## Index SVG Tutorial


**1 Vorbereitungen**

1.1 Grundlagen SVG
1.2 Notwendige Voraussetzungen
1.3 SVG unter Windows
1.4 SVG unter Linux allgemein
1.5 SVG unter Debian Woody (mit Adobe Plugin)
1.6 SVG unter Debian Woody (mit Batik)
1.7 SVG unter SuSE ab 8.2 (mit Adobe Plugin)
1.8 SVG unter SuSE 8.x (mit Batik)

**2 Darstellung von SVG-Grafiken**

2.1 SVG-Grafiken in Webdokumenten
2.2 Rendern von SVG-Grafiken

**3 Dokumentstruktur I**

3.1 Das Dokument - Die Grafik
3.2 Grundgerüst einer SVG-Grafik
3.3 Verschachtelung von SVG-Elementen
3.4 Titel und Beschreibung einer SVG-Grafik
3.5 Das Attribut viewBox
3.6 Metadaten

**4 Grundformen**

4.1 Rechtecke - das rect-Element
4.2 Kreise - das circle-Element
4.3 Ellipsen - das ellipse-Element
4.4 Linien - das line-Element
4.5 Polylinien - das polyline-Element
4.6 Polygone - das polygon-Element

**5 Styling**

5.1 Globale Formatierung in der Grafik
5.2 Globale Formatierung durch externe Style-Datei

**6 Hyperlinks**

6.1 Besonderheiten des a-Elements

**7 Dokumentstruktur II**

7.1 Elemente gruppieren - das g-Element
7.2 Das defs-Element und das use-Element
7.3 Symbole - das symbol-Element
7.4 Einbinden von externen Grafiken - das image-Element
7.5 Allgemeine Attribute - stdAttrs, langSpaceAttrs, testAttrs
7.6 Auswahlmöglichkeiten - das switch-Element

**8 Text**

8.1 Das text-Element und seine Koordinaten
8.2 Text im Text - das tspan-Element
8.3 Texte referenzieren - das tref-Element
8.4 Rotation einzelner Zeichen
8.5 Schriftformatierung

8.6 Ausrichtung und Dekoration von Text
8.7 Zeichen- und Wortabstände
8.8 Mögliche Grundlinien und das tpath-Element
8.9 Darstellung von Zeichen

**9 Transformationen**

9.1 Verschieben - die Anweisung translate
9.2 Skalieren - die Anweisung scale
9.3 Rotieren - die Anweisung rotate
9.4 Verzerren - die Anweisungen skewX und skewY
9.5 Transformtion der Matrix - die Anweisung matrix

**10 Pfade**

10.1 M und m - die Moveto Anweisung
10.2 L und l, H und h, V und v - die lineto Anweisungen
10.3 Z und z - die closepath Anweisung
10.4 C und c - kubische Bezierkurven
10.5 S und s - Kurzform fuer kubische Bezierkurven
10.6 Q und q - quadratische Bezierkurven
10.7 T und t - Kurzform fuer quadratische Bezierkurven
10.8 A und a - Bogenkurven
10.9 Pfeilspitzen - das marker-Element
10.10 Tipp zur Erstellung von Pfaden

**11 Painting**

11.1 Eigenschaften von Füllungen und Randlinien
11.2 Eigenschaften zur Darstellung der Objekte
11.3 Eigenschaften zur Farbdarstellung

**12 Verläufe und Muster**

12.1 Lineare Verläufe - das linearGradient-Element
12.2 Radiale Verläufe - das radialGradient-Element
12.3 Muster - das pattern-Element

**13 Animationen**

13.1 Animationen erstellen - animate
13.2 Attribute zur Zielbestimmung - animAttributeAttrs
13.3 Attribute zur Zeitsteuerung - animTimingAttrs
13.4 Attribute zur Wertänderung - animValueAttrs
13.5 Weitere Attribute - animAdditionAttrs
13.6 Attributwerte verändern - set
13.7 Bewegung entlang eines Pfades - animateMotion
13.8 Animierte Farbveränderungen - animateColor
13.9 Animierte Transformationen - animateTransform

**14 Filtereffekte**

14.1 Attribute für das filter-Element
14.2 Alle primitiven Filter - Übersicht
14.3 Primitive Filter - filter_primitive_attributes
14.4 Primitive Filter - filter_primitive_attributes_with_in
14.5 Primitive Filter - PresentationAttributs-FilterPrimitives
14.6 feConvolveMatrix
14.7 feDisplacementMap
14.8 feGaussianBlur
14.9 feOffset
14.10 feMorphology

14.11 feBlend
14.12 feComposite
14.13 feMerge
14.14 feColorMatrix
14.15 feComponentTransfer
14.16 feDistantLight, fePointLight, feSpotLight
14.17 feDiffuseLighting
14.18 feSpecularLighting
14.19 feFlood
14.20 feImage
14.21 feTile
14.22 feTurbulence

**15 Masken**

15.1 Clipping - das clipPath-Element
15.2 Masking - das mask-Element

**16 Scripting**

16.1 Übersicht Event-Attribute
16.2 Skriptsprachen einbinden - das script-Element
16.3 SVG Inhalte verfügbar machen
16.4 Methoden für den Zugriff auf Elemente
16.5 Methoden für den Zugriff auf Knoten
16.6 Methoden für den Zugriff auf Attribute
16.7 Methoden für den Zugriff auf Text
16.8 Methoden zur Manipulation von Attributen/Eigenschaften
16.9 Methoden zur Manipulation von Elementen/Knoten
16.10 Methoden zur Manipulation von Text

**17 SVG fonts**

17.1 Grundlagen SVG fonts
17.2 SVG fonts mit Apache Batik erzeugen
17.3 Externe SVG Fonts
17.4 Das font-Element
17.5 Beschreibung eines font - das Element font-face
17.6 Zeichen definieren - die Elemente missing-glyph und glyph
17.7 Kerning - die Elemente hkern und vkern

# 1 Vorbereitungen

In diesem Tutorial finden Sie eine schnelle, schnörkellose Anleitung zur Erstellung von SVG-Grafiken nicht nur für das Internet.

Da SVG-Grafiken für die Darstellung im Internet bzw. auf Webseiten konzipiert wurden, sind grundlegende Kenntnisse in HTML/XHTML, XML, CSS und JavaScript für das Verständnis der behandelten Thematik äußerst empfehlenswert. Zu diesen Themen gibt es bereits eine große Anzahl von Publikationen, ich empfehle SelfHTML, so dass ich hier nicht näher darauf eingehe, um den Umfang des Tutorials nicht unnötig aufzublähen.

In den ersten zwei Kapiteln dieses Tutorials erfahren Sie, welche Möglichkeiten SVG bietet und welche Voraussetzungen notwendig sind, um SVG-Grafiken darstellen und erstellen zu können. Außerdem werde ich Ihnen die grundlegende Arbeitsweise von SVG vorstellen.

In den weiteren Kapiteln finden Sie viele komplette SVG-Beispiele, da meiner Ansicht nach SVG-Quellcodes das optimale Lernmedium darstellen - "Learning by doing". Sie können die Beispiel-Grafiken also auch dazu verwenden ihre eigene Kreativität auszuleben. Ein paar Änderungen hier und da im jeweiligen Quellcode bewirken neue Variationen der entsprechenden Grafik und tragen zum weiteren Verständnis von SVG bei. Lassen Sie sich inspirieren und entwickeln Sie eigene Grafiken - schon nach dem Erlernen der grundlegenden Dokumentstruktur und einigen Grundformen sind ansprechende Grafiken in SVG möglich.

## 1.1 Grundlagen SVG

SVG - Scalable Vektor Graphics - ist eine XML-basierte, textorientierte Auszeichnungssprache um zweidimensionale, skalierbare Grafiken zu beschreiben. Dabei erlaubt SVG Vektorgrafiken, Bilder und Text als grafische Objekte. Die Darstellung der Grafiken erfolgt durch sogenannte user agents. Das sind in der Regel Browser, Browser-Plugins oder eigenständinge SVG-Viewer.

Die Features von SVG:

- SVG ist eine auf XML basierende Sprache
- SVG ist ein offizieller Standard des W3C
- SVG Grafiken bestehen aus ASCII-Code
- SVG Grafiken können mit jedem beliebigen Texteditor erstellt werden
- SVG Grafiken (Vektorgrafiken) sind ohne Qualitätsverlust skalierbar
- SVG unterstützt Stylesprachen (z.B. CSS)
- SVG unterstützt Scriptsprachen (z.B. JavaScript, ECMAScript)

Kompatibilitäten von SVG: Die Verweise führen zu den jeweiligen W3 Recommandations oder Heimatseiten der entsprechenden Organisationen Unicode bzw. WAI.

- XML - eXtensible Markup Language 1.0
- XML-NS - Namespaces in XML
- Xlink - XML Linking Language
- XML-Base - Base URI Spezification
- XPTR - Xpointer
- CSS2 - Cascading Stylesheets 2.0
- XSLT - Extensible Stylesheet Language Transformations Version 1.0
- DOM 2 - Document Object Model Level 2
- SMIL 1 - Synchronized Multimedia Integration Language
- SMILANIM - SMIL Animation
- HTML - Hypertext Markup Language 4.01
- XHTML - Extensible Hypertext Markup Language
- UNICODE - Unicode Zeichensatz
- CHARMOD - Charmod Zeichensatz
- WAI - Zugänglichkeitsrichtlinieren für Webinhalte

SVG bietet folgende Möglichkeiten:

- Grundlegende geometrische Objekte erzeugen
- Pfade zu erzeugen
- Texte zu erzeugen, die für Suchmaschinen indizierbar sind
- Grafische Symbole für den mehrmaligen Gebrauch zu definieren
- Farben, Farbverläufe und Muster als Füllung für Objekte festzulegen
- Ausschnitte und Masken zu verwenden
- Filtereffekte auf Objekte anzuwenden
- Interaktiv auf Objekte zuzugreifen
- Hyperlinks für Objekte zu definieren
- Animationen zu erstellen

Der erste offizielle SVG 1.0 W3C Recommendation des World Wide Web Consortiums ist auf September 2001 datiert, d.h. einige der Möglichkeiten von SVG werden momentan von den Herstellern der Anwendungssoftware (wie z.B. Browser oder Browsererweiterungen, aber auch eigenständige SVG-Viewer oder SVG-Editoren) noch nicht unterstützt.

Seit dem 14. Januar 2003 existiert bereits ein zweiter offizieller Standard SVG 1.1 W3C Recommendation und die nächste Aktualisierung SVG 1.2 W3C Working Draft ist in Vorbereitung.

Grundlage dieses Tutorials ist die SVG 1.0 W3C Recommendation. Veränderungen oder Erweiterungen innerhalb der neueren Standards werden natürlich nach und nach auch in dieses Tutorial einfließen. Denn wie die SVG Standards (und die SVG Programme) wird auch dieses Tutorial permanent erweitert und aktualisiert. Den Zeitpunkt der **letzten Aktualisierung des SVG Tutorials** finden Sie am Ende jeder Seite.

Nun müssen Sie ihren Rechner SVG-fähig machen, d.h. ein Programm installieren, das SVG Grafiken darstellen kann (user agent). Dieses kann ein eingenständiger Betrachter (viewer) oder eine Erweiterung (plugin) sein. Mehr darüber im nächsten Kapitel ...

## 1.2 Notwendige Voraussetzungen

Für das Training mit diesem Tutorial benötigen Sie

- Kenntnisse im Umgang mit einem der Betriebssysteme Windows oder Linux,
- einen **Web-Browser** der neuesten Generation,
- ein SVG **user agent** (Programm) zur Darstellung von SVG Dokumenten, wie z.B. ein plugin oder einen eingenständigen viewer
- und einen guten **Texteditor**.

Als zusätzliche SVG Informationsquellen sind folgende zu nennen:

- SVG 1.0 W3C Recommendation (erste Version) SVG 1.1 W3C Recommendation (aktuelle Version) Der offizielle, normative Standard zu SVG vom World Wide Web Consortium. Alle sonstigen, guten Dokumente zu SVG - inclusive dieses Tutorial - basieren auf diesem Standard. Laden Sie deshalb den aktuellen Standard 1.1 zur offline-Verwendung vom W3C herunter. Im Zweifelsfalle (nobody is perfect) hat die Beschreibung im Standard die höchste Priorität.
- SelfHTML 8.0 Alle notwendigen Informationen zur Metasprache XML sowie, anderen Sprachen wie HTML, CSS oder Javascript/DOM finden Sie in Stefan Münz's Dokumentation. Kann zur Offline-Verwendung heruntergeladen werden.
- W3C Heimatseite von SVG Hier finden sie alle offiziellen Informationen zu SVG. Dort finden Sie ebenfalls eine gute Übersicht aktueller Software, mit denen Sie SVG-Dokumente darstellen, edietieren, exportieren oder konvertieren kÖnnen.Sollte man auf jeden Fall besucht haben.
- www.scale-a-vector.de Sehr gute Online-Informationsquelle zu SVG. Enthält außerdem eine umfangreiche Linkliste zu Seiten mit SVG relevanter Software, weiteren SVG-Infos und -Beispielen im Web.

Natülich finden Sie im Internet noch eine große, wachsende Anzahl weiterer Publikationen rund um das Thema SVG: Linklisten, Tipps, Beispiele, Anleitungen, ... Was darauf hinweist, dass dieser wunderbare Standard noch eine große Zukunft vor sich hat :-). Es ist daher äußerst lohnenswert im Web nach SVG zu googlen .. .. eine kleine Auswahl empfehlenswerter Einstiegs-Links finden Sie auf der Linkliste diese Tutorials.

In den nachfolgenden Kapiteln dieses Tutorials finden Sie notwendige Informationen, um SVG Grafiken mit Hilfe eines Web-Browsers und einer geeigneten SVG Applikation unter **Windows** und **Linux** darzustellen und zu editieren. Wenn Sie **Windows** als Betriebssystem verwenden, können Sie die Kapitel zu Linux einfach überspringen. Innerhalb der **Linux** Kapitel habe ich mich auf das Wesentliche beschränkt. Daher wenden sich diese Kapitel an erfahrene Linux Anwender. Und: auch wenn Sie Linux verwenden, werfen Sie trotzdem einen Blick in das Kapitel SVG unter Windows. Dort sind Informationen zu den gängigsten Browserversionen zu finden.

## 1.3 SVG unter Windows

Software zur Darstellung und Bearbeitung von SVG Dokumenten für Windows. Momentan werden die meisten SVG Features vom Adobe SVG Viewer 3.0 (Browser-Plugin) in Zusammenarbeit mit dem Opera Browser >= 7.0 oder dem Internet-Explorer >= 5.5 auf dem Betriebssystem Windows unterstützt.

**Einen Browser der neusten Generation**

- MS IExplorer 5.5/6.x Stellt zur Zeit zusammen mit dem Adobe SVG Viewer (siehe unten) die meisten Features von SVG dar und ist daher derzeit zu empfehlen (siehe auch Opera 7.0).
- Opera 7.x Nach Installation des Adobe SVG Viewers (dazu müssen die Dateien NPSVG3.DLL und NPSVG3.ZIP aus dem Installationsordner des Adobe SVG Viewers in das Plugin-Verzeichnis des Opera kopiert werden) konnte auch der Opera 6.x SVG Grafiken eingeschränkt darstellen. Animationen und Javascript werden nicht unterstützt. Opera 7.0 unterstützt in Zusammenarbeit mit dem Adobe SVG Viewer Javascript und - im Gegensatz zu 7.0beta-Version, jetzt auch - Animationen. Opera 7.0 und Adobe SVG Viewer stellen also ebenfalls die meisten Features von SVG dar. Daher ist der Opera Browser momentan eine wirkliche Alternative zum Internet Explorer.
- Mozillla 1.x/Mozilla SVG Kommende Versionen des (vom Autor bevorzugten) Mozilla Browsers werden das SVG Format ohne die zusätzliche Verwendung eines fremden Viewers bzw. Plugins unterstützen. Der Mozilla-Browser implementiert also einen eigenen SVG Viewer. Aktuelle Informationen zum Stand der Dinge finden Sie bei www.mozilla.org/projects/svg und www.croczilla.com/svg. Der Adobe SVG Viewer wird daher von Mozilla offiziell nicht unterstützt (Versionen 1.0 und 1.2 funktionieren jedoch eingeschränkt mit dem Adobe SVG Viewer).
- Netscape Navigator 6.x/7.x Der Netscapeversionen 6.2 und 7.0 stellen zusammen mit dem Adobe SVG Viewer die meisten Features von SVG dar. Dazu müssen in jedem Fall die Dateien NPSVG3.DLL und NPSVG3.ZIP (die befinden sich im Installationsordner des Adobe SVG Viewers) in das Plugin-Verzeichnis des Netscape kopiert werden. In der Übersicht der installierten Plugins - findet sich über das Netscape Hilfe-Menü - sollte dann der SVG-Viewer angeführt und aktiviert sein. Bei Netscape können allerdings Probleme auftreten. Bei Tests mit der Version 6.2 öffnete sich zum Beispiel immer zuerst das Dialogfenster zum Speichern oder Ausführen der SVG Datei. Über den Dialog Erweitert mußte bei jedem Zugriff als ausführendes Programm erneut netscape6 festgelegt werden - erst danach wurde die Grafik korrekt dargestellt. Desweiteren kam es auch gelegentlich zu Abstürzen.
- Amaya Webbrowser/Editor des W3C. (Free) Erläuterung unter Texteditoren.

**Eine SVG Applikation**

- Adobe SVG Viewer (plugin) Installieren Sie einen der oben genannten Browser und den Adobe SVG Viewer als Plugin, um SVG Dokumente im Browser anzeigen zu können. Bei Adobe können Sie auf der Seite SVG Plugin Test ermitteln, ob ihr Browser in Zusammenarbeit mit dem Viewer SVG Grafiken darstellen kann.
- Apache Batik (viewer - und mehr) SVG Dokumente anzeigen, konvertieren und mehr. Auch für Linux erhältlich. Apache Batik ist ein auf JAVA basierendes Projekt, das aus mehreren SVG Programmen besteht. Es beinhaltet einen eigenständigen SVG Viewer (Batik Squiggle) und ist KEIN Browser-Plugin! Alle wichtigen Informationen zur Installation, Verwendung und Funktionsweise finden Sie auf den Heimatseiten von Apache Batik: xml.apache.org/batik/.

**Ein komfortabler Texteditor**

- Sodipodi (Free) Ein sehr empfehlenswerter WYSIWYG-Editor für SVG-Grafiken. Obwohl noch eine 0.-Version, bietet dieser SVG-Editor, der eigenlich von Linux kommt, schon eine große Anzahl von Möglichkeiten. Über die XML-Editor Funktion kann ebenfalls bequem und ohne Einschränkungen (!) auf den Quelltext des SVG-Dokuments zugegriffen werden. Ein weiterer Vorteil dieses Editors: er ist für Windows und Linux verfügbar.
- Inkscape (Free) Ein weiterere sehr empfehlenswerter WYSIWYG-Editor für SVG. Ebenfalls noch eine 0.-Version; aber auch dieser Editor bietet schon eine große Anzahl von Möglichkeiten. Dieser Editor ist ebenfalls für Windows und Linux verfügbar.
- Phase 5 HTML Editor (Free) Ein einfacher ASCII-Text-Editor ihrer Wahl ist zur Erstellung von SVG-Grafiken ausreichend. Da ich den Phase 5 HTML-Editor von Uli Meybohm verwende, empfehle ich ihn hier. Er stellt einige Arbeitserleichterungen bei der Erstellung von XML-basierten Dokumenten zur Verfügung, obwohl er eigentlich ein Editor zur Erstellung von HTML Dokumenten ist. So können Sie über den Menüeintrag "Benutzer" eigene Kurzbefehle für SVG-Elemente erzeugen :-). Es gibt

aber bereits auch einige SVG-Editoren - auch WYSIWYG (what you see is what you get) - die Sie verwenden können (siehe oben Sodipodi). Einige XML-Editoren bieten ebenfalls Annehmlichkeiten bei der Tipparbeit.

- Amaya Webbrowser/Editor des W3C. (Free) Der Amaya Webbrowser beinhaltet einen eingenen SVG viewer. Allerdiings können SVG Grafiken vom Amaya Browser nur eingeschränkt dargestellt werden. Mit Hilfe der Editorfunktion des Amaya können Sie geometrische Figuren und Pfade in einem WYSIWYG Modus erstellen und auf den Quellcode zugreifen. Eine wunderbare Hilfe bei der Erstellung von komplexen Pfaden.
- SVG Builder von NBProgs. (Free) Eine Art Quelltext-Editor der beim Einfügen von SVG Elementen und Attributen hilfreich ist.
- IMS Webdwarf V2 der Firma VirtualMechanics. (Free und Kommerziell) WYSIWYG Editor zum Erstellen von SVG Grafiken. über das Preview Menü können Sie Ihre Grafik im SVG Format in einem SVG-fähigen Browser betrachten und haben so auch Zugriff auf den Quelltext.

## 1.4 SVG unter Linux allgemein

Mittlerweile gibt es eine Reihe von Applikationen unter Linux, die SVG unterstützen. Einige sind bereits Bestandteil verschiedenster Distributionen wie SuSE, Debian, etc. So können Sie z.B. mit dem Zeichentool von **OpenOffice 1.x** oder dem Programm **dia**, SVG exportieren - es lohnt sich auf jeden Fall die SVG-Fähigkeiten seiner Linux-Tools mal näher zu untersuchen. Grundsätzlich benötigen Sie für den Umgang mit SVG:

**Einen Browser der neusten Generation**

- Mozilla 1.x Funktioniert mit dem Adobe-SVG-Plugin für Linux sehr gut. Es kommt allerdings auch der Tag, da kann Mozilla alle Features von SVG auch ohne zusätzliche Applikationen anzeigen. Ein paar Features, wie einfache geometrische Formen, kann Mozilla SVG schon :-) - natürlich auch unter Linux. Siehe auch Mozilla SVG Project.
- Opera 7.x Als rpm, deb oder Tarball erhältlich. Sehr empfehlenswert.
- KDE Konquerer - nicht nur Webbrowser Kann mit Hilfe von KSVG (s.u.) SVG Dokumente darstellen*. (*) Habe ich noch nicht getestet - steht auf meiner SVG-TODO-List.

**Eine SVG Applikation (SVG-Viewer)**

- Adobe SVG Viewer Seit Ende 2003 als Linux Version 3.01 verfügbar - funktioniert prima :-).
- KSVG Ein KDE Projekt zur Darstellung von SVG unter KDE, seit KDE 3.2 Bestandteil von KDE. ErmÖglicht die Darstellung von SVG in KDE-Applikationen, wie z.B. dem Webbrowser Konqueror.
- Apache Batik SVG Dokumente anzeigen, konvertieren und mehr. Auch für Windows erhältlich. Apache Batik beinhaltet einen eigenständigen SVG Viewer (Batik Squiggle) und ist KEIN Browser-Plugin!

**Ein komfortabler Texteditor**

- Sodipodi (Free) Ein sehr empfehlenswerter WYSIWYG-Editor für SVG-Grafiken. Obwohl noch eine 0.-Version, bietet dieser SVG-Editor, schon eine große Anzahl von Möglichkeiten. Über die XML-Editor Funktion kann ebenfalls bequem und ohne Einschränkungen (!) auf den Quelltext des SVG-Dokuments zugegriffen werden. Ein weiterer Vorteil dieses Editors: er ist für Windows und Linux verfügbar.
- Inkscape (Free) Ein weiterere sehr empfehlenswerter WYSIWYG-Editor für SVG. Ebenfalls noch eine 0.-Version; aber auch dieser Editor bietet schon eine große Anzahl von Möglichkeiten. Dieser Editor ist ebenfalls für Windows und Linux verfügbar.
- Weiterhin sind hier vom **vi** bis zu **Quanta+** natürlich jeder beliebige Texteditor möglich. Tipps zu besonders geeigneten Editoren sind äußerst willkommen ..

## 1.5 SVG unter Debian Woody (mit Adobe Plugin)

Seit Ende 2003 funktioniert das SVG-Plugin von Adobe zusammen mit dem Opera Browser oder dem Mozilla Browser unter Linux sehr zufriedenstellend. Nachfolgend sind alle notwendigen Schritte aufgeführt um SVG Grafiken unter Debian Linux Woody in den beiden oben genannten Browsern betrachten zu können.

**Benötigte Pakete**:

- adobesvg-3.01x88-linux.i386.tar.gz
- mozilla-686-pc-linux-gnu-1.6-installer.tar.gz
- opera-static_7.23-20031119.1-qt_en_i386.deb
- openmotif_2.1.30-5_i386.deb (beinhaltet libXm.so.2 für Opera)

**Installation von Mozilla:** Am besten auf einer grafischen Benutzeroberfläche im Terminal auszuführen. > mkdir /usr/local/src/mozilla > cp mozilla-686-pc-linux-gnu-1.6-installer.tar.gz /usr/local/src/mozilla/ > cd /usr/local/src/mozilla > tar xvfz mozilla-686-pc-linux-gnu-1.6-installer.tar.gz > cd mozilla-installer > ./mozilla-installer Dann Anweisungen auf dem Bildschirm ausführen und Mozilla ist installiert. Kommandozeile zum Start von Mozilla: > /usr/local/mozilla/mozilla

**Installation des Adobe SVG Plugins:** > cp adobesvg-3.01x88-linux.i386.tar.gz /usr/local/src > cd /usr/local/src > tar xvfz adobesvg-3.01x88-linux.i386.tar.gz > cd adobesvg-3.01/ > ./install.sh

Mozilla ist jetzt SVG-fähig.

