 Adobe SVG-Viewer

Der Stand dieses Viewers kann sich momentan noch täglich ändern, so dass sich meine Tests nur auf die beiden Versionen 1.0 und 2beta (release 3) beschränken. Die aktuellste Version liegt unter http://www.adobe.com/svg/viewer/install/ zum kostenlosen Download bereit.

Mit dieser Software bekundet Adobe ihr echtes Interesse an der Fortentwicklung von SVG. Von allen momentan existierenden Viewern schreibe ich diesem die vielversprechenste Zukunft zu, zumindest solange es noch keine vergleichbare SVG-Unterstützung von Standard-Browsern gibt. Es handelt sich dabei um ein PlugIn (für Windows und MacOS), das die beiden grössten Browser Netscape Navigator und Internet Explorer so erweitert, dass wir SVG-Graphiken direkt im Browser-Fenster anschauen können.

Der Hauptvorteil liegt in der Funktionalität, die uns der Viewer selbst bietet. Dazu gehören ebenso die Zoom- und Verschiebe-Möglichkeiten, wie die Finden-Funktion, mit der wir direkt nach SVG-Texten suchen können. Nachfolgend sehen wir das Sub-Menue, das wir mit gedrückter *rechter Maustaste* oder (beim Mac) über die Kombination *‚ctrl‘ + gedrückte Maustaste* erhalten:

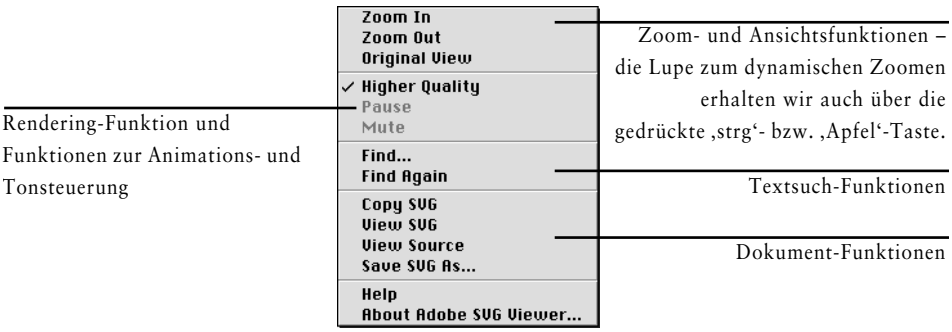


Abb. 2-4: Funktionalitäten des Adobe SVG-Viewers

Zum Navigieren auf einer SVG-Graphik müssen wir die *‚alt‘*-Taste drücken. Damit erhalten wir die *Verschiebe-Hand*, die wir auch in zahlreichen Graphikprogrammen finden.

Schliesslich unterstützt der Viewer bereits einen sehr grossen Teil der vom W3C vorgegebenen Empfehlungen. Auf die wenigen Einschränkungen werde ich detailliert im Kapitel *‚SVG-Graphiken erstellen‘* zurückkommen (in den betreffenden Abschnitten unter „Software-Hinweis“). Eine generelle Einschränkung möchte ich allerdings bereits hier erwähnen: Unter MacOS unterstützt der Viewer für den Internet-Explorer 5.0 noch keine Scripts.

2.1.6.2.2 IBM SVG-View

Die bis dato existierende (Alpha-)Version 0.4a lässt sich kostenlos von der Al-

phaworks-Website herunterladen (leider nur für Windows). Dabei handelt es sich um einen in Java geschriebenen Standalone-Viewer. Er enthält neben der reinen Anzeige von SVG-Dateien auch eine graphische, interaktive Ansicht des XML-DOM-Baums.

Generell ist das Programm sehr klar strukturiert und damit auch einfach zu bedienen (wenn auch die Java-typische Installation gewöhnungsbedürftig ist). Dennoch, für den anspruchsvollen Gebrauch ist der Viewer noch nicht geeignet, da beispielsweise DOM und Animationen kaum unterstützt werden.

2.1.6.2.3 Csiro SVG Toolkit

Dieser SVG-Viewer ist mit einer Open-Source-Lizenz erhältlich. Er ist ebenfalls recht einfach zu bedienen. Und auch hier müssen wir einschränken, dass der Viewer noch kein DOM unterstützt und Animationen nur z. T. wiedergibt.

Anmerkung: Zahlreiche weitere Software-Produkte finden wir in der SVG-Spezifikation aufgelistet (unter http://www.w3.org/Graphics/SVG/SVG-Implementations).

2.2 SVG-Graphiken erstellen

In den nachfolgenden Kapiteln werde ich den Inhalt des praktischen Teils der Arbeit, des Tutorials, aufführen. Es handelt sich gewissermassen um ein Handbuch, dessen einzelne Unter-Kapitel wie in einem solchen für sich stehen sollen. D. h., der Benutzer muss einen Bereich nachschlagen und verstehen können, ohne die anderen Kapitel bis ins Detail zu kennen. Das hat wiederum zur Folge, dass auf wichtige Einzelheiten redundant hingewiesen wird.

Ein weiterer Punkt, den ich anmerken möchte, ist der Bildreichtum, des Handbuchs. Er ist meines Erachtens notwendig, da es sich um eine didaktische Arbeit handelt, die klar und eindeutig vermitteln soll. Und wie lässt sich ein Graphikstandard besser vermitteln als über Graphiken und Bilder?

Schliesslich sind in dieser hiesigen Ausarbeitung gegenüber dem Tutorial einige Ergänzungen gemacht worden, die dort aus verschiedenen Gründen weggelassen wurden:

- Die momentan verfügbaren Adobe SVG-Viewer, auf die das Tutorial abgestimmt wurde, unterstützen noch nicht alle vom W3C vorgeschlagenen Elemente bzw. Attribute, weswegen sie nicht angezeigt werden können, da das Tutorial mit reinem SVG umgesetzt wurde.
- XML-Standards wie XLink und XPointer werden ebenfalls nicht komplett unterstützt.
- Nicht alle Elemente und Attribute sind für den „Normalgebrauch“ von SVG wirklich sinnvoll oder können gar durch bessere Alternativen ersetzt werden.
- Die Zeit für die Arbeit war sehr knapp bemessen.

Dafür habe ich in dieser Ausarbeitung auf die Lektion ‚XML' des Tutorials verzichtet, da ich die Kenntnisse über die Metasprache voraussetze.

2.2.1 Grundlagen

2.2.1.1 SVG

Wie wir gehört haben ist SVG eine XML-Instanz, d. h., es wurde zunächst eine spezielle DTD geschaffen, die die Grundregeln für SVG enthält und die wir beim Erstellen einer SVG-Graphik berücksichtigen sollten. Da es sich um ein Graphikformat handelt, gibt es für uns zwei verschiedene Nutzungsvarianten:

- als Datei, die in ein übergeordnetes Dokument (z. B eine HTML-Site) eingebunden wurde oder
- als eigenständige Datei, die wir wie eine HTML-Site lokalisieren und verlinken können.

Generell tragen SVG-Dateien das Suffix '.svg'.

2.2.1.1.1 Das Tag <svg>

Das Root-Tag, mit dem wir jedes SVG-Dokument beginnen heisst `<svg>`.

Die Syntax für das Root-Tag `<svg>` lautet:

```
<svg width="[SVG-Fensterbreite]" height="[SVG-Fensterhöhe]">
```

Werte zu ‚width' und ‚height':
(Massangaben in jerder beliebigen Einheit)

2.2.1.1.2 Das Grundgerüst eines SVG-Dokuments

Das Grundgerüst eines SVG-Dokuments sieht folgendermassen aus:

```
[1] <?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
[2] <!Doctype svg Public "-//W3C/DTD SVG 20001102 Stylable//EN"
     "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-
     20001102.dtd">
[3] <svg width="21cm" height="29.7cm" xml:space="preserve">
    ...
[4] </svg>
```

Erläuterungen:

[1] Der XML-Prolog enthält die Attribute ‚`encoding`' und ‚`standalone`'. Mit ‚`encoding`' geben wir einen Zeichenstandard an (in diesem Fall den ISO-8859-1-Standard), auf den das Dokument zurückgreift, damit auch Sonderzeichen wie Umlaute im Quelltext verwendet werden können. Mit `standalone="no"` drücken wir die Abhängigkeit von einem anderen Dokument aus, in dem Fall der DTD.

[2] Im Doctype verweisen wir auf die SVG-DTD, die öffentlich abgelegt ist (‚`Public`').

[3] Über die Attribute ‚`width`' und ‚`height`' legen wir die Ausmasse für

das SVG-Fenster fest (hier: 21 cm x 29.7 cm). Mit ‚xml:space' können wir angeben, ob die sog. Whitespaces (Leerschläge, Wagenrückläufe etc.) innerhalb des Source-Codes berücksichtigt, d. h. im Browser angezeigt werden (‚preserve') oder nicht (‚default').

2.2.1.2 Kommentare

Ein wichtiger Bestandteil einer jeden Programmier- oder Auszeichnungssprache ist der Kommentar, eine Dokumentbeschreibung im Source-Code, die vom Parser ignoriert wird. Ihn sollten wir so oft wie möglich einsetzen, da er der Übersichtlichkeit dient. Das ist vor allem dann wichtig, wenn wir nach längerer Zeit ein Dokument überarbeiten wollen. Für Auszeichnungssprachen sieht die Kommentar-Syntax folgendermassen aus:

```
<!-- Dies ist ein Kommentar -->
```

Sie gilt gleichermassen für DTDs wie für Instanzen.

Zwischen den Kommentarzeichen ‚<!--' und ‚-->' können alle beliebigen Zeichen stehen, mit Ausnahme von ‚--', da diese das Ende eines Kommentars markieren.

2.2.1.2.1 Strukturierungselemente in SVG

<svg> unterstehen diverse Child-Tags, die ausschliesslich der Strukturierung des Dokuments dienen. Die wichtigsten sind:

- <g> für die Gruppenbildung von mehreren Zeichenobjekten,
- <defs> für die Definition von Referenzobjekten, Texten und Effekten, die an anderer Stelle aufgerufen werden sollen,
- <title> sowie <desc> für Dokumentbeschreibungen, ähnlich den Kommentaren und
- <use> für die Referenzierung von Zeichenobjekten oder Texten an einer anderer Stelle im Dokument.

Beispiel:

```
[1]  <?xml version="1.0" encoding="iso-8859-1"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN""http://www.w3.org/
      TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
[3]  <svg width="8cm" height="3cm">
[4]   <title id="titel">Titel Dokument-Struktur</title>

[5]   <!-- ******** Es folgt eine Definition ******** -->
[6]   <defs id="definitionen">
[7]    <desc id="beschreib">Hier werden Definitionen für Referenzobjekte
        vorgenommen</desc>
[8]     <linearGradient id="verlaufs_defs">
[9]       <stop offset="20%" style="stop-color:#39F"/>
[10]      <stop offset="90%" style="stop-color:#F3F"/>
[11]    </linearGradient>
[12]    <rect id="pfad_defs" x="1cm" y="1cm" width="6cm"
         height="1cm"/>
[13]  </defs><!-- Ende der Definition -->
```

```
[14]   <!-- ******** Es folgt eine Gruppe ********* -->
[15]   <g id="gruppe">
[16]     <use xlink:href="#pfad_defs" style="fill:url(#verlaufs_defs)" />
[17]     <text x="1cm" y="2cm">Text</text>
[18]   </g><!-- Ende der Gruppe -->
[19] </svg>
```

Erläuterungen:

[4] Das Tag <title> können wir in einem SVG-Dokument, im Ggs. zu HTML, mehrfach und auch innerhalb zahlreicher anderer Elemente, wie beispielsweise <g> verwenden.

[6-13] Mit <defs> leiten wir Definitionen ein. In diesem Fall handelt es sich um die eines Verlaufs und eines Rechtecks, das an einer anderen Stelle referenziert werden soll.

[7] Auch <desc> können wir wie <title> mehrfach verwenden.

[15] Die „Gruppenbildung“ setzen wir ein, damit wir beispielsweise Attribute, die bei allen Einzelkomponenten gleich sind, nur einmal <g> zuweisen müssen. Diese werden dann an die Child-Elemente „vererbt“, was dementsprechend den Sourcecode verkürzt.

[16] <use> weisen wir hier mit Hilfe des Attributs ‚xlink:href‘ das oben definierte Rechteck über den eindeutigen Identifikator ‚id‘ zu. Gleichzeitig füllen wir dieses Rechteck mit dem Verlauf über das Attribut ‚style‘, das wir im Kapitel ‚*Formatierungen*‘ noch ausführlich behandeln werden.

Seite 28

Über das Tutorial hinaus:

Nachfolgend werden drei Elemente beschrieben, die in der SVG-Recommendation an entsprechender Stelle (‚*Document Structure*‘) aufgeführt werden:

- switch und
- symbol.
- image

Seite 21

2.2.1.2.2 System- und Ressourcen-Einstellungen für die Wiedergabe von Graphiken ermitteln

Aufgrund der uneinheitlichen Systemvoraussetzungen der vielen Client-Rechner dieser Welt stehen Website-Entwickler immer wieder vor dem Problem, ihre Seiten auf möglichst jeden Rechner, jede Plattform usw. abzustimmen. Wir benötigen Wege, die Einstellungen des Client-Rechners zu ermitteln, um festzustellen, ob ein Inhalt überhaupt angezeigt werden kann und wenn ja, wie er angezeigt wird.

In SVG gibt es das Tag <switch> samt den Attributen ‚requiredExtensions‘, ‚requiredFeatures‘ und ‚systemLanguage‘, mit denen wir solche Systemvoraussetzungen abfragen können. Dazu gehört beispielsweise auch, ob eine andere Auszeichnungssprache vom verwendeten Browser unterstützt wird. Ist das der Fall, wird der Wert ‚true‘ an <switch> zurückgegeben. Andernfalls der

Wert ‚false'. `<switch>` seinerseits vererbt den jeweiligen Wert an sein *erstes* Child-Element. Ist der Wert ‚true', wird dieses angezeigt, andernfalls wird alternativ auf das *zweite* Child-Element „umgeschaltet" (engl. ‚switch').

Beispiel (aus der Recommendation):

```
[1]  <?xml version="1.0" standalone="yes"?>
[2]  <svg width="4in" height="3in" xmlns = 'http://www.w3.org/2000/svg'>
[3]  <desc>This example uses the switch element to provide a fallback
      graphical representation of an equation, if XMHTML is not supported.
[4]  </desc>
[5]  <!-- The <switch> element will process the first child element
      whose testing attributes evaluate to true.-->
[6]   <switch>
[7]   <!-- Process the embedded HTML if the requiredExtensions attribute
       evaluates to true (i.e., the user agent supports XHTML embedded
       within SVG). -->
[8]    <foreignObject width="100" height="50" requiredExtensions="http://
        example.com/SVGExtensions/EmbeddedXHTML">
[9]         <!-- XHTML content goes here -->
[10]   </foreignObject>

[11]   <!-- Else, process the following alternate SVG. Note that there are
        no testing attributes on the <g> element.If no testing attributes
        are provided, it is as if there were testing attributes and they
        evaluated to true.-->
[12]   <g>
[13]     <!-- Draw a red rectangle with a text string on top. -->
[14]     <rect width="20" height="20" style="fill: red"/>
[15]     <text>Formula goes here</text>
[16]   </g>
[17]  </switch>
[18] </svg>
```

Erläuterungen:

Seite 87

[6] Das Tag `<switch>` enthält als erstes Child-Element ein `foreignObject` mit XHMTL-Code (der in diesem Beispiel nicht weiter ausgeführt wurde).

[8] Mit Hilfe des Attributs ‚`requiredExtensions`' wird ermittelt, ob in SVG eingebettetes XHTML vom Browser unterstützt wird. Dazu wird eine (Beispiel-)URI angegeben, die auf die zugehörige (Beispiel-)Extension verweist.

[12] Das zweite Child-Element ist eine Gruppe, bestehend aus einem roten Rechteck und Text. Diese dient als Alternativ-Graphik, wenn der Inhalt des ersten Child-Elements nicht unterstützt wird, sprich, wenn der Browser kein eingebettetes XHTML versteht.

Software-Hinweis:
Selbst die neueste Version des Adobe SVG-Viewers (2beta) unterstützt das Tag `<switch>` noch nicht.

2.2.1.2.3 Symbole und Pictogramme

Diverse Forschungen haben bereits ergeben, dass der Mensch ca. 80 % seiner Umwelt visuell wahrnimmt. Dieser Aspekt und natürlich auch die fortlaufende Internationalisierung führen dazu, dass verstärkt Graphiken anstelle von Texten eingesetzt werden. Das beste Beispiel hierzu ist mit Sicherheit der Flughafen, auf dem wir den Weg zur Toilette über ein ganz bestimmtes Pictogramm gewiesen bekommen. Ein anderes ist die Kartographie. Hier werden Symbole bewusst eingesetzt, um komplexe, immer wiederkehrende Inhalte auf kleinstem Raum symbolisch (stellvertretend) darzustellen.

Für diesen besonderen Einsatz definieren wir Zeichenobjekte innerhalb des Tags `<symbol>`.

Beispiel (aus der Recommendation):

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
[3]  "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
[4]  <svg width="10cm" height="3cm" viewBox="0 0 100 30">
[5]  <desc>Example Use02 - 'use' on a 'symbol'</desc>
[6]   <defs>
[7]    <symbol id="MySymbol" viewBox="0 0 20 20">
[8]    <desc>MySymbol - four rectangles in a grid</desc>
[9]      <rect x="1" y="1" width="8" height="8"/>
[10]     <rect x="11" y="1" width="8" height="8"/>
[11]     <rect x="1" y="11" width="8" height="8"/>
[12]     <rect x="11" y="11" width="8" height="8"/>
[13]   </symbol>
[14]  </defs>
[15]  <use x="45" y="10" width="10" height="10" xlink:href="#MySymbol" />
[16] </svg>
```

Erläuterungen:

[7] `<symbol>` dient als Rahmenlement für die Definition des eigentlichen Symbols. Über das Attribut ‚`viewBox`' legen wir einen Anzeigebereich fest, der in diesem Fall zwei Punkte grösser ist als Breite und Höhe der Signatur (10 + 8 = 18).

[15] Mit `<use>` rufen wir das Symbol auf und plazieren es mit Hilfe der Koordinaten-Attribute ‚`x`' und ‚`y`'.

Seite 23

Ergebnis:

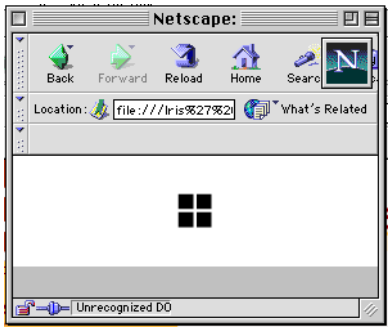


Abb. 2-5: Beispiel <symbol>

Anmerkung: Im Grunde können wir das Tag <symbol> auch durch <g> ersetzen, da es nur darum geht, Graphikelemente als Gruppe zu definieren und sie somit als solche abrufbar zu machen. Beide Elemente unterscheiden sich jedoch in einem Punkt voneinander: Für <g> existiert das Attribut ‚viewBox' nicht, das in diesem Fall die Graphik in halber Grösse erscheinen lässt, da es zusammen mit den <use>-Attributen ‚width' und ‚height' die Skalierung beeinflusst (siehe Kapitel ‚*Gesteuerte Ansichten*').

Seite 23

Software-Hinweis:
<symbol> wird erst seit der Version 2beta des Adobe SVG-Viewers vollständig unterstützt (mit der Version 1 funktionierte das Attribut ‚viewBox' noch nicht).

2.2.1.2.4 Einbinden von (Raster-)Bildern in ein SVG-Dokument

Auch wenn wir mit SVG sehr viel graphisch umsetzen können, gibt es dennoch auch Fälle (typischerweise Photographien), in denen wir auf Rasterbilder ausweichen sollten. Zum Einbinden solcher Bilder in ein SVG-Dokument gibt es das Tag <image> (vgl. <img> in HTML). Mit dem Attribut ‚xlink:href' (vgl. ‚src' in HTML) geben wir die URL für die eigentliche Bilddatei an.

```
<image xlink:href="../bilder/beispiel.jpg"
  width="[Breite des Bildes]"
  height="[Höhe des Bildes]"
  x="[y-Position der Lage des Bildes]"
  y="[y-Position der Lage des Bildes]"/>
```

Werte für ‚width', ‚height', ‚x' und ‚y':
Koordinaten- bzw. Massangaben in jeder beliebigen Einheit

DTD: ‚xlink:href', ‚width' und ‚height' sind Pflicht.

2.2.1.3 Koordinatensysteme und Einheiten

2.2.1.3.1 Der Ursprung des SVG-Koordinatensystems

Der Koordinaten-Ursprung liegt bei SVG in der linken oberen Ecke.

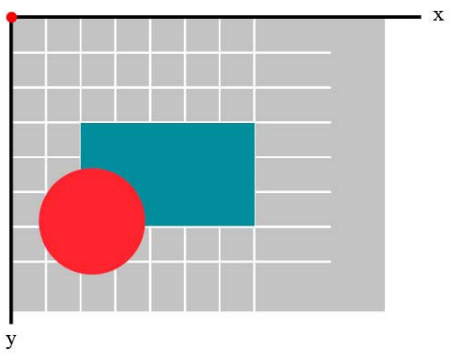


Abb. 2-6: Das SVG-Koordinatensystem

Demzufolge müssen wir bei der Übertragung von Graphiken des uns bekannten kartesischen Koordinatensystems relativen y-Werten ein negatives Vorzeichen geben.

2.2.1.3.2 Einheiten

Die in SVG verwendeten Einheiten, wurden von CSS2 (Cascading Stylesheets Version 2) übernommen.

Wir unterscheiden relative von absoluten Einheiten. Zu den relativen gehören:

- **px** für Pixel (abhängig von der Bildschirmauflösung),
- **em** für die relative Font-Grösse und
- **ex** für die relative Höhe des Buchstabens „x“ (klein geschrieben) eines verwendeten Fonts (diese Einheit kann allerdings auch für Fonts verwendet werden, die kein „x“ enthalten).

An absoluten Einheiten stehen uns zur Verfügung:

- **in** für inch,
- **cm** für Zentimeter,
- **mm** für Millimeter,
- **pt** für Points (72. Teil von einem Inch) und
- **pc** für Picas (1 Pica entspricht 12 Points).

Schliesslich können Masse auch in **Prozentwerten** ausgedrückt werden. Diese sind relativ zur Breite und Höhe des SVG-Fensters.

2.2.1.4 Gesteuerte Ansichten

Oft benötigen wir verschiedene **Detail-Ansichten oder gezielte Zoomstufen** innerhalb einer Graphik. Für diesen Zweck unterstehen einigen Tags, allen voran `<svg>`, die beiden Attribute

- ‚viewBox‘ und
- ‚preserveAspectRatio‘.

Mit ‚viewBox‘ können wir sowohl proportionale als auch nicht-proportionale (verzerrte) Ansichten erzeugen.

Mit ‚preserveAspectRatio‘ steuern wir darüberhinaus die Anzeige einer Zeichnung innerhalb ‚viewBox‘, wobei hier die Ansichten ausschliesslich proportional verändert werden.

2.2.1.4.1 Das Attribut ‚viewBox‘

Über das Attribut ‚viewBox‘ können wir die linke obere (P1) und die rechte untere Ecke (P2) eines Sichtfeldes beschreiben, so dass nur dieser Teil der Graphik zu sehen ist.

Die Syntax für das Attribut ‚viewBox' sieht folgendermassen aus:

```
<svg width="[Fensterbreite]" height="[Fensterhöhe]"
  viewBox="[x1] [y1] [x2] [y2]">
```

Werte für x1, y1, x2 und y2:
(Angaben in [px])

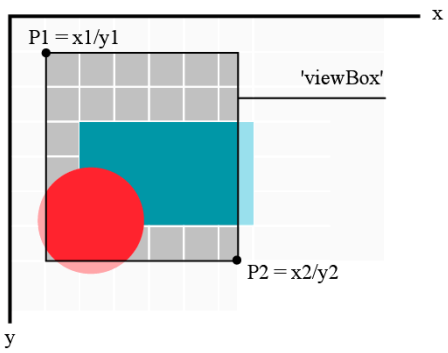


Abb. 2-7: Der ‚viewBox'-Anzeigebereich

Beispiele:

```
[1]  <!-- Beispiel 1: viewBox="-50 0 100 100" -->
[2]  <svg width="100px" height="100px" viewBox="-50 0 100 100">
[3]   <circle r="50" cy="50" style="fill:blue"/>
[4]    <text x="10px" y="50" style="fill:white; font-size:12pt">
        ganz</text>
[5]  </svg>
```

```
[1]  <!-- Beispiel 2: viewBox="0 0 100 100" -->
[2]  <svg width="100px" height="100px" viewBox="0 0 100 100">
[3]   <circle r="50" cy="50" style="fill:blue"/>
[4]    <text x="10px" y="50" style="fill:white; font-size:12pt">
        halb</text>
[5]  </svg>
```

Ergebnis:

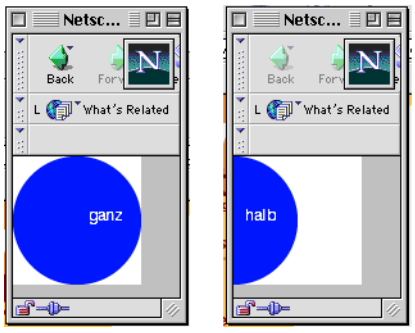


Abb. 2-8: Beispiel ‚viewBox'

Wie funktioniert ‚viewBox'?

Die Angabe über die Lage des Sichtfeldes ergibt sich aus den Punkten P1 und P2, die wir über die Koordinaten-Liste von ‚viewBox' ([x1] [y1] [x2] [y2]) festlegen. Dabei handelt es sich ausschliesslich um Pixelwerte (ohne Angabe

der Einheit).

Im Zusammenhang mit den beiden Attributen ‚width' und ‚height', die diversen Tags unterstehen, können wir zusätzlich einen Skalierungsfaktor (nicht-proportional) ermitteln, in dem unsere Graphik auf dem Bildschirm erscheint.

Die Formel dazu lautet folgendermassen:

x2 [px] - x1 [px] = Δx [px]
y2 [px] - y1 [px] = Δy [px]

'**width**' [px] : Δx [px] = Skalierungsfaktor in x-Richtung
'**height**' [px] : Δy [px] = Skalierungsfaktor in y-Richtung

Anmerkung: Da die Parameter von ‚viewBox' Pixelwerte sind, also abhängig von der Auflösung des Bildschirms, stimmt diese Formel nur, wenn wir für die Attribute ‚width' und ‚height' ebenfalls Pixelwerte eingeben.

2.2.1.4.2 Das Attribut ‚preserveAspectRatio'

Um die Ansicht innerhalb des durch ‚viewBox' festgelegten Anzeigebereichs (P1, P2) zu steuern verwenden wir ‚preserveAspectRatio' (letzteres ist also von ‚viewBox' abhängig). Für die Spezifizierung greifen wir in diesem Fall auf vorgegebene Parameter zurück.

Die Syntax für ‚preserveAspectRatio' sieht so aus:

```
<svg width="[Anzeigebreite]" height="[Anzeigehöhe]"
  viewBox="[x1] [y1] [x2] [y2]"
  preserveAspectRatio="[Ausrichtung] [Skalierung]">
```

Parameter für ‚Ausrichtung': (Standard: ‚xMidYMid')

`none` (keine Veränderung) |
`xMinYMin` (x: links, y: oben) |
`xMidYMin` (x: mittig y: oben) |
`xMaxYMin` (x: rechts y: oben) |
`xMinYMid` (x: links y: mittig) |
`xMidYMid` (x: mittig y: mittig) |
`xMaxYMid` (x: rechts y: mittig) |
`xMinYMax` (x: links y: unten) |
`xMidYMax` (x: mittig y: unten) |
`xMaxYMax` (x: rechts y: unten)

Parameter für ‚Skalierung': (Standard: ‚meet')

`meet` (die komplette Zeichnung wird auf ‚viewBox' skaliert) |
`slice` (die Zeichnung füllt ‚viewBox' komplett aus und wird dafür beschnitten)

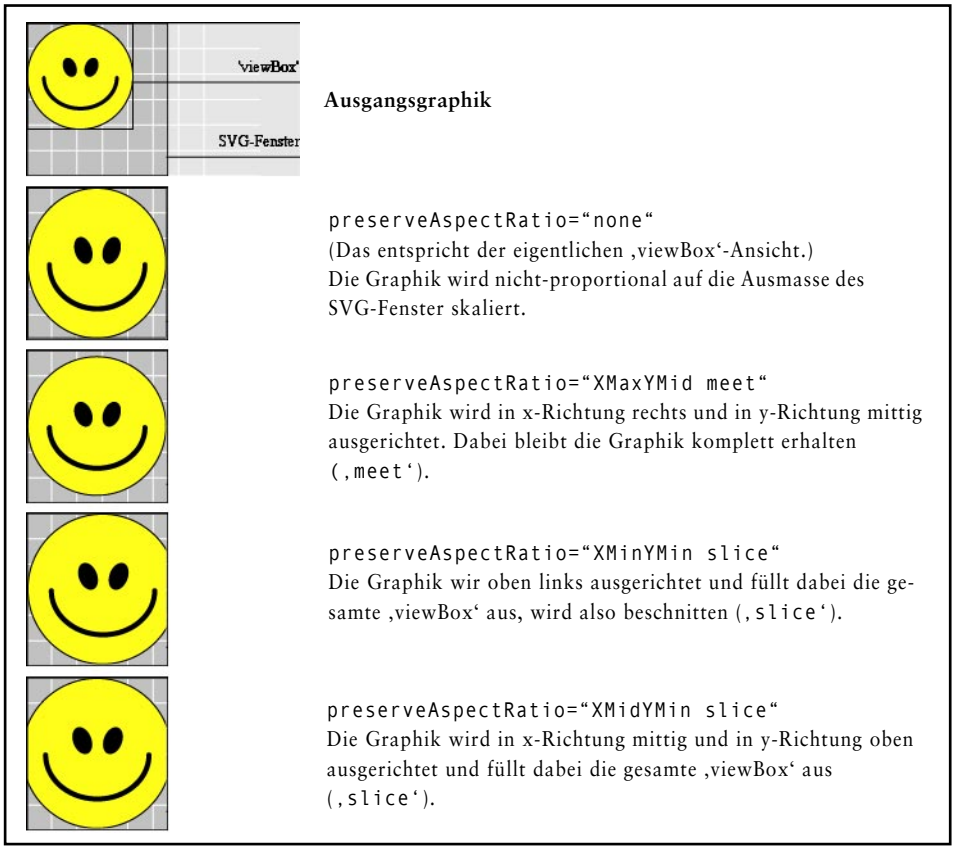


Abb. 2-9: Diverse Ansichten mit ‚preserveAspectRatio'

2.2.2 Graphik

2.2.2.1 Formatierungen

Graphiken besitzen neben ihrer Form auch Eigenschaften, wie Farbe und Strichstärke.

Bei Texten entspricht die Form gewissermassen ihrer inhaltlichen Aussage. Zu deren Eigenschaften zählen wir u. a. Schriftart und Schriftgrösse (näheres dazu im Kapitel ‚*Texte formatieren*').

Seite 49

In SVG stehen uns mehrere Möglichkeiten zur Verfügung, um solche Eigenschaften zu beschreiben oder – anders ausgedrückt – zu formatieren:

- über **C**ascading **S**tylesheets (CSS),
- über die Auszeichnung mit dem Attribut ‚style',
- über die SVG-interne Attributsauszeichnung oder
- über die „XML-eigene" e**X**tensible **S**tylesheet **L**anguage (XSL).

2.2.2.1.1 Cascading Stylesheets (CSS)

Eine elegante und vor allen Dingen Sprachen übergreifende Methode der Formatierung von Schriften und graphischen Elementen ist die Festlegung von **C**ascading **S**tylesheets (CSS). Sie wird von vielen Markup Languages gleichermassen unterstützt, so dass wir nicht ständig neue Schlüsselbegriffe lernen

müssen. Auch lassen sich CSS an einer Stelle definieren (dabei kann es sich um ein eigenständiges Dokument mit dem Suffix ‚.css' handeln) und an verschiedenen anderen Stellen referenzieren. Hierdurch sparen wir Zeit und Arbeit, wenn es darum geht, Änderungen vorzunehmen.

Damit nach dem Parsen eines Dokuments Stylesheets auch eindeutig zugeordnet werden können, erhalten sie Namen, sogenannte CSS-Bezeichner. Das können entweder Elementnamen sein oder eigendefinierte Strings, die später über die Attribute ‚id' oder ‚class' einem Element zugewiesen werden.

Die Schreibweise für CSS-Stylesheets sieht folgendermassen aus:

```
[Elementname]             { [Property]:[Wert] }
[Elementname].[id-String] { [Property]:[Wert] }
.[class-String]           { [Property]:[Wert] }
```

CSS-Stylesheets im Dokument einbinden

Innerhalb des SVG-Dokuments zeichnen wir CSS-Stylesheets über das Tag `<style>` aus.

```
[1] <style type="[text/css]">
[2]  <![CDATA[
[3]    text { fill:#000066; font-size:14pt; font-family:Verdana }
[4]    text.id_style { font-size:12pt; font-weight:bold }
[5]    .class_style { fill:red; stroke:none }
[6]  ]]>
[7] </style>
```

Erläuterungen:

[2] Mit `<![CDATA[ ... ]]>` wird ein ganzer Abschnitt als nicht zur Auszeichnung zugehörig gekennzeichnet. Hierin könnte z. B. auch Java-Code stehen oder Text, der mit offenen spitzen Klammern („<") durchsetzt ist, was im normal ausgezeichneten Text dazu führen würde, dass der darauffolgende Text nicht mehr erscheint.

Externe CSS-Stylesheets verknüpfen

Wie oben angedeutet können wir auch externe ‚.css'-Dateien mit einem SVG Dokument verknüpfen. Dazu legen wir zunächst eine solche Datei in einem einfachen Texteditor an und geben ihr beim Speichern das Suffix ‚.css':

stylesheet.css

In die Datei schreiben wir unsere Stylesheets hinein:

```
rect { fill:red; stroke-width:0.3pt; stroke:#006600 }
.kuchen { fill:brown; stroke:none }
```

Danach verknüpfen wir ‚stylesheet.css' mit dem SVG-Dokument, indem wir sie über dessen **Stylesheet-Prolog** ([2]) referenzieren:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <?xml-stylesheet href="stylesheet.css" type="text/css"?>
```

Jetzt sind die CSS-Stylesheets mit den entsprechenden Elementen verbunden:

```
<rect width="2cm" height="2cm" x="2cm" y="2cm"/>
<circle class="kuchen" cx="5cm" cy="3cm" r="3cm"/>
```

2.2.2.1.2 Das Attribut ‚style'

SVG bietet uns zusätzlich die Möglichkeit, CSS-Stylesheets direkt in die Auszeichnung einzubinden. Dazu benötigen wir das Attribut ‚style':

```
<text x="2in" y="3in" style="fill:red; font-familiy:Times;
  font-size:14pt">Times, 14 Punkt, rot</text>
```

Über das Tutorial hinaus: Formatierungen über spezielle Attribute

2.2.2.1.3 Formatierungen über spezielle Attribute

In der SVG-DTD (unter dem Abschnitt *‚Entity-Definitions for the presentation Attributes'*) finden wir auch interne Attribute, die die selben Bezeichnungen tragen wie die CSS-Stylesheets. Wir können demnach die meisten Formatierungen auch folgendermassen vornehmen:

```
<rect fill="blue" stroke="black" width="10" height="10"/>
```

Software-Hinweis:
Diese Art der Auszeichnung wird von den wenigsten Softwareprodukten verwendet.

2.2.2.1.4 Entities

Gibt es mehrer Zeichenobjekte, auf die wir ein und dasselbe Format anwenden wollen, können wir, ähnlich wie bei CSS, für Formatdefinitionen Platzhalter festlegen, die sogenannten Entities. Diese definieren wir im Doctype des Dokuments und rufen sie dann an entsprechender Stelle auf:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
     www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd" [
[3]   <!ENTITY style1 "stroke:#0000FF;stroke-width:3;">
[4]   <!ENTITY style2 fill="#0000FF">
[5] ]>
[6] <svg width="10cm" height="5cm">
[7]   <rect style="&style1;" width="3cm" height="5cm"/>
[8]   <circle &style2; r="1.5cm"/>
[9] </svg>
```

Anmerkung: Im gleichen Stil wie wir über Entities Platzhalter für Formate definieren können, steht es uns auch offen, darüber ganze Elemente oder Graphiken festzulegen, um sie an anderer Stelle zu referenzieren.

2.2.2.1.5 Die eXtensible Stylesheet Language (XSL)

Bisher sind wir bei der Formatierung so vorgegangen, dass wir diese entweder direkt in unser SVG-Dokument eingetragen oder aus einem externen Dokument referenziert haben.

Für die Formatierung mit der e**X**tensible **S**tylesheet **L**anguage (XSL) müssen wir einen anderen Weg einschlagen. Hier werden die Formate über ein externes ‚.xsl'-Dokument in ein Ergebnisdokument hineingeneriert. D. h., die dem SVG-Dokument zugrundeliegende Baumstruktur wird so umgewandelt, dass ein Element beispielsweise neue Formatierungsattribute erhält.

Beispiel:

XSL-Dokument:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/ Transform"
       version="1.0">

[3]  <!-- Weise allen circle-Elementen eine rote Füllung zu -->
[4]   <xsl:template match="circle">
[5]    <xsl:copy>
[6]     <xsl:copy-of select="@*"/>
[7]      <xsl:attribute name="fill">red</xsl:attribute>
[8]    </xsl:copy>
[9]   </xsl:template>
[10] </xsl:stylesheet>
```

SVG-Ausgangsdokument:

```
[11] <?xml version="1.0" standalone="no"?>
[12] <svg width="100" height="100">
[13]   <circle cx="50" cy="50" r="50"/>
[14] </svg>
```

SVG-Ergebnisdokument:

```
[15] <?xml version="1.0" standalone="no"?>
[16] <svg width="100" height="100">
[17]   <circle cx="50" cy="50" r="50" fill="red"/>
[18] </svg>
```

Erläuterungen:

- [4] Mit ‚`match`' geben wir an, auf was das Template angewendet werden soll: alle `circle`-Elemente.
- [6] Mit `<xsl:copy-of select="@*">` wird eine Kopie aller nachfolgenden Attribute den `circle`-Elementen zugewiesen.
- [13] Im Ausgangsdokument finden wir keine Formatierungsattribute.
- [17] Nach dem Parsen ist im Ergebnisdokument dem Tag `<circle>` das Attribut ‚`fill`' (mit dem Wert ‚`red`') zugeordnet worden.

2.2.2.2 Farben

Seite 26

Im vorangegangenen Kapitel ‚*Formatierungen*' haben wir CSS kennengelernt, mit deren Hilfe wir in erster Linie das Aussehen eines Zeichenobjekts oder eines Textes beeinflussen können. Ein wichtiger Bestandteil dieses Aussehens ist die **Farbgebung**. Zu deren Festlegung stehen uns in CSS diverse Properties, wie ‚`color`', ‚`fill`', ‚`stroke`' etc., zur Verfügung. Ausserdem haben wir die

Möglichkeit, Farbprofile zu zuweisen, die zum Farbstandard des **I**nternational **C**olor **C**onsortiums (ICC) konform sind.

2.2.2.2.1 Farben festlegen

Mit CSS-Properties, wie den gerade genannten, definieren wir Farben, die später Elementen, Klassen oder IDs zugeordnet werden.

Dazu können wir auf eine Liste von Systemfarben zurückgreifen, die jeweils in englisch-sprachigen Farbnamen (*red*, *blue*, *grey* etc.) ausgedrückt werden. Ausserdem unterstützt CSS den sRGB-Farbraum, der speziell für Bildschirm-Anwendungen eingeführt wurde.

Wir können also Farben in CSS auf verschiedene Weisen festlegen:

- über die Farbnamen:

```
[Elementname] {color:red}
```

- über den sRGB-Farbraum:

```
[Elementname] { color:rgb(255, 0, 0) }
[Elementname] { color:rgb(100%, 0%, 0%) }
```

- und schliesslich auch über die Hexadezimal-Schreibweise:

```
[Elementname] { color:#FF0000 }
[Elementname] { color:#F00 }
```

2.2.2.2.2 Farbprofile des International Color Consortiums (ICC)

1993 wurde das **I**nternational **C**olor **C**onsortium (ICC) gegründet, um ein einheitliches **plattform- und ausgabeunabhängiges Farbmanagementsystem** aufzu bauen. Daraus entstanden die ICC-Farbprofile, die zahlreiche Software-Hersteller in ihrer Software berücksichtigen (darunter auch Adobe Illustrator).

CSS (und damit auch SVG) unterstützt diese Farbprofile, so dass wir hier die Möglichkeit haben, ICC-konforme Farben zu definieren, die, vorausgesetzt, wir haben auf dem Computer ein Farbmanagementsystem installiert, immer gleich aussehen.

Mit Hilfe des Descriptors ‚`@color-profile`' beschreiben wir ein Eingabeprofil näher.

```
@color-profile { src:[Quelle des Farbprofils]; name:"meineFarbe";
  rendering-intent:[Rendering-Modell]; color-profile:[Farbprofil] }
```

Werte zu ‚src': (Standard: ‚sRGB')
`sRGB` | [URI] | [lokales Farbprofil]+ | `inherit`

Werte zu ‚rendering-intent': (Standard: ‚auto')
`auto` | `perceptual` | `relative-colorimetric` | `saturation` | `absolute-colorimetric`

Werte zu ‚color-profile': (Standard: ‚auto')
auto | sRGB | ([URI] | [lokales Farbprofil])+ | inherit

Schliesslich rufen wir das festgelegte Profil mit ‚icc-color' innerhalb einer Farbdefinition auf, wobei wir für Rechner, auf denen kein Farbmanagementsystem installiert ist, optional noch eine Ersatzfarbe (z. B. ‚color:green') angeben können.

```
rect {  color:green icc-color(meineFarbe, 0, 255, 0) }
```

2.2.2.2.3 Füllungen und Linien

Vektorgraphiken setzen sich zusammen aus den drei Bestandteilen:

- Knoten oder Punkten (die wir als x-/y-Koordinaten definieren),
- Linien (den Verbindungen zwischen Knoten) und
- Flächen (hier gilt: Anfangsknoten = Endknoten).

Für alle drei Bestandteile können wir mit Hilfe von Attributen die äussere Gestaltung beschreiben. Sämtliche Zeichenattribute von SVG stammen dabei wiedereinmal aus CSS. So haben wir auch hier die Möglichkeit, sie als Stylesheets festzulegen oder direkt in die Auszeichnung einzubinden.

2.2.2.2.4 Füllungsattribute

Für die Füllungsgestaltung eines Vektors können wir auf folgende Attribute zurückgreifen:

- ‚fill' (zum Festlegen der Flächenfarbe),
- ‚fill-rule' (zum Festlegen der Füllmethode) und schliesslich
- ‚fill-opacity' (zum Festlegen der Flächentransparenz).

‚fill'

Das Attribut ‚fill' entspricht im Wesentlichen dem Attribut ‚color', das wir im vorangegangenen Kapitel ‚*Farben*' kennengelernt haben, nur dass sich hier die Farbgebung ausschliesslich auf die Füllung bezieht.

```
<rect width="3cm" height="3cm" style="fill:blue">
```

‚fill-rule'

Mit ‚fill-rule' können wir zwei verschiedene Füllmethoden festlegen:

- ‚nonzero' (Standard) oder
- ‚evenodd'.

```
<g style="fill:blue; fill-rule:nonzero">
```

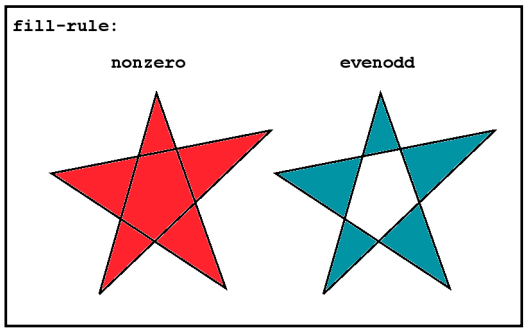


Abb. 2-10: Füllmethoden mit ‚fill-rule'

‚fill-opacity'

Mit ‚fill-opacity' steuern wir die Opazität (Deckkraft) eines Zeichenobjekts.

Sein Wertebereich liegt zwischen 0 und 1.

```
<g style="fill:blue; fill-rule:nonzero; fill-opacity:.5">
```

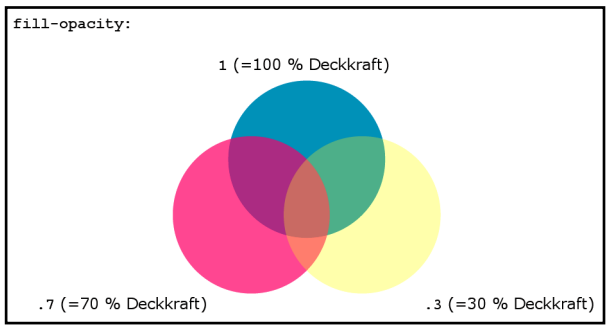


Abb. 2-11: Flächen-Opazität mit ‚fill-opacity'

2.2.2.2.5 Linienattribute

Für die Liniengestaltung stehen uns eine ganze Reihe von Attributen zur Verfügung:

- ‚stroke' (zum Festlegen der Linienfarbe),
- ‚stroke-width' (zum Festlegen der Linienstärke),
- ‚stroke-linecap' (zum Festlegen des Verhaltens der Linienenden),
- ‚stroke-linejoin' (zum Festlegen des Eckenverhaltens in Polylinien),
- ‚stroke-miterlimit' (zum Festlegen der Gehrungsgrenze),
- ‚stroke-dasharray' (zum Festlegen von gestrichelten Linien),
- ‚stroke-dashoffset' (zum Festlegen des Startpunkts des Linienmusters einer gestrichelten Linie) und
- ‚stroke-opacity' (zum Festlegen der Linienopazität).

‚stroke‘

Mit ‚stroke‘ können wir einem Zeichenobjekt unabhängig von der Flächenfarbe eine Linienfarbe zuweisen.

```
<ellipse rx="2.5in" ry="1.25in" style="fill:red; stroke:#0000FF">
```

‚stroke-width‘ und ‚stroke-linecap‘

Mit den Attributen ‚stroke-width‘ und ‚stroke-linecap‘ bearbeiten wir eine Linie in ihrer Form. Dabei legen wir mit ‚stroke-width‘ zunächst die Strichstärke fest: die Linie wird breiter.

Mit ‚stroke-linecap‘ steuern wir das Aussehen von Anfangs- und Endknoten: die Linie bekommt runde oder eckige Enden.

Für ‚stroke-linecap‘ haben wir die vier Werte

- ‚butt‘ (Standard),
- ‚round‘,
- ‚square‘ und
- ‚inherit‘(wie der Voränger)

zur Auswahl.

```
<line x1="50" y1="50" x2="150" y2="50" style="fill:none; stroke:#FF0;
  stroke-width:5pt; stroke-linecap:round"/>
```

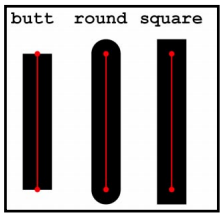


Abb. 2-12: Gestaltung der Linienenden mit ‚stroke-linecap‘

‚stroke-linejoin‘

Mit ‚stroke-linejoin‘ geben wir die Form eines Eckpunkts in einer Linie oder einem Pfad mit mehr als zwei Punkten an:

- ‚miter‘ (Standard),
- ‚round‘,
- ‚bevel‘ und
- ‚inherit‘ (wie der Voränger).

```
<polyline points="50,375 150,375 150,325 250,325 250,375"
  style="stroke:red; stroke-width:5; stroke-linejoin:round"/>
```

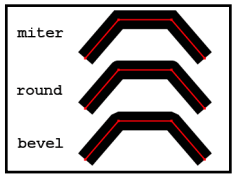


Abb. 2-13: Gestaltung von Eckpunkten mit ‚stroke-linejoin'

‚stroke-miterlimit'

Als Gehrungsgrenze bezeichnen wir ein Faktor, der vorgibt, wann ein Eckpunkt einer Polylinie nicht mehr spitz, sondern abgeflacht verlaufen soll. Z. B. besagt eine Gehrungsgrenze von 4 (Standard), dass die Ecke von dem Winkel an spitz verläuft, ab dem die Länge der 4-fachen Breite der Strichstärke entspricht.

Für deren Festlegung verwenden wir das Attribut ‚stroke-miterlimit'.

```
<polyline points="50,375 150,375 150,325 250,325" style="stroke:red;
  stroke-width:5; stroke-linejoin:round; stroke-miterlimit:7"/>
```

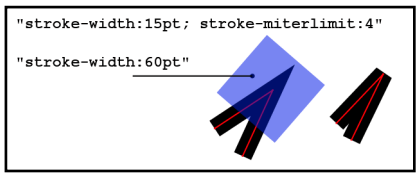


Abb. 2-14: Festlegung der Gehrungsgrenze mit ‚stroke-miterlimit'

‚stroke-dasharray' und ‚stroke-dashoffset'

Zum Zeichnen einer gestrichelten (unterbrochenen) Linie bietet uns SVG die beiden Attribute:

- ‚stroke-dasharray', über dessen Parameterliste wir die jeweiligen Linienlängen und Unterbrechungen eintragen und
- ‚stroke-dashoffset', mit dem wir einen Startpunkt innerhalb des Linienmusters festlegen.

```
<line x1="5" y1="50" x2="100" y2="5" style="stroke:green;
stroke-width:3; stroke-dasharray:30, 10, 30, 20; stroke-dashoffset:15"/>
```

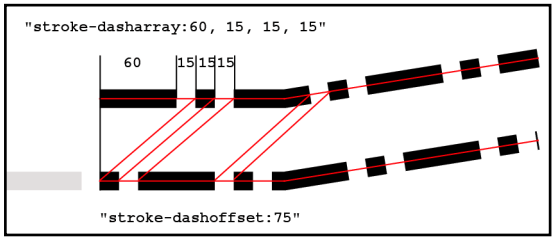


Abb. 2-15: Gestrichelte Linien mit ‚stroke-dasharray' und ‚stroke-dashoffset'

‚stroke-opacity‘

Mit ‚stroke-opacity‘ steuern wir unabhängig von der Fläche die Deckkraft einer Linie.

Sein Wertebereich liegt zwischen 0 und 1.

```
<polyline points="5, 5 5, 10 10, 10" style="fill:none; stroke:blue;
  stroke-opacity:.3"/>
```

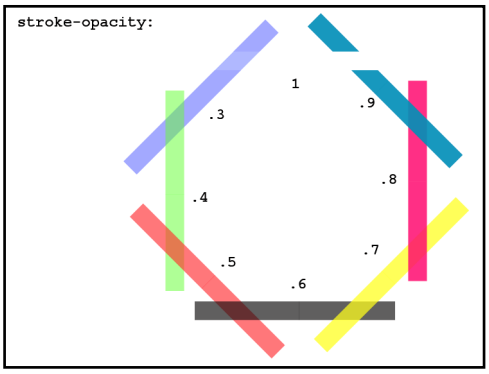


Abb. 2-16: Linien-Opazität mit ‚stroke-opacity‘

2.2.2.3 Vektoren

SVG enthält diverse Tags, mit deren Hilfe wir graphische Formen beschreiben können. Dazu gehören sowohl eindeutige, wie Rechtecke, Kreise, Ellipsen und Linien, als auch variable, wie Polylinien und Polygone. Ausserdem können wir mit dem Tag <path> klassische Bézierkurven erzeugen, die sich über sog. Anfasser verändern lassen.

SVG bietet uns also in punkto Vektorgestaltung die gleichen Möglichkeiten wie jedes gute Graphikprogramm.

2.2.2.3.1 Grundformen

Die uns zur Verfügung stehenden Grundformen-Tags heissen:

- <rect> für Rechtecke,
- <circle> für Kreise,
- <ellipse> für Ellipsen,
- <line> für Linien mit jeweils einem Anfangs- und Endpunkt,
- <polyline> für Linien mit mindestens zwei Punkten und
- <polygon> für Polygone.