**Installation von Opera:** Installation des Debian-Pakets opera-static_7.23-20031119.1-qt_en_i386.deb mit apt-get oder dpkg. Installation des Debian-Pakets openmotif_2.1.30-5_i386.deb mit apt-get oder dpkg. Nach der Installation von Openmotif sollte sich auf dem System die Bibliothek libXm.so.2 finden lassen. Opera benötigt das SVG-Plugin noch in seinem eigenen Plugin-Verzeichnis: > cp /usr/local/adobesvg/libNPSVG3.so /usr/lib/opera/plugins/

Opera ist jetzt SVG-fähig.

## 1.6 SVG unter Debian Woody (mit Batik)

Webbrowser Konquerer und Apache Batik Projekt unter KDE.

Apache Batik ist ein Projekt (d.h. es besteht aus mehreren unterschiedlichen Programmen) um SVG Dateien anzuzeigen, zu konvertieren und mehr.

Das Debian Paket **libbatik-java** stellt die SVG Batik Grundkomponenten von Apache Batik als Java **jar**-Dateien zur Verfügung. Da Apache Batik aus einer Java Programm Ansammlung besteht, muß auf Ihrem Debian System eine Java-Laufzeitumgebung installiert sein. Leider ist das Debian Paket **libbatik-java** noch sehr BETA. Die Dokumentation der Abhängikeiten ist lückenhaft und wichtige **jar**-Dateien zur Ausführung der Apache Batik Programme sind nicht im Paket enthalten. Aus diesem Grund folgt hier ein kleines "Howto" zur Installation von Batik unter Debian. Auf meinem Rechner läufts :-).

**TODO 1:** Alle nachfolgend aufgeführten Pakete (incl. Abhängigkeiten) müssen installiert werden. Die j2*-Pakete (Sun Java 2) sind aufgrund Ihrer Lizenzbestimmungen nicht im Distributionsumfang von Debian enthalten, können aber im Internet frei bezogen werden, z.B. bei www.blackdown.org.

- java-common-0.14 - Grundvoraussetzung für Java auf Debian
- j2se-common-1.1 - Voraussetzung für Sun Java
- kaffe-1:1.0.5e-0.5 - Java Bytecodeinterpreter
- ant-1.4.1-4 - Makefile für Java
- j2re1.3-1.3.1.02b-2 - Java Runtime-Environment von Sun
- j2sdk1.3-1.3.1.02b-2 - Java Software Develpment Kit von Sun
- libxerces-java-1.4.3-1 - Apache Xerces XML Parser
- libxerces2-java-2.0.1-1 - Apache Xerces XML Parser Version 2
- libxalan2-java-2.0.1-1 - Apache Xalan XSLT Prozessor
- libavalon-framework-java-4.1.2-1 - Framework für Java Applikationen
- libbsf-java-1:2.2-1 - Framework für Java Applikationen
- liblogkit-java-1.0.1-1 - Logger für Java
- libbatik-java-1.5beta2-4 - Apache Batik SVG Applikationen

**TODO 2:** Die im **libbatik-java** Paket fehlenden Java **jar**-Dateien befinden sich in der offiziellen Apache Batik binary-Version **batik-1.5beta4b.zip**, erhältlich bei xml.apache.org/batik/dist. Die Datei **batik-1.5beta4b.zip** also in ein beliebiges Unterverzeichnis kopieren (herunterladen), und dann mit dem Programm **jar** entpacken: > cd Unterverzeichnis > jar xf batik-1.5beta4b.zip Erzeugt wird ein Verzeichnis **batik-1.5/**. .. alle Dateien mit der Dateinamenerweiterung **.jar** aus dem Verzeichnis **batik-1.5/** und das komplette Unterverzeichnis **lib/** müssen nun in das Verzeichnis **/usr/share/java/** kopiert werden: > cp batik-1.5/*.jar /usr/share/java/ > cp -r batik-1.5/lib/ /usr/share/java/

**Batik Applikationen ausführen** am Beispiel Squiggle Der Batik SVG-Browser "squiggle" wird unter der grafischen Benutzeroberfläche (bei mir ist das KDE) mit folgender Kommandozeile gestartet: > java -jar /usr/share/java/batik-squiggle.jar **Anwendungsbeispiel:** Eine SVG-Datei aus dem SVG-Tutorial anzeigen

1. Webbrowser Konquerer starten
2. svg.tutorial.aptico.de aufsuchen ;-)
3. auf eine SVG-Beispiel-Link klicken
4. im Dialog mit dem Konquerer "öffnen" wählen
5. im Dialog "öffnen mit .." dann folgendes in die Eingabezeile eingeben: > java -jar /usr/share/java/batik-squiggle.jar nach einer Download-Sequenz öffnet sich der Batik-SVG Browser und zeigt die entsprechende SVG-Datei an. Es werden noch nicht alle Features von SVG unterstützt.

Alle anderen Programme von Apache Batik werden ähnlich, d.h. ebenfalls über die entsprechenden **batik-*.jar**-Dateien im Ordner **/usr/share/java/** gestartet. Siehe Apache Batik Website.

## 1.7 SVG unter SuSE ab 8.2 (mit Adobe Plugin)

Seit Ende 2003 funktioniert das SVG-Plugin von Adobe zusammen mit dem Opera Browser oder dem Mozilla Browser unter Linux sehr zufriedenstellend. Nachfolgend sind alle notwendigen Schritte aufgeführt um SVG Grafiken unter SuSE Linux 8.x in den beiden oben genannten Browsern betrachten zu können.

**Benötigte Pakete** (openmotif ist Bestandteil der SuSE Distribution):

- adobesvg-3.01x88-linux.i386.tar.gz
- mozilla-686-pc-linux-gnu-1.6-installer.tar.gz
- opera-7.23-20031119.1-static-qt.i386-en.rpm
- openmotif-2.2.2-204.rpm (beinhaltet libXm.so.3 für Opera)

**Installation von Mozilla:** Am besten auf einer grafischen Benutzeroberfläche im Terminal auszuführen. > mkdir /usr/local/src/mozilla > cp mozilla-686-pc-linux-gnu-1.6-installer.tar.gz /usr/local/src/mozilla/ > cd /usr/local/src/mozilla > tar xvfz mozilla-686-pc-linux-gnu-1.6-installer.tar.gz > cd mozilla-installer > ./mozilla-installer Dann Anweisungen auf dem Bildschirm ausführen und Mozilla ist installiert. Kommandozeile zum Start von Mozilla: > /usr/local/mozilla/mozilla

**Installation des Adobe SVG Plugins:** > cp adobesvg-3.01x88-linux.i386.tar.gz /usr/local/src > cd /usr/local/src > tar xvfz adobesvg-3.01x88-linux.i386.tar.gz > cd adobesvg-3.01/ > ./install.sh

Mozilla ist jetzt SVG-fähig.

**Installation von Opera:** Installation des rpm-Pakets opera-7.23-20031119.1-static-qt.i386-en.rpm mit rpm oder yast. Installation des Pakets openmotif mit rpm oder yast. Nach der Installation von openmotif sollte sich auf dem System die Bibliothek libXm.so.3 finden lassen. Da Opera jedoch die Bibliothek libXm.so.2 verwendet, muss ein symbolischer Link erstellt werden: > cd /usr/X11R6/lib > ln -s libXm.so.3 libXm.so.2 Opera benötigt das SVG-Plugin noch in seinem eigenen Plugin-Verzeichnis: > cp /usr/local/adobesvg/libNPSVG3.so /usr/lib/opera/plugins/

Opera ist jetzt SVG-fähig.

## 1.8 SVG unter SuSE 8.x (mit Batik)

Webbrowser Konquerer und Apache Batik Projekt unter KDE.

Apache Batik ist ein Projekt (d.h. es besteht aus mehreren unterschiedlichen Programmen) um SVG Dateien anzuzeigen, zu konvertieren und mehr.

Da Apache Batik aus einer Java Programm Ansammlung besteht, müssen auf Ihrem SuSE System die Pakete **java2-jre** (Java Laufzeitumgebung) und/oder **java2** (Java Entwicklungsumgebung) installiert sein. Die momentan aktuelle Apache Batik binary-Version **batik-1.5beta4b.zip** ist bei xml.apache.org/batik/dist erhältlich und muß heruntergeladen werden.Grundsätzlich sollten Sie immer die aktuellste Version installieren (falls z.B. aus batik-1.5beta4b die Version batik-1.5ohnebeta :-) wird).

**TODO:** Entpacken Sie die Datei **batik-1.5beta4b.zip** in einem beliebigen Verzeichnis (in unserm Beispiel wurde die Datei in das Verzeichnis /usr/local/ kopiert, um des dort zu entpacken): > cd /usr/local > unzip batik-1.5beta4b.zip Dabei wird ein Verzeichnis mit dem Namen batik-1.5 unterhalb von /usr/local erzeugt.

Nun können Sie unter der grafischen Benutzeroberfläche KDE mit folgendem Befehl den Baktik-SVG Browser Squiggle starten: > java -jar /usr/local/batik-1.5/batik-squiggle.jar **Anwendungsbeispiel:** Eine SVG-Datei aus dem SVG-Tutorial anzeigen

1. Webbrowser Konquerer starten
2. svg.tutorial.aptico.de aufsuchen ;-)
3. auf eine SVG-Beispiel-Link klicken
4. im Dialog mit dem Konquerer "öffnen" wählen
5. im Dialog "öffnen mit .." dann folgendes in die Eingabezeile eingeben: > java -jar /usr/local/batik-1.5/batik-squiggle.jar nach einer Download-Sequenz öffnet sich der Batik-SVG Browser und zeigt die entsprechende SVG-Datei an. Es werden noch nicht alle Features von SVG unterstützt.

Alle anderen Programme von Apache Batik werden ähnlich, d.h. ebenfalls über die entsprechenden **batik-*.jar**-Dateien im batik-1.5 Ordner (in unserem Beispiel /usr/local/batik-1.5) startet. Siehe Apache Batik Website.

## 2 Darstellung von SVG-Grafiken

SVG-Quelltexte sind reine ASCII-Textdokumente, die geometrische Objekte durch Auszeichnungsbefehle beschreiben. Sie werden mit der Dateinamenerweiterung **.svg** abgespeichert.

Ein SVG **user agent** kann ein SVG Dokument mit der Endung .svg direkt am Bildschirm darstellen. Der user agent interpretiert die im SVG-Quelltext enthaltenen Auszeichnungsbefehle (Elemente und Attribute) als Beschreibung geometrischer Objekte und malt sie auf den Bildschirm (Rendern).

Wie Sie SVG-Dokumente auch in HTML/XHTML-Dokumenten darstellen können und genauere Informationen darüber, wie die geometrischen Objekte gerendert werden, erfahren Sie hier.

## 2.1 SVG-Grafiken in Webdokumenten

**Wenn ein Browser SVG-Grafiken darstellen kann, zeigt er die .svg-Dateien direkt im Browserfenster an.**

SVG-Quelltexte können aber auch in HTML/XHTML-Dokumente eingebettet werden.

Nachfolgend alle Möglichkeiten SVG-Grafiken in Webseiten zu verwenden und anschließend zu jeder Möglichkeit ein Beispiel. Achtung: Zur Zeit - Ende 2002 - werden noch nicht alle Möglichkeiten von den darstellenden Anwendungen (Browser und SVG Viewer) unterstützt.

- Eine alleinstehende SVG Grafik (können alle Browser).
- Durch einen Verweis in ein HTML/XHTML Dokument einbetten (kann nur IE)
- Intern in ein XHTML Dokument einbetten (kann noch kein Browser)
- Verlinkung in einem HTML/XHTML Dokument durch einen externen Verweis (können alle Browser)
- Als Referenz von Style-Eigenschaften innerhalb von HTML/XHTML Dokumenten (kann noch kein Browser)

**Eine alleinstehende SVG Grafik.** Das SVG Dokument wird mit der Dateinamenerweiterung .svg abgespeichert und kann direkt im Browser angezeigt werden. Alle (svg-fähigen) Browser können diese alleinstehenden SVG Grafiken anzeigen. Die Beispiele in diesem Tutorial sind in der Regel alleinstehende SVG Grafiken.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="280" height="280">
<title>dummy</title>
<desc>ein SVG Smilie</desc>
<defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:60px;
            font-weight:bold; fill:black; stroke:black; stroke-width:4px;}
    ]]>
    </style>
  <symbol id="smilie">
    <desc>ein Symbol-Smilie</desc>
    <circle cx="20" cy="20" r="15"
      fill="yellow" stroke="black" />
    <path d="M 13 26 A 5 3 0 0 0 27 26"
      stroke="black" fill="none" stroke-width="2" />
  </symbol>
</defs>
<use xlink:href="#smilie" transform="translate(-20,-20) scale(8)" />
<text x="79" y="119">S</text>
<text x="117" y="170">V</text>
<text x="156" y="119">G</text>
</svg>
```

**SVG Grafik als Verweisziel innerhalb eines HTML Dokuments: object-Tag.** Mit Hilfe des HTML-Tags <objekt> wird die SVG Grafik als Multimedia-Objekt in das HTML Dokument eingebettet. Der Text innerhalb des object-Elements - "Sie benötigen einen SVG-Viewer" - wird angezeigt, falls der darstellende Browser SVG-Dokumente nicht unterstützt.

Wenn Sie SVG Grafiken als Verweisziel einbinden, sollten Sie beachten, dass der Browser, bei der Darstellung der Grafik, die Angaben zu Breite und Höhe des HTML-Elements verwendet. Falls Sie also eine SVG-Grafik mit größeren Ausmaßen einbinden, als sie im HTML-Element angegeben haben, wird sie in der Darstellung abgeschnitten. Verwenden Sie also immer die wirkliche Breite und Höhe der SVG-Grafik als Werte für die Attribute width und height in den entsprechenden HTML-Elementen (object, embed, img).

Beispiel Quellcode

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html><head><title>SVG in HTML</title></head>
<body style="margin-left:20px; margin-top:10px;">
<p style="font-family:Verdana;">
SVG Grafik durch das object-Tag eingebunden.<br>
Wird vom IE korrekt dargestellt.</p>

<object data="dummy3.svg" type="image/svg+xml" width="280" height="280">
Sie benötigen einen SVG-Viewer
</object>

</body>
</html>
```

**SVG Grafik als Verweisziel innerhalb eines HTML Dokuments: embed-Tag.** Mit Hilfe des HTML-Tags <embed> wird die SVG Grafik als Multimedia-Objekt nach Netscape Syntax in das HTML Dokument eingebettet.

Beispiel Quellcode

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html><head><title>SVG in HTML</title></head>
<body style="margin-left:20px; margin-top:10px;">
<p style="font-family:Verdana;">
SVG Grafik mit dem embed-Tag eingebunden<br>
wird vom IE dargestellt</p>

<embed type="image/svg+xml" src="dummy3.svg" width="280" height="280">

</body>
</html>
```

**SVG Grafik als Verweisziel innerhalb eines HTML Dokuments: img-Tag.** Mit Hilfe des HTML-Tags <img> wird die SVG Grafik als normale Grafik in das HTML Dokument eingebettet.

Beispiel Quellcode

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html><head><title>SVG in HTML</title></head>
<body style="margin-left:20px; margin-top:10px;">
<p style="font-family:Verdana;">
SVG Grafik durch das img-Tag eingebunden<br>
wird noch nicht dargestellt</p>

<img src="dummy3.svg" width="280" height="280" border="0" alt="SVG Grafik">

</body>
</html>
```

**SVG Quellcode innerhalb eines XHTML Dokuments.** Die SVG Grafik bzw. der Quellcode der SVG Grafik wird direkt in einem XHTML Dokument eingebunden. Zur Unterscheidung zwischen XHTML-Tags und SVG-Elementen dient die Namensraumangabe im öffnenden svg-Element. Ein XML-, XHTML- und SVG-fähiger Browser kann das Dokument dann darstellen. (Berechigte Hoffnung des Autors: der Mozilla wird es irgendwann können ..)

Beispiel Quellcode

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
<head><title>SVG in XHTML</title></head>
<body style="margin-left:20px; margin-top:10px;">
<p style="font-family:Verdana;">
SVG Quellcode im XHTML Quellcode eingebettet<br />
wird noch nicht dargestellt</p>

<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="280" height="280">
<title>dummy3</title>
<desc>ein SVG Smilie</desc>
<defs>
    <style type="text/css">
    <!--
    <![CDATA[
      text {font-family:verdana,sans-serif; font-size:60px;
           font-weight:bold; fill:black; stroke:black; stroke-width:4px;}
    ]]>
    -->
    </style>
  <symbol id="smilie">
    <circle cx="20" cy="20" r="15"
      fill="yellow" stroke="black" />
    <path d="M 13 26 A 5 3 0 0 0 27 26"
      stroke="black" fill="none" stroke-width="2" />
  </symbol>
</defs>
<use xlink:href="#smilie" transform="translate(-20,-20) scale(8)" />
<text x="79" y="119">S</text>
<text x="117" y="170">V</text>
<text x="156" y="119">G</text>
</svg>

</body>
</html>
```

**SVG Grafik als Verweisziel innerhalb eines HTML Dokuments: a-Tag.** Die SVG Grafik wird als Ziel des Hyperlinks im HTML-Tag <a> angegeben. Nach Mausklick des Anwenders auf den Hyperlink, wird die SVG Grafik im Browserfenster dargestellt.

Beispiel Quellcode

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html><head><title>SVG in HTML</title></head>
<body style="margin-left:20px; margin-top:10px;">
<p style="font-family:Verdana";>
SVG Grafik als Verweisziel - mit dem a-Tag realisiert<br>
wird von allen SVG-fähigen Browsern dargestellt</p>

<a href="dummy3.svg">SVG Grafik</a>

</body>
</html>
```

**Innerhalb eines HTML Dokuments wird die SVG-Grafik als Hintergrundbild eingebunden.** Die SVG Grafik wird von der CSS Eigenschaft background-image im <style>-Bereich des HTML Dokuments referenziert. Dadurch wird die Grafik als Hintergrundgrafik des HTML Dokuments dargestellt.

Beispiel Quellcode

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/transitional.dtd">
<html>
<head>
<title>SVG in HTML</title>

<style type="text/css">
<!--
body {background-image:url(dummy3.svg);
      background-attachment:fixed;
      background-position:40px 40px;
      background-repeat:no-repeat;}
-->
</style>

</head>
<body style="margin-left:20px; margin-top:10px;">
<p>SVG Grafik als Referenz einer Style-Eigenschaft (Hintergrundbild)<br>
wird noch nicht dargestellt</p>
</body>
</html>
```

## 2.2 Rendern von SVG-Grafiken

Dieses Kapitel ist ein wenig der grundlegenden Theorie gewidmet.

SVG verwendet ein Painters-Modell beim Rendern. Farbe (paint) wird in aufeinander folgenden Operationen am Ausgabegerät in der Weise aufgetragen, dass jede Operation einen Bereich des Ausgabegeräts übermalt. Wenn der Bereich einen bereits bemalten Bereich überlappt, überdeckt die neue Farbe die alte ganz oder teilweise. Wenn die Farbe nicht hundertprozentig deckend ist (falls Sie z.B. mit Hilfe der Eigenschaft ***opacity*** eine Transparenz definiert haben), wird das Ergebnis auf dem Ausgabegerät durch die (mathematischen) Regeln für die Mischung definiert, die mit Einfaches Alpha Blending bezeichnet werden.

Elemente in einem SVG-Dokument haben eine implizite Reihenfolge, in der sie gemalt werden.

1. Das erste Element in einem SVG-Dokumentfragment wird als erstes gemalt.
2. Darauffolgende Elemente werden auf die Oberfläche der zuvor gemalten Elemente aufgetragen.

Gruppierende Elemente wie das **g**-Element, mit dessen Hilfe mehrere Elemente zu einer Gruppe vereinigt werden können (siehe Dokumentstruktur II), produzieren einen temporären eingenständigen Canvas (rechteckiger Hintergrund), der in transparentem Schwarz initialisiert wird. Auf diesen werden dann die Kind-Elemente gemalt.

Einzelne Grafikelemente werden so gerendert, als sei jedes Grafikelement seine eigene Gruppe; im Klartext bedeutet das, dass für jedes Grafikelement ein eigener temporärer Canvas gebildet wird. Das Element wird zuerst auf den temporären Canvas gemalt. Dann werden alle Effekte angewendet (z.B. Transformationen oder Filter, etc.), die für das Grafikelement angegeben sind, um den veränderten temporären Canvas zu erzeugen. Der veränderte temporäre Canvas wird dann unter Berücksichtigung aller Einstellungen für Clipping, Masking und die Objektdeckfähigkeit des Elements in den Hintergrund des Gruppen-Canvas integriert.

Die 3 möglichen Grafikelementtypen:

- Formen (Kombination aus geraden Linien und Kurven, d.h. Pfaden)
- Text (Kombination aus Glyphen)
- Rasterbilder (Werte-Array, der die Farbe und die Deckfähigkeit (Alpha) einer Reihe von Bildpunkten in einem rechteckigen Gitter angibt. Auch als Pixel- oder Bitmap- Grafiken bezeichnet.)

Für bestimmte Formtypen können weiterhin Markersymbole (Muster, die selbst aus einer Kombination aus Formen, Text und Bilder bestehen können, wie z.B. Pfeilspitzen) auf ausgewählte Scheitelpunkte gemalt werden.

Formen und Text können gefüllt: ***fill***, dem Inneren der Form eine Farbe geben, oder gestrichen werden: ***stroke***, eine Randlinienfarbe erzeugen.

Unterstützung von SVG für ***fill*** und ***stroke***:

- Feste Farben
- Verläufe (linear und radial)
- Muster (Pattern)

SVG erlaubt, die Maloperation auf einen begrenzten Bereich innerhalb des Ausgabegeräts mit Hilfe von Ausschneiden und Maskieren zu beschränken.

# 3 Dokumentstruktur I

SVG-Grafiken sind XML-Dokumente.

XML - eXtensible Markup Language, ist eine Metasprache zur Definition von Auszeichnungssprachen. Mit Hilfe der Regeln aus XML haben die Entwickler des W3C - World Wide Web Consoritium die Auszeichnunssprache SVG - Scalable Vector Graphics entwickelt.

## 3.1 Das Dokument - Die Grafik

SVG Dokument und SVG Grafik bezeichnen dasselbe - eine Textdatei, die in der Auszeichnungssprache SVG erstellt wurde. Es liegt also ein SVG Textdokument vor, das eine Grafik beschreibt. Daher werden die Begriffe SVG Dokument und SVG Grafik in diesem Tutorial synonym behandelt.

Nachfolgend die wichtigsten Informationen zu einem SVG Dokument bzw. einer SVG-Grafik.

- Dateinamenerweiterung für SVG Grafiken: **.svg**
- MIME-Type einer SVG-Grafik: **image/svg+xml**
- SVG Namespace (Bezeichner für den eindeutigen Namensraum von SVG): **http://www.w3.org/2000/svg**
- Xlink Namespace (Bezeichner für den eindeutigen Namensraum von XLink): **http://www.w3.org/1999/xlink**
- Public Identifier for SVG 1.0 (Bezeichner für die Document Type Definition von SVG): **PUBLIC "-//W3C//DTD SVG 1.0//EN"**
- System Identifier for SVG 1.0 (URI der Document-Type-Definition von SVG): **http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd**

## 3.2 Grundgerüst einer SVG-Grafik

Der einzige Inhalt der folgenden Beispielgrafik ist ein Kommentar. Sie sehen also nichts, d.h. möglicherweise können sie, durch einen schwachen Farbunterschied, den Umriss der Grafik erkennen. Sie sehen also eine leere Grafik.

Wenn Sie mit der rechten Maustaste in diese (oder jede beliebige) SVG Grafik klicken, erscheint bei Verwendung des Adobe SVG Viewers ein spezielles **SVG Kontextmenü**, das folgende Möglichkeiten bietet.

- Einzoomen - Grafik vergrößert darstellen
- Auszoomen - Grafik verkleinert darstellen
- Orginalansicht - Grafik in Orginalansicht darstellen
- Bessere Qualität - Einstellung des User Agents bezüglich der Darstellung der Grafik
- Unterbrechen - für Animationsabläufe
- Ton aus - für Grafiken mit integriertem Sound
- Suchen - Text innerhalb der Grafik suchen
- Weitersuchen - Text innerhalb der Grafik weitersuchen
- SVG kopieren - Quellcode der SVG Grafik in die Zwischenablage kopieren
- SVG anzeigen - SVG Grafik in einem neuen Fenster anzeigen
- Quelle anzeigen - Quellcode der SVG Grafik in einem neuen Fenster anzeigen
- SVG speichern unter - Quellcode der SVG Grafik auf einem Datenträger speichern
- Hilfe - Eine weitergehende Hilfe zum Kontextmenü
- Über Adobe SVG Viewer - Versionsnummer und mehr

Beispiel Quellcode

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">

  <!-- .. ich bin ein Kommentar in einer leeren Grafik ... -->

</svg>
```

SVG Grafiken werden immer mit der **XML-Deklaration** eingeleitet, die das Dokument als XML-Dokument ausweist.

Die folgende Zeile des SVG-Grundgerüstes ist die **Dokument-Typ-Deklaration**. Eine Dokument-Typ-Definition (DTD) wird von den Erstellern einer auf XML basierenden Auszeichnungssprache als normatives Text-Dokument für diese Sprache festgelegt. In dieser DTD sind sowohl jedes Element und jedes Attribut , wie auch Verschachtelungsregeln der Elemente und weitere Eigenheiten der Auszeichnungssprache festgelegt.

Damit ein Anwendungsprogramm weiß, welche Regeln für ein bestimmtes XML-Dokument (in unserem Falle einem SVG-Dokument) gelten, muß angegeben werden, auf welche DTD es sich bezieht.

In der Dokument-Typ-Deklaration eines SVG-Dokuments wird zuerst der allgemeine Name, der Public Identifier:"-//W3C//DTD SVG 20010904//EN", und zusätzlich noch der reale Speicherort, der Document Identifier: "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" der Document Type Definition (DTD) angegeben.

Das anschließende **svg**-Element ist das eigentliche **Wurzelelement** einer SVG-Grafik bzw. eines SVG-Dokuments. Das Wurzelelement ist das Element, das in einer auf XML basierenden Sprache alle anderen Element einschließt.

Das Attribut ***xmlns*** mit dem Wert "http://www.w3.org/2000/svg" legt einen eindeutigen Namensraum (name space) für die Elemente der Auszeichnungssprache SVG fest, und muß immer in dieser Form notiert werden. Da es jeder Person möglich ist, mit Hilfe von XML Auszeichnungssprachen zu erstellen, könnte es möglich sein, dass Element- oder Attributnamen von SVG auch in anderen, auf XML

basierenden Auszeichnungssprachen vorkommen. Die Angabe eines eindeutigen Namensraums, in Form einer URL besagt, dass alle verwendeten Elemente eindeutig zur Auszeichnungssprache SVG gehören.

Durch das Attribut ***xmlns:xlink*** mit dem Wert "http://www.w3.org/1999/xlink" wird der Namensraum für xlink angegeben. Im Namensraum von xlink sind die SVG Elemente zur Referenzierung von Objekten definiert, wie z.B. Hyperlinks. Der Firefox Browser von Mozilla (Version >= 1.5Beta) benötigt diese Angabe um SVG Grafiken, die xlink-Element enthalten, korrekt darstellen zu können. Um Fehler zu vermeiden, ist es grundsätzlich eine gute Idee, den xlink Namensraums immer im **svg**-Element anzugeben.

Mit den Attributen ***width*** und ***height*** wird die Breite und Höhe der gesamten Grafik festgelegt. Als Werte sind Zahlen mit optionaler Maßangabe möglich. Wird keine Maßangabe angegeben, wird Pixel verwendet.

**Mögliche Maßangaben in SVG** sind:

- em - relative Maßangabe, bezieht sich auf die verwendete Schriftgröße
- ex - relative Maßangabe, bezieht sich auf die Größe des großen X der verwendeten Schrft
- px - Pixel, default-Einheit
- pt - Punkt
- pc - Pica
- cm - Centimeter
- mm - Millimeter
- in - Inch
- % - Prozentangabe

**Kommentare** werden, wie in HTML bzw. XML, mit der Zeichenfolge <!-- eingeleitet und enden mit der Zeichenfolge -->. Kommentare werden nicht am Bildschirm dargestellt.

Verwenden sie Kommentare großzügig, um den Quellcode Ihrer SVG-Grafik mit erläuterndern Anmerkungen zu versehen. Wenn Sie sich selbst erstellte, unkommentierte Quelltexte einige Zeit nach ihrer Erstellung wieder anschauen, werden Sie feststellen wie nützlich Kommentierungen gewesen wären ... ;-). Daher empfehle ich den ausgiebigen Gebrauch von Kommentaren in SVG-Quellcodes.

## 3.3 Verschachtelung von SVG-Elementen

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <rect x="0" y="0" width="200" height="200" style="fill:red;" />
  <svg x="10" y="10" width="100" height="100">
    <rect x="0" y="0" width="100" height="100" style="fill:blue;" />
  </svg>
</svg>
```

Das **svg**-Element kann weitere **svg**-Elemente beinhalten, d.h. Sie können innerhalb einer Grafik weitere Grafiken definieren.Um diese "internen" Grafiken korrekt zu platzieren, können sie, zusätzlich zu den Attributen ***width*** und ***height***, die Attribute ***x*** und ***y*** verwenden. Diese legen die x,y-Koordinate der linken, oberen Ecke der inneren Grafik fest.

Im obigen Beispiel wird eine Grafik der Größe 200 x 200 Pixel definiert.

In dieser Grafik ist ein rotes Rechteck derselben Größe definiert, d.h. es füllt die gesamte Grafik aus. Weiterhin beinhaltet die Grafik eine zweite, innere Grafik der Größe 100 x 100 Pixel. Der linke obere Eckpunkt dieser zweiten Grafik ist im Koordinatenpunkt 10,10 der äußeren Grafik definiert, d.h. sie wird jeweils 10 Pixel vom oberen und vom rechten Rand der ursprünglichen Grafik dargestellt.

Diese innere Grafik wird durch ein blaues Rechteck ausgefüllt. Beachten Sie, dass der linke obere Eckpunkt des blauen Rechtecks, mit den Koordinaten 0,0 bestimmt ist. Dieser Koordinatenpunkt wird relativ zur inneren Grafik interpretiert. Das blaue Rechteck wird also am Punkt 0,0 der inneren Grafik, gleichbedeutend mit dem Punkt 10,10 der äußeren Grafik, dargestellt.

Rechtecke sind Thema im Kapitel Grundformen.

## 3.4 Titel und Beschreibung einer SVG-Grafik

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Beispiel: desc und title</title>
  <desc>Ein simples schwarzes Rechteck</desc>
  <rect x="5" y="5" width="190" height="190" />
</svg>
```

Das **title**-Element wird verwendet, um der Grafik einen Titel zu geben. Auf welche Art User Agents den Inhalt dieses Elements interpretieren, ist allein die Entscheidung der Entwickler. So wird der Inhalt dieses Elements z.B. in der Titelleiste des Browsers angezeigt.

Der Inhalt des **desc**-Elements dient dazu, die Grafik zu beschreiben (desc = description = Beschreibung).

Der Inhalt dieses Elements wird nur von Browsern angezeigt, die SVG nicht unterstützen.Suchmaschinen können diese Information auslesen. Die Verwendung der Elemente **title** und **desc** ist vom W3C empfohlen.

## 3.5 Das Attribut viewBox

Grundsätzlich wird die Ausdehnung einer SVG Grafik durch die Attribute ***width*** und ***height*** im äusseren **svg**-Element festgelegt. Damit wird dann auch das Koordinatensystem der Grafik (Viewport) bestimmt, dass Sie zum Platzieren der Objekte verwenden müssen.

Durch den Gebrauch des Attributs ***viewBox*** - i.d.R. im **svg**-Element - können Sie innerhalb der SVG Grafik ein unterschiedliches Koordinatensystem verwenden. Dieses Koordinatensystem kann z.B. größer oder kleiner sein kann als das der "normalen" SVG Grafik.

Dazu wird mit Hilfe des Attributs ***viewBox*** ein Rechteck definiert. Dieses Rechteck bestimmt eine weitere, andere Ausdehnung für die SVG Grafik und somit auch ein unterschiedliches Koordinatensystem. In der SVG Grafik sind somit 2 Koordinatensysteme definiert.

- Zum einen das "normale" Koordinatensystem, bestimmt durch die Attribute ***width*** und ***height*** im **svg**-Element und
- zum anderen das "neue" Koordinatensystem, bestimmt durch das, mit dem Attribut ***viewBox*** definierte Rechteck. **!Zur Platzierung der Objekte innerhalb der SVG Grafik muss dieses neue Koordinatensystem verwendet werden!**

Das eigentliche Feature des Attributs ***viewBox*** besteht nun in der sogenannten **automatic transformation**. Diese bewirkt, dass ein so definiertes, "neues" Koordinatensystem beim Rendern vom User Agent automatisch auf die, im **svg**-Element durch ***width*** und ***height*** angegebene normale Grösse der Grafik (den "normalen" Viewport) skaliert wird.

Um den neuen Viewport der SVG Grafik festzulegen, werden dem Attribut ***viewBox*** 4 numerische Werte mitgegeben, wie z.B.: *0 0 200 100* die in der Reihenfolge von links nach rechts folgende Bedeutung haben:

1. x-Koordinate des Rechtecks, das den neuen Viewport festlegt
2. y-Koordinate des Rechtecks, das den neuen Viewport festlegt
3. Breite des Rechtecks, das den neuen Viewport festlegt
4. Höhe des Rechtecks, das den neuen Viewport festlegt

Die folgenden 3 SVG Beispiele sollen die Verwendung des Attributs ***viewBox*** im **svg**-Element verdeutlichen.

Im ersten "normalen" SVG Dokument wird das Attribut ***viewBox*** nicht verwendet. Das Rechteck wird in Originalgrösse angezeigt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<!-- das nomale Dokument - 200 Pixel breit, 100 Pixel hoch -->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="200" height="100">

<!-- das Rechteck - 160 Pixel breit, 60 Pixel hoch -->
  <rect x="20" y="20" width="160" height="60"
    fill="limegreen" stroke="black" stroke-width="2px" />
</svg>
```

Im folgenden SVG Dokument werden die Werte ***width*** und ***height*** im **svg**-Element so gewählt, dass die Grafik um die Hälfte verkleinert wird. Durch das Attribut ***viewBox*** wird ein Viewport definiert, der die Aussmasse des ersten Beispieldokuments hat. Die Koordinaten des Rechtecks werden nicht verändert, da nun das Koordinatensystem des neuen Viewport im Dokument gültig ist. Der User Agent stellt das Rechteck jetzt verkleinert dar, d.h. er skaliert den Inhalt des des virtuellen Viewports automatisch auf die "normale" Grösse des SVG-Dokuments.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<!-- das verkleinerte Dokument - 100 Pixel breit, 50 Pixel hoch -->
<!-- Verwendung von viewBox 200 Pixel breit, 50 Pixel hoch -->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="100" height="50"
  viewBox="0 0 200 100">

<!-- die Koordinaten und Aussmasse des Rechtecks haben sich nicht verändert -->
  <rect x="20" y="20" width="160" height="60"
    fill="limegreen" stroke="black" stroke-width="2px" />
</svg>
```

Im dritten SVG Dokument werden die Werte ***width*** und ***height*** im **svg**-Element wie im 2. Beispiel so gewählt, dass die Grafik um die Hälfte verkleinert wird. Jedoch wird das Attribut ***viewBox*** nicht verwendet. Der User Agent stellt nun lediglich einen Ausschnitt des Rechtecks dar.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<!-- das verkleinerte Dokument - 100 Pixel breit, 50 Pixel hoch -->
<!-- keine Verwendung von viewBox -->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="100" height="50">

<!-- die Koordinaten und Aussmasse des Rechtecks haben sich nicht verändert -->
  <rect x="20" y="20" width="160" height="60"
    fill="limegreen" stroke="black" stroke-width="2px" />
</svg>
```

Das Attribut ***viewBox*** kann ausser im **svg**-Element auch innerhalb weiterer Elemente, die einen neuen Viewport erzeugen, verwendet werden. Elemente, die dieses Kriterium erfüllen sind: **svg**, **symbol** (wenn es durch das Element **use** referenziert wird), **image** und **foreignObject** (siehe auch Kapitel Dokumentstruktur II).

## 3.6 Metadaten

Metadaten sind strukturierte Daten, die Daten beschreiben - oder kurz gesagt: Metadaten sind Informationen über Daten. Bei SVG Dokumenten sind Metadaten Informationen über die Grafik, wie z.B.: Titel, Beschreibung, Herausgeber, Autor, Erstellungsdatum, etc.

Zur standarisierten Strukturierung dieser Metadaten existieren 3 Ansätze die unter SVG miteinander verquickt werden:

- Resource Description Framework Model and Syntax Specification
- Resource Description Framework (RDF) Schema Specification
- Dublin Core

Zur näheren Information über Dublin Core in RDF empfehle ich folgende Links:

- Expressing Simple Dublin Core in RDF/XML
- Dublin Core Metadata Element Set, Version 1.1: Reference Description
- Expressing Qualified Dublin Core in RDF/XML

Um Metadaten festzulegen, muss in SVG Dokumenten das Element **metadata** verwendet werden. Die Elemente innerhalb des **metadata**-Bereichs müssen zu den Namensräumen von RDF und DC gehören, d.h. explizit nicht zum SVG-Namensraum.

Der folgende Quelltext zeigt ein Beispiel für "Expressing Simple Dublin Core in RDF/XML". Im Beispiel wurden die gebräuchlichsten Dublin Core Elemente für die Beschreibung der Metadaten verwendet. Eine Übersicht aller Elemente und der Syntax von RDF und DC finden Sie unter den oben genannten Links.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="238" height="60"
  xmlns:xlink="http://www.w3.org/1999/xlink">

<!-- die Metadaten -->
<metadata>
  <!-- Elemente aus dem RDF und DC Namesraum -->
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://svg.tutorial.aptico.de/grafik_svg/kap2_5.svg">
    <!-- Ersteller, Autor -->
      <dc:creator>Ralf.Pohlmann@aptico.de</dc:creator>
    <!-- Verantwortlicher -->
      <dc:contributor>Ralf.Pohlmann@aptico.de</dc:contributor>
    <!-- Herausgeber -->
      <dc:publisher>http://aptico.de</dc:publisher>
    <!-- Schlagwort, Schlüsselworte -->
      <dc:subject>SVG, Metadaten, RDF, Dublin Core, Beispielgrafik</dc:subject>
    <!-- Beschreibung des Dokuments -->
      <dc:description>Beispielgrafik zum Thema Metadaten in SVG</dc:description>
    <!-- URI des Dokuments -->
      <dc:identifier>http://svg.tutorial.aptico.de/grafik_svg/kap2_5.svg</dc:identifier>
    <!-- Bezieht sich auf -->
      <dc:relation>http://svg.tutorial.aptico.de/3.5.php?</dc:relation>
    <!-- Übergeordnete Quelle -->
      <dc:source>http://svg.tutorial.aptico.de/</dc:source>
    <!-- Rechte, Copyright -->
      <dc:rights>Copyright 2002-2004 Ralf Pohlmann</dc:rights>
    <!-- MIME-Type des Dokuments -->
      <dc:format>image/svg+xml</dc:format>
    <!-- Typ des Dokuments -->
      <dc:type>Image</dc:type>
```

```
    <!-- Titel des Dokuments -->
      <dc:title>Meta-Grafik</dc:title>
    <!-- Datum der Erstellung -->
      <dc:date>2004-06-14</dc:date>
    <!-- Verwendete Sprache -->
      <dc:language>de</dc:language>
    </rdf:Description>
  </rdf:RDF>
</metadata>

<!-- und ein bisschen Inhalt - Rechteck und Text -->
<rect x="0" y="0"
  width="238"
  height="60"
  fill="yellow"/>
<text x="6" y="43"
  font-family="Verdana,sans-serif"
  font-size="36px"
  font-weight="bold"
  fill="yellow">
  METADATA
</text>
</svg>
```

# 4 Grundformen

SVG stellt 6 Elemente zur Darstellung von geometrischen Grundformen zur Verfügung.

- **rect**-Element  Rechteck
- **circle**-Element  Kreis
- **ellipse**-Element  Ellipse
- **line**-Element  Linie
- **polyline**-Element  Polylinie (eine Linie, die aus mehreren Linien besteht).
- **polygon**-Element  Polygon (Vieleck)

Diesen Elementen müssen Attribute zugeordnet werden, die die Platzierung der Form innerhalb der Grafik und die Ausdehnung der Form festlegen. Außerdem stellt SVG, durch den Einsatz von festgelegten Attributen oder CSS-Style-Sheets, Möglichkeiten zur Verfügung, den Formen Füllungen und Linienfarben zuzuordnen.

## 4.1 Rechtecke - das rect-Element

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="410" height="110" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das rect-Element</title>
  <desc>9 (8 sichtbare) Rechtecke</desc>

  <!-- vier Rechtecke in der ersten Zeile -->
  <rect x="10" y="10" width="90" height="40" />
  <!-- das folgende Rechteck wird nicht dargestellt -->
  <rect x="110" y="10" width="90" height="40"
    fill="none" />
  <rect x="210" y="10" width="90" height="40" rx="5" ry="10"
    fill="none" stroke="black" />
  <rect x="310" y="10" width="90" height="40" rx="5"
    fill="none" stroke="black" />

  <!-- die vier Rechtecke in der zweiten Zeile -->
  <rect x="10" y="60" width="90" height="40"
    fill="#ff0000" />
  <rect x="110" y="60" width="90" height="40" ry="5"
    fill="blue" stroke="black" />
  <rect x="210" y="60" width="90" height="40"
    fill="red" stroke="blue" stroke-width="4" />
  <rect x="310" y="60" width="90" height="40"
    style="fill:#33cc33; stroke:rgb(0,0,0);" />

  <!-- das Rechteck, welches die gesamte Grafik umrahmt -->
  <rect x="1" y="1" width="408" height="108"
    fill="none" stroke="blue" />
</svg>
```

Ein Rechteck wird mit dem **rect**-Element erzeugt. Dieses ist ein Element, das in der Regel ohne ein schließendes </rect>-Tag verwendet werden muß, da es keinen Inhalt besitzt.

Nach den Regeln von XML sind Elemente ohne schließendes Tag nicht zulässig. Die Regeln von XML schreiben für Elemente ohne Inhalt folgende Syntax vor, bei der der Schrägstrich direkt vor der schließenden spitzen Klammer des Tags eingefügt wird: <rect />

Um ein Rechteck zu definieren, sind mindestens 2 Attribute im **rect**-Element zu platzieren. Das Attribut ***width*** legt die Breite des Rechtecks fest, das Attribut ***height*** die Höhe. Mit diesen zwei Angaben kann das Rechteck von der Anwendungssoftware (dem SVG Viewer) konstruiert bzw. gezeichnet werden. Mit den Attributen ***x*** und ***y*** legen Sie, wie auch im svg-Element, den Koordinatenpunkt für die linke, obere Ecke des Rechtecks fest. Wenn Sie diese Attribute nicht verwenden, wird die Voreinstellung 0 verwendet, d.h. die linke obere Ecke des Rechtecks liegt dann auf dem Koordinatenpunkt (0,0).

Das Attribut ***fill*** erwartet eine RGB-Farbangabe oder das Schlüsselwort *none*, und legt die Füllfarbe für das Element fest. Voreingestellter Wert für das Attribut ***fill*** ist *black*. Da standardmäßig keine Rahmenlinie der Form angezeigt wird und dem Attribut ***fill*** der Wert *none* zugewiesen ist wird das zweite Rechteck der obigen Beispielgrafik nicht angezeigt.

RGB-Farbangaben können in SVG entweder durch eine festgelegtes Farbwort (*black*, *white*, *green*, ...), durch eine Raute gefolgt vom Farbwert in hexadezimaler Darstellung (z.B. #ff0000 für rot) oder durch eine dezimale Farbangabe (rgb(255,0,0) ebenfalls für rot) definiert werden. Eine übersicht aller Farbworte und -werte finden Sie in Stefan Münz's SelfHTML.

Die Rahmenlinie wird durch das Attribut ***stroke*** erzeugt, das ebenfalls eine Farbangabe oder das Schlüsselwort *none* als Wert erwartet. Voreingestellter Wert ist *none*. Beachten Sie: Rahmenlinien eines

Elements werden also nur dargestellt, wenn Sie dem Element das Attribut ***stroke*** mit einer Farbangabe zuordnen.

Das Attribut ***stroke-width*** bietet Ihnen die Möglichkeit, die Stärke der Rahmenlinie zu bestimmen. Voreingesteller Wert für stroke-width ist *1px*.

Die voreingestellte Maßangabe in einem SVG-Dokument ist px (Pixel). In unserem Beispiel wurde bei der Bestimmung der Rahmenlinienstärke keine Maßangabe angegeben. Die Rahmenlinie des vorletzten Rechtecks in der obigen Beispielgrafik wird also in einer Stärke von 4 Pixeln dargestellt.

Umfassende Informationen zu Füllungen und Rahmenlinien finden Sie im Kapitel Painting.

Rechtecke können mit abgerundeten Ecken dargestellt werden. Die Radien für die Abrundung, die sich auf die x-Achse (Breitenlinie) und die y-Achse (Höhenlinie) des Rechtecks beziehen, werden durch die Attribute ***rx*** und ***ry*** festgelegt. Wenn Sie nur ***rx*** oder nur ***ry*** verwenden werden beide Achsen um den zugewiesenen Wert abgerundet.

Eine weitere Möglichkeit Angaben zur Darstellung eines Elements festzulegen, ist die Verwendung von CSS (Cascading Stylesheets) einer Sprache zur Definition von **Formatvorlagen** für Elemente von Auszeichnungssprachen. Die CSS-Angaben werden dem Attribut ***style*** als Wert zugewiesen. SVG unterstützt die Angaben aus CSS 2 (ebenfalls ein W3C Standard). Außerdem können eine Vielzahl von SVG-Attributen auch als Style-Vorschrift verwendet werden, wie z.B. fill oder stroke.

In den Beispielen dieses Buches werden häufig CSS-Angaben zur Formatierung für Elemente verwendet, da hierdurch die Trennung von Element und Formatierung möglich ist. Außerdem können CSS-Angaben auch global definiert werden. Siehe Kapitel 4 Styling.

Weitere Informationen zu CSS finden Sie in Stefan Münz's SelfHTML oder bei www.sur.ping.de

## 4.2 Kreise - das circle-Element

Um Kreise konstruieren bzw. am Bildschirm darstellen zu können, benötigt die anzeigende Applikation, die x,y-Koordinate des Kreismittelpunkts und die Länge des Radius. Für die Festlegung des x,y-Koordinate stellt das **circle**-Element die Attribute ***cx*** und ***cy*** zur Verfügung, der Radius wird mit dem Attribut ***r*** bestimmt.

**Beachten Sie, dass Elemente, die später im SVG-Quellcode definiert werden, zuerst festgelegt Elemente überdecken. Dies gilt für alle Elemente in SVG Grafiken.**

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="410" height="210" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das circle-Element</title>
  <desc>ein Kopf aus Kreisen</desc>
    <!-- 2 schwarze Ohren -->
    <circle cx="150" cy="50" r="40" fill="black" />
    <circle cx="260" cy="50" r="40" fill="black" />

    <!-- der Kopf -->
    <circle cx="205" cy="100" r="80" fill="yellow" stroke="black" />

    <!-- die Augen -->
    <circle cx="180" cy="67" r="14" fill="white" stroke="black" />
    <circle cx="230" cy="67" r="14" fill="white" stroke="black" />
    <circle cx="180" cy="70" r="10" fill="black" />
    <circle cx="230" cy="70" r="10" fill="black" />

    <!-- die Nase -->
    <circle cx="205" cy="90" r="15" fill="red" />

    <!-- der Mund -->
    <circle cx="205" cy="140" r="17" fill="red" stroke="black" />
    <circle cx="205" cy="130" r="17" fill="yellow" />
</svg>
```

## 4.3 Ellipsen - das ellipse-Element

Bei einer Ellipse müssen, außer der x,y-Koordinate des Mittelpunkts, zwei Radien festgelegt werden. Den Radius der x-Achse und den Radius der y-Achse der Ellipse.