Rechtecke

Die Syntax für das Tag <rect> lautet:

```
<rect x="[x-Koordinate der linken oberen Ecke]"
  y="[y-Koordinate der linken oberen Ecke]"
  width="[Breite des Rechtecks]"
  height="[Höhe des Rechtecks]"
  rx="[x-Radius für abgerundete Ecken]"
  ry="[y-Radius für abgerundete Ecken]"/>
```

Werte für alle Attribute:
Koordinaten- bzw. Massangaben in jeder beliebigen Einheit

DTD: ‚width' und ‚height' sind Pflicht.

Beispiel:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
     www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3] <svg>
[4]   <g id="beispiel_rechtecke" style="fill:blue; stroke:black;
       stroke-width:1pt">
[5]     <rect x="2cm" y="1cm" width="2cm" height="3cm"/>
[6]     <rect x="6cm" y="1cm" width="2cm" height="3cm" rx="1cm" ry="1cm"/>
[7]   </g>
[8] </svg>
```

Ergebnis:

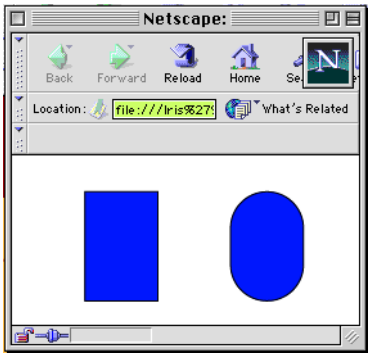


Abb. 2-17: Beispiel <rect>

Kreise

Die Syntax für das Tag <circle> lautet:

```
<circle
  cx="[x-Koordinate des Kreismittelpunkts]"
  cy="[y-Koordinate des Kreismittelpunkts]"
  r="[Kreisradius]"/>
```

Werte für alle Attribute:
Koordinaten- bzw. Massangaben in jeder beliebigen Einheit

DTD: ‚r' ist Pflicht.

Beispiel:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
     www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3] <svg>
[4]  <g id="beispiel_kreise">
[5]    <circle cx="5cm" cy="2.5cm" r="1.5cm" style="fill:green"/>
[6]    <circle cx="5cm" cy="2.5cm" r="0.5cm" style="fill:red"/>
[7]  </g>
[8] </svg>
```

Ergebnis:

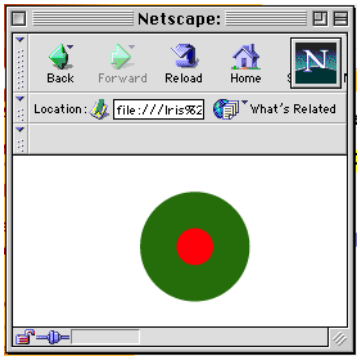


Abb. 2-18: Beispiel <circle>

Ellipsen

Die Syntax für das Tag `<ellipse>` lautet:

```
<ellipse
  cx="[x-Koordinate des Ellipsenmittelpunkts]"
  cy="[y-Koordinate des Ellipsenmittelpunkts]"
  rx="[x-Radius der Ellipse]"
  ry="[y-Radius der Ellipse]"/>
```

Werte für alle Attribute:
Koordinaten- bzw. Massangaben in jeder beliebigen Einheit

DTD: ‚rx' und ‚ry' sind Pflicht.

Beispiel:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
     www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3] <svg>
[4]  <g id="beispiel_ellipsen">
[5]   <ellipse cx="5cm" cy="3.2cm" rx="3cm" ry="1.5cm"
        style="fill:red"/>
[6]   <ellipse cx="5cm" cy="3.2cm" rx="1.5cm" ry="3cm"
        style="fill:blue; fill-opacity:.3"/>
[7]  </g>
[8] </svg>
```

Ergebnis:

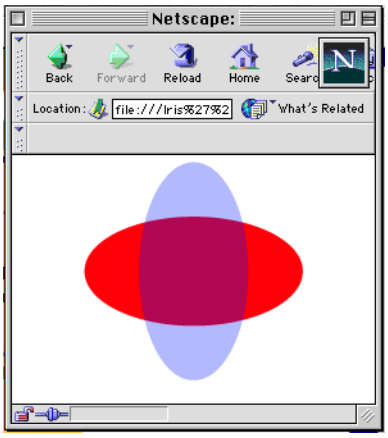


Abb. 2-19: Beispiel <ellipse>

Linien

Die Syntax für das Tag `<line>` lautet:

```
<line x1="[x-Koordinate des Anfangsknotens]"
  y1="[y-Koordinate des Anfangsknotens]"
  x2="[x-Koordinate des Endknotens]"
  y2="[y-Koordinate des Endknotens]"/>
```

Werte zu ‚x1', ‚y1', ‚x2' und ‚y2':
Koordinatenangaben in jeder beliebigen Einheit

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]   <g id="beispiel_linie">
[5]    <line x1="3cm" y1="3cm" x2="7cm" y2="3cm" style="stroke:red"/>
[6]    <line x1="5cm" y1="1cm" x2="5cm" y2="5cm" style="stroke:black"/>
[7]   </g>
[8]  </svg>
```

Ergebnis:

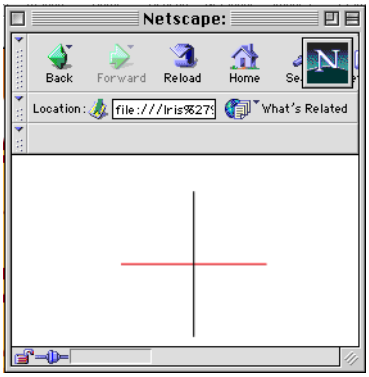


Abb. 2-20: Beispiel <line>

Polylinien

Die Syntax für das Tag <polyline> lautet:

```
<polyline points="[Punktliste mit x- und y-Koordinaten]"/>
```

Werte zu ‚points‘:
Einfache Punktliste in [px]

DTD: ‚points‘ ist Pflicht.

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]   <g id="beispiel_polylinie">
[5]     <polyline style="stroke:blue; fill:none"
[6]       points="0,100 50,100 50,50
                  100,50 100,100 150,100
                  150,0 200,0 200,100"/>
[7]   </g>
[8]  </svg>
```

Ergebnis:

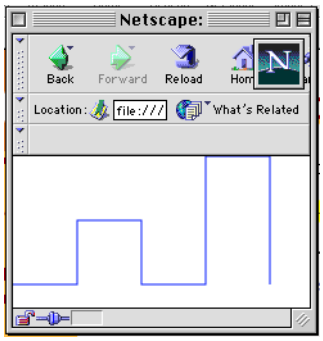


Abb. 2-21: Beispiel <polyline>

Polygone

Die Syntax für das Tag <polygon> lautet:

```
<polygon points="[Punktliste mit x- und y-Koordinaten]"/>
```

Werte zu ‚points‘:
Einfache Punktliste in [px]

DTD: ‚points‘ ist Pflicht.

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg viewBox="200 75 469 301">
[4]   <g id="beispiel_polygon">
```

```
[5]     <polygon style="fill:blue;stroke:yellow"
         points="350,75 379,161 469,161
                 397,215 423,301 350,250
                 277,301 303,215 231,161
                 321,161"/>
[6]   </g>
[7] </svg>
```

Ergebnis:

Abb. 2-22: Beispiel <polygon>

2.2.2.3.2 Beliebige Pfade mit dem Tag <path>

Die Grundformen in SVG decken schon einen Grossteil der möglichen Gestaltungsmittel ab. In einem Punkt sind wir dennoch eingeschränkt: in der Erstellung von Pfaden, die sowohl aus Eck- als auch aus über Anfasser freidefinierbaren Kurvenpunkten bestehen. Hierfür müssen wir auf das Tag <path> zurückgreifen. Über dessen Attribut ‚d' werden Knotenpunkte in ihrer Lage und Beschaffenheit genau beschrieben.

Die Syntax für das Tag <path> sieht folgendermassen aus:

```
<path d="[Punkliste mit Befehlen zur Beschreibung der Knoten]"/>
```

Mögliche Befehle:

M (moveto), z (closepath),
L, l (lineto), H, h (horizontal lineto), V, v (vertical lineto),
C, c (curveto), S, s (smooth curveto), A, a (elliptic arc)

DTD: ‚d' ist Pflicht.

Beispiel:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
     www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3] <svg>
[4]   <g id="beispiel_pfad">
[5]     <path d="M50,100 L50,50" style="stroke:black"/>
```

```
[6]      <path d="M125,50 L125,100" style="stroke:black"/>
[7]      <path d="M125,100 L125,150" style="stroke:black"/>
[8]      <path d="M200,150 L200,100" style="stroke:black"/>
[9]      <path d="M50,100 C50,50 125,50 125,100 S200,150 200,100"
          style="fill:black;stroke:red"/>
[10]   </g>
[11]  </svg>
```

Ergebnis:

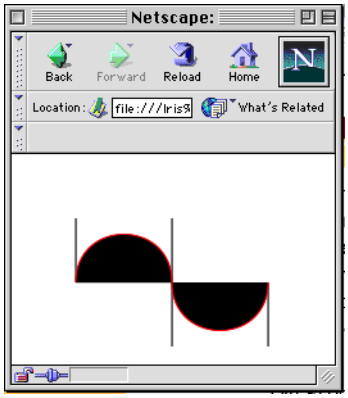


Abb. 2-23: Beispiel <path>

Was besagt die Befehlsfolge M, z, L usw.?

Das Attribut ‚d' enthält eine Befehlsfolge, die eingefleischten Graphikern bereits bekannt sein dürfte, nämlich von der Seitenbeschreibungssprache PostScript, mit der SVG im Prinzip verwandt ist.

Gross- und Kleinschreibung spielen bei den Befehlen übrigens eine Rolle. Grosse Buchstaben beschreiben absolute, kleine relative Positionen (bezogen auf den vorangegangenen Punkt).

Die Befehle für **Anfangs- und Endknoten**:

- `M` (moveto) gibt die Position des Anfangspunktes an:
  `M[x],[y]`
- `z` (closepath) verbindet Anfangs- mit Endpunkt, steht also bei geschlossenen Pfaden immer am Ende der Befehlsfolge.

Die Befehle für **Linien mit Eckpunkten**:

- `L`, `l` (lineto) zeichnet eine Linie zu [x] [y]:
  `L[x],[y]`
  `l[dx],[dy]`
- `H`, `h` (horizontal lineto) zeichnet eine horizontale Linie zu [x] [y]:
  `H[x]`
  `h[dx]`
- `V`, `v` (vertical lineto) zeichnet eine vertikale Linie zu [x] [y]:
  `V[y]`
  `v[dy]`

Wollen wir **Bézierkurven** erzeugen, benötigen wir folgende Befehle:

- `C, c` (curveto) zeichnet eine Bézierkurve zum Punkt [x] [y], die über zwei Anfasser eindeutig beschrieben wird:
  `C[x-Anfasser1],[y-Anfasser1] [x-Anfasser2],[y-Anfasser2] [x],[y]`
  `c[x-Anfasser1],[y-Anfasser1] [x-Anfasser2],[y-Anfasser2] [dx],[dy]`
- `S, s` (smooth curveto) zeichnet ebenfalls eine Bézierkurve zum Punkt [x] [y], bei der der erste Anfasser das Spiegelbild vom zweiten ist. Letzterer kann beliebig positioniert werden:
  `S[x-Anfasser2],[y-Anfasser2] [x],[y]`
  `s[x-Anfasser2],[y-Anfasser2] [dx],[dy]`

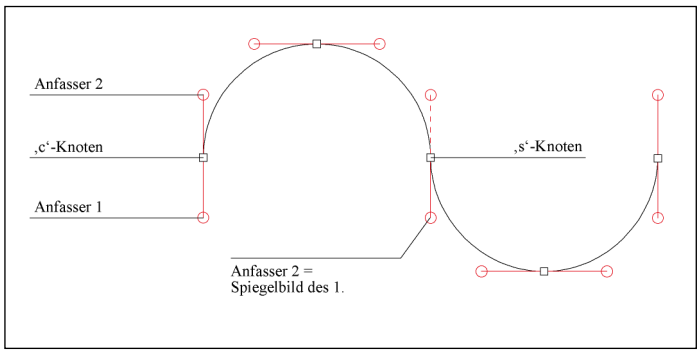


Abb. 2-24: Beschreibung der <path>-Befehle ‚curveto' (C) und ‚smooth curveto' (S)

Für eine vereinfachte Schreibweise der Beschreibung einer **elliptischen Kurve** gibt es den folgenden Befehl:

- `A, a` (elliptic arc) zeichnet eine elliptische Kurve, zum Punkt [x] [y], deren Ausmass über die Ellipsen-Radien ‚`rx`' und ‚`ry`' beschrieben wird:
  `A[rx],[ry] [Rotationswinkel der x-Achse] [Ellipsenbogen],[Richtung des Ellipsenbogens] [x],[y]`
  `a[rx],[ry] [Rotationswinkel der x-Achse] [Ellipsenbogen],[Richtung des Ellipsenbogens] [dx],[dy]`

Werte für ‚Ellipsenbogen':
(Angabe darüber, ob der kleine oder grosse Ellipsenbogen gezeichnet werden soll) `0` (= kleiner Ellipsenbogen) `1` (= grosser Ellipsenbogen)

Werte für ‚Richtung des Ellipsenbogens':
`0` (= negativer „Rotationswinkel") `1` (= positver „Rotationswinkel")

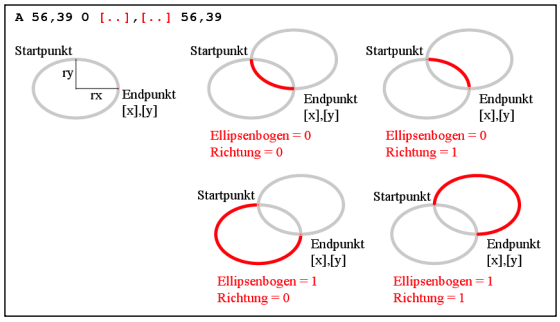


Abb. 2-25: Elliptische Kurven mit dem <path>-Befehl ‚A'

Über das Tutorial hinaus: Pfeile.

2.2.2.3.3 Pfeile

Im Grunde genommen haben wir mehrere Möglichkeiten, um in SVG Pfeile zu erstellen. Wir könnten beispielsweise ein Polygon als Dreieck definieren und dieses jeweils am Ende eines Pfades positionieren. Das Problem hier ist nur, dass wir, je nach dem in welche Richtung der Pfad weist, das Dreieck entsprechend transformieren müssen. Es ist also sinnvoll, eine Möglichkeit zu schaffen, mit der wir automatisch eine Pfeilspitze an den Anfangs- bzw. Endknoten eines Vektors setzen können. Auf diese Weise können wir die Linie, das Polygon, den Pfad drehen, verschieben oder skalieren und die Pfeilspitze passt sich in ihrer Richtung und Grösse an. In SVG gibt es das Tag `<marker>`, mit dem wir zunächst eine Pfeilform mit Hilfe einer der in den vorangegangenen Kapiteln beschriebenen Grundformen bzw. Pfade definieren können.

Zugewiesen wird die Pfeilform einem Pfad, einer Linie oder einer Polylinie über die CSS-Properties

- ‚marker-start' (für den Anfangsknoten),
- ‚marker-mid' (für jeden Knoten ausser Anfangs- und Endknoten) und
- ‚marker-end' (für den Endknoten).

Die Syntax für das Tag `<marker>` sieht folgendermassen aus:

```
<marker
  markerWidth="[Breite, auf die die Pfeilform skaliert wird]"
  markerHeight="[Höhe, auf die die Pfeilform skaliert wird]"
  markerUnits="[Koordinatensystem, in dem ‚markerWidth' und
                ‚markerHeight' abgebildet werden]
  refX="[x-Position des Referenzknotens, an dem die Pfeilform plaziert
         wird (relative Entfernung zum jeweiligen Knoten des Pfades)]"
  refY="[y-Position des Referenzknotens, an dem die Pfeilform plaziert
         wird (relative Entfernung zum jeweiligen Knoten des Pfades)]"
  orient="[Rotation der Pfeilform]">
    <!-- Pfad-Definition -->
</marker>
```

Werte für ‚markerWidth' und ‚markerHeight': (Standard: 0)

Massangaben in jeder beliebigen Einheit

Werte für ‚markerUnits‘: (Standard: `strokeWidth`)

`strokeWidth` | `userSpaceOnUse`
(mit ‚`strokeWidth`‘ werden ‚`markerWidth`‘ und ‚`markerHeight`‘ in Abhängigkeit von der Strichstärke abgebildet, mit ‚`userSpaceOnUse`‘ ist die Abbildung absolut – vgl. hierzu Kapitel ‚Koordinatensysteme der Effekte‘)

Werte für ‚refX‘ und ‚refY‘: (Standard: `3`)

Koordinatenangaben in jeder beliebigen Einheit

Werte für ‚orient‘: (Standard: `auto`)

`auto` | [Rotationswinkel]
(mit ‚`auto`‘ passt sich der Winkel automatisch an die Richtung des Pfades an)

Beispiel (aus der Recommendation):

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
      "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
[3]  <svg width="4in" height="2in" viewBox="0 0 4000 2000">
[4]   <defs>
[5]     <marker id="Triangle" viewBox="0 0 10 10" refX="0" refY="5"
         markerUnits="strokeWidth" markerWidth="4" markerHeight="3"
         orient="auto">
[6]        <path d="M 0 0 L 10 5 L 0 10 z" />
[7]     </marker>
[8]   </defs>
[9]   <rect x="10" y="10" width="3980" height="1980" style="fill:none;
       stroke:blue; stroke-width:10"/>
[10]  <desc>Placing an arrowhead at the end of a path.</desc>
[11]  <path d="M 1000 750 L 2000 750 L 2500 1250" style="fill:none;
       stroke:black; stroke-width:100; marker-end:url(#Triangle)"/>
[12] </svg>
```

Erläuterungen:

[5] Hier beginnt die Definition der Pfeilform (ein Dreieck). `<marker>` gehört zu den Elementen, denen auch die Attribute ‚`viewBox`‘ und ‚`preserveAspectRatio`‘ unterstehen, d. h. wir können über diese auch den Anzeigebereich genauer spezifizieren. Mit der ID geben wir der Pfeilform eine Adresse, über die wir sie später aufrufen können.

[11] Mit ‚`marker-end:url(#Triangle)`‘ rufen wir die Pfeilform auf und legen fest, dass diese am Endknoten des Pfades plaziert werden soll.

Seite 23

Software-Hinweis:
Auch von der neuesten Version des Adobe SVG-Viewers (2beta) wird `<marker>` noch nicht unterstützt.

2.2.2.4 Texte

Ein grosser Vorteil von SVG gegenüber den meisten anderen Graphikformaten ist die grosse Gestaltungsfreiheit in Bezug auf Texte. Für deren Auszeichnung stehen uns grundsätzlich drei Tags zur Verfügung:

- `<text>` für die Auszeichnung einzeiliger Texte,
- `<tspan>` für die Auszeichnung mehrzeiliger oder hervorgehobener Texte und
- `<tref>` für die Referenzierung von Texten.

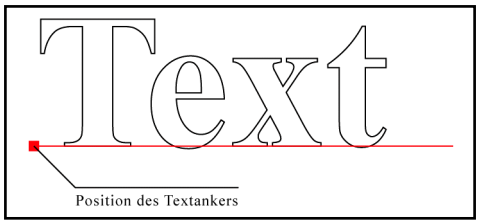
2.2.2.4.1 Texte auszeichnen

Ähnlich dem `<P>`-Tag in HTML enthält SVG eines, mit dem wir Texte im Browser sichtbar machen können: `<text>`. Anders als bei HTML ist jedoch, dass die Positionen für die Texte explizit angegeben werden müssen, da sie sonst Zeile für Zeile in der linken oberen Ecke erscheinen.

Die Syntax für das Tag `<text>` lautet:

```
<text
  x="[x-Position des Textankers]"
  y="[y-Position des Textankers]">
  Hier steht irgendwelcher Text.</text>
```

Werte zu ‚x' und ‚y': (Standard: 0)
Koordinatenangaben in jeder beliebigen Einheit



Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <text x="2cm" y="2cm" style="font-family:Verdana;
        font-size:16pt; fill:red; stroke:none;">
[5]        Das ist Text.
[6]    </text>
[7]  </svg>
```

Ergebnis:

Abb. 2-26: Mit <text> ausgezeichneter Text

Texte herausheben oder mehrzeilig anordnen

Wollen wir **besonders herausgehobene oder mehrzeilige Texte** auszeichnen, brauchen wir ein „Hilfselement", zu dem es ebenfalls ein Pendent in HTML gibt: das Tag . Dieses dient vor allem der besseren Strukturierung. In SVG nennt sich dieses Tag <tspan>. Auch hier müssen wir die Textpositionen mit angeben.

Die Syntax für das Tag <tspan> lautet:

```
<tspan
  x="[x-Position des Textankers]"
  y="[y-Position des Textankers]">
  Dies ist eine Zeile eines mehrzeiligen Textes</tspan>
```

Werte zu ‚x' und ‚y': (Standard: 0)
Koordinatenangaben in jeder beliebigen Einheit

DTD: <tspan> ist ein Child-Tag von <text>.

Anmerkung: Wenn wir statt den Attributen ‚x' und ‚y' ‚dx' und ‚dy' setzen, können wir eine Zeile relativ zur Vorgängerzeile verschieben, was die syntaktische Schreibweise erleichtert, da wir bei gleichbleibenden Zeilenabständen immer den gleichen Wert eintragen können.

Beispiel:

```
[1] <?xml version="1.0" standalone="no"?>
[2] <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
     www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3] <svg>
[4]   <text style="font-family:Verdana; font-size:16pt;fill:red;
       stroke:none">
[5]     <tspan x="1cm" y="1cm">Das ist Zeile 1</tspan>
[6]     <tspan x="1cm" y="2cm">Das ist Zeile 2</tspan>
[7]     <tspan x="1cm" y="3cm">Das ist Zeile 3</tspan>
[8]   </text>
[9] </svg>
```

Ergebnis:

Abb. 2-27: Mit <tspan> ausgezeichneter mehrzeiliger Text

Der nachfolgende Screenshot zeigt ein einzelnes Wort, das mit `<tspan>` herausgehoben wurde.

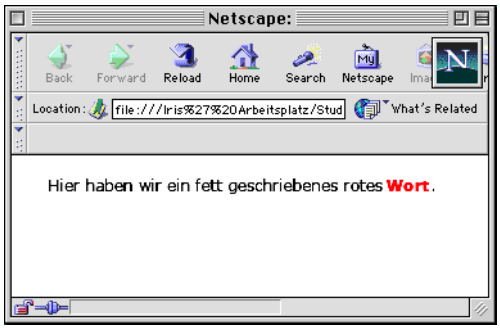


Abb. 2-28: Mit <tspan> hervorgehobenes Wort

Anmerkung: Tatsächlich würden wir das gleiche Ergebnis auch mit `<text>` ausgezeichneten Texten erhalten, jedoch wird vom SVG-Viewer die Zeile dann nicht mehr als Gesamtes betrachtet. Das sehen wir bereits daran, dass sich im SVG-Viewer nicht mehr die ganze Zeile markieren lässt:

Hier haben wir ein fett geschriebenes rotes **Wort**.	**mit <tspan> ausgezeichnetes Wort**
Hier haben wir ein fett geschriebenes rotes **Wort**.	**mit <text> ausgezeichnetes Wort**

Viel wesentlicher ist jedoch, dass Attribute von nicht mehr vererbt werden, d. h. wir müssten jede Zeile oder jedes herausgehobene Wort neu attributieren, was sich natürlich auf die Länge des Sourcecodes auswirkt.

Texte referenzieren

Ein Nachteil von HTML ist es, dass wir Texte, die auf einer Website mehrfach vorkommen (z. B. Adressen), redundant auszeichnen müssen. Schon bei der kleinsten Änderung entsteht ein hoher Nachführungsaufwand. Hier bietet uns SVG in Verbindung mit den beiden XML-Standards XLink und XPointer, eine gute Lösung: Mit dem Element `<tref>` und dem Attribut `‚xlink:href‘` (vergleichbar mit `‚href‘` in HTML) können wir **einen Text an mehrere Stellen referenzieren**.

Die Syntax für das Tag <tref> lautet:

```
<tref
  xlink:href="#ReferenzText"
  x="[x-Position des Textankers]"
  y="[x-Position des Textankers]"/>
```

Werte zu ‚x' und ‚y': (Standard: 0)
Koordinatenangaben in jeder beliebigen Einheit

DTD: <tref> ist ein Child-Tag von <text>.

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]   <defs>
[5]     <text id="referenz">Das ist zu referenzierender Text</text>
[6]   </defs>
[7]   <text style="font-family:'Verdana'; font-size:14pt;
       fill:blue; stroke:none">
[8]     <tref xlink:href="#referenz" x="1cm" y="1cm"/>
[9]   </text>
[10] </svg>
```

Erläuterung:

[4-6] Der zu referenzierende Text wird zentral innerhalb des Tags <defs> abgelegt, damit er der reinen Referenz dient. Sonst würde er in diesem Beispiel zweimal erscheinen.

Ergebnis:

Abb. 2-29: Mit <tref> referenzierter Text

Anmerkung: Für <tref> können wir auch das im Kapitel *‚Strukturierungselemente in SVG'* aufgeführte <use> verwenden.

Seite 18

2.2.2.4.2 Texte ausrichten

Für die Ausrichtung von Texten wird in SVG die Position des Bezugspunktes der Grundlinie (Textanker) entsprechend versetzt. Dies erreichen wir mit Hilfe des Properties ‚text-anchor'.

```
<text x="30" y="20" style="text-anchor:middle">
  zentrierter Text
</text>
```

Werte von ‚text-anchor': (Standard: `start`)

`start | middle | end | inherit`

2.2.2.4.3 Texte formatieren

Wie bereits erwähnt, steht uns mit SVG ein grosser Gestaltungsspielraum in Bezug auf Texte zur Verfügung. Der entspricht dem eines guten Graphikprogramms. Wir können Texten

- Textattribute,
- Farben und
- Effekte (Verläufe, Filter etc.)

zuweisen

Wie wir Füllungs- und Linienattribute festlegen, habe ich bereits im Kapitel *‚Füllungen und Linien'* aufgezeigt. Die gleichen Formatierungsprinzipien gelten auch für Textattribute. Dazu gibt es die CSS Attribute

Seite 31

- ‚`font-familiy`' für die Schriftart,
- ‚`font-size`' für die Schriftgrösse,

Werte zu ‚font-size': (Standard: ‚medium')

Angabe in jeder beliebigen Einheit | `inherit`

- ‚`font-weight`' für die Schriftstärke,

Werte zu ‚font-weight': (Standard: `normal`)

`normal | bold| bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit`

- ‚`font-style`' für Schriftneigung,

Werte für ‚font-style': (Standard: `normal`)

`normal | italic | oblique | inherit`

- ‚`font-variant`' für Kapitälchen und

Werte zu ‚font-variant': (Standard: `normal`)

`normal | small-caps | inherit`

- ‚font-stretch' für die relative horizontale Breite einer Schrift.