Folgende Attribute stehen dem **ellipse**-Element für diese Angaben zur Verfügung: ***cx*** und ***cy*** zur Bestimmung der Mittelpunktkoordinate (wie beim circle-Element) und ***rx*** für den x-Radius, sowie ***ry*** für den y-Radius.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="410" height="210" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das ellipse-Element</title>
  <desc>4 Ellipsen und ein Kreis</desc>
  <!-- 4 farbige Ellipsen -->
  <ellipse cx="200" cy="50" rx="20" ry="40"
    style="fill:green;" />
  <ellipse cx="250" cy="100" rx="40" ry="20"
    style="fill:blue;" />
  <ellipse cx="200" cy="150" rx="20" ry="40"
    style="fill:yellow;" />
  <ellipse cx="150" cy="100" rx="40" ry="20"
    style="fill:red;" />

  <!-- der Kreis in der Mitte -->
  <circle cx="200" cy="100" r="19"
    style="fill:black; stroke:white; stroke-width:3px" />
</svg>
```

## 4.4 Linien - das line-Element

Eine Linie besteht aus 2 Punkten ;-). Daher müssen Sie im **line**-Element mit Hilfe der Attribute ***x1*** und ***y1*** den ersten Koordinatenpunkt (Startpunkt) der Linie und mit Hilfe der Attribute ***x2*** und ***y2*** den zweiten Koordinatenpunkt der Linie (Endpunkt) bestimmen.

Da der Standardwert des Attributes ***stroke***, welches die Linienfarbe eines SVG-Elements festlegt, *none* ist, muß einer Linie mittels ***stroke*** (Attribute oder CSS-Eigenschaft) eine Farbe zugewiesen werden, damit sie dargestellt werden kann.

Füllungen sind für eine Linie logischerweise nicht möglich.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das line-Element</title>
  <desc>nur Linien</desc>
  <!-- schwarze Linien -->
  <line x1="80" y1="160" x2="320" y2="160"
    style="stroke:black; stroke-width:2px;" />
  <line x1="90" y1="150" x2="310" y2="150"
    style="stroke:black; stroke-width:2px;" />
  <line x1="100" y1="140" x2="300" y2="140"
    style="stroke:black; stroke-width:2px;" />
  <line x1="110" y1="130" x2="290" y2="130"
    style="stroke:black; stroke-width:2px;" />
  <line x1="120" y1="120" x2="280" y2="120"
    style="stroke:black; stroke-width:2px;" />
  <line x1="130" y1="110" x2="270" y2="110"
    style="stroke:black; stroke-width:2px;" />
  <line x1="140" y1="100" x2="260" y2="100"
    style="stroke:black; stroke-width:2px;" />
  <line x1="150" y1="90" x2="250" y2="90"
    style="stroke:black; stroke-width:2px;" />
  <line x1="160" y1="80" x2="240" y2="80"
    style="stroke:black; stroke-width:2px;" />
  <line x1="170" y1="70" x2="230" y2="70"
    style="stroke:black; stroke-width:2px;" />
  <line x1="180" y1="60" x2="220" y2="60"
    style="stroke:black; stroke-width:2px;" />
  <line x1="190" y1="50" x2="210" y2="50"
    style="stroke:black; stroke-width:2px;" />

  <!-- rote Linien -->
  <line x1="40" y1="165" x2="180" y2="30"
    style="stroke:red; stroke-width:2px;" />
  <line x1="360" y1="165" x2="220" y2="30"
    style="stroke:red; stroke-width:2px;" />
</svg>
```

## 4.5 Polylinien - das polyline-Element

Das **polyline**-Element erzeugt eine Linie durch mehere Koordinatenpunkte.

Die einzelnen Koordinatenpunkte werden nacheinander im Attribut ***points*** angegeben. Dabei müssen die einzelnen Zahlen mindestens durch ein Leerzeichen oder ein Komma voneinander getrennt werden. An welcher Stelle sie Leerzeichen oder Kommas verwenden ist Ihnen überlassen. Aus Gründen besserer Lesbarkeit empfiehlt es sich allerdings, nach jedem Koordinatenpaar ein Komma in die Liste der Zahlen einzufügen.

Weiterhin müssen Sie für das **polyline**-Element mit dem Attribut oder der Eigenschaft ***stroke*** eine Linienfarbe festlegen, da die Linie sonst nicht angezeigt wird.

Beachten Sie: Es ist in SVG möglich Polylinien zu füllen (siehe Beispiel). Von der Linie eingeschlossene Bereiche werden als innen betrachtet und können gefüllt werden. Um eine Polylinie als wirkliche Linie darzustellen, müssen Sie zusätzlich mit dem Attribut oder der Eigenschaft ***fill*** und dem Wert *none* dafür sorgen, dass der Polylinie keine Füllung zugewiesen wird - denn Standardwert für ***fill*** ist *black*.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="410" height="210" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das polyline-Element</title>
  <desc>eine Polylinie mit Füllung</desc>

  <!-- eine gefüllte (!) Polylinie -->
  <polyline fill="lightgray" stroke="red" stroke-width="5px"
    points="400 10, 120 10, 200 80, 280 20, 300 20
          220 100, 300 180, 280 180, 200 120, 120 180, 100 180
          180 100, 80 10, 10 10, 10 200, 400 200" />
</svg>
```

## 4.6 Polygone - das polygon-Element

Mit dem **polygon**-Element können Sie ein Vieleck erzeugen.

Die einzelnen Punkte eines **polygon**-Elementes, werden durch Angabe der Koordinatenpunkte (Eckpunkte des Vielecks) in korrekter Reihenfolge bestimmt. Wie im **polyline**-Element ist auch hier das Attribut ***points*** zu verwenden.

Der eigentliche Unterschied zwischen einem Polygon und einer Polyline besteht darin, dass die darstellende Anwendung bei einem Polygon den letzten angegebenen Koordinatenpunkt automatisch mit dem ersten angegeben Koordinatenpunkt durch eine weitere Linie verbindet.

Im obigen Beispiel wird ein Dreieck mittels des **polygon**-Elements erzeugt. Dazu ist lediglich die Angabe von 3 Koordinatenpunkten notwendig, da die letzte Linie vom user agent erzeugt wird. Das zweite Polygon im Beispiel definiert einen großen Stern. Durch die Verwendung des Attributs ***transform*** mit dem Wert *scale(.5)* wird dieses Polygon um die Hälfte seiner eigentlichen Größe verkleinert dargestellt - kurze Vorschau auf das Kapitel Transformationen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das polygon-Element</title>
  <desc>ein Dreieck und ein Stern</desc>

  <!-- ein Dreieck -->
  <polygon fill="darkblue"
    points="176 10, 26 160, 326 160" />

  <!-- ein (verkleinerter) Stern -->
  <polygon fill="yellow" stroke="none"
    points="350 75, 379 161, 469 161, 397 215,
          423 301, 350 250, 277 301, 303 215,
          231 161, 321 161"
    transform="scale(.5)" />
</svg>
```

# 5 Styling

In den bisher vorgestellten Beispielen wurden Füll- und Linienfarben der Elemente entweder mit Hilfe von Attributen oder durch die Angabe von CSS-Eigenschaften festgelegt. In diesem Kapitel werden wir uns nun ausführlicher mit der Formatierung der Elemente durch CSS-Eigenschaften, dem Styling, beschäftigen.

Folgende 4 Gründe sprechen für die Verwendung von Style-Vorschriften (Formatvorschriften, Style-Sheets), anstelle der Verwendung von Attributen, für die Formatierung von Elementen.

1. Durch Style-Vorschriften werden die Anweisungen für die Formatierung eines Elementes klar von den Anweisungen der eigentlichen Darstellung getrennt.
2. Style-Vorschriften bieten mehr Formatierungsmöglichkeiten als Attribute.
3. Style-Vorschriften können global festgelegt werden, d.h. Sie können zum Beispiel dafür sorgen, dass alle Rechtecke immer rot gefüllt dargestellt werden, indem Sie zu Beginn der SVG-Grafik eine Style-Vorschrift definieren.
4. Style-Vorschriften können in eine separaten Datei ausgelagert werden, die dann von mehreren beliebigen SVG-Grafiken verwendet werden kann. Durch eine Anweisung zu Beginn eines SVG-Dokuments kann eine Style-Datei unkompliziert in dieses SVG-Dokument eingebunden werden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das style-Attribut</title>
  <desc>Stylevorschriften mit CSS direkt im Element</desc>

  <!-- zwei gleich angemalte Rechtecke -->
  <rect x="50" y="50" width="125" height="100"
    style="fill:yellow; stroke:black; stroke-width:.1cm;" />
  <rect x="225" y="50" width="125" height="100"
    style="fill:yellow; stroke:black; stroke-width:.1cm;" />

  <!-- ein überlappender Kreis mit Transparenz -->
  <circle cx="200" cy="100" r="95"
    style="fill:blue; stroke:black; opacity:.4;" />
</svg>
```

Jedes Element im obigen Beispiel wird mit CSS-Eigenschaften formatiert, die mit Hilfe des Attibuts ***style*** direkt für das jeweilige Element festgelegt werden.

Die beiden Rechtecke werden gelb gefüllt und mit einer schwarzen Randlinie der Stärke von 0,1 cm am Bildschirm dargestellt. Sie erhalten also dieselbe Formatierung. Der Kreis, der die beiden Rechtecke überlappt, erhält eine blaue Füllfarbe und eine schwarze Randline der voreingestellten Stärke (1px). Außerdem wird für den Kreis mit der Eigenschaft ***opacity*** eine Transparenz von 0,4 festgelegt, so dass die überdeckten Teile der beiden Rechtecke durchscheinen.

Die **Transparenz** oder Durchsichtigkeit eines Elements wird mit dem Attribut oder der Eigenschaft ***opacity*** in einer Skala von *0 bis 1* eingestellt, wobei 0 "völlig durchsichtig" und 1 "keine Transparenz" bedeutet. Siehe auch Kapitel Painting.

## 5.1 Globale Formatierung in der Grafik

Das folgende SVG Beispiel bewirkt die gleiche Ausgabe, wie das vorhergegangene, jedoch werden die CSS-Eigenschaften global, zu Beginn der Grafik festgelegt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das defs-Element und das style-Element</title>
  <desc>Stylevorschriften mit CSS zu Begin des Dokuments</desc>

  <!-- Bereich für Definitionen: Style-Vorschrifen, Symbole, etc. -->
  <defs>
    <style type="text/css">
    <![CDATA[
      rect {fill:yellow; stroke:black; stroke-width:.1cm;}
      .blauTransparent {fill:blue; stroke:black; opacity:.4;}
    ]]>
    </style>
  </defs>

  <!-- zwei gleich angemalte Rechtecke -->
  <rect x="50" y="50" width="125" height="100" />
  <rect x="225" y="50" width="125" height="100" />

  <!-- ein überlappender Kreis mit Transparenz -->
  <circle cx="200" cy="100" r="95"
    class="blauTransparent" />
</svg>
```

Zuerst muß mit dem **defs**-Element am Anfang des SVG Dokuments ein Container (Bereich) definiert werden. Innerhalb dieses Bereichs können unter anderem folgende Definitionen vorgenommen werden:

- Scriptanweisungen, siehe Scripting,
- Filter, siehe Filtereffekte,
- Gradienten, siehe Farbverläufe und Muster,
- Symbole, siehe Dokumentstruktur II, und natürlich
- Stylevorschriften.

Weitere Informationen zum Element **defs** finden Sie im Kapitel Dokumentstruktur II.

Stylevorschriften werden innerhalb des **defs**-Container mit dem **style**-Element eingeleitet. Mit dem Attribut ***type*** muß der MIME-Typ der verwendeten Stylesprache angegeben werden. Die in unserm Falle verwendete Stylesprache CSS (Cascading Stylesheets) wird durch den MIME-Type *text/css* eindeutig identifiziert.

Vor der eigentlichen Notierung der Style-Vorschriften muß allerdings noch ein **CDATA**-Bereich definiert werden. Laut XML Spezifikation werden Daten innerhalb eines CDATA-Bereichs von der interpretierenden Anwendungssoftware nicht als XML-Code interpretiert, d.h. der SVG-Viewer versucht gar nicht erst, diesen Teil des Quellcodes am Bildschirm darzustellen.

Die erste Style-Vorschrift sorgt dafür, dass alle **rect**-Elemente innerhalb der Grafik gelb gefüllt und mit einer 4 Pixel breiten, schwarzen Randlinie dargestellt werden. Die zweite Style-Vorschrift definiert eine sogenannte allgemeine Unterklasse des Namens *blauTransparent*, mit folgenden Formatvorschriften: blaue Füllfarbe, schwarze Randline, eine Transparenz von 0.4. Mit Hilfe des Attributs ***class***, dem als Wert den Namen der Unterklasse zugeordnet wird, können die Formatierungen dieser Unterklasse einem Element zugeordnet werden.

Der Sprachumfang von Cascading Style Sheets ist nicht Thema dieses Tutorials. Ich verweise auf Informationen zu CSS bei Stefan Münz's SelfHTML, www.sur.ping.de oder natülich beim W3C.

## 5.2 Globale Formatierung durch externe Style-Datei

Einer der schönsten Vorteile bei der Verwendung von CSS-Formatvorlagen bietet die Möglichkeit, Style-Vorschriften in eine externe Datei auszulagern. Diese Datei kann auf diese Weise von beliebigen SVG Dokumenten verwendet werden.

Sie können Stylevorschriften einfach als Textdatei mit der Dateinamenerweiterung .css abspeichern. Im folgenden Beispiel beinhaltet diese Textdatei nur 2 Zeilen, nämlich die Stylevorschrift für das **rect**-Element und die allgemeine Unterklasse *blauTransparent*.

```
rect {fill:yellow; stroke:black; stroke-width:.1cm;}    .blauTransparent {fill:blue; stroke:black; opacity:.4;}
```

Dieser Datei wurde im Beispiel der Namen "kap4_2.css" gegeben.

Die so erzeugte CSS-Datei "kap4_2.css" wird direkt nach der einleitenden XML-Deklaration durch die XML-Anweisung <?xml-stylesheet ?> in das SVG-Dokument eingebunden.

Innerhalb dieser Anweisung wird mit Hilfe des Attributs ***type*** der MIME-Type der exteren Style-Datei (text/css) angebegen. Die Datei selbst wird durch das Attribut ***href***, das als Wert den Speicherort der Datei d.h eine URI enthält, referenziert. In unserem Fall befindet sich die CSS-Datei im gleichen Verzeichnis wie die SVG Grafik, daher muß lediglich der Name der CSS-Datei als Wert für ***href*** angegeben werden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<?xml-stylesheet type="text/css" href="kap4_2.css" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="200" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>die xml-stylesheet Anweisung</title>
  <desc>Stylevorschriften mit CSS in einer externen Datei</desc>

  <!-- zwei gleich angemalte Rechtecke -->
  <rect x="50" y="50" width="125" height="100" />
  <rect x="225" y="50" width="125" height="100" />

  <!-- ein überlappender Kreis mit Transparenz -->
  <circle cx="200" cy="100" r="95"
    class="blauTransparent" />
</svg>
```

# 6 Hyperlinks

In einer SVG-Grafik können Sie jedes Element als Hyperlink festlegen, d.h. Sie können jedes beliebige Element klickbar machen, um zum Beispiel eine andere SVG-Grafik oder ein beliebiges HTML/XHTML-Dokument im Browserfenster zu laden.

## 6.1 Besonderheiten des a-Elements

Um Hyperlinks zu markieren ist, ähnlich wie in HTML-Dokumenten, das **a**-Element zu verwenden. SVG verwendet die Spezifikation Xlink um Hyperlinks zu definieren. Dabei sind einfache (simple) Links möglich.

Nachfolgend die wichtigsten Attribute für das a-Element in SVG-Grafiken:

- ***xlink:href***="*URI*" Lokalisierung der anderen Ressource (die geladen werden soll)
- ***xlink:title***="*Zeichenkette*" eine frei wählbare Zeichenfolge zur Betitelung des Hyperlinks
- ***xlink:show***="*new*|*replace*" legt fest, ob die andere Ressource in einem neuen Browserfenster (new) oder im aktuellen Browserfenster (replace) geladen werden soll.
- ***target***="*Framebezeichnung*" Legt das Zielfenster (Zielframe) fest, in der die andere Ressource geladen werden soll.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="320px" height="170px" xmlns="http://www.w3.org/2000/svg"
 xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>das a-Element</title>
 <desc>Verweise in SVG-Grafiken</desc>
 <defs>
  <style type="text/css"><![CDATA[
   text {font-family:verdana; font-size:16px;}
  ]]>
  </style>
 </defs>

 <a xlink:href="webseite.html"
   xlink:title="Verweis zu einer HTML-Seite">
  <rect x="10" y="10" ry="5" width="40" height="40"
  style="fill:limegreen; stroke:black;" />
 </a>

 <a xlink:href="grafik.svg"
   xlink:title="Verweis zu einer SVG-Grafik">
  <rect x="10" y="60" ry="5" width="40" height="40"
  style="fill:mintcream; stroke:black;" />
 </a>

 <a xlink:href="http://www.w3.org/"
   xlink:title="Verweis zu www.w3.org"
   xlink:show="new">
  <rect x="10" y="110" ry="5" width="40" height="40"
  style="fill:chartreuse; stroke:black;" />
 </a>

 <text x="60" y="40">Verweis zu einer HTML-Seite</text>
 <text x="60" y="90">Verweis zu einer SVG-Grafik</text>
 <text x="60" y="140">Verweis zu www.w3.org</text>
</svg>
```

Im obigen Beispiel funktionieren 3 Rechtecke als Hyperlinks.

1. Klickt der User auf das erste, grüne Rechteck wird eine HTML-Seite des Namens webseite.html (die sich im selben Verzeichnis befindet, in der auch die Grafik liegt) im aktuellen Browserfenster geladen.
2. Beim Klick auf das zweite, blaue Rechteck wird eine andere SVG-Grafik des Namens grafik.svg im aktuellen Browserfenster geladen.
3. Das dritte, rote Rechteck verweist auf die Startseite des W3C im Internet. Diese Seite wird in einem neuen Browserfenster geladen.

## 7 Dokumentstruktur II

Begriffserläuterung: Unter einem SVG Objekt versteht man eine grafische Einheit, die aus einem oder mehreren Elementen bestehen kann.

In einer SVG Grafik können Sie Elemente bzw. Objekte wie Gruppen oder Symbole, mehrmals verwenden, d.h. ein bereits vorhandenes Element aus einem anderen Element heraus referenzieren.

Voraussetzung dafür, dass ein Objekt referenziert werden kann, ist eine eindeutige Namensvergabe für dieses Objekt mit dem ***id***-Attribut.

Weiterhin können Sie grafische Operationen (ebenfalls SVG-Elemente), wie z.B. Farbverläufe oder Filtereffekte, global definieren und diese dann als Eigenschaft für ein Element referenzieren.

Referenzen werden entweder mit dem Attribut ***xlink:href***="*#id*", oder Wert für eine Eigenschaft oder einem Attribut *url(#id)* eingebunden.

## 7.1 Elemente gruppieren - das g-Element

In SVG können Elemente gruppiert werden, d.h. Sie können Elemente mit Hilfe des **g**-Elements zu einer Gruppe zusammenschließen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="150px" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das g-Element</title>
  <desc>Gruppierungen festlegen</desc>
  <rect x="10" y="20" width="120" height="110" />

  <g stroke="red">
    <rect x="140" y="20" width="120" height="110"
      rx="10" ry="5"
      fill="none" stroke-width="1.5" />
    <rect x="270" y="20" width="120" height="110"
      style="fill:#efefef;" />
  </g>
</svg>
```

Das obige Beispiel zeigt 3 Rechtecke. Die hinteren beiden Rechtecke sind mit Hilfe des **g**-Elements gruppiert, d.h. sie bilden jetzt eine Einheit oder Gruppe.
Für die Elemente dieser Gruppe können die verschiedensten Eigenschaften auch gemeinsam festgelegt werden. Dazu werden dem **g**-Element die entsprechenden Attribute oder Eigenschaften zugeordnet.
Im Beispiel wurde dem **g**-Element das Attribut ***stroke*** mit dem Wert *red* zugeordnet, so dass die letzten beiden Rechtecke nun mit einer roten Linienfarbe dargestellt werden.
Gruppierungen sind ebenfalls sehr hilfreich für die Transformation von mehreren Elementen. So können Sie gruppierte Elemente, gemeinsam verschieben, skalieren, rotieren oder anderweitig transformieren. Weiterhin kann eine, mit dem Element **g** festgelegte, Gruppe von Elementen auch animiert werden.
Ein **g**-Element kann weitere **g**-Elemente enthalten, so dass es möglich ist eine komplexe Zeichnung in Gruppen zu strukturieren. Im folgenden ein einfaches Beispiel für die Verwendung von benannten Gruppen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="600" height="300" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das g-Element</title>
  <desc>Grundformen zu einem auto gruppiert</desc>

  <!-- Definition der Gruppe Auto -->
  <g id="auto">
    <rect x="55" y="180" width="195" height="45"
      ry="15" fill="red" />
    <rect x="55" y="200" width="195"      height="45"
      fill="red" />
    <rect x="100" y="132" width="100"      height="60"
      ry="15" fill="red" />
    <!-- Radkästen und Reifen -->
    <circle cx="85" cy="250" r="21"
      fill="white" />
    <circle cx="205" cy="250" r="21"
      fill="white" />
    <g id="rad_hinten">
      <circle cx="85" cy="250" r="18"
        fill="black" />
```

```
    <circle cx="85" cy="250" r="8"
      fill="grey" />
    <circle cx="75" cy="250" r="2"
      fill="red" />
  </g>
  <g id="rad_vorn">
    <circle cx="205" cy="250" r="18"
      fill="black" />
    <circle cx="205" cy="250" r="8"
      fill="grey" />
    <circle cx="195" cy="250" r="2"
      fill="red" />
  </g>
  <!-- Stoßstange Türen Scheiben -->
  <rect x="51" y="235" width="15"
    height="12" ry="15" fill="grey" />
  <rect x="222" y="235" width="32"
    height="12" ry="15" fill="grey" />
  <polygon fill="red" points="65 181, 106 135, 102 181" />
  <line x1="68" y1="180" x2="200" y2="180"
    style="stroke:grey; stroke-width:2px;" />
  <line x1="145" y1="132" x2="145" y2="245"
    style="stroke:grey; stroke-width:2px;" />
  <rect x="102" y="185" width="15"
    height="12" ry="15" fill="grey" />
  <line x1="103" y1="191" x2="116" y2="191"
    style="stroke:black; stroke-width:4px;" />
  <rect x="150" y="185" width="15"
    height="12" ry="15" fill="grey" />
  <line x1="151" y1="191" x2="164" y2="191"
    style="stroke:black; stroke-width:4px;" />
  <rect x="108" y="139" width="30"
    height="35" ry="8" fill="grey" />
  <rect x="152" y="139" width="39"
    height="35" ry="8" fill="grey" />
  <!-- Beleuchtung -->
  <rect x="54" y="185" width="10" height="20"
    ry="20" fill="firebrick" />
  <rect x="54" y="200" width="10" height="5"
    ry="2" fill="darkorange" />
  <rect x="243" y="210" width="10" height="5"
    ry="2" fill="darkorange" />
  <circle cx="248" cy="200" r="8"
    fill="gold" />
  <rect x="240" y="192" width="10" height="17"
    fill="red" />
</g>

<!-- die Strasse -->
<line x1="0" y1="268" x2="600" y2="268" stroke="black" />

<!-- hier beginnt die Animation -->
<animateTransform xlink:href="#rad_hinten"
  type="rotate" attributeName="transform"
  begin="0s" dur="2s"
  from="0 85 250" to="360 85 250"
  repeatCount="7" />
<animateTransform xlink:href="#rad_vorn"
  type="rotate" attributeName="transform"
  begin="0s" dur="2s"
  from="0 205 250" to="360 205 250"
  repeatCount="7" />
<animateTransform xlink:href="#auto"
  type="translate" attributeName="transform"
```

```
    begin="0s" dur="10s"
    from="-250" to="600" />
  <animateTransform xlink:href="#auto"
    type="translate" attributeName="transform"
    begin="10s" dur="4s"
    from="-250" to="50"
    fill="freeze" />
</svg>
```

Im obigen Beispiel werden verschiedenen Grundformen, die gemeinsam ein Auto darstellen, mit dem **g**-Element zu einer Gruppe zusammengeschlossen. Innerhalb dieser Gruppe, die durch das Attribut ***id*** eindeutig in *auto* benannt ist, sind zwei weitere Gruppen definiert, deren Elemente jeweils einen Reifen darstellen und ebenfalls durch ***id*** eindeutig in *rad_hinten* und *rad_vorn* benannt sind.

Das Attribut ***id*** gehört zu den allgemeinen Attributen und kann jedem Element einer SVG-Grafik zugeordnet werden, um dieses eindeutig zu identifizieren. Es ist also möglich beliebigen Elementen oder Objekten einen einmaligen Namen zu geben. Dabei ist zu beachten, das vergebene Werte für das Attribut ***id*** im Dokument auch wirklich einmalig sind, d.h. keine Zeichenfolge darf mehr als einmal als Wert für ***id*** verwendet werden.

Eindeutig benannte Elemente können von anderen Elementen (z.B. Elemente zur Animation) referenziert werden.

In der obigen Grafik wird jeder Reifen und das gesamte Auto unabhängig voneinander animiert: die Reifen rotieren und das Auto wird verschoben. Die Animation soll hier lediglich als ein Beispiel zur Verwendung von eindeutig benannten Gruppen dienen - weitergehende Informationen zu bewegten Grafiken finden Sie im Kapitel Animationen.

## 7.2 Das defs-Element und das use-Element

Das **defs**-Element ist ein Container-Element dessen einzige Aufgabe es ist einen Bereich festzulegen. Innerhalb dieses Bereichs (Containers) können Objekte definiert werden, die in der Grafik von anderen Objekten referenziert (verwendet) werden können, z.B. einzelne Elemente, Gruppen, Symbole, Farbverläufe oder Filtereffekte. Dabei ist zu beachten, dass im Definitions-Container eingeschlossenen Elemente oder Objekte nicht direkt in der Grafik dargestellt werden, sondern erst wenn sie innerhalb der Grafik von anderen Elementen referenziert werden. Es wird empfohlen, die Elemente die Referenzen darstellen, immer innerhalb des **defs**-Containers zu platzieren.

Mit dem **use**-Element wird eine Instanz einer bereits bestehenden Vorlage erzeugt, d.h. sie können mit dem Element **use** andere Elemente bzw. Objekte referenzieren.

Die Attribute ***x*** und ***y*** verlegen den Standort der Instanz, dabei ist der Nullpunkt der Nullpunkt des Elements und nicht der gesamten SVG-Grafik.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="250" height="120">
  <title>das defs-Element und das use-Element</title>
  <desc>Referenzen mit use</desc>

  <!-- Im Definitionsbereich (defs-Container) wird ein Farbverlauf
  und eine Linie definiert -->
  <!-- Die hier festgelegten Objekte werden erst
  nach Referenzierung angezeigt -->
  <defs>
    <linearGradient id="farbverlauf">
      <stop offset="0%" stop-color="red" />
      <stop offset="99%" stop-color="#33cc33" />
    </linearGradient>

    <line id="li"
      x1="5" y1="50" x2="205" y2="70"
      stroke="black" stroke-width="2" />
  </defs>

  <!-- Das Rechteck referenziert den Farbverlauf -->
  <rect x="5" y="5" width="200" height="40"
    style="fill:url(#farbverlauf);" />

  <!-- 5 Instanzen der Linie -->
  <use xlink:href="#li" />
  <use x="10" y="10" xlink:href="#li" />
  <use x="20" y="20" xlink:href="#li" />
  <use x="30" y="30" xlink:href="#li" />
  <use x="40" y="40" xlink:href="#li" />
</svg>
```

Im obigen Beispiel wird im **defs**-Container ein Farbverlauf mit der ID *farbverlauf* und eine Linie mit der ID *li* definiert.

Der Farbverlauf wird durch die Eigenschaft ***fill*** vom **rect**-Element referenziert. Der Farbverlauf soll hier allerdings nur als Beispiel für die Referenzierung (Verwendung) von, im **defs**-Bereich, definierten Objekten dienen. Farbverläufe werden in Kapitel Verläufe und Muster vorgestellt.

Die Linie wird durch das Attribut ***xlink:href*** vom **use**-Element verwendet. Jedes use-Element erzeugt genau eine Instanz der Linie *li* aus dem Definitionsbereich der Grafik. Die erste Instanz der Linie wird an der orginalen Position dargestellt. Die weiteren Instanzen werden durch die Attribute ***x*** und ***y*** entlang der x-Achse bzw. der y-Achse verschoben.

## 7.3 Symbole - das symbol-Element

Das Container-Element **symbol** ist speziell für ein grafisches Objekt (ein Element oder eine Gruppe von Elementen) gedacht, das mit Hilfe des **use**-Elements referenziert werden kann, und ist deshalb innerhalb eines **defs**-Containers zu verwenden.

Anstelle von **symbol** könnte auch das **g**-Element verwendet werden, um ein Objekt innerhalb des **defs**-Containers zur mehrmaligen Verwendung festzulegen, da das **symbol**-Element die gleichen Eigenschaften besitzt, wie das **g**-Element.

Der eigentliche Unterschied ist, dass Gruppen, die nicht im Definitionsbereich platziert werden am Bildschirm zu sehen sind, während Symbole immer (!) erst referenziert werden müssen. Weiterhin bewirkt die Verwendung des **symbol**-Elements eine bessere Strukturierung der Grafik, und daher eine bessere Lesbarkeit des Quellcodes. Und Symbole sind Grundlage für Marker (Kapitel 9.9 Pfeilspitzen - das marker-Element) und Muster (Kapitel 11.3 Muster - das pattern-Element).

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="100" height="100"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das symbol-Element</title>
  <desc>Referenzen - besondere Objekte: Symbole</desc>

  <defs>
    <symbol id="smilie1">
      <desc>ein Symbol-Smilie</desc>
      <circle cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>
  </defs>

  <use xlink:href="#smilie1" transform="scale(2)" />
</svg>
```

Im obigen Beispiel werden 3 Kreise, eine Linie und ein Pfad (mehr dazu im Kapitel Pfade) mit Hilfe des **symbol**-Elements zu einem Objekt mit der eindeutigen Bezeichnung *smilie1* gruppiert.

Dieses Objekt wird dann mit **use** referenziert und, durch die Verwendung des Attributs ***transform*** mit dem Wert *scale(2)*, doppelt so groß am Bildschirm dargestellt - (Transformationen werden im Kapitel Transformationen näher erläutert).

## 7.4 Einbinden von externen Grafiken - das image-Element

Mit dem **image**-Element wird eine externe Grafik referenziert, d.h. in die SVG Grafik eingebunden. Dabei sind nur die Grafikformate png, jpg und svg möglich.

Das **image**-Element erzeugt einen Bereich innerhalb der SVG-Grafik, in dem dann die externe Grafik dargestellt wird.

Das Attribut ***xlink:href*** wird benötigt, um die *URI* der externen Grafik festzulegen. Mit den Attributen ***x*** und ***y*** legen Sie die die x,y-Koordinate der linken, oberen Ecke des Bereichs fest, in dem die exteren Grafik angezeigt werden soll. Mittels der Attribute ***width*** und ***height*** bestimmen Sie die Ausmaße des Bereichs, in dem die Grafik angezeigt wird.

Das **image**-Element beinhaltet ein Child-Element (Kind-Element): das **title**-Element. Hiermit wird dem Bild ein Titel zugewiesen.

Beispiel Quellcode

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="200px" height="100px" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das image-Element</title>
<desc>Referenzen - Ein Bild im Bild</desc>

  <!-- hier wird eine png-Grafik in die SVG-Grafik eingebunden -->
  <image x="10" y="3.5" width="180px" height="93px"
    xlink:href="../grafik/aptilogo.jpg">
    <title>only jpg,png,svg</title>
  </image>
</svg>
```

## 7.5 Allgemeine Attribute - stdAttrs, langSpaceAttrs, testAttrs

Unter allgemeinen Attributen versteht man Attribute die man nahezu jedem SVG-Element zuordnen kann. Innerhalb der DTD (Document Type Definition) von SVG sind Attribute oft in benannten Gruppen zusammengefasst.

Es gibt fast 30 derartiger Attributgruppen, die innerhalb der DTD als sogenannte Entities (Abkürzungen, Umschreibungen) bezeichnet werden und eine Liste von Attributdefinitionen enthalten. Entities haben die Schreibweise: ProzentzeichenNameSemikolon. So ist also z.B. **%PresentationAttributes-FillStroke;** - eine Attributgruppe, die alle Attribute zur Festlegung von Eigenschaften für Füllungen und Randlinien umfasst oder **%stdAttrs;** eine Attributgruppe, welche die Standard-Attribute von SVG enthält.

Im Verlauf dieses Tutorials werden die wichtigsten Attributgruppen und deren Attribute im thematischen Zusammenhang innerhalb der entsprechenden Kapitel erläutert. Auch in der Übersicht 1: SVG Elemente und Attribute dieses Tutorials werden - entsprechenend der W3C SVG Recommendation - diese Attributgruppen zu Einteilung der Attribute verwendet.

Die Attribute der folgenden 3 Attributgruppen

- **stdAttrs**
- **langSpaceAttrs**
- **testAttrs**

können als "allgemeine Attribute" in nahezu allen Elementen verwendet werden.

Die Attributgruppe **stdAttrs** beinhaltet folgende Attributbeschreibungen:

***id*** Voreinstellung: keine. Mit diesem Attribut können Sie jedem Element oder Objekt einen eindeutigen Bezeichner zuordnen. Notwendig u.a. bei Referenzierungen.

***xml:base*** Voreinstellung: keine. Legt die Basis-URI (Ausgangsdokument) einer komplexen Web-Site fest, die logischerweise eine andere URI als das untergeordnete SVG-Dokument besitzt. Sie können hier also angeben auf welche Heimatseite einer Web-Site sich ihr SVG-Dokument bezieht.

Die Attributgruppe **langSpaceAttrs** beinhaltet folgende Attributbeschreibungen:

***xml:lang*** Voreinstellung: keine. Legt die, in den Inhalten der Elemente (z.B. text) oder in Attributwerten (z.B. id) verwendete Standardsprache des SVG-Dokuments fest. Möglich Werte sind die gebräuchlichen Sprachkürzel wie *de*, *en*, *fr*, .. .

***xml:space*** Voreinstellung: keine. Legt fest, wie die anzeigende Applikation standardmäßig Leerzeichen darstellt. Mögliche Werte sind *default* und *preserve*. Bei Verwendung von *preserve* werden alle editierten Leerzeichen dargestellt (wie im pre-Bereich eines HTML-Dokuments).

Die Attributgruppe **testAttrs** beinhaltet folgende Attributbeschreibungen:

***requiredFeatures*** Voreinstellung: keine. Wertet die SVG-Features aus, die die anzeigende Applikation unterstützt wie z.B. Animationen, Filtereffekte, .. . Im zutreffenden Fall wird true zurückgeliefert.

***requiredExtensions*** Voreinstellung: keine. Definiert eine Liste von benötigten SVG-Spracherweiterungen. Spracherweiterungen sind Möglichkeiten der anzeigenden Applikation, die über die in der SVG-Spezifikation festgelegen Möglichkeiten hinausgehen. Mögliche Werte sind *URI*s, welche die benötigten Spracherweiterungen identifizieren. Unterstützt der User-Agent die angegebenen Erweiterungen wird true zurückgeliefert.

***systemLanguage*** Voreinstellung: keine. Wertet die im System eingestellte Standardsprache aus (nicht die Browsereinstellungen). Mögliche Werte sind die gebräuchlichen Sprachkürzel wie *de*, *en*, *fr*, .. . Im zutreffenden Fall wird true zurückgeliefert.

## 7.6 Auswahlmöglichkeiten - das switch-Element

Das **switch**-Element wertet die Testattribute ***systemLanguage***, ***requiredFeatures*** oder ***requiredExtensions*** seiner direkten Kindelemente in ihrer Reihenfolge aus. Nur das erste (!) Kindelement, bei dem alle verwendeten Testattribute true zurückliefern, wird am Bildschirm dargestellt.

Alle (!) anderen bzw. weiteren Kindelemente werden nicht dargestellt, ungeachtet ob auch deren Testattribute true zurückliefern.

Falls es sich bei einem Kindelement von **switch** um ein Container-Element (wie das **g**-Element) handelt, werden alle Elemente innerhalb dieses Containers angezeigt.

Nachfolgend 3 Beispiele, welche die Verwendung der 3 unterschiedlichen Testattribute zeigen.

Im ersten Beispiel wird, je nach der im System eingestellten Sprache, ein entsprechendes Länderkürzel angezeigt. Dazu wird das Test-Attribut ***systemLanguage*** verwendet, das eine zulässiges Länderkürzel wie *de*, *en*, *fr*, etc. als Wert akzeptiert. Die Darstellung von Text in SVG Dokumenten wird im nächsten Kapitel erläutert.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das switch-Element</title>
<desc>eine Auswahl nach der Systemsprache</desc>

<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:verdana,sans-serif; font-size:20px; font-weight:bold;}
    circle {fill:lightgreen; stroke:black;}
  ]]>
  </style>
</defs>

<!-- der switch Bereich -->
<switch>
  <g systemLanguage="en">
    <circle cx="100" cy="95" r="20" />
    <text x="87" y="102">en</text>
  </g>

  <g systemLanguage="de">
    <circle cx="100" cy="95" r="20" />
    <text x="87" y="102">de</text>
  </g>

  <g systemLanguage="fr">
    <circle cx="100" cy="95" r="20" />
    <text x="87" y="102">fr</text>
  </g>
</switch>
</svg>
```

Im zweiten Beispiel wird das Test-Attribut ***requiredFeatures*** verwendet. Damit kann überprüft werden, ob der User Agent ein bestimmtes Feature von SVG, wie z.B. Verläufe oder Animationen unterstützt. Die zulässigen Werte für dieses Attribut, sogenannte **feature strings**, unterscheiden sich jedoch in den SVG Standards.

In SVG 1.0 werden **feature strings** mit folgender Präfix verwendet: **org.w3c.svg**. So ist z.B. *org.w3c.svg.static* eine gültige Wertzuweisung fü ***requiredFeatures*** nach SVG 1.0 und bedeutet, dass

die grundlegenden Möglichkeiten des SVG Standards unterstützt werden müssen. Dazu gehören: Grundlegende Datentypen, Elemente der Dokumentstruktur, CSS-Unterstützung, Transformationen, Pfade, Marker, geometrische Formen, Text, Attribute für Painting, Verläufe und Muster, Masken, Filter und SVG Fonts. Dazu gehören nicht: Erzeugung einer DOM-Struktur für Skriptsprachen, Animationen und Hyperlinks.

In SVG 1.1 werden **feature strings** mit folgender Präfix verwendet: **http://www.w3.org/TR/SVG11/feature#**. So ist z.B. *http://www.w3.org/TR/SVG11/feature#SVG-static* eine gültige Wertzuweisung für ***requiredFeatures*** nach SVG 1.1 und bedeutet ebenfalls, dass die grundlegenden Möglichkeiten des SVG Standards unterstützt werden müssen. In SVG 1.1 ist allerdings eine größere Anzahl von **feature strings** verfügbar, wobei man die unterschiedlichen Möglichkeiten auch einzeln abfragen kann (z.B. nur Gradienten oder Text). Dies macht eine genauere Überprüfung der vom User Agent unterstützten SVG Features möglich.

Eine ausführliche Beschreibung der **feature strings** finden Sie in der W3C SVG 1.1 Recommendation - Appendix O: Feature Strings.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das switch-Element 2</title>
<desc>eine Auswahl nach unterstützten Features</desc>

<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:verdana,sans-serif; font-size:20px; font-weight:bold;}
  ]]>
  </style>
 <linearGradient id="lgr1"
   x1="0" y1="0" x2="1" y2="0">
   <stop offset="0" stop-color="limegreen" />
   <stop offset="1" stop-color="yellow" />
 </linearGradient>
</defs>

<switch>
<!-- feature string nach SVG 1.0 -->
<!-- Anzeige der folgenden Gruppe, wenn z.B. Grandienten unterstützt werden -->
  <g requiredFeatures="org.w3c.svg.static">
    <g>
      <circle cx="100" cy="95" r="60" fill="url(#lgr1)" />
      <text x="100" y="102" text-anchor="middle">feature</text>
    </g>
  </g>

<!--Anzeige der folgenden Gruppe, im Sonst-Fall -->
  <g>
    <g>
      <circle cx="100" cy="95" r="60" fill="red" />
      <text x="100" y="102" text-anchor="middle">nix feature</text>
    </g>
  </g>
</switch>
</svg>
```

Im letzten Beispiel wird das Test-Attribut ***requiredExtensions*** verwendet. Damit kann überprüft werden, ob der User Agent bestimmte SVG Erweiterungen unterstützt. Das Attribut ***requiredExtensions*** benötigt eine URL, die i.d.R. einen Namensraum bezeichnet, als Wertzuweisung.

Im folgenden SVG Dokument wird überprüft ob der User Agent die Hyperlink-Erweiterungen des Adobe SVG Viewers unterstützt (was nur der Fall ist, wenn sie diesen Viewer verwenden). Im Ja-Fall wird - nach notwendiger Downloadzeit - eine kurze mp3-Sound-Datei immer wieder abgespielt und ein passender Text angezeigt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="100"
  xmlns:a="http://www.adobe.com/svg10-extensions" xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das switch-Element 3</title>
<desc>eine Auswahl nach unterstützten SVG Erweiterungen</desc>

<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:verdana,sans-serif; font-size:20px; font-weight:bold;}
  ]]>
  </style>
</defs>

<switch>
<!-- Abspielen eines Sounds und Anzeige eines Textes -->
<!-- wenn die Hyperlink-Erweitungen des Adobe SVG Viewers unterstützt werden -->
  <g requiredExtensions="http://www.adobe.com/svg10-extensions">
    <a:audio xlink:href="sound64.mp3" begin="0" volume="10" repeatCount="indefinite" />
    <text x="100" y="60" text-anchor="middle">Sound in SVG</text>
  </g>

<!-- Nur Anzeige eines Textes, im Sonst-Fall -->
  <g>
    <text x="100" y="60" text-anchor="middle">kein Sound</text>
  </g>
</switch>
</svg>
```

# 8 Text

Mit Hilfe des **text**-Elements können Texte in SVG-Grafiken eingefügt werden.

Diese Möglichkeit Text einzubinden ist ein großer Vorteil von SVG-Dokumenten gegenüber anderen Grafikformaten, da auf diese Weise der Textinhalt für sämtliche externen Applikationen, wie z.B. Suchmaschinen, Skriptprogramme, CGI-Programme oder ähnliches, zugänglich wird.

Ein weiterer Vorteil ist die Unterstützung von UNICODE. UNICODE ist ein Zeichensystem, das jedem Zeichen eine einzigartige Nummer zuweist, die platform-, sprachen- und programmunabhängig ist. Dadurch ist es möglich jedes beliebige Zeichen, wie z.B. deutsche Umlaute und das ß, chinesische Schriftzeichen, Sonderzeichen, mathematische Zeichen, etc. darzustellen. Eine Übersicht der UNICODE-Zeichen finden sie bei http://www.unicode.org.

Texte können durch Attribute oder Eigenschaften vielfältig formatiert werden. So können Sie folgende Eigenschaften von Text durch Attribute bestimmen:

- Schriftfarbe,
- Schriftart,
- Schriftgröße,
- Schriftdicke,
- Schriftstil,
- Schriftausdehnung,
- Schriftvariation,
- Textdekorationn,
- Laufrichtung,
- Orientierung,
- Bidirektionalität,
- Textbündigkeit,
- Leerzeichenverhalten,
- Zeichenabstand,
- Wortabstand,
- Grundlinienverschiebung,
- Text entlang Pfaden (hier wird ein SVG-Element verwendet),
- Kerning,
- Anti-Aliasing.

Diese Attribute zur Formatierung von Text sind hauptsächlich in folgende 3 Attributgruppen eingegliedert.

- **PresentationAttributes-FontSpecification**
- **PresentationAttributes-TextContentElements**
- **PresentationAttributes-TextElements**

Aber Achtung bei der Erzeugung von Text! Texte werden immer als "Einzeiler" dargestellt, d.h. SVG stellt automatisch keine Zeilenumbrüche dar. Um mehrzeilige Texte in SVG darzustellen, können Sie entweder mehrere text-Elemente verwenden oder das **tspan**-Element, mit dem Sie innerhalb eines text-Elements, Unterbereiche festlegen können.

## 8.1 Das text-Element und seine Koordinaten

Einem **text**-Element werden die Attribute ***x*** und ***y*** zur Festlegung der Position des Textes innerhalb der Grafik zugeordnet.

**Dabei ist unbedingt zu beachten, dass die y-Koordinate bei text-Elementen die Position der Grundlinie des Textes angibt. Die x,y-Koordinate bestimmt also die linke untere Ecke und nicht wie bei allen anderen Elementen in SVG die linke obere Ecke.**

Weisen Sie ***y*** daher immer eine Wert größer 0 zu, sonst liegt der Text möglicherweise außerhalb der Grafik und wird nicht dargestellt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das text-Element</title>
<desc>Text in SVG platzieren</desc>

<!-- etwas Grafik -->
<rect x="1" y="1" width="298" height="198"
  style="fill:none; stroke:black" />
<circle cx="150" cy="100" r="90" style="fill:lightgreen;" />

<!-- ein Einzeiler, mit style-Vorschriften formatiert -->
<text x="8" y="40"
style="font-family:verdana; font-size:20px; font-weight:bold;">
    Text in SVG-Dokumenten
</text>

<!-- ein Einzeiler, mit Attributen formatiert -->
<text x="61" y="105"
font-family="verdana" font-size="16px">
    .. ist eine tolle Sache!
</text>
</svg>
```

Es ist möglich für die Attribute ***x*** und ***y*** mehrere, durch Komma getrennte, Werte festzulegen. Dann bezieht sich jeder Wert auf das seiner Position entspechende einzelne Zeichen innerhalb des **text**-Elements. Also: erster Wert für das erste Zeichen, zweiter Wert für das zweite Zeichen, .. .

Die Attribute ***dx*** und ***dy***, bewirken Verschiebungen der einzelene Zeichen entlang der x- bzw. y-Achse. Die Verschiebung bezieht sich auf die direkt vorangegangenen Festlegung der x,y-Koordinate. Auch hier sind mehrere, durch Komma getrennte Werte möglich.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>nochmal das text-Element</title>
<desc>Text in SVG platzieren</desc>
<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:verdana,sans-serif; font-size:40px; font-weight:bold;}
  ]]>
  </style>
```

```
</defs>

<!-- wieder etwas Grafik -->
<circle cx="150" cy="100" r="90" style="fill:none; stroke:black;" />

<!-- mehrere Werte für x und y -->
<text class="big" x="110,135,160" y="70,110,150">
SVG
</text>

<!-- Verwendung von dx und dy -->
<text class="big" x="110,135,160" y="70,110,150"
  dx="-10,-10,-10" style="fill:red;">
SVG
</text>
</svg>
```

## 8.2 Text im Text - das tspan-Element

Das **tspan**-Element entspricht im wesentlichen dem span-Element in HTML.

Mit dem **tspan**-Element lassen sich innerhalb des **text**-Elements eigene Bereiche definieren, die Sie Ihren Wünschen entsprechend positionieren und layouten können. Auf diese Art ist es unter anderem möglich, mehrere Zeilen innerhalb eines **text**-Elements darzustellen, ohne mehrere dieser Elemente verwenden zu müssen. Dies kann von Bedeutung sein, damit auf Texte mit mehreren Zeilen komplett selektiert werden können, z.B. um Texte zu kopieren oder für externe Anwendungen, um Texte im Zusammenhang auszulesen.

Natürlich lassen sich Textinhalte von tspan-Elementen auch unabhängig vom Eltern-text-Element formatieren, so dass z.B. Schrift mit verschieden farbigen Zeichen möglich ist.

Für das **tspan**-Element sind alle Attribute gültig, die auch für das **text**-Element verwendet werden dürfen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das tspan-Element</title>
<desc>Text selektieren und lokal formatieren</desc>

<!-- und wieder etwas Grafik -->
<rect x="1" y="1" width="298" height="198"
  style="fill:none; stroke:black" />
<circle cx="150" cy="100" r="90" style="fill:lightgreen;" />

<!-- ein Einzeiler -->
<text x="8" y="40"
style="font-family:verdana; font-size:20px; font-weight:bold;">
    Text in SVG-Dokumenten
</text>

<!-- drei Zeilen innerhalb eines text-Elements,
     mit Hilfe des tspan-Elements
     verschoben und unterschiedlich formatiert -->
<text x="63" y="95"
  style="font-family:verdana; font-size:16px;">
    .. ist eine
    <tspan x="97" y="115" style="font-weight:bold; fill:red;">
      tolle Sache!
    </tspan>
    <tspan x="78" y="135">
      Was meinen Sie ?!
    </tspan>
</text>
</svg>
```

## 8.3 Texte referenzieren - das tref-Element

Das **tref**-Element ermöglicht die Einbindung von Textvorlagen an beliebigen Stellen in einem SVG-Dokument. Diese Textvorlagen müssen innerhalb des SVG-Dokuments deklariert werden und - durch das Attribut ***id*** - eindeutig bezeichnet sein. Externe Textdateien können nicht eingebunden werden.

Für das **tref**-Element sind alle Attribute gültig, die auch für das **text**-Element oder das **tspan**-Element verwendet werden dürfen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="330" height="180"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das tref-Element</title>
<desc>Texte referenzieren</desc>
<defs>
  <style type="text/css"><![CDATA[
    text {font-family:verdana,sans-serif; font-size:16px;}
  ]]></style>
  <text id="svgtxt">Scalable Vector Graphics</text>
</defs>

<text x="20" y="30" style="font-size:20px;">
SVG
<tref xlink:href="#svgtxt" x="20" y="60" />
<tref xlink:href="#svgtxt" x="30" dy="20" style="fill:red;" />
<tref xlink:href="#svgtxt" x="40" dy="20" style="fill:gray;" />
<tref xlink:href="#svgtxt" x="50" dy="20" style="fill:blue;" />
<tref xlink:href="#svgtxt" x="60" dy="20" style="fill:green;" />
</text>
</svg>
```

Im Beispiel wird eine Textvorlage mit der ID *svgtxt* im **defs**-Container der Grafik festgelegt, sie wird also an dieser Stelle nicht angezeigt. Mit dem **tref**-Element wird diese Vorlage dann mehrmals in die Grafik eingebunden.

## 8.4 Rotation einzelner Zeichen

Mit Hilfe des Attributs ***rotate*** können Zeichen vom Inhalt eines text-Elements rotiert werden.

Dabei ist zu beachten, dass lediglich das erste Zeichen rotiert wird, wenn dem Attribut ***rotate*** nur ein Wert zugeordnet wird. Es ist allerdings möglich rotate mehrere Werte zuzuordnen. Jeder Wert bezieht sich dann auf genau ein Zeichen: der erste Wert auf das erste Zeichen, der zweite Wert auf das zweite Zeichen, ...