Werte zu ‚font-stretch': (Standard: normal)

```
ultra-condensed | extra-condensed | condensed |
semi-condensed | normal | semi-expanded | expanded |
extra-expanded | ultra-expanded | inherit
```

Weitere Attribute finden wir in der CSS2-Spezifikation (unter http://www.w3.org/TR/CSS2).

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <text x="1cm" y="1cm" style="font-family:Arial; font-size:14pt;
        font-weight:bold; fill:blue; stroke:none">
[5]      Arial, 14 Punkt, fett
[6]    </text>
[7]    <text x="1cm" y="1.5cm" style="font-family:Times New Roman;
        font-size:12pt; font-style:oblique; stroke:none">
[8]      Times New Roman, 12 Punkt, kursiv
[9]    </text>
[10]   <text x="1cm" y="2cm" style="font-family:Arial; font-size:10pt;
        font-variant:small-caps; fill:red; stroke:none">
[11]     Arial, 10 Punkt, Kapitaelchen
[12]   </text>
[13]   <text x="1cm" y="2.5cm" style="font-family:Courier New;
        font-size:6pt; font-stretch:ultra-expanded; stroke:none">
[14]     Courier New, 6 Punkt, ultra-expanded
[15]   </text>
[16] </svg>
```

Ergebnis:

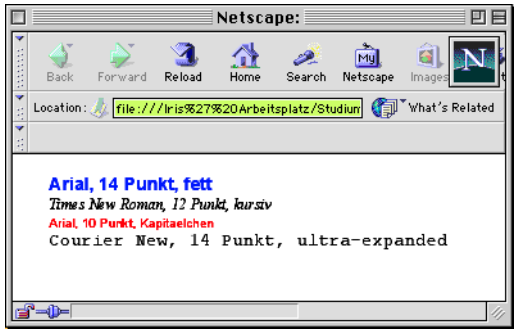


Abb. 2-30: Beispiel formatierte Texte

Software-Hinweis:
Kapitälchen und relative Fontbreite werden von Version 1 des Adobe SVG-Viewers unterstützt, während es die Version 2 beta (siehe Abbildung) nicht anzeigt.

2.2.2.4.4 Textpfade

Mit SVG können wir Texte auch an die Laufrichtung eines vorgegebenen Pfades anpassen. Das kennen wir u. a. von Landkarten, in denen sich beispielsweise Flussnamen an die zugehörige Flusslinie anschmiegen.

Für die Auszeichnung solcher „**Textpfade**“ benötigen wir das Tag `<textPath>`. Den Referenzpfad rufen wir über das Attribut ‚`xlink:href`‘ auf.

Die Syntax für das Tag `<textPath>` lautet:

```
<text>
  <textPath xlink:href="#referenzpfad"
   startOffset="[Startpunkt des Textes auf dem Pfad]">
    Dieser Text läuft entlang des Referenzpfades</textPath>
</text>
```

Werte für ‚startOffset‘: (Standard: `0`)
Angabe in jeder beliebigen Einheit

DTD: `<textPath>` ist ein Child-Tag von `<text>`.

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <defs>
[5]      <path id="Fluss" d="M0.4,0.92 c28,12, 54.67,29.33,75.33,35.33
          s48.67,4,64.67,4.67 s43.33,8.67,60.67,18.67"
          style="stroke:#00CCFF; stroke-width:2"/>
[6]    </defs>
[7]    <text style="font-family:Times New Roman; font-size:14pt;
       font-style:italic; fill:#0066FF; stroke:none">
[8]      <textPath startOffset="110" xlink:href="#Fluss">
         Flussname</textPath>
[9]    </text>
[10] </svg>
```

Ergebnis:

Abb. 2-31: Beispiel <textPath>

2.2.2.4.5 Fonts

In der Web-Graphik stehen wir immer wieder vor der Frage, auf welche Schriftarten wir zurückgreifen können, damit möglichst jeder Client das gleiche Bild vor sich hat. Denn die Websites greifen in der Regel auf den Zeichensatz des Client-Rechners zurück und da ist nicht jede Schriftart installiert. Als Folge davon wird das textuelle Erscheinungsbild fast jeder Website von Arial, Verdana und Times geprägt.

Externe Fonts

Mit Adobe Illustrator 9 ist es uns möglich, TrueType- und Type1-Fonts in ein stark komprimiertes Font-Format zu konverterieren, das vom Adobe SVG-Viewer unterstützt wird – das Compact Embedded Font-Format (CEF). Derartige Fonts können wir über den CSS-Font-Deskriptor ‚`@font-face`' und dessen Porperty ‚`src:url()`' mit einem SVG-Dokument verknüpfen.

Ausserdem werden über diesen Deskriptor sämtliche Formatierungen eingebetteter Fonts, die in Auszeichnungssprachen wie SVG in erster Linie der Semantik dienen, mit den eigentlichen Font-Daten, seien es eingebette externe Fonts oder Systemfonts, verbunden.

Der Deskriptor ‚`@font-face`' wird genau wie jedes CSS-Stylesheet im Rahmen des Tags `<style>` verwendet.

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]   <style type="text/css">
[5]   <![CDATA[
[6]      .style1{ font-family:'naked-monk-Regular'; fill:#663300;
          font-size:30; stroke:none }
[7]      .style2{ font-family:'AlmonteSnow'; fill:#FF0000; font-size:30;
          stroke:none }
[8]      @font-face{ font-family:'AlmonteSnow'; src:url(almontesnow.cef) }
[9]      @font-face{ font-family:'naked-monk-Regular';
          src:url(nakedmonk.cef) }
[10]  ]]>
[11]  </style>
[12]  <text x="25" y="40" class="style2">Almonte Snow</text>
[13]  <text x="50" y="90" class="style1">naked monk</text>
[14] </svg>
```

Ergebnis:

Abb. 2-32: Beispiel mit ‚@font-face' eingebettete Fonts

Über das Tutorial hinaus:

SVG-Fonts

Die Recommendation bietet uns noch eine weitere Alternative zum Einbetten von Fonts, die sog. **SVG-Fonts**. Dabei werden nach dem gleichen Prinzip, das wir von Graphik-Programmen kennen, in denen wir Texte in Pfade konvertieren können, einzelne Buchstaben oder Zeichen (Glyphen) in einem SVG-Dokument als Pfad abgelegt. Das hat vor allem den Vorteil, dass wir beim herausgeben eines SVG-Dokuments nicht an eine zweite Datei, den Font selbst, denken müssen. Damit sind solche Fonts auch plattform-neutral. Ausserdem lassen sich auf diese Weise Zeichen, die in einem Systemzeichensatz nicht existieren, hinzufügen.

Ausgezeichnet werden Glyphen innerhalb des Tags `<font>`. Damit wir später einem solchen Font auch Formatierungsattribute, wie Schriftart, Schriftschnitt, Zeichenabstände etc., geben können, erhalten sie eine zusätzliche Beschreibung über das Tag `<font-face>`. Schliesslich wird von Seiten der DTD noch ein `<missing-glyph>` erwartet, das eine Ersatzgraphik bereitstellt, falls ein bestimmter Glyph nicht vorhanden ist. So eine Ersatzgraphik könnte, wie wir es von den meisten Textverarbeitungsprogrammen kennen, ein Rechteck sein. Die Glyphen selbst werden über `<glyph>` definiert.

```
[1]  <font id="Beispiel_Font">
[2]   <font-face font-family="Verdana" font-weight="bold"
       font-style="normal" units-per-em="1000" cap-height="600"
       x-height="400" ascent="700" descent="300" horiz-adv-x="1000"
       alphabetic="0" mathematical="350" ideographic="400" hanging="500"/>
[3]   <missing-glyph><path d="M0,0h200v200h-200z"/></missing-glyph>
[4]   <glyph unicode="T">
[5]    <path d="M73 4504l-24 0 0 -214 -47 0 0 -23 118 0 0 23 -47 0 0 214z"/>
[6]   </glyph>
[7]  </font>
```

Werte für ‚unicode':
Ein oder mehrere Buchstaben (z. B. „ffl" -> Für die Eingabe „f"„f"„l")

Erläuterungen:

[1] <font> erhält eine ID, damit der Zeichensatz eine eindeutige Adresse hat, die später vom ‚@font-face'-Deskriptor (s. u.) aufgerufen werden kann.

[2] Innerhalb des Tags <font-face> werden diverse typographischen Eigenschaften der Glyphen näher beschrieben.

[3] Im Rahmen der Tag-Klammer <missing-glyph> wird ein Pfad in der Form eines Rechtecks definiert, der als Ersatzgraphik für fehlende Fonts dient.

[4] Schliesslich beschreiben wir den eigentlichen Glyphen. Über das Attribut ‚unicode' weisen wir diesem die entsprechende Unicode-Adresse zu. Damit weiss das Programm, für welchen Buchstaben welches Zeichen zu verwenden ist. Hier empfiehlt die Recommendation auch XML-Referenzen (Entities) zu verwenden, die in Hexadezimalen oder Dezimalen ausgedrückt werden (z. B. in Dezimal-Schreibweise: unicode="T")

[5] Mit <path> wird die Umrisslinie des jeweiligen Buchstabens „nachgezeichnet" (in diesem Fall ein „T").

Über ‚@font-face' wird der so definierte Zeichensatz aufgerufen und noch näher beschrieben:

```
[1]   <style type="text/css">
[2]    <![CDATA[
[3]     @font-face { font-family:"Verdana";src:url("#Beispiel_Font")
          format(svg) }
[4]     .systemfont { font-weight:normal;font-size:333;
          font-family:'Verdana' Beispiel_Font }
[5]    ]]>
[6]   </style>
```

Erläuterungen:

[1] Innerhalb des Tags <style> definieren wir unsere Stylesheets und Deskriptoren.

[3] Über das ‚@font-face'-Attribut ‚src' geben wir die URI unseres Beispiel-Fonts an.

[4] Mit ‚.systemfont' haben wir ein Stylesheet definiert, dass wir dem eigentlichen Text zuweisen. Dabei wird, falls der Font ‚Verdana' auf dem System installiert ist, dieser verwendet, andernfalls auf den alternativen ‚Beispiel_Font' zurückgegriffen.

Software-Hinweis:
CorelDraw verwendet beim Export diese „Form" der eingebetten Fonts. Allerdings in einer recht unschönen, nicht-validierbaren Weise: Die Font-Beschreibungen, die eigentlich innerhalb <font-face> vorgenommen werden, erscheinen hier als eigendefinierte Attribute des Tags <font>:

```
<font id="FontID0" fullFontName="AvantGarde Bk BT" fontVariant="normal"
   fontStyle="normal" fontWeight="400">
    <glyph unicode="84">
```

```
        <path d="M73 4504l-24 0 0 -214 -47 0 0 -23 118 0 0 23 -47
         0 0 214z"/>
      </glyph>
    </font>
```

Der Adobe SVG-Viewer unterstützt seit der Version 2beta SVG-Fonts (natürlich nicht die CorelDraw-Fonts).

2.2.2.5 Graphische Effekte

Seite 5

Bereits zu Anfang des Kapitels *‚Scalable Vector Graphics‘* habe ich darauf hingewiesen, dass wir in SVG auf Zeichnungen und Texte Filter anwenden können, die über das Potential der meisten vektororientierten Programme hinausgehen. Und das ist nur einer von vier graphischen Effekten, die alle vom Grundprinzip her gleich „funktionieren“. In diesem Kapitel werde ich auf

- Verläufe,
- Füllmuster,
- Beschneidungspfade und Alphamasken sowie
- Filter

näher eingehen.

2.2.2.5.1 Koordinatensysteme der Effekte

Beim Anwenden der o. g. Effekte haben wir in SVG die Wahl zwischen zwei Koordinatensystemen, in denen der jeweilige Effekt abgebildet werden soll:

- das Koordinatensystem des Dokuments, das bereits über das Root-Tag `<svg>` und dessen Attribute (`‚width‘`, `‚height‘` und `‚viewBox‘`) definiert wird, repräsentiert durch den Parameter `‚userSpaceOnUse‘`,
- de Rahmen des Zeichenobjekts, auf das sich der Effekt bezieht, ausgedrückt in dem Parameter `‚objectBoundingBox‘`.

Den Unterschied sehen wir in der nachfolgenden Graphik:

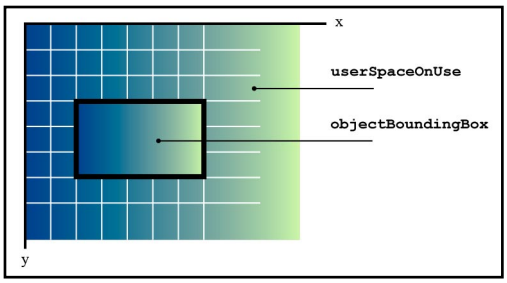
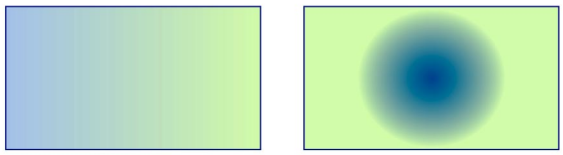


Abb. 2-33: Der Unterschied zwischen ‚userSpaceOnUse‘ und ‚objectBoundingBox‘

2.2.2.5.2 Verläufe

Unter dem Verlauf verstehen wir eine gleichmässige Abstufung von einer Quell- hin zu einer Zielfarbe. Dabei unterscheiden wir lineare von radialen Verläufen, die jeweils durch Form und Richtung des Verlaufsmusters charakterisiert werden.



Für die beiden Varianten stehen uns die Tags

- `<linearGradient>` und
- `<radialGradient>`

zur Verfügung.

Verläufe definieren

Allgemein bedienen wir uns für die Definition von Effekten, die an einer oder mehreren Stellen zugewiesen werden sollen, der besseren Strukturierung wegen des Tags `<defs>`. Danach beschreiben wir den eigentlichen Verlauf:

```
<defs>
  <linearGradient id="beispielverlauf">
    <stop offset="10%" style="stop-color:black"/>
    <stop offset="90%" style="stop-color:white"/>
  </linearGradient>
</defs>
```

Seite 31

Zugewiesen werden Verläufe schliesslich über das CSS-Property ‚`fill`‘:

```
<rect width="3cm" height="3cm" style="stroke-width:1pt;
  fill:url(#beispielverlauf)"/>
```

Lineare Verläufe spezifizieren

Einen **linearen Verlauf** spezifizieren wir in SVG grundsätzlich mit Hilfe:

- der *Verlaufsrichtung*, die durch eine Verlaufslinie repräsentiert wird,
- des *Koordinatensystems*, auf das sich die Verlaufslinie bezieht (Kapitel ‚*Koordinatensysteme der Effekte*‘),

Seite 55

Seite 68

- der *Transformation des Koordinatensystems* (siehe Kapitel ‚*Transformationen*‘) und
- des *Verlaufsverhaltens*, das besagt, ob ein Verlauf innerhalb eines Zeichenobjekts wiederholt oder gespiegelt werden soll.

Daraus ergibt sich für uns folgende Syntax zu `<linearGradient>`:

```
<linearGradient
  x1="[x-Koordinate des Startpunkts der Verlaufslinie]"
  y1="[y-Koordinate des Startpunkts der Verlaufslinie]"
  x2="[x-Koordinate des Endpunkts der Verlaufslinie]"
  y2="[x-Koordinate des Endpunkts der Verlaufslinie]"
  gradientUnits="[Koordinatensystem]"
  gradiantTransform="[Transformation des Koordinatensystems]"
  spreadMethod="[Verlaufsverhalten]"/>
```

Werte zu ‚x1', ‚y1', ‚x2' und ‚y2': (Standard für ‚x1', ‚y1' und ‚y2': `0%` / Standard für ‚x2': `100%`)

Koordinatenangaben in jeder beliebigen Einheit

Werte zu ‚gradientUnits': (Standard: `userSpaceOnUse`)

`userSpaceOnUse | objectBoundingBox`

Werte zu ‚gradientTransform':

Transformationsparameter (siehe Kapitel ‚Transformationen')

Werte zu ‚spreadMethod': (Standard: `pad`)

`pad | reflect | repeat`

Um innerhalb eines Zeichenobjekts gezielt **Verlaufsabschnitte** festzulegen, bedienen wir uns des Tags `<stop>`. Dessen Attribut ‚`offset`' kennzeichnet Beginn bzw. Ende des Verlaufmusters.

Abb. 2-34: Verlaufsabschnitte für lineare Verläufe

Quell- und Zielfarbe bzw. -opazität definieren wir über die CSS-Properties

- ‚`stop-color`' und
- ‚`stop-opacity`',

die wir beide über das Attribut ‚style' aufrufen.

```
<linearGradient>
   <stop offset="[Verlaufsabschnitt1]" style="stop-color:#330099"/>
   <stop offset="[Verlaufsabschnitt2]" style="stop-opacity:.1"/>
</linearGradient>
```

Werte für ‚offset':

Angaben in jeder beliebigen Einheit

DTD: ‚`offset`' ist Pflicht.

Radiale Verläufe spezifizieren

Für einen **radialen Verlauf** sind Attribute sinnvoll, die einen gedachten Kreis beschreiben:

- der *Kreismittelpunkt* und
- der *Kreisradius*.

Zusätzlich können wir einen *Fokuspunkt* definieren, über den wir das Verlaufszentrum vom Kreismittelpunkt wegbewegen.

Wir erhalten folgende Syntax für das Tag `<radialGradient>`:

```
<radialGradient
  cx="[x-Position des Kreismittelpunkts]"
  cy="[y-Position des Kreismittelpunkts]"
  r="[Kreisradius]"
  fx="[x-Position des Fokuspunkts]"
  fy="[y-Position des Fokuspunkts]"
  gradientUnits="[Koordinatensystem]"
  gradiantTransform="[Transformation des Koordinatensystems]"
  spreadMethod="[Verlaufsverhalten]"/>
```

Werte zu ‚cx', ‚cy', ‚fx' und ‚fy': (Standard für ‚cx' und ‚cy': `50%`)

Koordinatenangaben in jeder beliebigen Einheit

Werte zu ‚r':

Massangaben in jeder beliebigen Einheit

Werte zu ‚gradientUnits': (Standard: `userSpaceOnUse`)

`userSpaceOnUse | objectBoundingBox`

Werte zu ‚gradientTransform':

Transformationsparameter (siehe Kapitel ‚Transformationen')

Werte zu ‚spreadMethod': (Standard: `pad`)

`pad | reflect | repeat`

Auch hier definieren wir die **Verlaufsabschnitte** über das Tag `<stop>`.

```
<radialGradient>
  <stop offset="10%" style="stop-color:#FC6"/>
  <stop offset="90%" style="stop-opacity:.3"/>
</radialGradient>
```

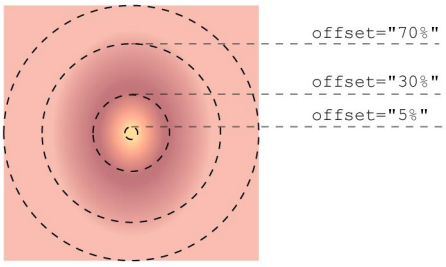


Abb. 2-35: Verlaufsabschnitte für radiale Verläufe

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]   <defs>
[5]     <radialGradient id="verl_rad" r="3cm" cx="4cm" cy="2cm"
         fx="6cm" fy="4cm" gradientUnits="objectBoundingBox">
[6]       <stop offset="0%" style="stop-color:red"/>
[7]       <stop offset="50%" style="stop-color:green"/>
[8]       <stop offset="100%" style="stop-color:blue"/>
[9]     </radialGradient>
[10]  </defs>
[11]  <circle r="1cm" cx="4.5cm" cy="1.5cm" style="fill:url(#verl_rad)"/>
[12] </svg>
```

Ergebnis:

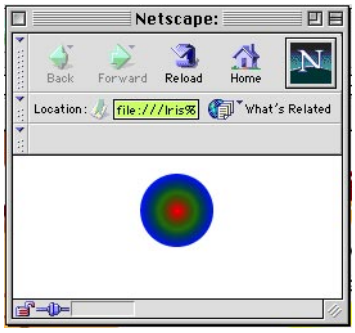


Abb. 2-36: Beispiel <radialGradient>

2.2.2.5.3 Füllmuster

Bei Füllmustern handelt es sich um Graphiken, Texte oder Bilder, die sich innerhalb der Fläche eines Zeichenobjekts so oft wiederholen, bis diese vollständig ausgefüllt ist.

Für deren Auszeichnung gibt es in SVG das Tag `<pattern>`.

Füllmuster definieren

Wir definieren zunächst das Füllmuster der besseren Strukturierung wegen innerhalb `<defs>`:

```
<defs>
  <pattern id="beispielmuster" width="1cm" height="1cm">
    <text x="1" y="1" style="font-family:Arial;
     font-size:7pt">text</text>
  </pattern>
</defs>
```

Seite 31

Danach weisen wir es mit Hilfe des CSS-Properties ‚`fill`' einem Zeichenobjekt zu:

```
<rect width="3cm" height="3cm" style="fill:url(#beispielmuster)"/>
```

Füllmuster spezifizieren

Mit Hilfe von <pattern> machen wir aus jeder „x“-beliebigen Graphik eine „Muster“-Graphik (Musterkachel). Dazu können wir folgende Angaben genauer spezifizieren:

- den *Startpunkt der Graphikelemente* innerhalb einer Musterkachel,
- die *Ausmasse einer Musterkachel*,
- das *Koordinatensystem*, in das die Kacheln projeziert werden, Seite 55
- Das *Koordinatensystem*, in das der *Inhalt* der Kacheln projeziert wird und
- die *Transformation des Koordinatensystems* (siehe Kapitel ‚*Transformationen*‘). Seite 68

Demnach erhalten wir folgende Syntax für <pattern>:

```
<pattern
  x="[x-Koordinate des Startpunkts der Graphikelemente]"
  y="[y-Koordinate des Startpunkts der Graphikelemente]"
  width="[Breite der Musterkachel]"
  height="[Höhe der Musterkachel]"
  patternUntits="[Koordinatensystem der Kachel]"
  patternContentUnits="[Koordinatensystem des Inhalts der Kachel]"
  patternTransform="[Transformation des Koordinatensystems]"/>
```

Werte für ‚x‘ und ‚y‘:

Koordinatenangaben in jeder beliebigen Einheit

Werte für ‚width‘ und ‚height‘:

Massangaben in jeder beliebigen Einheit

Werte für ‚patternUnits‘ und ‚patternContentUnits‘:
(Standard: `userSpaceOnUse`)

`userSpaceOnUse` | `objectBoundingBox`

Werte für ‚patternTransform‘:

Transformationsparameter

DTD: ‚`width`‘ und ‚`height`‘ sind Pflicht.

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]   <defs>
[5]     <pattern id="muster" width="1cm" height="1cm" x="2mm" y=2mm">
[6]       <rect width="6mm" height="6mm" style="stroke:none; fill:blue"/>
[7]     </pattern>
[8]    </defs>
[9]    <path d="M200,75H0V0h200v75z" style="stroke:black;
        fill:url(#muster)"/>
[10] </svg>
```

Erläuterungen:

[5-7] Hier wird eine Musterkachel mit den Ausmassen 1 cm x 1 cm definiert. Die darin enthaltenen Graphikelemente sollen 2 mm von links und 2 mm oberhalb der Kachel beginnen. Da hier keine Angabe über das Koordinatensystem gemacht wurde, wird standardmässig das Koordinatensystem des Dokuments (‚userSpaceOnUse') verwendet.

[6] Als Kachelinhalt wird ein blaues Rechteck mit der Kantenlänge 6 auf 6 mm festgelegt.

[9] Mit ‚fill:url' weisen wir das Füllmuster einem Pfad zu.

Ergebnis:

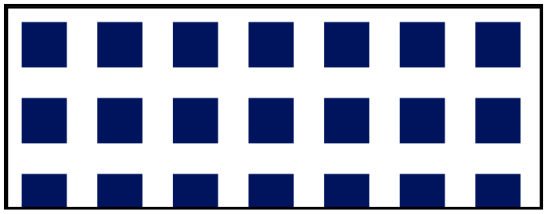
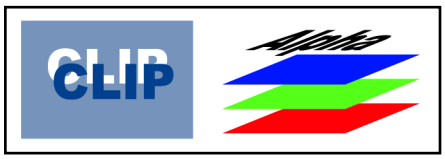


Abb. 2-37: Beispiel <pattern>

Software-Hinweis:
<pattern> soll, laut Support-Bericht, von der Version 2 des Adobe SVG-Viewers unterstützt werden. Das ist jedoch bei den momentan verfügbaren Beta-Versionen noch nicht der Fall.

2.2.2.5.4 Masken (‚clipping' und ‚masking')

Ein häufig benötigter graphischer Effekt ist die „Maskierung" (gängig sind in dem Zusammenhang auch die englischen Bezeichnungen „clipping" und „masking"). Dabei dient uns die Kontur eines beliebigen geschlossenen Pfades als Beschneidungsmaske für eine andere Graphik (Clipping). Darüberhinaus können wir beliebige Graphikelemente als zusätzliche Kanäle definieren, die zu den bereits bestehenden, wie RGB, bildbearbeitungstechnisch addiert werden – die Rede ist von sog. Alphamasken (Masking).



SVG bietet uns zum „Clippen" und „Maskieren" die beiden Elemente

- <clipPath> und
- <mask>.

Masken definieren

Innerhalb des Elements <defs> definieren wir, der besseren Strukturierung wegen, unsere Masken. Danach verwenden wir die Tags <clipPath> bzw. <mask> als Rahmenelemente für die eigentlichen Beschneidungspfade bzw. Alphamasken:

```
<defs>
  <clipPath id="beispielclip">
    <circle cx="2cm" cy="2cm" r="1cm"/>
  </clipPath>
</defs>
```

Zum Zuweisen der Masken verwenden wir die CSS-Properties ‚clip-path' bzw. ‚mask':

```
<text x="1cm" y="2cm" style="font-family:Arial; font-weight:bold;
  clip-path:url(#beispielclip)">CLIP</text>
```

Beschneidungspfade spezifizieren

Einen **Beschneidungspfad** definieren wir mit Hilfe von Pfaden, Grundformen und Texten. Dabei können wir das Koordinatensystem, in das die Einheiten des Inhalts des Beschneidungspfads übertragen werden, noch näher spezifizieren.

Wir erhalten die folgende Syntax für <clipPath>:

```
<clipPath clipPathUnits="[Koordinatensystem]"/>
```

Werte für ‚clipPathUnits': (Standard: userSpaceOnUse)
userSpaceOnUse | objectBoundingBox

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg width="250" height="200" viewBox="0 0 125 100">
[4]    <defs><!-- Beschneidungspfad -->
[5]      <clipPath id="clip" clipPathUnits="objectBoundingBox">
[6]        <text x="0.5cm" y="4.5cm" style="font-size:2cm; fill:none;
             font-family:'Arial-Black'">SKY</text>
[7]      </clipPath>
[8]    </defs>
[9]    <g><!-- Bild, das beschnitten wird -->
[10]     <image xlink:href="sky.jpg"  width="336" height="225"
         style="clip-path:url(#clip)"/>
[11]   </g>
[12] </svg>
```

Erläuterungen:

[5-7] Hier wird die Umrisslinie eines Textes als Beschneidungspfad definiert.