Der Drehpunkt eines Zeichens: Denken Sie sich ein Rechteck um das jeweilige Zeichen, das das Zeichen eng umschliesst - der Drehpunkt des Zeichens ist dann die linke, untere Ecke dieses gedachten Rechtecks.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das rotate-Attribut</title>
<desc>Rotation von Zeichen</desc>
  <defs>
    <style type="text/css"><![CDATA[
      text {font-family:Verdana,sans-serif;
            font-size:28px;}
    ]]></style>
  </defs>

  <!-- es wird immer nur das erste Zeichen gedreht -->
  <text x="20" y="40" rotate="40">
    rot
    <tspan rotate="40">ate</tspan>
  </text>
  <line x1="20" y1="40" x2="200" y2="40" stroke="black" />

  <!-- um den gesamten Text zu drehen muß transformiert werden -->
  <text x="20" y="70" fill="green" transform="rotate(40,20,70)">
    rotate
  </text>
  <text x="70" y="85" style="font-size:12px;" fill="gray">transform</text>
  <line x1="20" y1="70" x2="200" y2="70" stroke="black" />

  <!-- rotate mit mehreren Werten -->
  <text x="20" y="150" rotate="0,30,60,90,120,150,180,210,240,270,300,330,0">
    rotate-more-!
  </text>
  <line x1="20" y1="150" x2="200" y2="150" stroke="black" />

  <!-- Drehpunkt ist ist die linke, untere Ecke des rechteckigen Bereichs
       den das Zeichen einnimmt -->
  <text x="250" y="70">
    W
    <tspan x="250" y="70" rotate="90" fill="blue">W</tspan>
    <tspan x="250" y="70" rotate="180" fill="red">W</tspan>
  </text>
  <line x1="250" y1="70" x2="350" y2="70" stroke="black" />
</svg>
```

## 8.5 Schriftformatierung

Außer dem Attribut ***fill***, welches zur Attributgruppe **PresentationAttributes-FillStroke** gehört, sind alle weiteren Attribute dieses Kapitels in der Attributgruppe **PresentationAttributes-FontSpecification** zu finden.

- Schriftfarbe - ***fill***
- Schriftart - ***font-family***
- Schriftgröße - ***font-size***
- Schriftdicke - ***font-weight***
- Schriftstil - ***font-style***
- Schriftausdehnung - ***font-stretch***
- Schriftvariation - ***font-variant***

***fill*** Voreinstellung: *black*. Die Schriftfarbe wird mit Eigenschaft oder dem Attribut ***fill*** zugewiesen. Diese Attribut gehört zur Attributgruppe **PresentationAttributes-FillStroke**. Mögliche Werte sind Farbangaben in hexadezimaler Schreibweise, durch ein festgelegtes Farbwort oder in dezimaler Schreibweise, z.B.:
<tspan style="fill:#ff0000;"> ... roter text ... </tspan> <tspan style="fill:red;"> ... roter text ... </tspan>
<tspan style="fill:rgb(255,0,0);"> ... roter text ... </tspan> Alle drei Beispiele erzeugen eine rote Schrift. (siehe auch Kapitel Painting, dort werden alle Attribut der Attributgruppe **PresentationAttributes-FillStroke** vorgestellt).

***font-family*** Voreinstellung: vom User agent abhängig. Eine Schriftart können sie mit dem Attribut oder der Eigenschaft ***font-family*** festlegen. Dabei ist es möglich, mehrere Schriftarten, durch Komma getrennt, hintereinander anzugeben. Falls die zuerst angegebenen Schriftart auf dem System des Clients bzw. Users nicht dargestellt werden kann, versucht der Browser es mit der nächsten angegbenen Schriftart, usw. Wenn die Bezeichnung einer Schriftart Leerzeichen enthält, müssen Sie diese in einfache Hochkommatas einschließen, z.B.: <tspan style="font-family:'Courier New',Verdana,sans-serif;"> ... text ... </tspan>

***font-size*** Voreinstellung: *medium*. Mit Hilfe des Attributs oder der Eigenschaft ***font-size*** legen Sie die Schriftgröße fest. ***font-size*** erwartet eine Größenangabe oder ein festgelegtes Schlüsselwort als Wertzuweisung. Schriften werden meist in den Maßen Pixel oder Punkt angegeben, z.B: <tspan style="font-size:12px;"> ... 12 Pixel text ... </tspan> <tspan style="font-size:16pt;"> ... 16 Punkt text ... </tspan>

***font-weight*** Voreinstellung: *normal*. Die Schriftdicke oder das Schriftgewicht können Sie mit dem Attribut oder der Eigenschaft ***font-weight*** einstellen. Die wichtigsten, möglichen Werte für ***font-weight*** sind

- *normal* - keine Formatierung und
- *bold* - erzeugt fette Schrift.

Es sind ebenfalls noch die Werte *bolder* (noch fetter) und *lighter* (nicht ganz so fett) möglich. Und weiterhin können Sie mit den Zahlen *100*, *200*, *300*, *400*, *500*, *600*, *700*, *800* und *900* als Wertzuweisung für ***font-weight*** insgesamt 9 verschieden "Fett-Abstufungen" erzeugen - wenn der Browser das mitmacht ... Dabei entspricht der Wert *normal* dem Wert *500*.

***font-style*** Voreinstellung: *normal*. Den Schriftstil, also kursive Schrift, bestimmen Sie mit dem Attribut oder der Eigenschaft ***font-style***. Die Werte *italic* und *oblique* erzeugen am Bildschirm nicht unterscheidbare kursive Schriftzeichen, mit dem Wert *normal* findet keine Formatierung statt. <tspan style="font-style:oblique;"> ... kursiver text ... </tspan>

***font-stretch*** Voreinstellung: *normal*. Mit Schriftausdehnung kann der Abstand der Zeichen zueinander verändert werden. Möglich macht dieses das Attribut oder die Eigenschaft ***font-stretch***. Sie können den Zeichen also eine "Stretching" verordnen :-). Folgende Werte werden von ***font-stretch*** akzeptiert:

- *normal*
- *wider*
- *narrower*
- *ultra-condensed*
- *extra-condensed*
- *condensed*
- *semi-condensed*

- *semi-expanded*
- *expanded*
- *extra-expanded*
- *ultra-expanded*

Experimentieren Sie einfach ein wenig mit den verschieden Wertzuweisungen.

***font-variant*** Voreinstellung: *normal*. Dieses Attribut akzeptiert 2 Werte: *normal* und *small-caps*. Bei Verwendung von *small-caps* wird der Text als Kapitälchen (kleine Großbuchstaben) dargestellt. Der SVG-Viewer von Adobe unterstützt dieses Attribut noch nicht.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="210">
<title>Schriftformatierung</title>
<desc>
  Farbe, Art, Größe, Dicke, Stil, Ausdehnung und Variation von Text
</desc>

  <text x="40" y="20">
    <!-- Schriftfarbe -->
    <tspan fill="blue">Schriftfarbe</tspan>
    <tspan x="200" style="fill:blue;">Schriftfarbe</tspan>

    <!-- Schriftart -->
    <tspan x="40" y="40" font-family="Times">
      Schriftart
    </tspan>
    <tspan x="200" y="40" style="font-family:Times">
      Schriftart
    </tspan>

    <!-- Schriftgröße -->
    <tspan x="40" y="60" font-size="26px">
      Schriftgröße
    </tspan>
    <tspan x="200" y="60" style="font-size:26px">
      Schriftgröße
    </tspan>

    <!-- Schriftdicke -->
    <tspan x="40" y="80" font-weight="bold">
      Schriftdicke
    </tspan>
    <tspan x="200" y="80" style="font-weight:bold;">
      Schriftdicke
    </tspan>

    <!-- Schriftstil -->
    <tspan x="40" y="100" font-style="italic">
      Schriftstil
    </tspan>
    <tspan x="200" y="100" style="font-style:italic;">
      Schriftstil
    </tspan>

    <!-- Schriftausdehnung -->
    <tspan x="40" y="120" font-stretch="extra-expanded">
      Schriftausdehnung
    </tspan>
    <tspan x="200" y="120" style="font-stretch:extra-expanded;">
```

```
    Schriftausdehnung
   </tspan>

   <!-- Schriftvariation -->
   <tspan x="40" y="140" font-variant="small-caps">
    Schriftvariation
   </tspan>
   <tspan x="200" y="140" style="font-variant:small-caps;">
    Schriftvariation
   </tspan>
  </text>

  <text x="10" y="170"
   style="font-family:Verdana; font-size:14px;">
   Klicken Sie mit der rechten Maustaste in die Grafik
   <tspan x="20" dy="20">und wählen Sie
    <tspan dx="20" dy="7"
     style="font-size:28px; fill:yellow;
          stroke:black; stroke-width:.5px;">
     &amp;quot;Einzoomen&amp;quot;
    </tspan>
   </tspan>
  </text>
</svg>
```

## 8.6 Ausrichtung und Dekoration von Text

Außer dem Attribut ***writing-mode***, das einziges Attribut der Attributgruppe **PresentationAttributes-TextElements** ist, gehören alle weiteren Attribute dieses Kapitels der Attributgruppe **PresentationAttributes-TextContentElements** an.

- Laufrichtung - ***writing-mode***
- Orientierung - ***glyph-orientation-horizontal*** und ***glyph-orientation-vertical***
- Bidirektionalität - ***unicode-bidi*** und ***direction***
- Textbündigkeit - ***text-anchor***
- Textdekoration - ***text-decoration***

***writing-mode*** Voreinstellung: *lr-tb* (left-right/top-bottom). Die Laufrichtung eines Textes können Sie mit dem Attribut ***writing-mode*** bestimmen. Dieses Attribut ist das einzige Attribut der Attributgruppe **PresentationAttributes-TextElements**. Die wichtigsten Werte für dieses Attribut sind:

- *rl* - right-left: von rechts nach links
- *lr* - left-right: von links nach rechts
- *tb* - top-bottom: von oben nach unten
- *bt* - bottom-top: von unten nach oben

***glyph-orientation-horizontal*** Voreinstellung: *0*. Die Orientierung (Rotation) der einzelnen Zeichen kann bei den Laufrichtungen *lr* und *rl* mit dem Attribut ***glyph-orientation-horizontal***, bestimmt werden. Der Wert für dieses Attribut gibt in jeweils 90 deg-Schritten (deg = Grad) die Orientierung der Zeichen an. Mögliche Werte sind also *0*, *90*, *180*, *270* und *360*.

***glyph-orientation-vertical*** Voreinstellung: *90*. Die Orientierung (Rotation) der einzelnene Zeichen kann bei den Laufrichtungen *tb* und *bt* mit dem Attribut ***glyph-orientation-vertical*** bestimmt werden. Auch der Wert für dieses Attribut gibt in jeweils 90 deg-Schritten (deg = Grad) die Orientierung der Zeichen an. Mögliche Werte sind also *0*, *90*, *180*, *270* und *360*.

***unicode-bidi*** Voreinstellung: *normal*. Bidirektionalität bedeutet (visuell) die Zeichen in Spiegelschrift darzustellen. Durch das Attribut ***unicode-bidi*** können Sie die normale Darstellung der Zeichen, die in Abhängigkeit von der System-Sprache automatisch verwendet wird, aufheben. Verwenden Sie hierfür den Wert *bidi-override*. Erst dann können Sie mit Hilfe des Attributs ***direction*** die die Direktionalität verändern.

***direction*** Voreinstellung: *ltr* (left-to-right). Nach Aufhebung der normalen Zeichendarstellung durch Verwendung des Attributs ***unicode-bidi*** mit dem Wert *bidi-override*, kann man mit dem Attribut ***direction*** die Direktionalität verändern. Mögliche Werte sind:

- *ltr* - left to right: von links nach rechts und
- *rtl* - right to left: von rechts nach links

***text-anchor*** Voreinstellung: *start*. Die Textbündigkeit oder Ausrichtung von Text wird durch das Attribut ***text-anchor*** festgelegt. Im obigen Beispiel sehen Sie die Darstellung einer Zeichenkette bei Verwendung der für dieses Atrribut möglichen Werte: *begin*, *middle* und *end*.

***text-decoration*** Voreinstellung: *none*. Textdekoration findet durch das Attribut oder die Eigenschaft ***text-decoration*** statt. Sie können mit dieser Eigenschaft Linien am Text erzeugen, also unterstrichenen und durchgestrichenen Text, aber auch Text mit einer oberen Linie oder sogar blinkenden Text, wenn es der Browser will ... Dabei sind die möglichen Werte

- *none* - keine Formatierung.
- *underline* - erzeugt unterstrichenen Text, und
- *overline* - Linie oberhalb des Textes.
- *line-through* - durchgestrichenen Text.
- *blink* - blinkender Text.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="550" height="360">
<title>Ausrichtung und Leerzeichen</title>
<desc>Laufrichtung, Orientierung, Bidirektionalität, Textbündigkeit, Leerzeichen</desc>
  <defs>
    <style type="text/css"><![CDATA[
      text {font-family:Verdana,sans-serif;
            font-size:20px;}
    ]]></style>
```

```
  </defs>

<!-- Laufrichtung, Orientierung, Bidirektionalität 1 -->
  <text x="30" y="40"
    writing-mode="tb">
    top bottom
  </text>
  <text x="60" y="40"
    writing-mode="tb" glyph-orientation-vertical="0">
    top bottom
  </text>
  <text x="90" y="40"
    writing-mode="tb" glyph-orientation-vertical="270">
    top bottom 270
  </text>
  <text x="120" y="40"
    writing-mode="tb" glyph-orientation-vertical="270"
    unicode-bidi="bidi-override" direction="rtl">
    top bottom 270 right to left
  </text>

<!-- Laufrichtung, Orientierung, Bidirektionalität 2 -->
  <g fill="red">
    <text x="270" y="60"
      writing-mode="rl">
      right left
    </text>
    <text x="270" y="90"
      writing-mode="lr">
      left right
    </text>
    <text x="270" y="120"
      writing-mode="lr" glyph-orientation-horizontal="180">
      left right 180
    </text>
    <text x="270" y="150"
      writing-mode="lr" glyph-orientation-horizontal="180"
      unicode-bidi="bidi-override" direction="rtl">
      left right 180 right to left
    </text>
    <line x1="270" y1="40" x2="270" y2="160" stroke="black" />
  </g>

<!-- Textbündigkeit links, mittig, rechts -->
  <g fill="blue">
    <text x="220" y="230" text-anchor="begin">
      begin
    </text>
    <text x="220" y="260" text-anchor="middle">
      middle
    </text>
    <text x="220" y="290" text-anchor="end">
      end
    </text>
    <line x1="220" y1="200" x2="220" y2="310" stroke="black" />
  </g>

<!-- text-decoration -->
  <text x="330" y="240"
    text-decoration="underline">underline</text>
  <text x="330" y="270"
    text-decoration="line-through">line-through</text>
  <text x="330" y="300"
    text-decoration="overline">overline</text>
```

```
  <text x="330" y="330"
    text-decoration="blink">blink (vielleicht)</text>
</svg>
```

Im obigen Beispiel wird zuerst ein Text durch die Verwendung des Attributs ***writing-mode*** mit dem Wert *tb* zuerst 4 mal von oben nach unten verlaufend dargestellt, wobei die Orientierung der Zeichen der ersten Zeichenkette 90 Grad beträgt - Standardwert von ***glyph-orientation-vertical***. Die Orientierung der Zeichen in der zweiten Zeichenkette beträgt 0 Grad, die Zeichen stehen aufrecht. In der dritten Zeichenkette wurde die Orientierung der Zeichen auf 270 Grad eingestellt. Wie Sie im Beispiel sehen, wird jedes einzelne Zeichen um 270 Grad gedreht, so dass "quasi" eine Spiegelschrift erzeugt wurde. Die vierte Zeichenkette verdeutlicht die Bedeutung der Attribute ***unicode-bidi*** und ***direction***. Hier wurde die Direktionalität *rtl*, von rechts nach links, eingestellt. Auf diese Weise wird die Zeichenkette wieder "entspiegelt".

Die erste der darauffolgenden 4 Zeichenketten zeigt die Darstellung der Zeichen, wenn die Laufrichtung von rechts nach links verwendet wird, also die Anwendung des Attributs ***writing-mode*** mit dem Wert *rl*. Bei den folgenden drei Zeichenketten werden wieder Orientierung, diesmal mit dem Attribut ***glyph-orientation-horizontal*** und Bidirektionalität der Zeichen manipuliert.

Nachfolgend finden Sie 2 Beispielblöcke zur Verwendung des Attributs ***text-anchor*** mit allen möglichen Werten und des Attributs ***text-decoration***, ebenfalls mit allen möglichen Werten.

## 8.7 Zeichen- und Wortabstände

Die folgenden Attribute gehören, wie im Kapitel zuvor, ebenfalls zur Attributgruppe **PresentationAttributes-TextContentElements**.

- Zeichenabstand - ***letter-spacing***
- Wortabstand - ***word-spacing***

***letter-spacing*** Voreinstellung: *normal*. Den Abstand der einzelnen Zeichen einer Zeichenkette zueinander, also den Zeichenabstand, können Sie mit dem Attribut ***letter-spacing*** verändern. Mögliche Werte für dieses Attribut sind entweder *normal* oder eine Längenangabe, wobei auch negative Zahlen möglich sind.

***word-spacing*** Voreinstellung: *normal*. Der Wortabstand innerhalb einer Zeichenkette wird mit dem Attribut ***word-spacing*** eingestellt. Dieses Attribut akzeptiert ebenfalls *normal* oder eine Längenangabe als Wert.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="190"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Abstände im Text</title>
<desc>Die Attribute letter-spacing und word-spacing</desc>
  <defs>
    <style type="text/css"><![CDATA[
      text {font-family:Verdana,sans-serif;
            font-size:20px;}
      .rot {fill:red;}
      .blau {fill:blue;}
    ]]></style>
  </defs>

  <!-- Zeichenabstand -->
  <g class="rot">
    <text x="20" y="40" letter-spacing="10">
      Zeichenabstand 10
    </text>
    <text x="20" y="70" letter-spacing="-2">
      Zeichenabstand -2
    </text>
  </g>

  <!-- Wortabstand -->
  <g class="blau">
    <text x="20" y="120" word-spacing="10">
      Abstand der Worte 10
    </text>
    <text x="20" y="150" word-spacing="-5">
      Abstand der Worte -5
    </text>
  </g>
</svg>
```

## 8.8 Mögliche Grundlinien und das tpath-Element

Auch das Attribut ***baseline-shift***, das neben dem **tpath**-Element in diesem Kapitel behandelt wird, gehört zur Attributgruppe **PresentationAttributes-TextContentElements**.

- Grundlinienverschiebung - ***baseline-shift***
- Text entlang Pfaden - das **tpath**-Element

***baseline-shift*** Voreinstellung: *baseline*. Sie haben die Möglichkeit die Grundlinie eines Textes nach oben oder nach unten zu verschieben. Eine solche Grundlinienverschiebung stellen Sie mit dem Attribut ***baseline-shift*** ein. Eine positive Längenangabe als Wert bewirkt eine Verschiebung der Grundlinie nach oben, eine negative Längenangabe als Wert bewirkt eine Verschiebung der Grundlinie nach unten.

Mit dem Element **textPath** können Sie einer Zeichenkette aber auch einen selbst festgelegten Pfad, siehe Pfade, als Grundlinie zuordnen. In diesem Fall verläuft der Text am Pfad entlang, wobei die Orientierung der Zeichen an den Verlauf des Pfades automatisch erfolgt. Das Element **textPath** ist ein direktes Kind-Element des **text**-Elements.

Das **textPath**-Element muß einen bereits definierten Pfad referenzieren. Referenzen werden im Kapitel Dokumentstruktur II erläutert. Damit dies möglich wird, muß dem Pfad mit Hilfe des ***id***-Attributs ein eindeutiger Bezeichner zugeordnet werden. Daraufhin kann dieser Pfad eindeutig vom **textPath**-Element referenziert werden. Dies geschieht durch die Verwendung des Attributs ***xlink:href***.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="300"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das textPath-Element</title>
<desc>Grundlinienverschiebung mit dem Attribut basline-shift
  und die Verwendung eines Pfades als Grundlinie</desc>
  <defs>
    <style type="text/css"><![CDATA[
      text {font-family:Verdana,sans-serif;
            font-size:20px;}
    ]]></style>
  </defs>

  <!-- Grundlinienverschiebung -->
  <rect x="10" y="20" width="370" height="20" fill="lightgray" />
  <text x="20" y="40">
    Grundlinienverschiebung +/- Null
  </text>
  <line x1="10" y1="40" x2="380" y2="40" stroke="red" />

  <rect x="10" y="80" width="370" height="20" fill="lightgray" />
  <text x="20" y="100" baseline-shift="10">
    Grundlinienverschiebung +10
  </text>
  <line x1="10" y1="100" x2="380" y2="100" stroke="red" />

  <rect x="10" y="140" width="370" height="20" fill="lightgray" />
  <text x="20" y="160" baseline-shift="-10">
    Grundlinienverschiebung -10
  </text>
  <line x1="10" y1="160" x2="380" y2="160" stroke="red" />

  <!-- Hintergrund und Pfad -->
  <rect x="10" y="200" width="370" height="90" fill="lightgray" stroke="black" />
  <path id="textpfad" d="M 20,255 L 100,255
                         A 70 100 0 0 0 180,255
                         A 70 100 0 0 1 260,255
```

```
                    L 370,255" fill="none" stroke="red" />

  <!-- Text am Pfad -->
  <text>
    <textPath xlink:href="#textpfad">
      Hier läuft ein Text am Pfad entlang !
    </textPath>
  </text>
</svg>
```

Die erste Zeichenkette des obigen Beispiels wird ohne Grundlinienverschiebung dargestellt. Die Grundlinien der zweiten und dritten Zeichenkette sind dagegen nach oben bzw. nach unten verschoben.

Für die vierte Zeichenkette wird zuerst ein **path**-Element mit dem Bezeichner *textpfad* im Dokument platziert. Dann wird dieser Pfad durch das Attribut ***xlink:href*** im **textPath**-Element referenziert. Ergebnis: Der Pfad ist nun Grundlinie des Textes :-).

## 8.9 Darstellung von Zeichen

Das Attribut ***kerning*** stammt wieder aus der Attributgruppe **PresentationAttributes-TextContentElements**, das Attribut ***text-rendering*** gehört zur Attributgruppe **PresentationAttributes-Graphics**.

- Kerning - ***kerning***
- Anti-Aliasing - ***text-rendering***

***kerning*** Voreinstellung: *auto*. Kerning bedeutet die Bereichsunterschneidung von Zeichen. Bei einigen Schriftarten haben die einzelnen Zeichen unterschiedliche Ausmaße in der Breite - das "i" ist zum Beispiel schmaler als das "w". Mit dem Attribut ***kerning*** haben Sie in solchen Fällen die Möglichkeit die Darstellung der Zeichen Ihren Bedürfnissen anzupassen. Die Standardeinstellung *auto* ist allerdings in den meisten Fällen die beste Lösung ;-).

***text-rendering*** Voreinstellung: *optimizeQuality*. Mit Antialiasing wird die Fähigkeit des user agents bezeichnet, die Ränder von Textzeichen etwas weichgezeichnet darstellen zu können, d.h. die Kanten des Textes werden geglättet. Standardmäßig ist das Antialiasing eingeschaltet. Um es zu deaktivieren, müssen Sie das Attribut ***text-rendering*** mit dem Wert *optimizeSpeed* verwenden. Dies empfiehlt sich vor allem bei der Verwendung von kleinen Schriftgrößen. Dieses Attribut gehört zur Attributgruppe **PresentationAttributes-Graphics**.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="190"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Kerning und Aliasing</title>
<desc>Lesbarkeit des Textes verbessern</desc>
  <defs>
    <style type="text/css"><![CDATA[
      text {font-family:Verdana,sans-serif;}
      .kerning {font-size:40px; font-family:'Times New Roman';}
    ]]></style>
  </defs>

  <!-- Bereichsunterschneidung beeinflussen -->
  <text class="kerning" x="20" y="50">
    KERNING auto *
  </text>
  <text class="kerning" x="20" y="90" kerning="5">
    KERNING 5
  </text>

  <!-- Antialiasing abschalten -->
  <text x="30" y="115"
    style="text-rendering:optimizeSpeed; fill:red; font-size:10px;">
    .. die mit * gekennzeichnete ist die default-Einstellung ..
  </text>

  <text x="180" y="140" text-rendering="optimizeSpeed">
    Antialiasing ausgeschaltet
  </text>
  <text x="180" y="160">
    Antialiasing eingeschaltet *
  </text>
</svg>
```

# 9 Transformationen

In SVG können Elemente oder Objekte (Gruppierungen oder Symbole) transformiert werden. Mit Hilfe des ***transform***-Attributs sind folgende Transformationen möglich:

- Verschieben
- Skalieren
- Rotieren
- Verzerren oder Neigen

Die verschieden Transformationen werden durch unterschiedliche Wertzuweisungen für das Attribut ***transform*** realisiert. Dabei ist es auch möglich mehrere Transformantionsanweisungen hintereinander anzugeben.