Ergebnis:

Abb. 2-38: Beispiel <clipPath>

Über das Tutorial hinaus:

Mit dem CSS-Property ‚clip-rule' können wir ausserdem die Clip-Methode festlegen. Dazu stehen uns die gleichen Werte zur Verfügung wie für das Property ‚fill-rule':

Seite 31

- ‚nonzero',
- ‚evenodd' und
- ‚inherit'.

Alphamasken spezifizieren

Auf den ersten Blick erhalten wir mit „Clipping" und „Masking" ein ähnliches Ergebnis, die Funktionsweise ist jedoch ganz anders. Innerhalb des Elements <mask> definieren wir mit Hilfe von Pfaden, Grundformen und/oder Texten einen **Alphakanal**, der später einem Zeichenobjekt mit ‚mask:url()' zugewiesen wird. Wir erhalten hier nur den Eindruck einer „beschnittenen" Zeichnung.

Mit folgenden Festlegungen können wir einen Alphakanal in SVG näher beschreiben:

- der Lage eines (Puffer-)Rechtecks (x- und y-Koordinate), auf das der Alphakanal angewandt wird,
- den Ausmassen dieses Rechtecks,

Seite 55

- dem *Koordinatensystem*, in das dieses Rechteck hineinprojeziert wird,
- dem Koordinatensystem, in das die eigentliche Alphamaske (der Inhalt) hineinprojeziert wird.

Daraus ergibt sich die folgende Syntax zu <mask>:

```
<mask
  x="[x-Koordinate für die Lage des Rechtecks]"
  y="[y-Koordinate für die Lage des Rechtecks]"
  width="[Breite des Rechtecks]"
  height="[Höhe des Rechtecks]"
  maskUnits="[Koordinatensystem des Rechtecks]"
  maskContentUnits="[Koordinatensystem der Alphamaske]"/>
```

Werte für ‚x' und ‚y': (Standard: -10%)
Koordinatenangaben in jeder beliebigen Einheit

Werte für ‚width' und ‚height':
Massangaben in jeder beliebigen Einheit

Werte für ‚maskUnits' und ‚maskContentUnits': (Standard: userSpaceOnUse)
userSpaceOnUse | objectBoundingBox

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <defs><!-- Alphakanal -->
[5]      <mask id="alpha">
[6]        <circle cx="2cm" cy="1.5cm" r="1cm"/>
[7]        <circle cx="7cm" cy="1.5cm" r="1cm"/>
[8]      </mask>
[9]    </defs>
[10]   <!-- Beschneidungspfad fuer Alphakanal -->
[11]   <text id="Text" x="1cm" y="2cm" style="font-family:'Arial-Black';
        font-stretch: ultra-expanded; font-size:40pt;">MASK</text>
[12]   <!-- Referenz von Alphakanal u. Beschneidungspfad -->
[13]   <use xlink:href="#Text" style="fill:orange; mask:url(#alpha)"/>
[14] </svg>
```

Erläuterungen:

[5-7] Hier werden zwei Kreise als Alphamasken definiert.

[11] Dieser Text liegt im Hintergrund (schwarz).

[13] Mit <use> wird über ‚xlink:href' der selbe Text referenziert. Auf ihn wird über ‚mask:url' die Alphamaske gelegt. Somit sehen wir von dem referenzierten Text nur noch die „beschnittenen" orangenen Kreise.

Ergebnis:

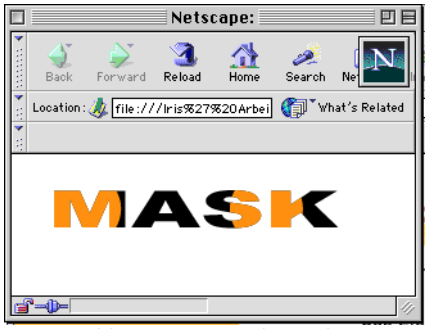
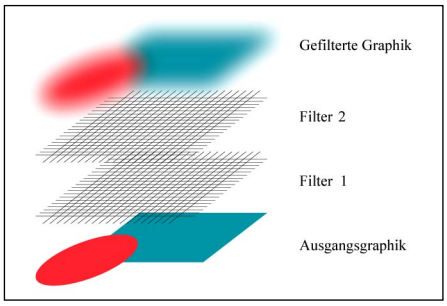


Abb. 2-39: Beispiel <mask>

2.2.2.5.5 Filter

Graphikfilter stammen ursprünglich aus der digitalen Bildverarbeitung, die sich ausschliesslich mit Rasterdaten beschäftigt. Mit SVG können wir solche auch auf Vektorgraphiken anwenden. Wie beim Kaffee filtrieren Filter auch in der Graphik einige Bestandteile heraus, während andere bestehen bleiben. Und wie beim Kaffee sieht auch hier das Filtrat anders aus als die Ausgangselemente.



Für die Auszeichnung eines Filters verwenden wir das Tag `<filter>`.

Filter definieren

Innerhalb des Tags `<defs>` nehmen wir unsere Definitionen vor. Darauffolgend setzen wir `<filter>` als Rahmenelement für die eigentlichen Filtereffekte.

```
<defs>
   <filter id="beispielfilter">
      <feGaussianBlur in="SourceAlpha" stdDeviation="2" result="blur"/>
   </filter>
</defs>
```

Aufgerufen werden Filter über das CSS-Property ‚`filter`‘:

```
<text x="2cm" y="2cm" style="font-family:Impact; font-size:30pt;
  filter:url(#beispielfilter)">Filter</text>
```

Filter spezifizieren

Mit Hilfe von `<filter>` können wir einen Filter auf der Anzeigefläche genau plazieren. Dazu enthält es Attribute, mit denen wir Angaben machen können zu:

- der Lage des Filters,
- den Ausmassen (Breite und Höhe) des Filterrechtecks,
- dem *Koordinatensystem*, auf das sich der Filter beziehen soll,
- dem Koordinatensystem, in das die Einheiten der Filtereffekte hineinprojeziert werden und
- der Auflösung (Pixelwerte von Breite und Höhe).

Daraus ergibt sich die folgende Syntax zu `<filter>`:

Seite 55

```
<filter
  x="[x-Koordinate für die Lage des Filters]"
  y="[y-Koordinate für die Lage des Filters]"
  width="[Breite des Filters]"
  height="[Höhe des Filters]"
  filterUnits="[Koordinatensystem]"
  primitiveUnits="[Koordinatensystem für die Filtereffekte]"
  filterRes="[Pixelauflösung]"/>
```

Werte für ‚x' und ‚y':

Koordinatenangaben in jeder beliebigen Einheit

Werte für ‚width' und ‚height':

Massangaben in jeder beliebigen Einheit

Werte für ‚filterUnits' und ‚primitiveUnits': (Standard: `userSpaceOnUse`)

`userSpaceOnUse` | `objectBoundingBox`

Werte für ‚filterRes':

Pixelbreite und -höhe (Höhe ist optional)

Anmerkung: Die Festlegung von Lage und Ausmass des Filters ist sinnvoll, da damit der Bildaufbau des für den Viewer ansonsten sehr rechen-intensiven Effekts stark beschleunigt werden kann.

Filter anwenden

Filter eröffnen uns ein riesiges Spektrum an graphischen Möglichkeiten wie Schattierungen und 3-D-Effekten. Um zu einem gewünschten Ergebnis zu gelangen, müssen wir mitunter mehrere Filter „übereinanderlegen" oder, anders ausgedrückt, sie miteinander verrechnen (letzteres ist ein Begriff aus der digitalen Bildverarbeitung, der besagt, dass Farb- bzw. Grauwerte eines Bildes Pixel für Pixel mit einer Matrix oder einem anderen Bild addiert, multipliziert etc. werden). Nachfolgend wird anhand eines Beispiels beschrieben, wie wir in SVG vorgehen müssen, um dieses „Hintereinanderschalten" von Filtern zu steuern.

Für die Steuerung unterstehen den Filtereffekten zunächst die beiden Attribute

- ‚`in`' und
- ‚`result`'.

Mit ‚`in`' geben wir an, auf welche Quelle ein Filtereffekt angewendet werden soll. Dabei kann es sich um ein Bild, einen Alphakanal, eine Farbe oder das Ergebnis (result) eines anderen Filtereffekts handeln.

Über ‚`result`' erstellen wir eine Filter-Referenz, die einem anderen Filtereffekt als Input-Parameter (in) zugewiesen werden kann.

```
<filter id="beispiel_filter">
  <feGaussianBlur in="SourceAlpha" stdDeviation="3" result="erg_blur"/>
  <feOffset in="erg_blur" dx="5" dy="5" result="erg_offset"/>
  ...
</filter>
```

Was passiert nun im einzelnen, wenn mehrere Filter „hintereinandergeschaltet“ werden?

```
<!-- Beispiel aus der SVG-Recommendation (CR-SVG-20001102) -->

[1]  ...
[2]  <filter id="MyFilter">
[3]  <desc>Produces a 3D lighting effect.</desc>
[4]    <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
[5]    <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
[6]    <feSpecularLighting in="blur" surfaceScale="5" specularConstant="1"
        specularExponent="10" style="lighting-color:white"
        result="specOut">
[7]       <fePointLight x="-5000" y="-10000" z="20000"/>
[8]    </feSpecularLighting>
[9]    <feComposite in="specOut" in2="SourceAlpha" operator="in"
        result="specOut"/>
[10]   <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
        k1="0" k2="1" k3="1" k4="0"  result="litPaint"/>
[11]   <feMerge>
[12]     <feMergeNode in="offsetBlur"/>
[13]     <feMergeNode in="litPaint"/>
[14]   </feMerge>
[15] </filter>
[16] ...
```

Erläuterungen:

Unsere Ausgangsgraphik das Logo des Tutorials:

[4] Der Gauss’sche Weichzeichner wird auf den Alphakanal der Ausgangsgraphik angewendet. Ergebnis: ‚`blur`‘.

[5] Das Ergebnis ‚`blur`‘ wird jeweils um 4 Pixel nach links und nach unten verschoben. Ergebnis: ‚`offsetBlur`‘.

[6] Die Oberfläche des Ergebnis’ ‚`blur`‘ wird zunächst um den Skalierungsfaktor 5 „erhöht“. Danach wird für den plastischen Eindruck ein Beleuchtungseffekt zugewiesen. Dazu wird über das Child-Tag `<fePointLight>` ein Punkt im Raum als „Lichtquelle“ festgelegt. Ergebnis: ‚`specOut`‘. (Damit man diesen Effekt überhaupt sehen kann, wurde

ein graues Rechteck hinterlegt.)

[9] Das Ergebnis ‚specOut' wird mit dem Alphakanal der Quellgraphik verrechnet. Ergebnis: ‚specOut'.
(Auch hier wurde ein graues Rechteck hinterlegt.)

[10] Nun wird die Quellgraphik mit dem Ergebnis ‚specOut' verrechnet. Ergebnis: ‚litPaint'.

[11] Abschliessend werden über <feMerge> und dessen Child-Tag <feMergeNode> die Ergebnis-„Filtrate" ‚litPaint' (Graphik) und ‚offsetBlur' (Schatten) aufeinandergelegt.

Software-Hinweis:
Dem Filtereffekt <feSpecularLighting> können wir laut Recommendation das Property ‚lighting-color' zuweisen, über das wir die Farbe für den „Glanz" festlegen. Damit funktioniert der Effekt im Adobe SVG-Viewer allerdings nicht. Hier müssen wir auf eine „Eigen-Kreation" von Adobe zurückgreifen: das Attribut ‚lightColor' (Beispiel: lightColor="white").

2.2.2.6 Transformationen

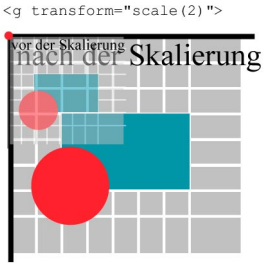
SVG bietet uns mehrere Möglichkeiten, um Graphiken zu skalieren, zu rotieren, zu verschieben oder zu neigen. Alle Transformationen werden innerhalb des Attributs ‚transform' vorgenommen.

```
<g transform="scale(2)">
```

Werte (Befehle) für ‚transform':
scale (zum Skalieren) |
rotate (zum Rotieren) |
translate (zum Verschieben) |
skewX | skewY (zum Neigen in x- oder y-Richtung) |
matrix (zum allgemeinen Verzerren einschliesslich Skalieren, Rotieren, Verschieben und Neigen).

Skalieren

Beim **Skalieren** werden die Achsen-Einheiten des Koordinatensystems um den entsprechenden Faktor vergrössert oder verkleinert:



Die Syntax für den Befehl ‚scale' lautet:

```
<... transform="scale([Skalierungsfaktor])" ...>
```

oder für die nicht-proportionale Skalierung

```
<... transform="scale([Skalierung-x], [Skalierung-y])" ...>
```

Beispiel:

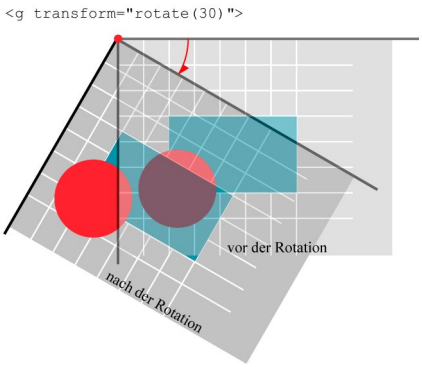
```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <g style="stroke:none"><!-- keine Transformation -->
[5]      <text x="5" y="30" style="font-family:Verdana;
          font-size:12; fill:blue">mittel</text>
[6]    </g>
[7]    <!-- ******** verkleinern ******** -->
[8]    <g transform="scale(0.5)" style="stroke:none">
[9]      <text x="5" y="30" style="font-family:Verdana; font-size:12;
          fill:red">klein</text>
[10]   </g>
[11]   <!-- ******** vergroessern ******** -->
[12]   <g transform="scale(3, 1.5)" style="stroke:none">
[13]     <text x="5" y="30" style="font-family:Verdana; font-size:12;
          fill:red">GROSS UND BREIT</text>
[14]   </g>
[15] </svg>
```

Ergebnis:

Abb. 2-40: Beispiel ‚scale'

Rotieren

Bei der Rotation drehen wir das Koordinatensystem im entsprechenden Rotationswinkel um den Ursprung. Dabei wird im Ggs. zu Graphikprogrammen bei positiven Winkelwerten im Uhrzeigersinn gedreht:



Die Syntax für den Befehl ‚rotate' lautet:

```
<... transform="rotate([Rotationswinkel])" ...>
```

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <g style="stroke:none"><!-- keine Transformation -->
[5]      <text x="50" y="30" style="font-family:Verdana; font-size:10;
          fill:red">Text und Linie wurden nicht rotiert</text>
[6]      <line x1="50" y1="35" x2="270" y2="35" style="stroke:black;
          stroke-width:2"/>
[7]    </g>
[8]    <g transform="translate(26,-20)">
[9]    <!-- ******** rotieren ******** -->
[10]     <g transform="rotate(30)" style="stroke:none">
[11]       <text x="50" y="30" style="font-family:Verdana; font-size:10;
            fill:red">Text und Linie wurden um 30 Grad rotiert</text>
[12]       <line x1="50" y1="35" x2="270" y2="35"
            style="stroke:black;stroke-width:2"/>
[13]     </g>
[14]   </g>
[15] </svg>
```

Erläuterungen:

[8] Dieses Beispiel zeigt, dass wir mehrere Transformationen aufeinander folgen lassen können. An dieser Stelle wurde der eigentlichen Rotation eine Translation vorweggenommen (die im nächsten Abschnitt beschrieben wird). Sonst würden sich die Linien nicht im Start-Punkt berühren.

Ergebnis:

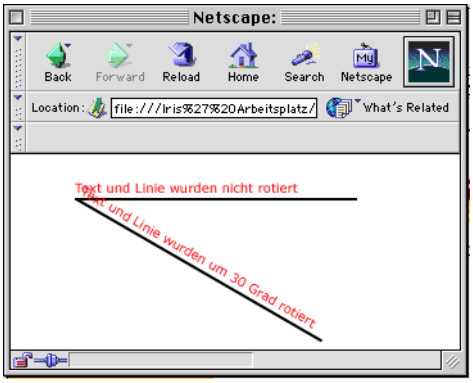
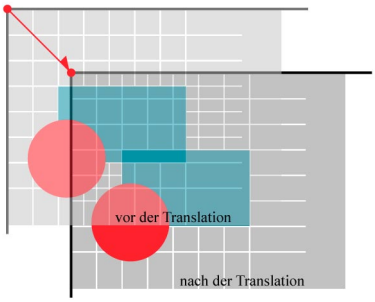


Abb. 2-41: Beispiel ‚rotate'

Verschieben

Bei der Translation wird der Ursprung des Koordinatensystems um die entsprechenden Werte verschoben:

```
<g transform="translate(50, 50)">
```



Die Syntax für den Befehl ‚translate' lautet:

```
<... transform="translate([x-Translation], [y-Translation])" ...>
```

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <g style="stroke:none"><!-- keine Transformation -->
[5]      <text x="0" y="30" style="font-family:Verdana; font-size:10;
          fill:red">Text und Linie wurden nicht verschoben</text>
[6]      <line x1="0" y1="35" x2="360" y2="35" style="stroke:black;
          stroke-width:2"/>
[7]    </g>
[8]    <!-- ******** verschieben ******** -->
[9]    <g transform="translate(50,50)" style="stroke:none">
[10]     <text x="0" y="30" style="font-family:Verdana; font-size:10;
          fill:red">Text und Linie wurden um 30 Punkte in x- und y-Richtung
          verschoben</text>
[11]     <line x1="0" y1="35" x2="360" y2="35" style="stroke:black;
          stroke-width:2"/>
[12]   </g>
[13] </svg>
```

Ergebnis:

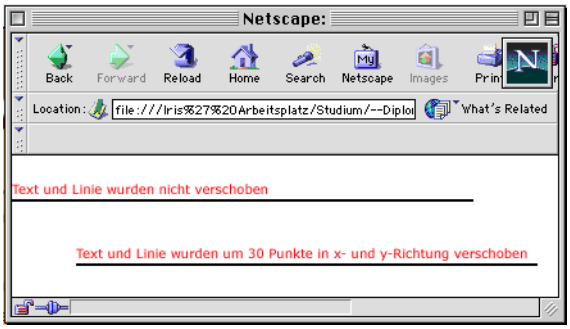
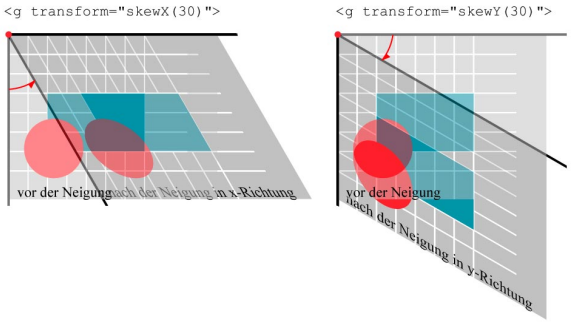


Abb. 2-42: Beispiel ‚translate'

Neigen

Bei der Neigung wird der Winkel zwischen den beiden Koordinatenachsen über den entsprechenden Neigungswert verändert. Dabei wird bei positivem Neigungswinkel in x-(y-)Richtung die x-(y-)Achse gegen den (im) Uhrzeigersinn gedreht:



Die Syntax für den Befehl ‚skewX' bzw. ‚skewY' lautet:

```
<... transform="skewX([Neigungswinkel in x-Richtung])" ...>

<... transform="skewY([Neigungswinkel in y-Richtung])" ...>
```

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <g style="stroke:none"><!-- keine Transformation -->
[5]      <text x="20" y="30" style="font-family:Verdana; font-size:10;
          fill:red">Text und Linie wurden nicht geneigt</text>
[6]      <line x1="20" y1="35" x2="350" y2="35" style="stroke:black;
          stroke-width:2"/>
[7]    </g>
[8]    <!-- ******** neigen ******** -->
[9]    <g transform="skewX(30)" style="stroke:none">
[10]     <text x="20" y="60" style="font-family:Verdana; font-size:10;
          fill:red">Text und Linie wurden um 30 Grad in x-Richtung
          geneigt</text>
[11]     <line x1="20" y1="65" x2="350" y2="65" style="stroke:black;
          stroke-width:2"/>
[12]   </g>
[13] </svg>
```

Ergebnis:

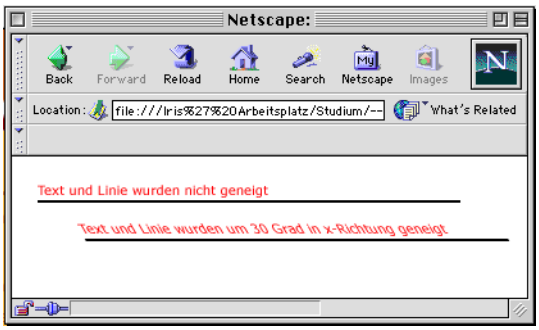


Abb. 2-43: Beispiel ‚skew'

Matrizentransformation

Allgemein lassen sich mit Hilfe einer 3 x 3 Matrix alle möglichen Transformationen vornehmen:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Da es innerhalb der Matrix nur 6 Variablenwerte gibt (a - f), genügt es, die Matrix in einem Vektor [a b c d e f] auszudrücken.

Wir erhalten demnach folgende Schreibweise für den Befehl ‚`matrix`':

```
<... transform="matrix([a] [b] [c] [d] [e] [f])">
```

Beispiel:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN" "http://
      www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg 20001102.dtd">
[3]  <svg>
[4]    <g style="stroke:none"><!-- keine Transformation -->
[5]      <text x="50" y="30" style="font-family:Verdana; font-size:10;
          fill:red">Text und Linie wurden nicht veraendert</text>
[6]      <line x1="50" y1="35" x2="265" y2="35" style="stroke:black;
          stroke-width:2"/>
[7]    </g>
[8]    <!-- ******** Matrix ******** -->
[9]    <g transform="matrix(.707 .707 -.707 .707 40 -25)">
[10]     <text x="50" y="30" style="font-family:Verdana; font-size:10;
          fill:red;stroke:none">Der Text wurde gedreht und verschoben
         </text>
[11]     <line x1="50" y1="35" x2="265" y2="35" style="stroke-width:2"/>
[12]   </g>
[13] </svg>
```

Erläuterungen:

[9] Die ersten 4 Parameter von ‚`matrix`' bewirken eine Drehung, während die letzten beiden eine Verschiebung zur Folge haben (siehe unten ‚*Was geschieht bei der Matrizentransformation?*'). Bei einer reinen Drehung würden sich hier die Linien nicht berühren.

Ergebnis:

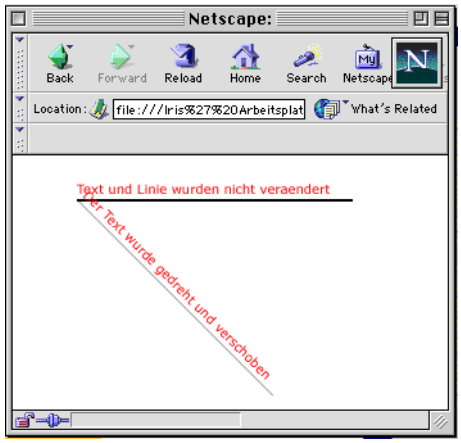


Abb. 2-44: Beispiel ‚matrix'

Was geschieht eigentlich bei der Matrizentransformation?

Grundsätzlich wird ein Vektor, der die Koordinaten x und y enthält, mit der oben erwähnten 3x3-Matrix multipliziert.

$$\begin{bmatrix} x_{neu} \\ y_{neu} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \mathbf{X} \begin{bmatrix} x_{alt} \\ y_{alt} \\ 1 \end{bmatrix}$$

Daraus ergeben sich die Matrizen für die einzelnen Transformationen „Skalieren", „Rotieren", „Verschieben" und „Neigen":

Skalieren:

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
matrix([sx] 0 0 [sy] 0 0)
```

Rotieren:

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
matrix([cos(a)] [sin(a)]
[-sin(a)] [cos(a)] 0 0)
```

Verschieben:

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

```
matrix(1 0 0 1 [tx] [ty])
```

Neigen:

in x-Richtung

$$\begin{bmatrix} 1 & \tan(a) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

in y-Richtung

$$\begin{bmatrix} 1 & 0 & 0 \\ \tan(a) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
matrix(1 0 [tan(a)] 1 0 0)
matrix(1 [tan(a)] 0 1 0 0)
```

Anmerkung: Die Tatsache, dass in SVG das Koordinatensystem und nicht die Graphik selbst transformiert wird, finde ich nachteilig, da es sehr benutzerunfreundlich ist. Denn man wundert sich im ersten Moment schon, wenn eine Graphik nach der Rotation nicht mehr auf dem Bildschirm erscheint, da sie sich nun ausserhalb des Fensterbereichs befindet.

Hier wäre es meines Erachtens besser, wenn wir neben dem Transformationsmodus und dessen Parameter zusätzliche Festlegungen treffen könnten, wie wir sie auch von Graphikprogrammen her kennen:

- den Transformationsmittelpunkt,
- das Koordinatensystem auf das sich die Transformation bezieht (vgl. Kapitel *‚Koordinatensysteme der Effekte‘*).

Seite 55

Um das umzusetzen, gäbe es zwei Möglichkeiten:

- die Erweiterung der Parameterliste der einzelnen Befehle oder
- die Definition eines neuen Elements, z. B. `transform`. Diesem könnten wir die o. g. Festlegungen in Form von Attributen zuschreiben. Über die ID liese sich die Transformation dann beispielsweise einer Gruppe zuordnen.

Der letzte Punkt wiederspricht vielleicht der Semantik, denn SVG-Elemente stehen, mit Ausnahme der Animationen, die allerdings aus einem anderen Standard übernommen wurden, für klar festgelegte Zustände, wie der Form eines Zeichenobjekts, während die Transformation eine Veränderung derselben bewirkt. Hinzu kommt, dass Transformationen sehr häufig und in zu unterschiedlicher Form gebraucht werden, was unzählige Transformationsdefinitionen zur Folge hätte.

Seite 75

2.2.3 Multimedia

(Dieser und die folgenden Abschnitte des Kapitels 2.2 wurden im Rahmen dieser Arbeit für das Tutorial nicht mehr umgesetzt.)

2.2.3.1 Animationen

SVG enthält einige Elemente und Attribute des XML-Multimediastandards **S**ynchronized **M**ultimedia **I**ntegration **L**anguage (SMIL), mit denen wir diverse Eigenschaften eines Zeichenobjekts in Abhängigkeit von der Zeit steuern können. Solche Eigenschaften sind beispielsweise

- Position (Bewegung),
- Sichtbarkeit,
- Farbe und
- Transformation.

Zum Erstellen einer Animation, bedürfen wir einiger grundlegender Komponenten. Hierzu finden wir im Mulitmedia-Programm Macromedia Director Werkzeuge, die Bezeichnungen tragen, die an ein reales Filmstudio erinnern sollen: „Besetzung“ bzw. „Darsteller“ und „Drehbuch“. Das Drehbuch enthält einen Zeitstrahl, anhand dessen wir genau festlegen können, wann welcher Darsteller „in Aktion“ zu treten hat. Da es sich bei Director um ein WYSIWYG-Tool handelt, können wir dessen Werkzeuge nur als Vergleichsobjekte behandeln, denn was wir hier nicht bewusst wahrnehmen: Es sind eigentlich die Eigenschaften eines Darstellers, die animiert werden. Und so müssen wir in SMIL bzw. SVG explizit die Eigenschaften eines Zeichenobjekts

oder Textes ansprechen um eine Veränderung zu erzielen. Als Rahmenelemente dienen uns spezielle Tags. Das sind zunächst die aus SMIL übernommenen

- `<animate>` (zum Auszeichnen von Animationen numerischer Attribute und Properties),
- `<set>` (entspricht eigentlich `<animate>`, kann aber auch für nicht-numerische Attribute und Properties eingesetzt werden, z. B. ‚`visibility`‘),
- `<animateMotion>` (zum Auszeichnen von Bewegungen entlang eines Pfads. Dieser wird über das Attribut(!!!) ‚`path`‘ festgelegt) und
- `<animateColor>` (zum Auszeichnen von Farbveränderungen)

Darüberhinaus wurden in SVG auch noch zusätzliche Elemente und Attribute definiert, die mit den SMIL-Elementen zusammenarbeiten:

- `<animateTransform>` (zum Auszeichnen einer animierten Transformation) und
- `<mPath>` (zum spezifizierteren Auszeichnen eines Bewegungspfades).

DTD: `<mPath>` ist ein Child-Tag von `<animateMotion>`.

Für die Spezifizierung der **„Darsteller“-Eigenschaften** verwenden wir die Attribute

- ‚`attributeName`‘ und

Werte für ‚attributeName‘:
Bezeichnung des Properties bzw. Attributs, das animiert werden soll

- ‚`attributeType`‘.

Werte für ‚attributeType‘: (Standard: `auto`)
`CSS` (für CSS-Properties) | `XML` (für XML-Attribute) | `auto`

Hinzu kommen einige Steuerungsattribute, mit denen wir die **Zustände von Eigenschaften** angeben können, die die eigentliche Animation zum Ausdruck bringen (*von* Zustand1 *zu* Zustand2). Die wichtigsten sind:

- ‚`from`‘,
- ‚`to`‘ und
- ‚`by`‘ (für eine relative Zustandsveränderung).

Schliesslich benötigen wir noch Attribute zur **Steuerung der Zeit.** Auch hier werde ich nur die wichtigsten auflisten:

- ‚`begin`‘ (Zeitpunkt des Beginns der Animation),

Werte für ‚begin‘:
Liste mit verschiedenen Werten z. T. aus der SMIL-Spezifikation, von denen die meisten in Zeitangaben (`h`, `min` und `s`) ausgedrückt werden | `indefinite`

- ‚dur‘ (Dauer der Animation),

SVG-relevante **Werte für ‚dur‘:**
Zeitangabe in `h`, `min` und `s` | `indefinite`

- ‚end‘ (Zeitpunkt der Beendigung der Animation),

SVG-relevante **Werte für ‚end‘:**
Liste mit verschiedenen Werten z. T. aus der SMIL-Spezifikation, von denen die meisten in Zeitangaben (`h`, `min` und `s`) ausgedrückt werden | `indefinite`

- ‚repeatCount‘ (Anzahl der Wiederholungen einer Animation) und

Werte für ‚repeatCount‘:
Anzahl der Wiederholungen | `indefinite`

- ‚fill‘ (Zustand des Darstellers nach der Animation).

Werte für ‚fill‘:
`freeze` | `remove`

Hierzu ein Beispiel aus der Recommendation:

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
      "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
[3]  <svg width="8cm" height="3cm" viewBox="0 0 800 300">
[4]  <desc>Example anim01 - demonstrate animation elements</desc>
[5]    <rect id="RectElement" x="300" y="100" width="300" height="100"
        style="fill:rgb(255,255,0)" >
[6]      <animate attributeName="x" attributeType="XML"
          begin="0s" dur="9s" fill="freeze" from="300" to="0" />
[7]      <animate attributeName="y" attributeType="XML"
          begin="0s" dur="9s" fill="freeze" from="100" to="0" />
[8]      <animate attributeName="width" attributeType="XML"
          begin="0s" dur="9s" fill="freeze" from="300" to="800" />
[9]      <animate attributeName="height" attributeType="XML"
          begin="0s" dur="9s" fill="freeze" from="100" to="300" />
[10]   </rect>
[11]   <g transform="translate(100,100)" >
[12]     <text id="TextElement" x="0" y="0" style="font-family:Verdana;
          font-size:35.27; visibility:hidden">It's alive!
[13]       <set attributeName="visibility" attributeType="CSS"
            to="visible" begin="3s" dur="6s" fill="freeze" />
[14]       <animateMotion path="M 0 0 L 100 100" begin="3s" dur="6s"
            fill="freeze" />
[15]       <animateColor attributeName="fill" attributeType="CSS"
            from="rgb(0,0,255)" to="rgb(128,0,0)" begin="3s" dur="6s"
            fill="freeze" />
[16]       <animateTransform attributeName="transform"
            attributeType="XML" type="rotate" from="-30" to="0"
            begin="3s" dur="6s" fill="freeze" />
[17]       <animateTransform attributeName="transform"
            attributeType="XML" type="scale" from="1" to="3"
            additive="sum" begin="3s" dur="6s" fill="freeze" />
```

```
[18]       </text>
[19]     </g>
[20]  </svg>
```

Erläuterungen:

[6] Ein gelbes Rechteck wird definiert, das animiert werden soll.

[7] Mit `<animate animateName="x">` wird die Animation der x-Koordinate des linken oberen Eckpunkts des Rechtecks eingeleitet. Dieser bewegt sich von Beginn der Dokument-Aktivierung an (`begin="0s"`) von Position „300" zu „0". Die Dauer der Bewegung beträgt 9 Sekunden. Am Ende soll die Position so beibehalten werden (`fill="freeze"`).

[8-10] Nach dem gleichen Schema wird mit der y-Koordinate sowie den Attributen ‚`width`' und ‚`height`' fortgefahren.

[11] Mit `transform="translate(100,100)"` wird zunächst der Ursprung des Koordinatensystems verschoben, damit der folgende Text noch im Fensterbereich zu sehen ist.

[12] Der Text „It's alive!" wird ausgezeichnet und über das CSS-Property ‚`visibility`' zunächst unsichtbar (`hidden`) gemacht. Alle nachfolgenden Animationen werden innerhalb des Tags `<text>` ausgezeichnet, d. h. sie beziehen sich auch auf den Text.

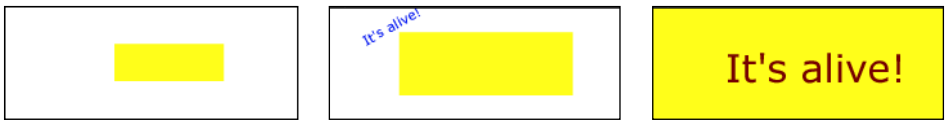
[13] Mit `attributeName="visibility"` wird das CSS-Property angesprochen. Der Text soll von der 3. Sekunde an sichtbar gemacht werden (`to="visible" begin="3s"`). Die Dauer hat hier keine weitere Bedeutung, zumal mit ‚`fill="freeze"`' die Sichtbarkeit „einfriert".

[14] Mit `<animateMotion>` leiten wir eine Bewegung ein. Der Text „It's alive!" bewegt sich von der 3. Sekunde an 6 Sekunden entlang eines Pfades (Attribut(!!!) ‚`path`') fort und bleibt danach stehen (`freeze`).

[15] Mit `<animateColor>` wird die Farbe des Textes gesteuert: von blau zu violett.

[16-17] Schliesslich wird der Text über `<animateTransform>` transformiert: zunächst rotiert [16] und danach skaliert [17].

Ergebnis:



nach 0 Sekunden **nach 3 Sekunden** **nach 9 Sekunden**

Abb. 2-45: Beispiel Animation

Den Animationsablauf kontrollieren

Es gibt noch eine ganze Reihe weiterer Möglichkeiten, Bewegungen und Eigenschaften von Graphiken zu kontrollieren. Ein recht wichtiges „Kontrollinstrument" ist dabei das Attribut ‚`calcMode`', mit dem wir die **Art des Animationsablaufs** angeben. Soll beispielsweise eine Bewegung ruckartig (nicht-interpoliert) ablaufen oder übergangslos (interpoliert)?

```
<animateColor attributeName="fill" attributeType="CSS"
  from="red" to="yellow" begin="3s" dur="6s" calcMode="discrete"/>
```

Werte für ‚calcMode':
(Standard: `linear`. Ausnahme: für `<animateMotion>`: `paced`)

`discrete` (für die nicht-interpolierte Zustandsveränderung) |
`linear` (für die interpolierte Zustandsveränderung) |
`paced` (für die schrittweise Veränderung von numerischen Zuständen, wie Position, Breite und Höhe eines Zeichenobjekts) |
`spline` (für die Veränderung von Werten einer Bézierkurve in Abhängigkeit von bestimmten Zeitschritten [s. u. ‚`values`', ‚`keyTimes`' und ‚`keySplines`'])

Animationen optimieren

Bei den oben beschriebenen Animationen ist ein Zeichenobjekt immer von einem Anfangs- zu einem Endzustand gelangt. Um auf diese Weise zu einem dritten Zustand zu kommen, z. B. von rot über grün zu blau, müssten wir eine weitere eigenständige Animation auszeichnen. Es gibt aber auch die Möglichkeit, mit Hilfe eines bestimmten Attributs eine **Liste von Zuständen** direkt anzugeben, in die sich ein Darsteller während der Animation begeben soll. Ausserdem können wir über ein anderes Attribut die **Zeiten für die jeweiligen Zustände in der Liste** festlegen. Die beiden Attribute heissen:

- ‚`values`' (für eine Liste von [durch Semikolon getrennten] Zuständen).
- ‚`keyTimes`' (für eine Liste von [durch Semikolon getrennten] Zeiten, mit denen wir die jeweiligen Werte von ‚`values`' zeitlich steuern können).

```
<animate attributeName="fill" values="green;blue;red" keyTimes="0;.5;1"
  calcMode="spline" dur="5s" repeatCount="indefinite"/>
```

Werte für ‚keyTimes':

Gleitkommazahl zwischen 0 und 1

Spezifiaktion: Wenn wir ‚`keyTimes`' verwenden, *muss* ‚`calcMode`' den Wert ‚`spline`' haben, sonst wird es vom Parser ignoriert!

Neben dem Element `<animateMotion>` (s. o.) gibt es auch die Möglichkeit, die Bewegung eines Zeichenobjektes entlang eines „Pfades" auf rein „mathematische" Weise zu lösen: mit Hilfe des Attributs ‚`keySplines`'. Damit beschreiben wir die Anfasser einer gedachten Bézierkurve. Dabei werden „Sets" aus vier Koordinatenwerten (für jeweils zwei Anfasser) x1 y1 x2 y2 als Parameter dem Attribut zugewiesen. Diese haben Werte zwischen 0 und 1, die einer Skala von 0 bis 100 % entsprechen. Mehrere solcher Sets werden durch Semikolon voneinander getrennt.

Die eigentlichen „Knoten" der Bézierkurve entsprechen den Werten des Attributs ‚`values`'.

Mit dem nachfolgenden Beispiel werden die x- und y-Koordinate eines Zeichenobjektes entlang einer Kurve verschoben:

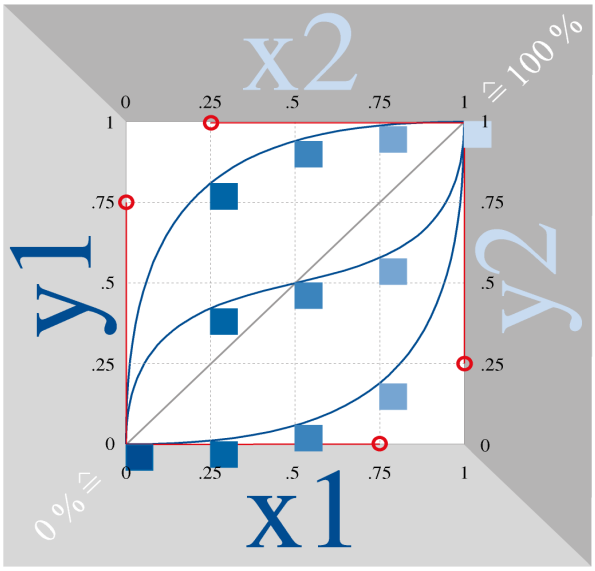
```
<animate attributeName="x" values="0;100" keySplines=".75 0 .75 1"
  calcMode="spline"/>
<animate attributeName="y" values="100;0" keySplines="0 0 1 1"
  calcMode="spline"/>
```

Werte für ‚keySplines': (Standard: `0 0 1 1`)

Koordinatenliste aus Gleitkommazahlen zwischen 0 und 1

Spezifiaktion: Wenn wir ‚`keySplines`' verwenden, *muss* ‚`calcMode`' den Wert ‚`spline`' haben, sonst wird es vom Parser ignoriert!

Die nachfolgende Graphik zeigt, wie sich ein Rechteck entlang von Bézierkurven fortbewegt, deren Anfasser über die ‚`keySplines`'-Parameter `x1 y1 x2 y2` beschrieben werden:



Anfasser-Positionen

Abb. 2-46: Funktionsweise von ‚keySplines'

Die komplette Vielfalt an Animationsmöglichkeiten finden wir in der SVG-Spezifikation im Abschnitt ‚*Animations*' und in der SMIL-Spezifikation (SMIL 2.0 (Working Draft) unter http://www.w3.org/TR/smil20).

2.2.3.2 Andere Medien einbinden

Adobe hat einen eigenen sogenannten XML-Namensraum (namespace) entwickelt, der von Adobe SVG-Viewer unterstützt wird: ‚`a`'. Mit diesem können wir MP3- und WAV-Dateien in ein SVG-Dokument integrieren. Dazu geben wir im Root-Tag `<svg>` im Attribut ‚`xmlns`' die URI für die dazugehörende SVG-Extension an:

```
<svg xmlns:a="http://www.adobe.com/svg10-extensions">
```

Nach der Namensraum-Deklaration können wir sämtliche in dieser Extension definierten Elemente und Attribute verwenden, indem wir jeweils das „a“ getrennt durch einen Doppelpunkt voran stellen:

```
<a:audio xlink:href="rev3.mp3" volume="10" begin="1.56s">
```

Seite 87

Siehe hierzu auch das Kapitel ‚*Namensräume (Namespaces)*‘

2.2.4 Linking und Scripting

2.2.4.1 Linking

Die wohl einfachste Art, einem Online-Dokument eine gewisse Interaktivität zu verleihen, ist der Link, also die Verbindung zwischen zwei Dokumenten (Hyperlink) oder bestimmten Bereichen (Anker) innerhalb eines Dokuments. Von HTML kennen wir das `<a href=″″>`, mit dem wir die Position (URL) einer HTML-Site angeben, um diese nach dem Mausklick (Event), im Browser aufzurufen.

In XML haben wir für diese Funktionalität drei Sprachen: XLink, XPath und XPointer. Dabei entspricht XLink im Wesentlichen dem `<a href=″″>`. Über XPath beschreiben wir den Weg innerhalb des DOM-Baums (siehe Kapitel ‚*Das Document Object Model (DOM)*‘) und mit XPointer definieren wir eine bestimmte Stelle oder einen Bereich (Adresse) in einem Dokument, auf die verwiesen werden soll.

Seite 82

In der SVG-Recommendation wurden zunächst nur XLink und XPointer berücksichtigt. Das heisst wiederum nicht, das diese auch so funktionieren, wie wir uns das vorstellen, zumal die Software, in dem Fall unser SVG-Viewer, nicht alle Standards und Möglichkeiten in vollem Umfang unterstützt. Nachfolgend werde ich die Syntaxen, die von Seiten der Recommendation möglich sind, aufführen.

Die Auszeichnung für einen Hyperlink sieht in SVG zunächst nicht viel anders aus als in HTML:

```
<a xlink:href=“beispiel.svg“><!--  Inhaltsdefinition --></a>
```

Der Unterschied zu HTML besteht nur darin, dass wir hier auf den XLink-Namensraum zurückgreifen, was wir wiederum durch Voranstellen von `,xlink:‘` zum Ausdruck bringen.

Ebenso dürfte uns auch der Verweis auf eine (ID-)Stelle im Dokument bekannt vorkommen:

```
<a xlink:href=“#id_stelle“><!--  Inhaltsdefinition --></a>
```

Spätestens beim nächsten Schritt stossen wir jedoch an die Grenzen von HTML: beim Verweis auf eine Stelle in einem anderen Dokument:

```
<a xlink:href=“beispiel.svg#id_stelle“><!--  Inhaltsdefinition --></a>
```

Schliesslich bietet uns SVG auch noch ein echtes Graphik-Feature, nämlich die Veränderung einiger Attribute, z. B. zur Steuerung des Anzeigebereichs mit ‚viewBox':

```
<a xlink:href="#svgView(viewBox(0 10 10 10))">
  <!--  Inhaltsdefinition --></a>
```

Anmerkung: Es gibt in SVG auch das Attribut ‚xlink:target', jedoch ist das nur für die Verwendung von in HTML oder XHTML eingebettete SVG-Graphiken vorgesehen.

Software-Hinweis:
Die letzten beiden Link-Features funktionieren leider auch mit der neuesten Version (2beta) des Adobe SVG-Viewers noch nicht.

2.2.4.2 Scripting

2.2.4.2.1 Das Document Object Model (DOM)

Seite 6

Im Kapitel *‚Die Recommendation des W3C'* habe ich darauf hingewiesen, dass die Empfehlung einige SVGDOM-Interfaces enthält, die wir in Programmiersprachen wie Java, JavaScript oder ECMAScript (darunter verstehen wir eine Art standardisiertes JavaScript) verwenden können. Das **D**ocument **O**bject **M**odel (DOM) ist grundsätzlich ein Modell zur Beschreibung von Objekt-Zusammenhängen, die in einer Baumstruktur, wie wir sie in XML generell vorliegen haben, bestehen. Dabei wird der Baum von oben nach unten durchlaufen, um zu einem gewünschten Element zu gelangen. „Oben", darunter verstehen wir im DOM-Sprachgebrauch zunächsteinmal das Dokument. Danach kommt das Root-Element, dann dessen erstes Child-Element usw. Auf das DOM müssen wir zurückgreifen, wenn wir in SVG Scripts implementieren. Dabei ist zu berücksichtigen, dass noch nicht alle DOM-Spezifika von den jeweiligen Viewern unterstützt werden.

2.2.4.2.2 Die script-gesteuerte Interaktion

Um eine script-gesteuerte Interaktion auszulösen, bedarf es zunächst zweier Komponenten:

- des Programms (Script), das die Aktion ausführt und
- des Events (Mausklick, Tastatureingabe etc.), der die Ausführung des Programms auslöst und über einen sogenannten Event-Handler erfasst wird.

In der nachfolgenden Tabelle sehen wir alle Event-Handler, die für SVG vorgesehenen sind:

Event-Handler	Beschreibung
onfocusin	Erfasst, dass ein Element hervorgehoben, z. B. ein Text markiert wird.
onfocusout	Erfasst, dass ein Element wieder in den Normalzustand gebracht, z. B. ein Text demarkiert wird.

Event-Handler	Beschreibung
onactivate	Erfasst, dass ein Element aktiviert wird, z. B. durch einen Mausklick oder eine Tastatureingabe.
onclick	Erfasst, dass auf ein Element ein Mausklick angewandt wird.
onmousedown	Erfasst, dass die Maustaste über einem Element gedrückt wird.
onmouseup	Erfasst, dass die Maustaste über einem Element losgelassen wird.
onmouseover	Erfasst, dass sich der Mauszeiger über einem Element befindet.
onmousemove	Erfasst, dass sich der Mauszeigers über einem Element bewegt.
onmouseout	Erfasst, dass der Mauszeiger ein Element verlässt.
onload	Erfasst, dass ein Dokument vollständig geladen (geöffnet) ist.

Tabelle 2-1: Event-Handler in SVG

Für die Integration von Scripts verwenden wir, wie in HTML auch, das Tag `<script>`. Darin müssen wir, anders als in HTML, die CDATA-Klammer ‚`<![CDATA [ ... ]]>`' setzen, um einen Bereich zu kennzeichnen, der nicht zur Auszeichnung gehört (vgl. CSS).

Seite 26

Ein Beispiel-Script aus dem SVG-Tutorial:

```
[1]  <script type="text/javascript">
[2]  <![CDATA[
[3]  // Funktion mit der ein Zielelement sichtbar gemacht wird
[4]  function show (evt,id_ziel)
[5]          {
[6]          var evtelem = evt.getTarget();
[7]          var doc = evtelem.getOwnerDocument();
[8]          var zielelem = doc.getElementById (id_ziel);
[9]          if (zielelem)
[10]                 {
[11]                 var style = zielelem.getStyle();
[12]                 if (style)
[13]                         style.setProperty ('visibility', 'visible');
[14]                 }
[15]         }
[16] ]]>
[17] </script>
```

Nachfolgend sehen wir eine Zeile aus dem Tutorial, in die die Event-Handler ‚`onmouseover`' und ‚`onmouseout`' eingefügt wurden:

```
[18] <tspan x="43.826" y="27" style="fill:#A80C0A;"
      onmouseover="show(evt,'verw_richt')"
      onmouseout="hide(evt,'verw_richt')">Verlaufsrichtung</tspan>
```

Erläuterungen:

[1] Mit dem Attribut ‚`type`' geben wir die verwendete Sprache an: ‚JavaScript'.

[4] Entscheidend, um in SVG eine Interaktion über einen Event auszulösen, ist das Event-Objekt, repräsentiert durch die Event-Variable ‚`evt`'. Es

wird über die Parameterliste ‚()' der entsprechenden Funktion, in dem Fall ‚show()', übergeben, um den eigentlichen Event zu festzustellen.

[6] In dieser Zeile wird über die Methode ‚getTarget()' das Element ermittelt, bei dem der Event (‚evt') ausgelöst wurde. Dieser wird der Variablen ‚evtelem' zugewiesen.

[7] Um den DOM-Baum abzulaufen, müssen wir zunächst an die Wurzel springen: das Dokument. Das erreichen wir mit der Methode ‚getOwnerDocument()'. In die Umgangssprache übersetzt heisst diese Zeile: Ermittle das Dokument des Event-Elements und weise dieses der Variablen ‚doc' zu.

[8] Nun müssen wir eine Möglichkeit finden, um im DOM-Baum das Element ausfindig zu machen, auf das sich der Event auswirken soll. Der einfachste Weg ist der über die ID eines Elements. Dazu gibt es die Methode ‚getElementByID()'. In die Umgangssprache übersetzt heisst diese Zeile: Ermittle in dem Dokument (der Wurzel) das Element mit der ID ‚id_ziel' und weise dieses der Variablen ‚zielelem' zu. Bei ‚id_ziel' handelt es sich um einen von mir definierten Platzhalter, der in der Parameterliste der Funktion steht. Dieser kann durch die ID eines beliebigen Elements im Dokument ersetzt werden, so dass sich der Event auf dieses Element auswirkt (es sichtbar macht).

[9] Hier wurde eine Bedingungsanweisung eingefügt, die klar macht: Nur wenn es das Element mit der ‚id_ziel' wirklich gibt, führe den Inhalt der IF-Anweisung aus.

[10] Gibt es dieses Element, wird mit der Methode ‚getStyle()' dessen Attribut ‚style' ermittelt und dieses der Variablen ‚style' zugeschrieben.

[11] Danach müssen wir noch eine Abfrage einfügen, denn: Nur wenn es das Attribut ‚style' gibt, kann die nachfolgende Zeile ausgeführt werden.

[12] Diese lautet: Setze das Property ‚visibility' auf den Wert ‚visible' (das Element wird sichtbar).

[18] ‚id_ziel' wurde hier durch 'verw_richt' ersetzt (die Hochkommatas müssen in der Parameterliste gesetzt werden, damit das Script verw_richt nicht als undefinierte Variable interpretiert und dadurch abgebrochen wird).

2.2.4.2.3 Die script-gesteuerte Animation

Der Unterschied zwischen einer Interaktion und einer Animation liegt im Wesentlichen nur darin, dass wir bei letzterem keinen Einfluss auf den Ablauf haben. Das ist auch der Grund, weshalb ich die *script-gesteuerte Animation* nicht, wie vielleicht zu erwarten, im Kapitel ‚*Animationen*' aufführe. Dennoch, die gleichen Animationen, die wir über die Auszeichnung steuern können, erzielen wir auch mit Hilfe des DOMs. Damit können wir alle Attribute oder Properties ansprechen und um diese mit Hilfe eines Scripts über mathematische Funktionen steuern.

Auch hier benötigen wir die beiden Komponenten

- Programm (Script) und
- Event.

Letzterer entspricht bei einer Animation häufig dem Laden des Dokuments (‚onload').

Ein Beispiel aus der Recommendation:

(mit JavaScript umgesetzt, damit das Script auch im Adobe SVG-Viewer läuft)