Bei einer Transformation ist folgendes zu beachten:

1. Vom user agent wird zuerst ein temporäres Koordinatensystem erzeugt, welches mit dem orginären Koordinatensystem identisch ist, dessen einziger Inhalt jedoch das zu transformierende Objekt ist.
2. Dann wird die Transformationsanweisung auf dieses temporäre Koordinatensystem angewandt, d.h. das gesamte temporäre Koordinatensystem wird verändert.
3. Somit wird das Objekt ebenfalls transformiert.

Wenn Sie also mehrere Transformationen auf ein Objekt anwenden, ist die Reihenfolge der Veränderungen von enormer Bedeutung.

Im folgenden Beispiel wird ein SVG-Objekt, das Symbol "ks" verwendet, welches aus der Gruppierung "smilie1" und einem kleinen, roten Koordinatensystem, der Gruppierung "koord", besteht. Das rote Koordinatensystem repräsentiert das temporäre Koordinatensystem, in welchem sich das gesamte Objekt befindet. Dieses temporäres Koordinatensystem wird nun zuerst verschoben und dann rotiert, und danach zuerst rotiert und dann verschoben dargestellt um die die Wirkungsweise von Transformationsanweisungen zu verdeutlichen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das transform-Attribut</title>
  <desc>Anwendung und Besonderheiten von Transformationen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;}
    ]]>
    </style>

    <symbol id="ks">
    <g id="smilie1">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </g>
    <g id="koord">
      <line x1="0" y1="0" x2="100" y2="0"
```

```
      stroke="red" />
    <line x1="0" y1="0" x2="0" y2="100"
      stroke="red" />
    <rect x="0" y="0" width="3" height="3"
      fill="red" />
    <rect x="0" y="97" width="3" height="3"
      fill="red" />
    <rect x="97" y="0" width="3" height="3"
      fill="red" />
  </g>
  </symbol>
 </defs>

 <svg x="10" y="10" width="380" height="380">
  <rect x="0" y="0" width="380" height="380"
   fill="none" stroke="blue" />
  <line x1="20" y1="0" x2="400" y2="315"
   stroke="blue" />

  <use xlink:href="#ks" transform="translate(250)" />
  <use xlink:href="#ks" transform="translate(250) rotate(40)" />
  <use xlink:href="#ks" transform="rotate(40)" />
  <use xlink:href="#ks" transform="rotate(40) translate(250)" />

  <text x="200" y="115">erst verschoben dann rotiert
   <tspan x="200" y="130">translate(250) rotate(40)</tspan>
  </text>
  <text x="5" y="155">erst rotiert dann verschoben
   <tspan x="5" y="170">rotate(40) translate(250)</tspan>
  </text>
 </svg>
</svg>
```

## 9.1 Verschieben - die Anweisung translate

Sie können ein Objekt verschieben, wenn Sie dem Attribut ***transform*** als Wert die Anweisung *translate* mit Längenangaben zur Verschiebung entlang der Achse(n) zuweisen.

Die Längenangabe(n) werden in runden Klammern direkt nach der Anweisung ***translate*** gesetzt. Es sind maximal 2 Angaben (Verschiebung der x- und der y-Achse), durch Komma getrennt, möglich. Wenn Sie nur eine Längenangabe verwenden, wird diese als Verschiebungslänge entlang der x-Achse interpretiert.

Beachten Sie, dass bei Transformationen immer das gesamte temporäre Koordinatensystem transformiert wird. Der Verlauf und die Skalierung (Größe) der x- und/oder y-Achse kann also, je nach vorangegangener Transformationsanweisung, auch vom "normalen" Verlauf abweichen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das transform-Attribut</title>
<desc>translate - Verschieben</desc>
<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:Verdana,sans-serif; font-size:12px;}
  ]]>
  </style>

  <symbol id="ks">
  <g id="smilie1">
    <desc>ein lachendes Smilie</desc>
    <circle id="gesicht" cx="20" cy="20" r="15"
      fill="yellow" stroke="black" />
    <circle id="auge-links" cx="15" cy="15" r="2"
      fill="black" stroke="black" />
    <circle id="auge-rechts" cx="25" cy="15" r="2"
      fill="black" stroke="black" />
    <line id="nase" x1="20" y1="18" x2="20" y2="23"
      stroke="black" stroke-width="2" />
    <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
      stroke="black" fill="none" stroke-width="2" />
  </g>
  <g id="koord">
    <line x1="0" y1="0" x2="100" y2="0"
      stroke="red" />
    <line x1="0" y1="0" x2="0" y2="100"
      stroke="red" />
    <rect x="0" y="0" width="3" height="3"
      fill="red" />
    <rect x="0" y="97" width="3" height="3"
      fill="red" />
    <rect x="97" y="0" width="3" height="3"
      fill="red" />
  </g>
  </symbol>
</defs>

  <svg x="10" y="10" width="380" height="380">
    <rect x="0" y="0" width="380" height="380"
      fill="none" stroke="blue" />

    <use xlink:href="#ks" />
    <use xlink:href="#ks" transform="translate(150)" />
```

```
    <use xlink:href="#ks" transform="translate(0,150)" />
    <use xlink:href="#ks" transform="translate(150,150)" />

    <text x="10" y="70">keine Verschiebung</text>
    <text x="160" y="70">150 px nur x-Achse
      <tspan x="160" y="85">translate(150)</tspan>
    </text>
    <text x="10" y="220">150 px nur y-Achse
      <tspan x="10" y="235">translate(0,150)</tspan>
    </text>
    <text x="160" y="220">150 px in beide Richtungen
      <tspan x="160" y="235">translate(150,150)</tspan>
    </text>
  </svg>
</svg>
```

Im obigen Beispiel wird - das bereits bekannte - Symbol "ks" verschoben: zuerst nur entlang der x-Achse: *translate(150)*, dann nur entlang der y-Achse: *translate(0,150)* und zuletzt in beide Richtungen: *translate(150,150)*.

## 9.2 Skalieren - die Anweisung scale

Sie können ein Objekt skalieren (in seiner Größe verändern), wenn Sie dem Attribut ***transform*** als Wert die Anweisung *scale* mit der Angabe des Skalierungsfakors zuweisen.

Wenn Sie, wie im obigen Beispiel nur einen Skalierungsfaktor angeben, so wird dieser für beide Achsen des temporären Koordinatensystems verwendet. Es findet also eine proportionale Vergrößerung - durch einen Faktor größer 1 oder eine proportionale Verkleinerung - durch einen Faktor zwischen 0 und 1 statt.

Es besteht jedoch auch die Möglichkeit zwei Faktoren zu verwenden. Dann wird der erste Wert zu Skalierung der x-Achse und der zweite Wert zur Skalierung der y-Achse verwendet. Sie erzeugen dann eine nicht-proportionale Vergrößerung bzw. Verkleinerung des Objekts.

Wenn sie auf ein skaliertes Koordinatensystem weitere Transformationen anwenden, beachten Sie, dass sich die Längen im Koordinatensystem entsprechend verändert haben.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das transform-Attribut</title>
<desc>scale - vergrößern oder verkleinern</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;}
    ]]>
    </style>

    <symbol id="ks">
    <g id="smilie1">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </g>
    <g id="koord">
      <line x1="0" y1="0" x2="100" y2="0"
        stroke="red" />
      <line x1="0" y1="0" x2="0" y2="100"
        stroke="red" />
      <rect x="0" y="0" width="3" height="3"
        fill="red" />
      <rect x="0" y="97" width="3" height="3"
        fill="red" />
      <rect x="97" y="0" width="3" height="3"
        fill="red" />
    </g>
    </symbol>
  </defs>

  <svg x="10" y="10" width="380" height="380">
    <rect x="0" y="0" width="380" height="380"
      fill="none" stroke="blue" />
```

```
    <use xlink:href="#ks" transform="translate(20,10)" />
    <use xlink:href="#ks" transform="translate(20,150) scale(.5)" />
    <use xlink:href="#ks" transform="translate(150,150) scale(2)" />

    <text x="25" y="90">orginale Darstellung</text>
    <text x="25" y="250">halb so groß
      <tspan x="25" y="265">scale(.5)</tspan>
    </text>
    <text x="155" y="250">doppelt so groß
      <tspan x="155" y="265">scale(2)</tspan>
    </text>
  </svg>
</svg>
```

## 9.3 Rotieren - die Anweisung rotate

Sie können ein Objekt rotieren, wenn Sie dem Attribut ***transform*** als Wert die Anweiung *rotate* mit der Angabe des Winkels zuweisen.

Zusätzlich zu Angabe des Winkels können Sie die x,y-Koordinate des Drehpunktes angeben. Alle Werte werden durch Kommata getrennt angegeben.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das transform-Attribut</title>
<desc>rotate - rotieren</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;}
    ]]>
    </style>

    <symbol id="ks">
    <g id="smilie1">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </g>
    <g id="koord">
      <line x1="0" y1="0" x2="100" y2="0"
        stroke="red" />
      <line x1="0" y1="0" x2="0" y2="100"
        stroke="red" />
      <rect x="0" y="0" width="3" height="3"
        fill="red" />
      <rect x="0" y="97" width="3" height="3"
        fill="red" />
      <rect x="97" y="0" width="3" height="3"
        fill="red" />
    </g>
    </symbol>
  </defs>

  <svg x="10" y="10" width="380" height="380">
    <rect x="0" y="0" width="380" height="380" fill="none" stroke="blue" />

    <use xlink:href="#ks" transform="translate(100,150)" />
    <use xlink:href="#ks" transform="translate(100,150) rotate(30)" />
    <use xlink:href="#ks" transform="translate(100,150) rotate(-130)" />
    <use xlink:href="#ks" transform="translate(250,150)" />
    <use xlink:href="#ks" transform="translate(250,150) rotate(30,20,20)" />

    <text x="105" y="230">orginal</text>
    <text x="105" y="230" transform="rotate(30,105,230) translate(-40,15)">
```

```
    rotate(30)
   </text>
   <text x="105" y="230" transform="rotate(-130,105,230) translate(70,120)">
    rotate(-130)
   </text>
   <text x="255" y="230" transform="rotate(30,255,230) translate(-30)">
    rotate(30,20,20)
   </text>
  </svg>
</svg>
```

Im obigen Beispiel wird das Objekt "ks" zuerst durch zwei Transformationsanweisungen in zwei Richtungen gedreht: *rotate(30)* - positive Winkel drehen das Objekt im Uhrzeigersinn, *rotate(-130)* - negatige Winkel drehen das Objekt gegen den Uhrzeigersinn. Da der Drehpunkt bei beiden Rotationsanweisungen nicht angegeben ist, wird der Nullpunkt des temporären Koordinatensystems als Drehpunkt verwendet.

Bei der dritten Rotationsanweisung wird zusätzlich die x,y-Koordinate des Objektmittelpunkts als Drehpunkt angegeben: *rotate(30,20,20)* - Drehpunkt ist die Koordinate 20,20. Beachten Sie, dass sich die Koordinate des Objektmittelpunkts nicht verändert hat, obwohl das Objekt vor der Rotationsanweisung verschoben wurde: Es wurde ja das gesamte temporäre Koordinatensystem verschoben.

.. und auch bei Rotationsanweisungen ist darauf zu achten, dass das gesamte temporäre Koordinatensystem gedreht wird. Nachfolgende Transformationsanweisungen werden also auf eine gedrehte (!) x- und y-Achse angewandt.

## 9.4 Verzerren - die Anweisungen skewX und skewY

Um ein Objekt bzw. das temporäre Koordinatensystem des Objekts zu verzerren, stehen zwei Möglichkeiten zur Verfügung: die Anweisung *skewX* - eine Verzerrung entlang der x-Achse und die Anweisung *skewY* - eine Verzerrung entlang der y-Achse.

Sie können ein Objekt also verzerren, wenn Sie dem Attribut ***transform*** als Wert die Anweisung *skewX* und/oder die Anweisung *skewY* mit der Angabe eines Winkels zuweisen.

Dabei gibt der Winkel an, um wieviel Grad sich die jeweils "andere" Achse, auf der in der Eigenschaft genannten Achse, hin- oder wegdreht. Bei einer positiven Wikelangabe dreht sich die jeweils "andere" Achse zur genannten Achse hin: es entsteht eine Stauchung. Bei einer negativen Winkelangabe dreht sich die jeweils "andere" Achse von der genannten Achse weg: es entsteht eine Streckung.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das transform-Attribut</title>
<desc>skewX und skewY - Verzerren</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;}
    ]]>
    </style>

    <symbol id="ks">
    <g id="smilie1">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </g>
    <g id="koord">
      <line x1="0" y1="0" x2="100" y2="0"
        stroke="red" />
      <line x1="0" y1="0" x2="0" y2="100"
        stroke="red" />
      <rect x="0" y="0" width="3" height="3"
        fill="red" />
      <rect x="0" y="97" width="3" height="3"
        fill="red" />
      <rect x="97" y="0" width="3" height="3"
        fill="red" />
    </g>
    </symbol>
  </defs>

  <svg x="10" y="10" width="380" height="380">
    <rect x="0" y="0" width="380" height="380" fill="none" stroke="blue" />

    <use xlink:href="#ks" transform="translate(50,50)" />
    <use xlink:href="#ks"
```

```
    transform="translate(50,50) skewX(25)" />
  <use xlink:href="#ks" transform="translate(200,50)" />
  <use xlink:href="#ks" transform="translate(200,50)
    skewY(25)" />

  <use xlink:href="#ks" transform="translate(50,200)" />
  <use xlink:href="#ks"
    transform="translate(50,200) skewY(25) skewX(25)" />
  <use xlink:href="#ks" transform="translate(200,200)" />
  <use xlink:href="#ks"
    transform="translate(200,200) skewY(-25) skewX(-25)" />

  <text x="55" y="140">skewX(25)</text>
  <text x="205" y="140">skewY(25)</text>
  <text x="55" y="290">skewX(25) skewY(25)</text>
  <text x="205" y="290">skewX(-25) skewY(-25)</text>
 </svg>
</svg>
```

Im obigen Beispiel wird unser Symbol "ks" zuerst um 25 Grad zur X-Achse gestaucht, d.h. die y-Achse dreht sich um 25 Grad auf die X-Achse zu: *skewX(25)*, dann wird es um 25 Grad zu Y-Achse gestaucht, die x-Achse dreht sich also um 25 Grad auf die Y-Achse zu: *skewY(25)*.

Die unteren beiden Verzerrungen zeigen eine Stauchung und eine Streckung des Objekts entlang beider Achsen. Die Reihenfolge der Verwendung von *skewX* und *skewY* ist - für Transformationsanweisungen ungewöhnlich - nicht von Bedeutung.

Wenn Sie allerdings andere, weitere Transformationen auf ein Objekt anwenden möchten, sollten sie darauf achten, dass sich der Verlauf der Achsen geändert hat.

## 9.5 Transformtion der Matrix - die Anweisung matrix

Bei allen vorangegangenen Transformationsarten wurde die Matrix des temporären Koordinatensystems auf eine besondere Art verändert. Das diese speziellen Transformationsarten, wie Verschieben, Skalieren, Rotieren und Verzerren durch eigene, einfach zu verwendende Anweisungen angewendet werden können, ist ein Komfort von SVG. Denn: Alle Transformationen lassen sich auch durch Matrix-Transformationen erzeugen.

Allerdings ist die Anwendung von Matrix-Transformationen eher etwas für diplomierte Mathematiker. Daher verweise ich alle Wißbegierigen zum entsprechenden Kapitel in der SVG Specification des W3C.

## 10 Pfade

Pfade bieten die Möglichkeit alle erdenklichen Linienformen (offenen Pfade) oder Objektformen (geschlossene Pfade) zu erzeugen. So können auch alle 6 Grundformen von SVG durch einen Pfad erzeugt werden.

Pfade werden mit Hilfe des **path**-Elements festgelegt.

Den eigentlichen Verlauf des Pfades bestimmen Sie durch das, im **path**-Element zwingend anzugebende Attribut ***d***. Das Attribut ***d*** erwartet als Wert einen alphanumerischen Wert, wobei die Buchstaben jeweils den Verlauf des Pfades festlegen und die Zahlen die notwendigen Koordinaten-, Winkel- oder sonstige Angaben für den entsprechenden "Verlaufsbuchstaben" bestimmen. Die Buchstaben und Zahlen können durch ein Leerzeichen und/oder ein Kommazeichen beliebig von einander getrennt werden.

Folgende Buchstaben bzw. Pfadverläufe sind möglich:

- moveto: *M*, *m* - Startpunkt (das Aufsetzen des imaginären Stiftes)
- lineto: *L*, *l* und *H*, *h* und *V*, *v* - eine gerade Linie
- closepath: *Z*, *z* - eine Pfad schließen
- cubic Bézier curve: *C*, *c* und *S*, *s* - eine kubische Bézier-Kurve
- quadratic Bézier curve: *Q*, *q* und *T*, *t* - eine quadratische Bézier-Kurve
- elliptical arc curve: *A*, *a* - einen elliptischen Bogen

Großbuchstaben leiten eine absolute Angabe ein, d.h. alle nachfolgenden Werte werden als absolut im Koordinatensystem interpretiert. Kleinbuchstaben leiten eine relative Angabe ein. Hier werden alle Werte im Bezug zum direkt vorher definierten Punkt des Pfades interpretiert.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="360" height="160"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>Pfade mit absolutem und relativem Verlauf</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:12px; fill:black;}
      path {fill:red; stroke:black;}
    ]]>
    </style>
  </defs>

  <path d="M 50,10 L 10,50 L 90,50 Z" />
  <path d="M 250,10 l -40,40 l 80,0 z" />

  <text x="10" y="70">absolute Positionierung
  <tspan x="10" dy="20">Moveto M: 50,10</tspan>
  <tspan x="10" dy="16">Lineto L: 10,50</tspan>
  <tspan x="10" dy="16">Lineto L: 90,50</tspan>
  <tspan x="10" dy="16">Closepath Z</tspan>
  </text>
  <text x="210" y="70">relative Positionierung
  <tspan x="210" dy="20">Moveto M: 250,10</tspan>
  <tspan x="210" dy="16">Lineto l: -40,40</tspan>
  <tspan x="210" dy="16">Lineto l: 80,0</tspan>
  <tspan x="210" dy="16">Closepath z</tspan>
  </text>
</svg>
```

Das obige Beispiel zeigt zwei gefüllte Pfade, die jeweils ein Dreieck darstellen. Bis auf den Startpunkt sind die beiden Dreiecke sind identisch.

Das erste Dreieck wird durch absolute Angaben festgelegt: das Aufsetzen des Stifts M am absoluten Punkt (50,10), dann eine gerade Linie L zum absoluten Punkt (10,50), dann wieder eine gerade Linie L zum absoluten Punkt (90,50), zuletzt das Schließen des Pfades mit Z.

Das zweite Dreieck wurde mit relativen Angaben festgelegt. *M 250,10 l -40,40 l 80,0 z*. Um die absoluten Punkte dieses Dreiecks zu ermitteln benötigen sie nur ein wenig Addition. Die erste lineto Anweisung *l -40,40* bezieht sich auf den zuletzt festgelegten Punkt, also den Startpunkt. Die relative Anweisung muß mit den Koordinaten des vorherigen Punktes auf folgende Weise addiert werden: (-40)+250 und (40)+10. Der absoluten Punkt ist also (210,50). Die zweite lineto Anweisung *l 80,0* bezieht sich wieder auf zuletzt festgelegt Punkt, also den Punkt (210,50). Die notwendigen Berechnungen sind also: (80)+210 und (0)+50. Der absolute Punkt ist demzufolge (290,50). Das zweite Dreieck hätte daher auch mit folgenden Angaben definiert werden können: *M 250,10 L 210,50 L 290,50 Z*.

Wenn Sie jetzt in der Lage sind, das erste Dreieck mit relativen Angaben zu erstellen, haben Sie das Prinzip der Groß- und Kleinbuchstaben für Pfadverläufe verstanden ;-).

## 10.1 M und m - die Moveto Anweisung

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>moveto</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:12px; fill:black;}
      path {fill:none; stroke:red; stroke-width:2px;}
    ]]>
    </style>
  </defs>

  <path d="M 80,90
           l -70,0
           l 0,20
           l 70,0
           M 110,120
           l 0,70
           l -20,0
           l 0,-70" />

  <text x="15" y="40" style="font-weight:bold; fill:red; font-size:14px;">
  Ein (1!) Pfad mit 2 Startpunkten
  </text>
  <circle cx="80" cy="90" r="3" />
  <text x="80" y="85">Startpunkt 1: M 80,90</text>
  <circle cx="110" cy="120" r="3" />
  <text x="110" y="115">Startpunkt 2: M 110,120</text>
</svg>
```

Durch die moveto-Anweisung mit Hilfe der Buchstben *M* oder *m* wird der "virtuelle Stift" am nachfolgend angegebenen Koordinatenpunkt aufgesetzt.

Am Anfang einer Pfaddefinition durch das ***d***-Attribut muß immer eine moveto-Anweisung platziert werden. Eine Pfaddefinition fängt also immer mit einem *M* und einem nachfolgenden Koordinatenpunkt an.

Wenn die Pfaddefinition eine weitere moveto-Anweisung enthält, so springt der "virtuelle Stift" vom zuletzt festgelegen Punkt direkt zu dem mit *M* der *m* angegebenen Koordinatenpunkt - ohne eine Spur zu hinterlassen.

Im obigen Beispiel wurde genau ein unterbrochener Pfad definiert. Der "virtuelle Stift" wird einmal neu aufgesetzt. An Stelle der absoluten Anweisung *M* hätte natürlich auch die relative Anweisung *m* verwendet werden können. Der letzte Punkt des Pfades vor der moveto-Anweisung wird durch eine relative lineto-Anweisung bestimmt und liegt absolut in 80,110. Daher müßte die entsprechende relative moveto-Anweisung folgendermaßen festgelegt werden: *m 30,10*. Dies entspricht dem absoluten Koordinatenpunkt 110,120.

Beachten Sie: Auch unterbrochene Pfade können gefüllt sein, wenn die Teile des Pfades Flächen einschließen - diese werden dann gefüllt. Damit der Pfad also nicht schwarz gefüllt dargestellt wird (Voreinstellung) muß ***fill*** ausdrücklich mit dem Wert *none* gesetzt werden.

## 10.2 L und l, H und h, V und v - die lineto Anweisungen

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="220" height="220"
 xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>das path-Element</title>
 <desc>lineto</desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana; font-size:12px; fill:black;}
   path {fill:none; stroke:saddlebrown; stroke-width:2px;}
  ]]>
  </style>
 </defs>

 <path d="M 20,190
        v -90
        l 80,-80
        l 80,80
        h -160
        l 160,80
        h -160
        l 160,-80
        v 70" />

 <circle cx="20" cy="190" r="3" />
 <text x="25" y="202">Start</text>
 <circle cx="180" cy="170" r="3" />
 <text x="185" y="167">Ende</text>
 <circle cx="20" cy="136" r="9" fill="saddlebrown" />
 <text x="16.5" y="140" style="fill:white;">1</text>
 <circle cx="64" cy="56" r="9" fill="saddlebrown" />
 <text x="60" y="60" style="fill:white;">2</text>
 <circle cx="136" cy="56" r="9" fill="saddlebrown" />
 <text x="132" y="60" style="fill:white;">3</text>
 <circle cx="104" cy="100" r="9" fill="saddlebrown" />
 <text x="100" y="104" style="fill:white;">4</text>
 <circle cx="64" cy="120" r="9" fill="saddlebrown" />
 <text x="60" y="124" style="fill:white;">5</text>
 <circle cx="104" cy="180" r="9" fill="saddlebrown" />
 <text x="100" y="184" style="fill:white;">6</text>
 <circle cx="64" cy="158" r="9" fill="saddlebrown" />
 <text x="60" y="162.5" style="fill:white;">7</text>
 <circle cx="180" cy="136" r="9" fill="saddlebrown" />
 <text x="176" y="140" style="fill:white;">8</text>
</svg>
```

SVG stellt 3 unterschiedliche lineto-Anweisungen zu Verfügung:

- *L* oder *l* - erzeugt eine beliebige gerade Linie
- *V* oder *v* - erzeugt eine beliebige vertikale gerade Linie
- *H* oder *h* - erzeugt eine beliebige horizontale gerade Linie

Wie für alle Pfadanweisung gilt auch hier: Großbuchstaben für absolute Angaben, Kleinbuchstaben für relative Angaben.

Die lineto-Anweisung *L* oder *l* erwartet die nachfolgende Angabe eines Koordinatenpunkts. Der user agent zeichnet dann eine gerade Linie vom vorherigen Punkt des Pfades zu dem in der lineto-Anweisung angegebenen Koordinatenpunkt.

Die lineto-Anweisung *V* oder *v* zeichnet eine vertikale Linie. Nachfolgend muß daher nur die y-Koordinate angegeben werden. Die lineto-Anweisung *H* oder *h* zeichnet eine horizontale Linie. Nachfolgend muß daher nur die x-Koordinate angegeben werden.

Aber natürlich ist es ebenso möglich vertikale oder horizontale Linien mit der lineto-Anweisung *L* oder *l* zu erzeugen. Bei relativen Angaben genügt es, entweder die x- oder die y-Koordinate mit dem Wert 0 festzulegen. Bei absoluten Angaben muß der vorherige Wert der x- oder y-Koordinate verwendet werden.