```
[1]  <?xml version="1.0" standalone="no"?>
[2]  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
[3]  "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
[4]  <svg width="12cm" height="2cm" viewBox="0 0 1200 200"
      onload="StartAnimation(evt)" >
[5]    <script type="text/javascript">
[6]    <![CDATA[
[7]      var timevalue = 0;
[8]      var timer_increment = 50;
[9]      var max_time = 5000;
[10]     var text_element;
[11]     function StartAnimation(evt)
[12]      {
[13]       text_element =
         evt.getTarget().getOwnerDocument().getElementById("TextElement");
[14]       ShowAndGrowElement();
[15]      }
[16]     function ShowAndGrowElement()
[17]      {
[18]       timevalue = timevalue + timer_increment;
[19]       if (timevalue > max_time)
[20]        {
[21]         return;
[22]        }
[23]       scalefactor = (timevalue * 20.) / max_time;
[24]       text_element.setAttribute('transform',
           "scale(" + scalefactor + ")");
[25]       opacityfactor = timevalue / max_time;
[26]       text_element.setAttribute('style', "opacity:" + opacityfactor);
[27]       setTimeout('ShowAndGrowElement()', timer_increment)
[28]      }
[29]   ]]>
[30]   </script>
[31]   <g transform="translate(50,150)" style="fill:red; font-size:7">
[32]     <text id="TextElement">SVG</text>
[33]   </g>
[34] </svg>
```

Erläuterungen:

[4] Über den Event-Handler ‚onload' wird das Script gestartet, sobald das Dokument vollständig geladen ist.

[7-10] Im Script werden zunächst einige globale Variablen deklariert. Gleichzeitig erhalten die „Timer"-Variablen einen festen Wert (Initialisierung).

[11-14] Hier beginnt die Funktion, mit der wir den eigentlichen Event erfassen

und die wir später über ‚onload' aufrufen. Im eigentlichen Funktionskörper wird zuerst das Element mit der ID „TextElement" ermittelt und der globalen Variablen ‚text_element' zugewiesen. Danach wird die zweite Funktion, mit der wir die eigentliche Animation steuern, initialisiert (ShowAndGrowElement).

[16] Hier leiten wir die zweite Funktion ein. Sie berechnet die Skalierung und Opazität des Elements (in dem Fall ein Text), auf das wir die Animation anwenden wollen.

[18] Um den Zeitablauf zu steuern, benötigen wir eine Zählvariable. Dazu verwenden wir die globale Variable ‚timevalue', die wir bereits mit 0 initialisiert haben. Die einzelnen Zeitabschnitte (time_increment) betragen 50 Millisekunden, d. h., die Zählvariable wird um jeweils 50 Millisekunden weitergezählt.

[19] Durch die if-Anweisung wird solange hochgezählt, bis 5000 Millisekunden (max_time) vorbei sind.

[23-26] Dieser Script-Abschnitt bewirkt die eigentlichen Veränderungen des Textes.

[23] Dabei wird zunächst ein Skalierungsfaktor definiert, der abhängig ist von der Zählvariablen ‚timevalue'.

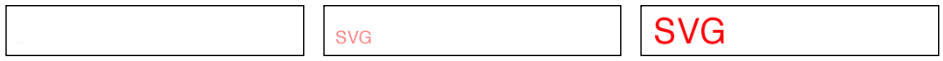
[24] Mit ‚setAttribute()' weisen wir dem Text das Attribut ‚transform' und dessen Befehl ‚scale' zu. Diesem geben wir die Variable scalefactor als Wert.

[25] Nach dem gleichen Prinzip wird ein Opazitätsfaktor festgelegt und ...

[26] ... dem Text zugewiesen.

[27] Mit dieser Zeile ruft sich die Funktion ShowAndGrowElement() alle 50 Millisekunden selbst auf und übergibt dabei den jeweils neuen Wert der Veränderung [23-26].

Ergebnis:



Start **Halbzeit** **Ergebnis**

Abb. 2-47: Beispiel script-gesteurte Animation

Anmerkung: Zahlreiche Beispiel-Scripts finden wir auf der Adobe-Websites (unter http://www.adobe.com/svg/demos) und auf der KevLinDev-Site (unter http://www.kevlindev.com/basics/index.htm).

Software-Hinweis:
Der Adobe SVG-Viewer unterstützt die standardisierte Sprache ECMAScript nicht (ob Java unterstützt wird, habe ich nicht ausgetestet, es gibt jedoch auch von Adobe weder Beispiele noch Anmerkungen dazu).

2.2.5 SVG und andere Standards

2.2.5.1 Andere Sprachkonzepte in SVG integrieren

2.2.5.1.1 Namensräume (namespaces)

Da SVG ein XML-Standard ist, also erweiterbar sein soll, stellt sich für uns die Frage, in wie weit wir diese Sprache zu anderen Sprachkonzepten ergänzen können.

Seite 80

Im Kapitel *‚Andere Medien einbinden‘* habe ich bereits darauf hingewiesen, dass in XML sogenannte Namensraum-Deklarationen gemacht werden können, um Elemente oder Attribute einer anderen XML-Sprache in einem XML-Dokument zu verwenden. Von Seiten der SVG-Recommendation ist dabei in erster Linie die Unterstützung von *Attributen* externer Sprachen vorgesehen.

Ein Namensraum wird grundsätzlich über das Attribut ‚`xmlns`‘ (für „XML-Namespace“) in ein Dokument eingebunden.

Ein Beispiel aus der Recommendation:

```
[1]  <?xml version="1.0" standalone="yes"?>
[2]  <svg width="4in" height="3in" xmlns="http://www.w3.org/2000/svg">
[3]   <defs>
[4]    <myapp:piechart xmlns:myapp="http://example.org/myapp"
         title="Sales by Region">
[5]      <myapp:pieslice label="Northern Region" value="1.23"/>
[6]      <myapp:pieslice label="Eastern Region" value="2.53"/>
[7]      <myapp:pieslice label="Southern Region" value="3.89"/>
[8]      <myapp:pieslice label="Western Region" value="2.04"/>
[9]      <!-- Other private data goes here -->
[10]   </myapp:piechart>
[11]  </defs>
[12]  <desc>This chart includes private data in another namespace
[13]  </desc>
[14]  <!-- In here would be the actual SVG graphics elements which
        draw the pie chart -->
[15] </svg>
```

Erläuterungen:

In diesem Beispiel werden zwei Namensräume deklariert:

- [2] der Namensraum von SVG selbst und ...
- [4] der Namenraum für die zu integrierende Sprache „myapp“. (In diesem Beispiel sollen die Sektoren-Werten eines Kuchendiagramms der SVG-Graphik zugewiesen werden.)

2.2.5.1.2 Das Tag <foreignObject>

Seite 19

(Wir haben bereits zu Anfang im Kapitel *‚System- und Ressourcen-Einstellungen für die Wiedergabe von Graphiken ermitteln‘* ein Beispiel betrachtet, das dieses Tag verwendet.)

Eine weitere Möglichkeit, externe Sprachen in ein SVG-Dokument zu integrieren, ist die Verwendung des Tags `<foreignObject>`.

Der Unterschied zu Namensraum-Deklarationen besteht zunächst darin, dass wir mit Hilfe von `<foreignObject>` ganze Dokument-Fragmente einer anderen Sprache in ein SVG-Dokument einbinden können. So hätten wir damit beispielsweise die Möglichkeit, XHTML-Tags auch in SVG zu nutzen, was toll wäre, denn so könnten wir direkt in SVG-Dokumente Tabellen oder Formulare einbinden. Doch leider sind solche Ideen noch Theorie, da wir immer noch auf die alles-könnende Software warten müssen. Einen weitereren Unterschied finden wir in der Attributsliste des `<foreignObject>`: Hierüber können wir nämlich eine Reihe von Angaben machen, die die spätere Ausgabe des externen Dokument-Fragments beeinflussen. Dazu gehören

- die Festlegung der vorausgesetzten System- bzw. Ressourcen-Anforderungen, die zum Rendern der externen Elemente notwendig sind,
- die *Transformation* des `<foreignObject>`,
- die Lage von `<foreignObject>` innerhalb des SVG-Fensters,
- die Ausmasse von `<foreignObject>`.

Seite 68

Wir erhalten folgende Syntax für `<foreignObject>`:

```
<foreignObject
  requiredFeatures="[Vorausgesetzte SVG- bzw. DOM-Features]"
  requiredExtensions="[Vorausgesetzte Extensions]"
  systemLanguage="[Vorausgesetzte Systemsprache]"
  transform="[Transformation von <foreignObject>]"
  x="[x-Koordinate von <foreignObject>]"
  y="[y-Koordinate von <foreignObject>]"
  width="[Anzeigebreite von <foreignObject>]"
  height="[Anzeigehöhe von <foreignObject>]">
   <!-- Dokument-Fragment einer anderen XML-Sprache -->
</foreignObject>
```

Werte für ‚requiredFeatures':

Feature-Strings wie
`"org.w3c.svg"` (für alle Sprachen-,Graphik- und Dynamik-Features der SVG-Spezifikation einschliesslich der DOM-Interfaces),
`"org.w3c.svg.animation"` (für alle SVG-Animationen).

Werte für ‚requiredExtensions':

URI von extern abgelegten Sprach-Extensions (vgl. hierzu Kapitel ‚Andere Medien einbinden' oder ‚System-, Ressourcen- und Browsereinstellungen für die Wiedergabe von SVG-Graphiken ermitteln')

Werte für ‚systemLanguage':

`en`, `fr`, `de`, `da`, `el`, `it` und andere Sprachkürzel

Werte für ‚transform':

Transformationsparameter (siehe Kapitel ‚Transformationen')

Werte für ‚x', ‚y', ‚width' und ‚height':
Koordinaten- und Massangaben in jeder beliebigen Einheit

DTD: ‚width' und ‚height' sind Pflicht.

Die Attribute ‚requiredFeatures', ‚requiredExtensions' und ‚systemLanguage' liefern die evaluierten Werte ‚true' oder ‚false' zurück. Alle drei arbeiten in erster Linie mit dem Tag <switch> zusammen, das je nach Voraussetzung den Einsatz einer Alternativ-Graphik steuert.

Seite 19

2.2.5.2 SVG-Graphiken in andere Sprachkonzepte integrieren

2.2.5.2.1 SVG-Graphiken in HTML einbinden

Graphiken oder Bilder benötigen wir am häufigsten im Rahmen eines übergeordneten Dokuments, wie HTML bzw. XHTML. Gerade für SVG-Graphiken hat das auch in sofern Vorteile, dass wir nur in HTML Frames (Rahmen) einsetzen können, in die wir SVG-Dateien direkt einbinden können. Anders ausgedrückt: Wir rufen eine SVG-Datei innerhalb eines Framesets auf, so dass sie im entsprechenden Rahmen angezeigt wird.

Eine weitere Möglichkeit ist die vom Adobe SVG-Viwer unterstützte Verwendung der HTML-Tags <object> oder <embed>.

```
<embed src="[URL der Graphik]" width="[Breite der Graphik]"
  height="[Höhe der Graphik]"/>
```

Ein HTML-Beispiel:

```
[1]  <html>
[2]   <head>
[3]    <title>HTML-Beispiel mit eingebetteter SVG-Graphik</title>
[4]   </head>
[5]   <body bgcolor="#ffffff">
[6]    <p>Unter diesem Text befindet sich eine SVG-Graphik</p>
[7]    <embed src="rect.svg" width="300" height="100">
[8]   </body>
[9]  </html>
```

Ergebnis:

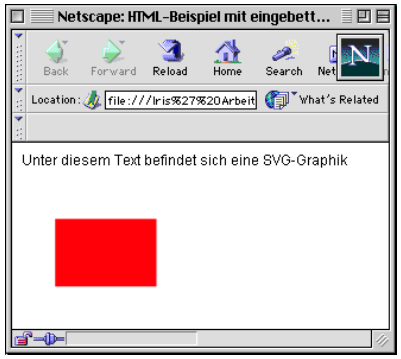


Abb. 2-48: Beispiel einer in HTML eingebetteten SVG-Graphik

Software-Hinweis:
Momentan unterstützt das Website-Gestaltungstool Adobe GoLive diese Art des Einbettens. Laut Adobe soll allerdings eine zukünftige Version des Programms auch die Möglichkeit bieten, SVG-Code direkt zu bearbeiten.

2.2.5.2.2 SVG-Graphiken über Formatting Objects (fo) in externe Dokumente transformieren

Mit Formatting Objects wurde ein Katalog von Gestaltungsmitteln geschaffen, der die Möglichkeiten von CSS bei weitem übertrifft. Dabei handelt es sich zunächst um einen Namensraum (`fo:`) der XML-Transformierungssprache XSL. D. h., wir müssen Formatting Objects noch in andere Dokumente hineingenerieren. „Noch" deswegen, weil es sich dabei um eine „Übergangssituation" handelt. Formatting Objects sollen eigentlich, wie jede andere XML-Sprache auch, als Namensraum innerhalb aller XML-Sprachen verwendet werden können. Hierzu fehlt nur wieder einmal die entsprechende Software.

Mit Hilfe des Formatierungsobjekts ‚`fo:instream-foreign-object`' können wir auch SVG-Graphiken als `foreignObject` in andere Dokumente hineingenerieren. Im zugehörig sind einige Properties, mit denen wir u. a.

- den Anzeigebereich,
- die Ausrichtung,
- die relative Position und
- die Skalierung

der Graphik festlegen können.

3 Das SVG-(Online-)Tutorial

Die im Kapitel *‚SVG-Graphiken erstellen‘* beschriebenen „Erkenntnisse“ sollten in ein Online-Tutorial verpackt werden, das sich besonders dadurch auszeichnet, dass es selbst so gut wie ausschliesslich mit SVG erstellt wurde (siehe beigelegte CD-ROM).

3.1 Konzept

3.1.1 Zielgruppe

Die Zielgruppen, die ich mit dem Online-Tutorial ansprechen möchte sind zunächst

- Graphiker,
- Kartographen,
- Technische Zeichner und Redakteure,
- Informatiker sowie
- Webentwickler allgemein.

Dabei sollten HTML-Kenntnisse vorhanden sein. XML-Kenntnisse sind von Vorteil, aber nicht notwendig, da es eine kleine Einführung zu der Metasprache gibt (die zumindest das Lesen einer DTD ermöglichen sollten).

3.1.2 Funktionalität des Tutorials

Eine grundsätzliche Idee lautete, mit diesem Tutorial nicht nur Basis-Wissen über die Sprache SVG zu vermitteln, sondern darüberhinaus auch eine Art laien-verständliche Referenz zu schaffen. Anders ausgedrückt, das Tutorial soll dem Nutzer neben dem klassischen Rundgang auch die Möglichkeit bieten, gezielt etwas zu finden.

Um diesen Zweck zu erfüllen, habe ich ins Layout einen Navigationsbereich aufgenommen. Dieser enthält:

- den variabelen Teil mit der Überschrift der betreffenden Lektion (z. B. „Verläufe“) einschliesslich deren Unterteilung in Form von Fragestellungen. (Mit den Fragestellungen wollte ich der typischen Tutoriumsdidaktik gerecht werden, die den Anwender direkter ansprechen soll.) Es entstanden zwei Kategorien:
  - ➔ „Kurz gefragt“ mit Fragen, die auf den eigentlichen Rundgang bezogen sind (z. B. „Wie rotieren wir ein Zeichenobjekt?“) und
  - ➔ „Hinterfragt“ mit den Fragen, die über das im Rundgang vermittelte Wissen hinausgehen (z. B. „Was sagt die DTD zu diesem Thema?“)
- den festen Teil mit den lektionsübergreifenden Punkten:
  - ➔ „Zu Abschnitt ...“, über den man menüartig in die einzelnen Abschnitte bzw. Lektionen gelangt,

- „DTD“ (hiermit haben echte „Tiefgänger“ die komplette DTD auf einen Blick) und
- „Index“ mit verlinkten Stichworten.

3.1.3 Aufbau des Tutorials

An der Stelle möchte ich die Ideen, die dem Aufbau des Tutorials zugrunde liegen beschreiben, auch wenn diese im Rahmen der Diplomarbeit nicht mehr komplett umgesetzt werden konnten.

Website (HTML):

Grundgedanke: Die Website soll eine Zugangsseite darstellen, die auch nicht-endgültige Fakten enthält.

Dazu gehören:

- der Überblick zu SVG:
  - Was ist SVG?
  - Angabe zum Stand der Empfehlung (Candidate Recommendation),
  - Werbung für Tutorial,
- der Hinweis auf den empfohlenen Viewer,
- der Link auf das Tutorial.

Diese Seite entspricht der Themen-Seite der Projekt-Website.

Tutorial (SVG/HTML)

Grundgedanke: Hier sollten die Inhalte wenn möglich dauerhaft sein.

Dabei war es mir besonders wichtig, einen relativ modularen Aufbau zu schaffen, der sich durch fehlende Teile ergänzen lässt (gerade, weil die Zeit sehr knapp bemessen war). Ich legte drei Stufen fest:

- die sechs Abschnitte *‚Überblick‘*, *‚Grundlagen‘*, *‚Graphik‘*, *‚Multimedia‘*, *‚Interaktivität‘* und *‚Erweiterbarkeit‘*, die das Grundgerüst bilden, an dem sich der Benutzer schon einmal grob orientieren kann,
- die eigentlichen Lektionen innerhalb der Abschnitte und
- die Fragestellungen innerhalb der Lektionen.

Gerade die letzte Strukturstufe lässt sich ohne grösseren Aufwand durch neue Seiten oder Inhalte ergänzen, da sie nur innerhalb der Lektionen selbst auftaucht.

Die Reihenfolge der Lektionen legte ich unter pragmatischen Gesichtspunkten fest. Dabei war mir die Logik genauso wichtig wie die Anwendung (was auch kein Wiederspruch ist). Also versuchte ich den Aufbau der logisch strukturierten Recommendation mit der „Denkweise“ des Anwenders zu verknüpfen. So hielt ich mich zwar überwiegend an die Grobstruktur der Spezifikation, setzte aber Lektionen, bei denen es mir sinnvoll erschien an eine ganz andere Stelle: Beispielsweise finden wir in der Recommendation das Thema *Transformatio-*

nen unter dem ziemlich zu Anfang befindlichen Abschnitt *‚Coordinate Systems, Transformations and Units‘*, was logisch ist, da in SVG immer das gesamte Koordinatensystem transformiert wird. In der Regel können wir aber davon ausgehen, dass der Benutzer nicht das Koordinatensystem, sondern eine Zeichnung umwandeln will, und die muss er erst einmal erstellen können. Also setzte ich diese Lektion ans Ende des Abschnitts *‚Graphik‘*.

Insgesamt erhält das Tutorial folgenden Aufbau (in Klammern finden wir die dazugehörigen Verzeichnis- bzw. Datei-Bezeichnungen):

- Startanimation

die sechs Abschnitte, die sich wiederum in Lektionen verzweigen:

- Überblick (ab_ueber):
  - → Was ist SVG? (Redundanz zur Zugangsseite der Vollständigkeit wegen [lekt_ueber/lekt_ueber1.svg])
  - → Welche Editoren gibt es? (Kein Hinweis mehr auf Viewer, da der ja Voraussetzung ist [lekt_ueber/lekt_ueber2.svg])
  - → Einführung in das Tutorial (in Form einer Animation [lekt_ueber/lekt_ueber3.svg])
  - → Inhalt (lekt_ueber/lekt_ueber_inh.svg)
- Grundlagen (ab_grund):
  - → XML (kurze Einführung [lekt_xml/lekt_xml...])
  - → Kommentare (lekt_komment/lekt_komment1.svg)
  - → SVG (lekt_svg/lekt_svg...)
  - → Koordinatensysteme (und Einheiten [lekt_koord/lekt_koord...])
  - → Gesteuerte Ansichten (lekt_ansicht/lekt_ansicht...)
- Graphik (ab_graphik):
  - → Formatierungen (lekt_format/lekt_format...)
  - → Farben (lekt_farbe/lekt_farbe...)
  - → Zeichnungen (Füllungen und Linien [lekt_zeich/lekt_zeich...])
  - → Vektoren (lekt_vektor/lekt_vektor...)
  - → Texte (lekt_text/lekt_text...)
  - → Verläufe (lekt_effekte/lekt_verlauf/lekt_verlauf... [Effekt*])
  - → Füllmuster (lekt_effekte/lekt_muster/lekt_muster... [Effekt*])
  - → Maskierung (lekt_effekte/lekt_mask/lekt_mask... [Effekt*])
  - → Filter (lekt_effekte/lekt_filter/lekt_filter... [Effekt*])

* Die Effekte enthalten alle einen Verweis auf die Lektion Koordinatensysteme (der Effekte). Hier soll auch der Schema-F-Charakter in Erscheinung treten.

  - → Transformationen (lekt_transform...)
- Multimedia (ab_multi):
  - → Animationen (lekt_anim/lekt_anim...)

- ➔ Einbindung von Medien (lekt_media/lekt_media...)
- Interaktivität (ab_inter):
  - ➔ Verlinkung (lekt_link/lekt_link...)
  - ➔ Scripts (lekt_script/lekt_script...)
- Erweiterbarkeit (ab_erw):
  - ➔ Einbetten in HTML (lekt_svghtml/lekt_svghtml...)
  - ➔ Integrieren anderer XML-Sprachen (lekt_integ/lekt_integ...)

Übrigens: Für die Ausarbeitung vertauschte ich manche Teile, da das Hintergrundwissen, auf das wir in einem Online-Tutorial über zusätzliche Links hinweisen können, in einer Print-Version an den Anfang gesetzt werden muss.

3.1.4 Gewählte Sprache für's Tutorial

Die Wahl der richtigen Ausdrucksform in einem Online-Tutorial ist in sofern von Bedeutung, dass die zu vermittelnde Materie mit wenigen Worten erschöpfend und möglichst leicht verständlich erklärt sein muss. Auf der anderen Seite soll der Anwender, sprich der Lernende, „bei der Stange gehalten werden". D. h., eine trockene Formulierung, in die man unter den genannten Voraussetzungen schnell hineingerät, führt auch schnell zur Ermüdung des Lesers.

Ich versuchte also einen Stil zu finden, der ebenso kreativ wie inhaltlich prägnant ist. Ausserdem entschied ich mich für die Verwendung der „wir"-Form, damit sich der/die LeserIn(nen) angesprochen fühlt/fühlen („Wir" die Angehörigen der „SVG-Gemeinde" – damit soll gleichzeitig für SVG geworben werden).

Im Detail stand ich u. a. vor der Frage der konsequenten Begriffsverwendung, z. B.: Welcher der beiden Begriffe „Element" und „Tag" eignet sich besser? Der grösseren Verbreitung wegen und soweit es inhaltlich möglich war, sprach ich von „Tags" und setzte demzufolge auch die Elementnamen zwischen „<" und „>".

3.1.5 Äussere Gestaltung des Tutorials

Die Graphik gilt sicherlich im Hinblick auf die klassischen Tätigkeitsgebiete eines technischen Redakteurs zunächst als zweitrangig. Nicht so bei diesem Thema. Es ist ein Thema, das gewisse Assoziationen wecken soll. Es geht um *das* Thema „Graphik" (mit dem Teilgebiet SVG). Aber das ist nicht allein der „springende" Punkt. Der Anwender soll sich schliesslich im Tutorial wohlfühlen. Er soll weder vor einem langweiligen, weiss-flimmernden Bildschirmhintergrund sitzen, noch darf er durch (womöglich animierte) „Ablenkungsmanöver" vom Wesentlichen abgebracht und damit überfordert werden.

3.1.5.1 Das Layout

Beim Layouten, also dem Anordnen der graphischen Elemente „Zeichnung", „Text" und „Bild" auf einer Zeichengrundlage, ging es mir weniger darum, ein überaus professionelles, mit allen DTP-Finessen ausgestattetes Bild zu entwerfen, als mehr darum, die bereits erwähnte Assoziation zum Thema hervorzurufen. Dabei arbeitete ich mich in den beiden Stufen Makro- und Mikro-Ebene vor.

Die Makro-Ebene

In der Stufe „Makro" stellte ich mir zwei Fragen:

- Welche Ausmasse sollte das Gesamtbild des Tutorials haben?
- Wie sollen die wichtigsten Layout-Bestandteile (Navigations- und Inhaltsbereich) organisiert sein?

Für die Beantwortung der ersten Frage griff ich auf die allgemeingültigen Regeln der Website-Gestaltung zurück und legte zunächst die Fensterausmasse auf 800 x 600 Pixel fest.

Zur zweiten Frage können mehrere Antworten gegeben werden:

- ➔ die Navigation links und die Inhalte des Tutorials rechts,
- ➔ die Navigation oben und die Inhalte unten,
- ➔ die Navigation für die festen Menüpunkte oben und die für die variablen links oder
- ➔ die sicherlich ungewohnte Variante: Navigation unten und Inhalte oben.

Ich entschied mich für die klassische Version, die erste Antwort. Sie erschien mir am schnellsten erfassbar, schon wegen der Leserichtung „von links nach rechts" (vergleichbar einer Kartenlegende, die optimaler Weise auch auf der linken Seite in der Anordnung ‚Signatur – Erklärung(– Karte)' angebracht sein sollte).

Die Mikro-Ebene

Wie soll das Erscheinungsbild aussehen?

Auffallend bei fast allen Websites ist das rechteckige Gesamtbild, da sich die Form des Bildschirms und der Fenster leider nicht ändern lässt. Ausserdem sind die beiden einzigen Gestaltungsmittel in HTML – neben dem Text – die ‚Tabelle' und die ‚waagrechte Linie' (`<HR>`). So kommen wir hier in punkto „Formgebung" um den Einsatz von Bildern nicht umhin. Diese sind speicherintensiv und machen folglich eine Site am Browser schwerfällig.

SVG ist dazu gemacht, beliebige Formen und Farben auf den Bildschirm bzw. in den Browser zu bringen (obwohl derartige Dateien je nach Grösse auch schwerfällig werden können). Für mich war klar, dass der Betrachter ein „andersgeformtes" Bild als Gesamteindruck erhalten sollte, um wieder auf unsere eingangs erwähnte Assoziation zu kommen. Der Kreis als Blickfang im Hintergrund des Tutorial-Inhalts erschien mir als geeignet (hätte genauso gut ein

Dreieck sein können).

3.1.5.2 Die Farbwahl

Für die Wahl der Farben legte ich diverse graphische Richtlinien zugrunde. Zu diesen zählen u. a.,

- dass warme, nicht-komplementäre Farben das menschliche Auge weniger überfordern als kalte oder flimmernde (komplementäre),
- dass schwarze Schriften auf weissem oder hellem Hintergrund am besten lesbar sind und
- dass die Farbe nach den Gesetzen der graphischen Prägnanz noch vor der Form als bester optischer Selektor gilt.

Der Hintergrund bekam einen orange/weinroten Verlauf.

Den Hintergrund des inhaltlichen Teils, dem soeben erwähnten Kreis, machte ich hellgrau.

Das Beispielsfenster, das wie ein Layer in DHTML eingeblendet werden kann, erhielt einen grauen Hintergrund mit blauem Rand (der soll an ein „Windows"-Fenster erinnern).

3.1.5.3 Die Schrift

Hier unterschied ich vier Kategorien:

- Fliesstext (Verdana, schwarz),
- Text, der eine Interaktion auslöst (Verdana, weinrot),
- Schlüsselbezeichnungen und Quellcode (Courier, schwarz oder weinrot),
- Text, der auf ein Beispiel hinweist (Verdana, fett, blau).

3.2 Dynamische Umsetzung

Bei der Umsetzung wollte ich mich generell an die Machart einer klassischen Online-Hilfe, wie sie beispielsweise mit RoboHelp verwirklichbar wäre, halten. So wollte ich eigentlich für *Beispiele*, *DTD zum Thema*, *DTD* und *Index* Zweitfenster erstellen, die parallel zur thematischen Erklärung stehen können. Jedoch scheiterte ich bei der nötigen Script-Erstellung an der Software (zumal der Internet Explorer die JavaScript-Funktion `window.open()` problemlos unterstützte, während Netscape sich dabei aufhängte) und demzufolge auch an meinem Programmier-Latein. Ich musste also Alternativen finden, was in SVG nicht besonders problematisch ist. Wie bereits oben erwähnt, erstellte ich wenigstens für die *Beispiele*, genauso wie für die *DTD zum Thema*, Layer, die zunächst unsichtbar gemacht (`visibility:hidden`) beim Mausklick auf den entsprechenden Hinweis-Text sichtbar werden (vgl. DHTML).

Auch die Online-Hilfe-typischen Popups, die beim Überfahren eines Elements mit dem Mauszeiger (`onmousover` bzw. `onmousout`) sichtbar bzw. unsichtbar werden, wollte ich für manche Inhalte nutzen. Zu diesen gehören in erster Linie

- Werte, die ein Attribut oder Property einnehmen kann (in dieser Ausarbeitung finden wir dafür die grauen Tabellen),
- Hintergrundinformationen und
- Tipps.

Da SVG ein Online-Standard ist und damit aus Einzelkomponenten besteht, die über Verweise miteinander verbunden werden können, wäre es schön gewesen, auf diese Weise Redundanzen zu vermeiden. XML bietet hierfür die im Kapitel *‚Linking‘* angesprochenen Sprachen XLink und XPointer. Da diese aber von der Software leider noch nicht in vollem Umfang unterstützt werden, blieb mir gerade in Bezug auf Popups nichts anderes übrig, als (auf althergekömmliche Weise) jedes Popup, das für ein Dokument benötigt wurde, vollständig darin einzubinden.

Seite 81

4 SVG im Einsatz

4.1 Graphik

Generell wurde SVG in erster Linie für den Einsatz im www konzipiert. Das Web soll endlich eine graphische Erneuerung erleben – vektorbasiert und standardisiert. So finden wir für den Bereich Graphik natürlich den Hauptbedarf befriedigt.

4.1.1 Allgemeine Graphik

Unter allgemeiner Graphik verstehe ich eigentlich alle graphischen Ausrichtungen, die nichts mit den später beschriebenen technischen Bereichen zu tun haben, also in erster Linie Werbe-Graphik und Graphik-Design. Wohl unerwarteter Weise stossen wir hier auf die wenigsten Anwendungen (Ausnahme: die von den Software-Herstellern herausgegebenen SVG-Beispiele).

Hier denke ich, dass sich das mit wachsender Anzahl an WYSIWYG-Tools noch ändern wird, zumal SVG uns fast uneingeschränkte graphische Möglichkeiten bietet. Auf der anderen Seite ist SVG aber auch ein sehr technisch ausgerichteter Standard, d. h., Graphikern bringt SVG im Vergleich zu momentan vorherrschenden Formaten nicht den erhofften Vorteil. Gerade für statische Graphiken sind zur Zeit Rasterformate wie GIF, JPEG oder PNG besser, da sie z. T. noch schnellere Ladezeiten aufweisen.

4.1.2 Kartographie und Geoinformationssysteme

Für die Kartographie ist SVG schon wegen der koordinatenabhängigen Positinierung von Vektoren und Texten prädestiniert. In der Tat sind in diesem Bereich auch schon zahlreiche Diplomarbeiten (im In- und Ausland) umgesetzt worden. Das wohl bekannteste Beispiel ist der Europa-Atlas von André Winter und Andreas Neumann (unter http://www.carto.net/papers/svg/index.html). Hierbei handelt es sich um ein kleines „Web“-GIS, mit dem ausgewählte Themenbereiche mit Hilfe von Diagrammen und Tabellen auf einer Europakarte angezeigt werden können.

Was in diesem Bereich sicherlich auch grossen Zuspruch findet, sind die Funktionalitäten, die uns der Adobe SVG-Viewer bietet. Eine davon, die Such-Funktion, erinnert fast schon an ein mit Datenbanken verknüpftes Abfragesystem. Und da wir in SVG die Möglichkeit haben, Texte zu verbergen (`opacity:0`), wäre es auch möglich, verschiedene beispielsweise internationale Schreibweisen in die Abfrage miteinzubinden, ohne dabei das Kartenbild zu überladen. Auch die Tatsache, dass wir beliebig Zoomen und mit der Verschiebe-Hand navigieren können, spricht für das Medium Karte.

Seite 15

4.1.3 Technische Zeichnungen

Auch hier haben wir einen der wirklich prädestinierten Bereiche, da gerade Online-Bedienungsanleitungen und -Lexikas immer gefragter sind. Die breite

Software-Unterstützung in Sachen Import von CAD-Daten und Export in SVG liefert auch die notwendige (Tool-)Grundlage. Ausserdem können wir, dank der guten Scripting-Möglichkeiten, mit SVG interaktive Zeichnungen anfertigen, bei denen wir beispielsweise per Mausklick ins Innere einer Maschine blicken können.

4.2 Multimediale Präsentationen

Hier wird wohl noch eine Weile lang Microsoft PowerPoint vorherrschen, was vielleicht auch seine Berechtigung hat. Dennoch: Wollen wir einem echten W3C-Standard den Vorzug geben, dann können wir sagen: SVG ist besser. Denn SVG bietet uns

- bessere graphische Möglichkeiten,
- bessere Animationsmöglichkeiten und
- bessere Interaktionsmöglichkeiten (vor allem mit standardisierten Script-Sprachen).

Darüberhinaus sind die Dateien wesentlich kleiner und so kommt auch die Prozessorleistung des Client-Rechners weniger zum Tragen. Allerdings sind die Ladezeiten momentan noch einiges höher.

Generell spricht für diesen Standard die Tool-Unabhängigkeit, die auf PowerPoint natürlich nicht zutrifft. Im gleichen Atemzug muss ich aber auch den einzigen echten „Noch"-Nachteil nennen: Es gibt *noch* keinen SVG-Viewer mit „Präsentationsmodus".

Unterstützung in Sachen Präsentationen mit SVG erhalten wir bereits heute von Software-Firmen, die spezielle Programme entwickelt haben, mit denen sich aus beliebigen XML-Dokumenten – z. T. „on the fly" – SVG-Präsentationen erstellen lassen (siehe hierzu Kapitel *‚Weitere Programme'* – Catwalk).

4.3 Dokumenation

Unter diese Rubrik würde ich das Tutorial einordnen, und da kann ich nur sagen: Mit SVG Fliesstexte zu erstellen, ist nicht empfehlenswert!

Vergleichen wir jedoch SVG mit PDF, so schneidet der Graphik-Standard immer noch besser ab, denn: Der einzige Nachteil in Bezug auf die Erstellung von Dokumentationen ist die separate Auszeichnung einzelner Textzeilen, und das ist genau genommen bei PDF auch nicht anders: Wollen wir PDF-Texte im Acrobat Exchange ändern, müssen wir ebenfalls zeilenweise „schieben" oder „ziehen". So ist es auch hier wohl nur eine Frage der Zeit, wann das richtige Tool mit „Schiebe-" und „Zieh-"Funktionalität auf den Markt kommt. Optimal wäre eine Art Macromedia Dreamweaver, der parallel alle im WYSIWYG-Editor vorgenommenen Änderungen direkt in den Quelltext überträgt.

5 Eigene Einschätzungen zu SVG

Ist SVG der Graphikstandard der Zukunft? Nun, diese Frage dürfen mal meine Nichten beantworten. Auf jeden Fall konnte ich im Rahmen dieser Arbeit feststellen, dass SVG gerade gegenüber vergleichbaren Formaten klare Vorteile bietet. Wobei die Gründe dafür weniger etwas mit Benutzerfreundlichkeit oder guter Tool-Unterstützung zu tun haben. Zunächst zählt das Prinzip. SVG ist ein XML-Standard und damit semantisch strukturiert. Alle Daten liegen „offen". Wir können auf jedes noch so kleine Detail zugreifen. Und ... zum x-ten Mal ..., wir benötigen kein bestimmtes Tool, um SVG-Graphiken zu erstellen oder weiterzuverarbeiten. Ganz anders als beim Haupt-Konkurrenten Macromedia Flash, der zwar viele graphische und multimediale Möglichkeiten bietet, uns aber in jeder Hinsicht abhängig macht (eigener Editor, eigene Script-Sprache usw.).

Einen besonderen Vorteil sehe ich in einem Punkt, den ich bisher gar nicht erwähnt habe: dem Workflow. Ich könnte mir gut vorstellen, dass, vergleichbar dem PDF, das aufgrund seiner hohen Qualität vermehrt in Workflow-Prozesse intgriert wird, auch SVG auf Dauer gesehen nicht mehr nur als einfacher Online-Standard Einsatz findet. Es wäre in jedem Fall besser. Denn wir können ein SVG-Dokument aufgrund der guten Strukturierung als „Datenbank" einzelner Graphik-Bausteine betrachten. Wir haben also nicht nur die Datenbank mit den Dokumenten, wir haben auch eine *in* den Dokumenten, was gezieltere Zugriffe erlaubt und damit den Workflow präziser und noch redundanzfreier ablaufen lässt. Was diesen Punkt betrifft, stimmen bereits jetzt schon zahlreiche Software-Ideen optimistisch.

Schliesslich stelle ich mir noch die Frage, wie der optimale SVG-Editor auszusehen hätte, zumal die momentan verfügbaren Produkte natürlich noch ein wenig „reifen" müssen. Optimal wäre hier sicherlich nicht das Einer-für-Alles-Prinzip, denn gute Software zeichnet sich in erster Linie durch stabile Funktionalität und hohe Beständigkeit aus. Das wiederum erfordert, dass Programme eine klar definierte Aufgabe haben, wie Textverarbeitung, Graphik oder Multimedia. SVG ist ein Standard, der, wie bereits zu Anfang erwähnt, alles unter einen „Hut" bringt. D. h., es wird schwer werden hier eine stabile Allround-Software zu konzipieren. Im voran gegangenen Kapitel habe ich die Idee geäussert, eine Art Dreamweaver müsste her. Seine WYSIWYG-Möglichkeiten und seine einfachen Eingaben in Bezug auf Formatierungen, Linking und Scripting sind in jedem Fall vorbildlich. Ausserdem haben wir hier die Möglichkeit, Templates zu erstellen, die uns zeitraubende Anpassungen der „Child"-Dokumente abnehmen. Toll wäre es hier noch, wenn sich Textrahmen erstellen liesen, in denen sich der Textfluss automatisch anpasst, so dass später die einzeln ausgezeichneten Zeilen direkt vom Editor in den Quelltext übertragen werden. Ideen gibt es also genug.

Aber noch ist nicht aller Tage Abend und die Entwicklungen gehen weiter.

Literatur

ADOBE: The SVG Zone. http://www.adobe.com/svg/. 18. März 2001.

CLOSS, Sissi 1999: FrameMaker+SGML: Schulungunterlagen. München.

CLOSS, Sissi 1999: SGML/XML: Schulungunterlagen. München.

HAROLD, Elliot Rusty 2000: Die XML-Bibel. Aus dem Amerikanischen von Reinhard Engel. Bonn.

MICHEL, Thomas 1999: XML kompakt: Eine praktische Einführung. München, Wien.

WORLD WIDE WEB CONSORTIUM (W3C): Extensible Stylesheet Language (XSL) Version 1.0: W3C Candidate Recommendation 21 November 2000. 13. März 2001.

WORLD WIDE WEB CONSORTIUM (W3C): Scalable Vector Graphics (SVG) 1.0 Specification: W3C Candidate Recommendation 02 November 2000. http://www.w3.org/TR/SVG/. 21. März 2001.

WORLD WIDE WEB CONSORTIUM (W3C): Synchronized Multimedia Integration Language (SMIL) 1.0 Specification: W3C Recommendation 15-June-1998. 18. März 2001.

WORLD WIDE WEB CONSORTIUM (W3C): Synchronized Multimedia Integration Language (SMIL 2.0) Specification: W3C Working Draft 01 March 2001. 18. März 2001.

Index

Symbols

@color-profile 30
@font-face 52

A

Adobe FrameMaker + SGML 5.5 3
Adobe Illustrator 7
Adobe SVG-Viewer 15
Adresse 81
Alphakanal 63
Alphamasken 61, 63
  spezifizieren 63
Animation 75
  kontrollieren 78
  optimieren 79
Animationsablauf 78
Anker 81
attributeName 76
attributeType 76
Attributsauszeichnung 26

B

Beschneidungspfad 62
Bézierkurven 42

C

calcMode 78
Candidate Recommendation 6
Cascading Stylesheets 26
Catwalk (SchemaSoft) 14
CDATA 27, 83
CEF 52
Clipping 61
color 30
Compact Embedded Font-Format 52
CorelDraw 10
Csiro SVG Toolkit 16
CSS 26
CSS2 23, 50

D

Directory-Server 2
Doctype 17, 28
Document Object Model 82
Document Type Definition 4
Dokumentbeschreibung 18
DOM 82
Druckertreiber 13
DTD 4, 17

E

ECMAScript 82
Effekte 55
Einheiten 23
Elemente
  a 81
  animate 76
  animateColor 76
  animateMotion 76
  animateTransform 76
  circle 36
  clipPath 61
  defs 18
  desc 18
  ellipse 37
  feComposite 67
  feGaussianBlur 67
  feMerge 67
  feMergeNode 67
  feOffset 67
  fePointLight 67
  feSpecularLighting 67
  filter 65
  font 53
  font-face 53
  foreignObject 87
  g 18
  glyph 53
  image 19, 22
  line 38
  linearGradient 56
  mask 61
  missing-glyph 53
  mPath 76
  path 40
  pattern 59
  polygon 39
  polyline 39
  radialGradient 56, 58
  rect 36
  set 76
  stop 57, 58
  svg 17
  switch 19
  symbol 19, 21
  textPath 51
  title 18
  tref 48
  tspan 46
  use 18
Ellipsen 37
elliptische Kurve 42
Entities 28
Event 82, 85
Event-Handler 82
eXtensible Stylesheet Language 4, 26, 28

F
Farben
festlegen 30
Farbmanagementsystem 30
Farbnamen 30
Farbprofile 30
fill 31
fill-opacity 31, 32
fill-rule 31
Filter 65
anwenden 66
definieren 65
spezifizieren 65
Flächen 31
fo 90
font-familiy 49
Fonts 52
externe 52
SVG- 53
font-size 49
font-stretch 50
font-style 49
font-variant 49
font-weight 49
Formatierungen 26
Formatting Objects 4, 90
Frames 89
Füllmuster 59
definieren 59
spezifizieren 60
Füllungsattribute 31

G
Geoinformationssysteme 98
Geschichte
von SVG 6
Glyphen 53
Graphik 98
Grundformen 35

H
height 17
HTML 89
Hyperlink 81

I
IBM SVG-View 15
ICC 30
Illustrator 7
Instanzen 4
Interaktion 82
Interaktivität 81
International Color Consortium 30

J
Jasc Webdraw 11
Java 82
JavaScript 82

K
Kapitälchen 49
Kartographie 98
keySplines 79
keyTimes 79
Knoten 31
Kommentare 18
Syntax 18
Koordinatensystem 22
Effekte 55
Kreise 36
Kurve
elliptische 42

L
Linien 31, 38
Linienattribute 32
Link 81

M
mask 63
Masken 61
definieren 62
Masking 61
Medien 80
Muster 59

N
Namensraum 80, 87
namespace 80, 87

O
objectBoundingBox 55

P
Parser 4
Pfade 40
Befehle 41
PGML 6
PICT to SVG (Chuck Houpt) 14
Polygone 39
Polylinien 39
PowerPoint 99
Precision Graphics Markup Language 6
preserveAspectRatio 23, 25
Property 27
pstoedit (GeoCities) 14

R
Rechtecke 35
Recommendation 6
requiredExtensions 19, 88
requiredFeatures 19, 88
Root-Tag 17

S
Scalable Vector Graphics 4, 5
Schriftart 49
Schriftbreite 50

Schriftgrösse 49
Schriftneigung 49
Schriftstärke 49
Script 82, 85
SMIL 75
Software 6
Sphinx open 12
sRGB-Farbraum 30
stop-color 57
stop-opacity 57
stroke 33
stroke-dasharray 34
stroke-dashoffset 34
stroke-linecap 33
stroke-linejoin 33
stroke-miterlimit 34
stroke-opacity 35
stroke-width 33
style 26, 28
Stylesheet-Prolog 27
Stylesheets 26, 28
SVG 5, 17
  Dateien 17
  Koordinatensystem 22
  Toolkit 16
  Working Group 6
SVG2PDF (Digital Applications Inc.) 14
SVG-Dateien 17
SVG-Dokument 17
SVG-Fenster 18
SVG-Fonts 53
SVGMaker 13
SVGObjects 14
SVG-Tutorial 91
SVG-View 15
SVG-Viewer 15
Synchronized Multimedia Integration Language 75
systemLanguage 19, 88

T

Technische Zeichnungen 98
Textattribute 49
Texte 45
  auszeichnen 45
  formatieren 49
  herausheben oder mehrzeilig anordnen 46
  referenzieren 47
Textpfade 51
transform 68
Transformationen 68
  Matrizen 73
  neigen 72
  rotieren 70
  skalieren 69
  verschieben 71

U

Uniform Ressource Identifier 5
Uniform Ressource Locator 5
URI 5
URL 5
userSpaceOnUse 55

V

Validierung 4
Vector Markup Language 6
Vektoren 35
Verläufe 55
  definieren 56
  linear 56
  radial 57
Verlaufsabschnitte 57, 58
viewBox 23
VML 6

W

WebDraw 11
WebObjects (Apple) 14
width 17
Working Group 6

X

XHTML 89
XLink 4, 81
xlink
  href 81
  target 82
XML 4
  + Directory-Server 2
  Instanzen 4
xmlns 80, 87
XML-Prolog 17
XPath 4, 81
XPointer 4, 81
XSL 4, 26, 28

Z

Zeichenstandard 9
  ISO-8859-1 (Latin-1) 9
  Latin-1 9
  UTF-16 9
  UTF-8 9

Anhang

Wer sagt was zu SVG?

Adobe

Artikel von Matthew Rothenberg „Adobes's InDesign debuts at Seybold", unter: http://www.zdnet.com/eweek/stories/general/0,11011,1014053,00.html 3/1999:

> *„‚We want to upgrade the Web'*
>
> *Adobe also touted its support for the proposed W3C (World Wide Web Consortium) Scalable Vector Graphics standard for resolution-independent Web images.*
>
> *‚We want to seriously upgrade the Web,' Warnock[, CEO of Adobe Systems Inc.,] said. Adobe is working „in lock step" with the W3C to refine the SVG standard; when the standard is ratified this summer, Warnock said, Adobe will immediately release plug-ins that provide SVG support for all its core graphics applications."*

Schön. Das war bereits 1999. Heute ist das PlugIn auf dem Markt.

http://www.adobe.com/svg/overview/overview.html 2001

> *„The Scalable Vector Graphics (SVG) format is set to revolutionize the way graphics look on the Web.*
>
> *...*
>
> *Adobe is supporting SVG across its product line, including Adobe® Illustrator® and Adobe GoLive™. Adobe and other industry leaders recognize in SVG the clear future of Web graphics."*

Ausserdem:

> *Foundation technology for the Internet of the future Adobe believes SVG will fundamentally transform nearly all aspects of the Web experience. Because it combines XML with professional-quality graphics, SVG will play a fundamental role in e-commerce and the future of the Internet.*
>
> *... Adobe applications will both import and generate SVG. Additionally, Adobe will be adding SVG-specific enhancements to its authoring tools.*

Adobe möchte also ihre gesamte Produktepalette mit SVG-Im- und Export-Möglichkeiten ausstatten. Interessant wären hier noch die Programme FrameMaker, InDesign und LiveMotion.

Macromedia

Artikel von Andreas Asanger „Flash geht in die fünfte Runde", Macwelt 11/2000:

> *... wir bemerken jedoch die offensichtliche Distanz [von Macromedia] zum kommenden SVG-Format ...*
>
> *Nach Macromedias Meinung ist SVG eher dafür geeignet, das PDF-Format zu ersetzen, als dem Flash-Format Konkurrenz zu machen. Mit dem Wissen im Rücken, dass über 90 Prozent der Internet-Nutzer Flash-Inhalte standardmässig in ihrem Browser sehen können, fühlt sich der Flash-Hersteller auf der sicheren Seite. Nachdem das SVG-Format schon in den Starlöchern steht, wird sich bald zeigen, ob Macromedia Recht behält.*

Dabei wäre neben Flash auch gerade Macromedias Freehand prädestiniert für diesen Standard. Und, um nach der bisherigen Qualität des Programms zu urteilen, wäre es auch um einiges besser als der Haupt-Konkurrent Adobe Illustrator.

Quark

Artikel von Matthew Rothenberg „Quark response to Adobe challenge at Seyold“, unter: http://www.zdnet.com/eweek/stories/general/0,11011,1014071,00.html 2000:

> *[Chief Technical Officer Tim] Gill [(Quark Inc.)] also pledged support for the World Wide Web Consortium's proposed SVG standard [in QuarkXPress 5.0].*

Corel

Derek Burney, CEO and President, Corel Corporation:

> *„Corel is extremely pleased to support the World Wide Web Consortium efforts to develop an industry-wide standard for Scalable Vector Graphics. As an active participant in the SVG working group, we are committed to tracking the SVG working drafts and to embracing SVG throughout our graphics applications.“*

Presse

Artikel von Holger Reibold „Graphiken mit XML“, Internet Professional 4/2000:

> *„Ob man es will oder nicht: Grafiken beherrschen das Web, nicht Inhalte. Und jeder, der seine Site mit grafischen Komponenten schmückt, kennt die Probleme: Versucht man, eine Grafik zu vergrößern oder auch zu verkleinern, sind Qualitätseinbußen unvermeidlich. Mit Scalable Vector Graphics, kurz SVG, ist damit Schluss. SVG verspricht Grafiken in Truecolor-Qualität, die sich mit Effekten versehen lassen, wie sie heute nur mit Flash möglich sind.“*

Sonstige (wichtige Persönlichkeiten)

Artikel von Roberta Holland „XML takes on graphics“, unter: http://

www.zdnet.com/eweek/stories/general/0,11011,2612024,00.html 8/2000:

> *„‚The state of graphics on the Web today is awful,' said Chris Lilley, graphics activity lead for the W3C, in Sophia-Antipolis, France, and chairman of the SVG working group. ‚When this takes off, it's going to change the way we look at things'.*
>
> *...*
>
> *Phillip Torrone, director of technology for Braincraft Technologies, in New York, said he believes Macromedia Inc.'s Flash has already changed the face of graphics. Torrone uses Macromedia's vector graphics tool in building e-commerce and other Web sites.*
>
> *SVG ‚is a great way to exchange vector data between systems, but I don't think it's going to be used for Web display,' said Torrone, who added that the current SVG plug-in is too cumbersome. ‚It's just not ready for prime time yet.'"*

Geht man jedoch davon aus, dass das SVG-PlugIn durch die Implementierung von SVG in die Standard-Browser ersetzt werden soll (dafür sind die Standards des W3C ja schliesslich da), dann wird sich die momentane Schwerfälligkeit als Gegenargument erübrigen.

Nützliche Links zu SVG

SVG-Portal

http://www.dmoz.org/Computers/Data_Formats/Graphics/Vector/SVG

Tutorials

http://www.adobe.com/svg/ (Die Adobe SVG Zone mit Tutorials Beispielen und Informationen zu SVG)

http://www.kevlindev.com/basics/index.htm (Tutorial für SVG, SMIL und DOM)

SVG-Reference

http://zvon.org/xxl/svgReference/Output/index.html

Beispiele für verschiedene Anwendungsbereiche

http://www.xml.com/2000/03/22/style/parts-catalog.htm (Technische Zeichnung)

http://www.usbyte.com/index_SVG.htm (SVG-Animationen und -Interaktionen, z. T. Technische Zeichnungen)

http://www.pinkjuice.com/SVG (SVG-Animationen)

http://www.carto.net/papers/svg/ (Kartographie)

Das Projekt „XML + Directory Server“

http://www.fbwi.fh-karlsruhe.de/lin02/xmldirectory