Im obigen Beispiel wird mit 8 relativen lineto-Anweisungen das "Haus vom Nikolaus" gezeichnet. Eine schöne Übung: das Häuschen mal mit absoluten Angaben zu erzeugen ...

## 10.3 Z und z - die closepath Anweisung

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>closepath</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:12px; fill:black;}
      path {fill:red; stroke:black; stroke-width:2px;}
    ]]>
    </style>
  </defs>
  <rect x="1" y="1" width="198" height="198"
    style="fill:dodgerblue;" />

  <path d="M 10,80
           v 40
           h 70
           v 70
           h 40
           v -70
           h 70
           v -40
           h -70
           v -70
           h -40
           v 70
           z" />

  <text x="20" y="98" style="fill:white;">
    closepath z
  </text>
  <line x1="10" y1="80" x2="80" y2="80"
    style="stroke:white; stroke-width:2px;" />
  <circle cx="10" cy="80" r="3" />
  <text x="5" y="72">Start</text>
</svg>
```

Durch die closepath-Anweisung *Z* oder *z* wird ein Pfad geschlossen. Der user agent zeichnet eine gerade Linie vom zuletzt festgelegten Punkt zu dem mit *M* festgelegent Startpunkt des Pfades.

Im obigen Beispiel ist die mit Hilfe der closepath-Anweisung erzeugte Linie weiß dargestellt.

## 10.4 C und c - kubische Bezierkurven

Die Bézier-Kurve ist grundsätzlich eines der wichtigsten grafischen Elemente, weil fast alle Formen und Zeichenumrisse aus ihren Segmenten zusammengesetzt werden können. Die Bézier-Kurve ist nach Pierre Bézier benannt, der als Pionier des CAD-Bereiches zwischen 1960 und 1970 für den Französischen Autohersteller Renault eine Modellierungshilfe für Autokarosserien entwickelte. (gegen Rost hat das aber auch nicht geholfen, A.d.Lektors)

Bézier-Kurven sind Kurvenverläufe, die durch zwei Endpunkte einer Kurve und die Besonderheiten der an ihnen angelegten Tangenten definiert sind. Die Gestalt einer Bézier-Kurve wird durch die Position ihrer Kontrollpunkte und der daraus resultierenden Neigung der so festgelegten Tangenten beeinflusst.

SVG unterscheidet zwischen einer kubischen Bézier-Kurve, die durch zwei Kontrollpunkte definiert wird und einer quadratischen Bézier-Kurve, die durch lediglich einen Kontrollpunkt definiert wird, durch welchen allerdings ebenfalls 2 Tangenten festgelegt werden.

Kubische Bézier-Kurven werden mit der Anweisung *C* oder *c* definiert. Außerdem gibt es in SVG eine Kurzform für die Erstellung von kubischen Bézier-Kurven: *S* oder *s*. Diese wird im nächsten Kapitel erläutert.

Wenn Sie eine Bézier-Kurve mit *C* oder *c* erstellen ist folgendes zu beachten: der Anfangspunkt der Bézier-Kurve ist immer der direkt zuvor festgelegte Punkt (Startpunkt *M*, Endpunkt einer vorherigen lineto-Anweisung, Endpunkt einer vorherigen Kurven-Anweisung, ...).

Die Anweisung *C* oder *c* erwartet 3 nachfolgende Koordinatenpunkte:

- die Koordinaten des ersten Kontrollpunktes(die imaginäre Linie, die zwischen dem Anfangspunkt der Bézier-Kurve und dem ersten Kontrollpunkt verläuft, bildet die erste Tangente, welche den Verlauf der Kurve beeinflußt),
- die Koordinaten des zweiten Kontrollpunktes(die imaginäre Linie, die zwischen dem Endpunkt der Bézier-Kurve und dem zweiten Kontrollpunkt verläuft, bildet die zweite Tangente, welche den Verlauf der Kurve beeinflußt),
- die Koordinaten des Endpunkts.

Die oben angegebene Reihenfolge der Koordinatenpunkte muß eingehalten werden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="460" height="280"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>cubic bezier curves - C und c</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:14px; fill:black;}
      path {fill:none; stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
      circle {fill:black;}
    ]]>
    </style>
  </defs>

  <text x="20" y="30" style="font-size:16px;">
  <tspan style="font-weight:bold">C|c</tspan>
    &apos;Kontrollpunkt 1&apos;
    &apos;Kontrollpunkt 2&apos;
    &apos;Endpunkt&apos;
  </text>

  <path d="M 50,150
```

```
        c 0,-50 150,-50 150,0" />

  <circle cx="50" cy="150" r="3" />
  <circle cx="50" cy="100" r="3" />
  <circle cx="200" cy="100" r="3" />
  <circle cx="200" cy="150" r="3" />
  <line x1="50" y1="150" x2="50" y2="100" />
  <line x1="200" y1="150" x2="200" y2="100" />

  <text x="50" y="170">M 50,150</text>
  <text x="50" y="70">Kontrollpunkt 1</text>
  <text x="50" y="90"><tspan style="font-weight:bold;">
    c
  </tspan> 0,-50</text>
  <text x="200" y="70">Kontrollpunkt 2</text>
  <text x="200" y="90">150,-50 </text>
  <text x="200" y="170">150,0</text>
  <text x="200" y="190">Endpunkt</text>
  <text x="50" y="230">absolute Angabe:
  <tspan x="70" dy="20">M 50,150
  <tspan style="font-weight:bold;">
    C
  </tspan> 50,100 200,100 200,150
  </tspan>
  </text>
</svg>
```

Im obigen Beispiel sind die Kontrollpunkte und die dadurch entstandenen Tangenten der Kurve zum besseren Verständnis ebenfalls dargestellt. Die Kurve wird mit der Anweisung *c* (Kleinbuchstabe) erstellt, d.h. die 3 folgenden Koordinatenpunkte sind relativer Natur und beziehen sich alle auf den direkt zuvor festgelegten Punkt des Pfades: in diesem Falle auf den, mit *M* festgelegten Startpunkt des Pfades.

Die Lage der Kontrollpunkte bestimmt die Länge und den Verlauf der Tangenten. Die Länge und der Verlauf der Tangenten bestimmen wiederum den Verlauf der Kurve.

Im folgenden SVG Dokument sind eine Anzahl von kubischen Bézier-Kurven definiert. Sie zeigen mögliche Verläufe einer mit *c* oder *C* definierten Kurve durch unterschiedlichste Tangenten bzw. Kontrollpunkte.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="510" height="580"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>cubic bezier curves - einige Beispiele mit c</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:14px; fill:black;}
      path {fill:none; stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
      circle {fill:black;}
    ]]>
    </style>
  </defs>

  <!-- Kurve 1 -->
  <text x="90" y="30">Kurve 1</text>
  <path d="M 160,10
        c 50,20 150,60 200,80" />

  <circle cx="160" cy="10" r="3" />
```

```
  <circle cx="210" cy="30" r="3" />
  <circle cx="310" cy="70" r="3" />
  <circle cx="360" cy="90" r="3" />
  <line x1="160" y1="10" x2="210" y2="30" />
  <line x1="360" y1="90" x2="310" y2="70" />

  <!-- Kurve 2 -->
  <text x="90" y="150">Kurve 2</text>
  <path d="M 160,130
        c 50,-60 210,20 200,80" />

  <circle cx="160" cy="130" r="3" />
  <circle cx="210" cy="70" r="3" />
  <circle cx="370" cy="150" r="3" />
  <circle cx="360" cy="210" r="3" />
  <line x1="160" y1="130" x2="210" y2="70" />
  <line x1="360" y1="210" x2="370" y2="150" />

  <!-- Kurve 3 -->
  <text x="90" y="270">Kurve 3</text>
  <path d="M 160,250
        c 130,-50 120,-30 200,80" />

  <circle cx="160" cy="250" r="3" />
  <circle cx="290" cy="200" r="3" />
  <circle cx="280" cy="220" r="3" />
  <circle cx="360" cy="330" r="3" />
  <line x1="160" y1="250" x2="290" y2="200" />
  <line x1="360" y1="330" x2="280" y2="220" />

  <!-- Kurve 4 -->
  <text x="90" y="390">Kurve 4</text>
  <path d="M 160,370
        c 190,-30 -10,-170 200,80" />

  <circle cx="160" cy="370" r="3" />
  <circle cx="350" cy="340" r="3" />
  <circle cx="150" cy="200" r="3" />
  <circle cx="360" cy="450" r="3" />
  <line x1="160" y1="370" x2="350" y2="340" />
  <line x1="360" y1="450" x2="150" y2="200" />

  <!-- Kurve 5 -->
  <text x="90" y="510">Kurve 5</text>
  <path d="M 160,490
        c 330,-50 -140,-190 200,80" />

  <circle cx="160" cy="490" r="3" />
  <circle cx="490" cy="440" r="3" />
  <circle cx="20" cy="300" r="3" />
  <circle cx="360" cy="570" r="3" />
  <line x1="160" y1="490" x2="490" y2="440" />
  <line x1="360" y1="570" x2="20" y2="300" />
</svg>
```

## 10.5 S und s - Kurzform fuer kubische Bezierkurven

Die Kurzform für kubische Bézier-Kurven wird mit der Anweisung *S* oder *s* eingeleitet. Diese Anweisung erwartet nachfolgend nur 2 Koordinatenpunkte:

- die Koordinaten des zweiten Kontrollpunktes(die imaginäre Linie, die zwischen dem Endpunkt der Bézier-Kurve und dem zweiten Kontrollpunkt verläuft, bildet die zweite Tangente, welche den Verlauf der Kurve beeinflußt),
- die Koordinaten des Endpunkts.

Der erste Kontrollpunkt wird automatisch aus dem letzten Kontrollpunkt der direkt zuvor festgelegten Bézier-Kurve generiert.

Die so entstandene Tangente entspricht in der Länge der Tangente des zuvor festgelegen Kontrollpunkts, verläuft aber in genau entgegengesetzte Richtung.Sie ist also das "Spiegelbild" der vorherigen Tangente.

Die Verwendung der Anweisungen *S* oder *s* sind daher nur im Zusammenhang mit einer vorangegangen *C* oder *c* Anweisung sinnvoll.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="350"
 xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>das path-Element</title>
 <desc>cubic bezier curves - S und s</desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana; font-size:14px; fill:black;}
   path {fill:none; stroke:red; stroke-width:2px;}
   line {fill:none; stroke:black;}
   circle {fill:black;}
  ]]>
  </style>
 </defs>
 <text x="20" y="30" style="font-size:16px;">
 <tspan style="font-weight:bold">S|s</tspan>
 &apos;Kontrollpunkt&apos; &apos;Endpunkt&apos;
 </text>

 <path d="M 50,150
       c 0,-50 150,-50 150,0
       s 150,50 150,0" />

 <circle cx="50" cy="150" r="3" />
 <circle cx="50" cy="100" r="3" />
 <circle cx="200" cy="100" r="3" />
 <circle cx="200" cy="150" r="3" />
 <circle cx="200" cy="200" r="3"
  style="fill:none;stroke:black;" />
 <circle cx="350" cy="200" r="3" />
 <circle cx="350" cy="150" r="3" />
 <line x1="50" y1="150" x2="50" y2="100" />
 <line x1="200" y1="150" x2="200" y2="100" />
 <line x1="200" y1="150" x2="200" y2="200"
  style="stroke-dasharray:8,2,8,2,8,2,8,2,8,2;" />
 <line x1="350" y1="200" x2="350" y2="150" />
 <text x="50" y="170">M 50,150</text>
 <text x="50" y="70">Kontrollpunkt 1</text>
 <text x="50" y="90"><tspan
  style="font-weight:bold;">c</tspan> 0,-50</text>
```

```
  <text x="200" y="70">Kontrollpunkt 2</text>
  <text x="200" y="90">150,-50 </text>
  <text x="210" y="145">Endpunkt <tspan
    style="font-weight:bold;">c</tspan></text>
  <text x="210" y="165">150,-50 </text>
  <text x="200" y="220"><tspan
    style="font-weight:bold;">s</tspan></text>
  <text x="200" y="240">generierter
  <tspan x="200"dy="20">
    Kontrollpunkt
  </tspan>
  </text>
  <text x="360" y="205">150,50</text>
  <text x="360" y="225">Kontrollpunkt</text>
  <text x="360" y="155">150,0</text>
  <text x="360" y="175">Endpunkt</text>
  <text x="50" y="290">absolute Angabe:
  <tspan x="70" dy="20">M 50,150
  <tspan style="font-weight:bold;">
    C
  </tspan> 50,100 200,100 200,150
  </tspan>
  <tspan x="142" dy="20">
  <tspan style="font-weight:bold;">S</tspan> 350,200 350,150
  </tspan>
  </text>
</svg>
```

## 10.6 Q und q - quadratische Bezierkurven

Quadratische Bézier-Kurven werden mit der Anweisung *Q* oder *q* definiert. Für die Erstellung von quadratischen Bézier-Kurven gibt es, analog zu den kubischen Bézier-Kurven, ebenfalls eine Kurzform: *T* oder *t*. Diese wird im folgenden Kapitel erläutert.

Wenn Sie eine Bézier-Kurve mit *Q* oder *q* erstellen ist folgendes zu beachten: der Anfangspunkt der Bézier-Kurve ist, wie auch bei kubischen Bézier-Kurven, immer der direkt zuvor festgelegte Punkt des Pfades (Startpunkt *M* oder Endpunkt einer vorherigen lineto-Anweisung oder Endpunkt einer vorherigen Kurven-Anweisung oder ...).

Die Anweisung *Q* oder *q* erwartet 2 nachfolgende Koordinatenpunkte:

- die Koordinaten des einzigen Kontrollpunktes(die imaginäre Linie, die zwischen dem Anfangspunkt der Bézier-Kurve und dem einzigen Kontrollpunkt verläuft, bildet die erste Tangente, welche den Verlauf der Kurve beeinflußt; die imaginäre Linie, die zwischen dem Endpunkt der Bézier-Kurve und dem einzigen Kontrollpunkt verläuft, bildet die zweite Tangente, welche den Verlauf der Kurve beeinflußt),
- die Koordinaten des Endpunkts.

Die oben angegebenen Reihenfolge der Koordinatenpunkte muß eingehalten werden.

Wenn Sie versuchen quadratische Bézier-Kurven mit der Anweisung *C* oder *c* zu erstellen, d.h. wenn die 2 festzulegenden Kontrollpunkte der kubischen Bézier-Kurve mit ihren Koordinaten übereinstimmen, werden Sie allerdings festellen, daß die kubische Kurve eine weitaus stärkere Ausbuchtung besitzt. Für diesen Hinweis Dank an Tobias.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="460" height="280"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>quadratic bezier curves - Q und q</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:14px; fill:black;}
      path {fill:none; stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
      circle {fill:black;}
    ]]>
    </style>
  </defs>
  <text x="20" y="30" style="font-size:16px;">
  <tspan style="font-weight:bold">Q|q</tspan>
  Kontrollpunkt Endpunkt
  </text>

  <path d="M 50,150
         q 75,-75 150,0" />

  <circle cx="50" cy="150" r="3" />
  <circle cx="200" cy="150" r="3" />
  <circle cx="125" cy="75" r="3" />
  <line x1="50" y1="150" x2="125" y2="75" />
  <line x1="200" y1="150" x2="125" y2="75" />
  <text x="50" y="170">M 50,150</text>
  <text x="145" y="70">
  <tspan style="font-weight:bold">q</tspan> 75,-75
  <tspan x="145" dy="20">Kontrollpunkt</tspan>
  </text>
```

```
  <text x="210" y="160">150,0
  <tspan x="210" dy="20">Endpunkt</tspan>
  </text>
  <text x="50" y="230">absolute Angabe:
  <tspan x="70" dy="20">M 50,150
  <tspan style="font-weight:bold;">Q</tspan> 125,75 200,150
  </tspan>
  </text>
</svg>
```

Das wichtigste Merkmal einer quadratische Bézier-Kurve: Es ist lediglich ein (1) Kontrollpunkt festzulegen. Die beiden Tangenten verlaufen dann jeweils durch den einzigen Kontrollpunkt und dem Start- bzw. Endpunkt der Kurve.

Im obigen Beispiel wird die quadratische Bézier-Kurve mit Hilfe von relativen Angaben erstellt: *M 50,150 q 75,-75 150,0*. Alternativ können Sie die Kurve natürlich auch durch absolute Angaben erzeugen: *M 50,150 Q 125,75 200,150*.

In der folgenden Beispiel-Grafik finden Sie einige Beispiele für quadratische Bézier-Kurven.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="510" height="580"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>cubic bezier curves - einige Beispiele mit q</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:14px; fill:black;}
      path {fill:none; stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
      circle {fill:black;}
    ]]>
    </style>
  </defs>

  <!-- Kurve 1 -->
  <text x="90" y="30">Kurve 1</text>
  <path d="M 160,10
          q 100,40 200,80" />

  <circle cx="160" cy="10" r="3" />
  <circle cx="260" cy="50" r="3" />
  <circle cx="360" cy="90" r="3" />
  <line x1="160" y1="10" x2="260" y2="50" />
  <line x1="360" y1="90" x2="260" y2="50" />

  <!-- Kurve 2 -->
  <text x="90" y="150">Kurve 2</text>
  <path d="M 160,130
          q 100,-60 200,80" />

  <circle cx="160" cy="130" r="3" />
  <circle cx="260" cy="70" r="3" />
  <circle cx="360" cy="210" r="3" />
  <line x1="160" y1="130" x2="260" y2="70" />
  <line x1="360" y1="210" x2="260" y2="70" />

  <!-- Kurve 3 -->
  <text x="90" y="270">Kurve 3</text>
```

```
  <path d="M 160,250
        q 0,-90 200,80" />

  <circle cx="160" cy="250" r="3" />
  <circle cx="160" cy="160" r="3" />
  <circle cx="360" cy="330" r="3" />
  <line x1="160" y1="250" x2="160" y2="160" />
  <line x1="360" y1="330" x2="160" y2="160" />

  <!-- Kurve 4 -->
  <text x="90" y="390">Kurve 4</text>
  <path d="M 160,370
        q 300,0 200,80" />

  <circle cx="160" cy="370" r="3" />
  <circle cx="460" cy="370" r="3" />
  <circle cx="360" cy="450" r="3" />
  <line x1="160" y1="370" x2="460" y2="370" />
  <line x1="360" y1="450" x2="460" y2="370" />

  <!-- Kurve 5 -->
  <text x="90" y="510">Kurve 5</text>
  <path d="M 160,490
        q -140,-120 200,80" />

  <circle cx="160" cy="490" r="3" />
  <circle cx="20" cy="370" r="3" />
  <circle cx="360" cy="570" r="3" />
  <line x1="160" y1="490" x2="20" y2="370" />
  <line x1="360" y1="570" x2="20" y2="370" />
</svg>
```

## 10.7 T und t - Kurzform fuer quadratische Bezierkurven

Die Kurzform für quadratische Bézier-Kurven wird mit der Anweisung *T* oder *t* eingeleitet. Diese Anweisung erwartet nachfolgend nur einen (1) Koordinatenpunkt:

- die Koordinaten des Endpunkts.

Der einzige Kontrollpunkt wird automatisch aus dem letzten Kontrollpunkt der direkt zuvor festgelegten quadratischen Bézier-Kurve generiert.

Die erste so entstandene Tangente entspricht in der Länge der letzten Tangente des zuvor festgelegen Kontrollpunkts, verläuft aber in genau entgegengesetzte Richtung.Sie ist also das "Spiegelbild" der vorherigen Tangente.

Die Verwendung der Anweisungen *T* oder *t* sind daher nur im Zusammenhang mit einer vorangegangen *Q* oder *q* Anweisung sinnvoll.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="350"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>quadratic bezier curves - T und t</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:14px; fill:black;}
      path {fill:none; stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
      circle {fill:black;}
    ]]>
    </style>
  </defs>
  <text x="20" y="30" style="font-size:16px;">
  <tspan style="font-weight:bold">T|t</tspan>
  Endpunkt
  </text>

  <path d="M 50,150
           q 75,-75 150,0
           t 150,0" />

  <line x1="50" y1="150" x2="125" y2="75" />
  <line x1="200" y1="150" x2="125" y2="75" />
  <line x1="200" y1="150" x2="275" y2="225"
    style="stroke-dasharray:8,2;" />
  <line x1="350" y1="150" x2="275" y2="225"
    style="stroke-dasharray:8,2;" />
  <circle cx="50" cy="150" r="3" />
  <circle cx="200" cy="150" r="3" />
  <circle cx="125" cy="75" r="3" />
  <circle cx="275" cy="225" r="3"
    style="fill:white; stroke:black;" />
  <circle cx="350" cy="150" r="3" />
  <text x="50" y="170">M 50,150</text>
  <text x="145" y="65">
  <tspan style="font-weight:bold">q</tspan> 75,-75
  <tspan x="145" dy="20">Kontrollpunkt</tspan>
  </text>
  <text x="210" y="130">150,0
  <tspan x="210" dy="20">Endpunkt q</tspan>
```

```
  </text>
  <text x="290" y="230">generierter
  <tspan x="290" dy="20">Kontrollpunkt</tspan>
  </text>
  <text x="360" y="160">
  <tspan style="font-weight:bold;">t</tspan> 150,0
  <tspan x="360" dy="20">Endpunkt</tspan>
  </text>
  <text x="50" y="275">absolute Angabe:
  <tspan x="70" dy="20">M 50,150
  <tspan style="font-weight:bold;">Q</tspan> 125,75 200,150
  </tspan>
  <tspan x="143" dy="20">
  <tspan style="font-weight:bold;">T</tspan> 350,150
  </tspan>
  </text>
</svg>
```

## 10.8 A und a - Bogenkurven

Mit den Anweisungen *A* oder *a* können Sie eine elliptische Bogenkurve erzeugen. Bei einer elliptischen Bogenkurve wird zuerst eine Ellipse festgelegt, um welche dann der Pfad bzw. die Kurve verläuft. Die Bogenkurve stellt quasi Teile der Umrandung der Ellipse dar.

Der Anweisung *A* oder *a* müssen 5 Werte und ein Koordinatenpunkt folgen:

1. der Radius der x-Achse der Ellipse
2. der Radius der y-Achse der Ellipse
3. die Rotation der x-Achse der Ellipse in Grad(0 bedeutet keine Rotation)
4. das large-arc-flag: 0 für den kurzen Weg um die Ellipse 1 für den langen Weg um die Ellipse
5. das sweep-flag: 0 für Zeichnung entgegen den Uhrzeigersinn 1 für Zeichnung mit dem Uhrzeigersinn
6. die Koordinaten des Endpunktes.

Die Reihenfolge ist unbedingt einzuhalten. Der Mittelpunkt der Ellipse wird aus allen Werten von A durch den user agent generiert. Anfangspunkt der elliptischen Bogenkurve ist immer der direkt zuvor festgelegte Punkt des Pfades.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="350"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das path-Element</title>
  <desc>elliptical arc curve - A und a</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana; font-size:14px;}
      path {fill:none; stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
    ]]>
    </style>
  </defs>
  <g transform="scale(1.7)">

    <path d="M 150,100
             A 70 50 0 1 0 250,150" />
    <path d="M 150,100
             A 70 50 0 1 1 250,150"
             style="stroke:gray; stroke-width:1px;" />
    <path d="M 150,100
             A 70 50 0 0 0 250,150"
             style="stroke:blue; stroke-width:1px;" />
    <path d="M 150,100
             A 70 50 0 0 1 250,150"
             style="stroke:green; stroke-width:1px;" />

  <!-- Die Winkellinien -->
  <path d="M 111.5,145.8 h 70 v 50"
          style="stroke:gray; stroke-width:.5px;" />
  </g>

  <text x="20" y="30">Startpunkt: M 150,100</text>
  <text x="20" y="55">
  <tspan style="font-weight:bold">elliptische Bogenkurve A</tspan>
  <tspan x="20" dy="20">Radius x: 70 Pixel</tspan>
  <tspan x="20" dy="16">Radius y: 50 Pixel</tspan>
  <tspan x="20" dy="16">Rotation x: 0 Grad</tspan>
  <tspan x="20" dy="16" style="fill:red;">langer Weg: 1, gegen Uhr: 0</tspan>
```

```
  <tspan x="20" dy="16" style="fill:gray;">langer Weg: 1, mit Uhr: 1</tspan>
  <tspan x="20" dy="16" style="fill:blue;">kurzer Weg: 0, gegen Uhr: 0</tspan>
  <tspan x="20" dy="16" style="fill:green;">kurzer Weg: 0, mit Uhr: 1</tspan>
  <tspan x="20" dy="16">Endpunkt: 250,150</tspan>
  </text>
  <text x="260" y="185">Start</text>
  <text x="430" y="270">Ende</text>
  <text x="240" y="244">x</text>
  <text x="312" y="293">y</text>
</svg>
```

## 10.9 Pfeilspitzen - das marker-Element

Ein mit dem **marker**-Element definierter Marker ist strenggenommen ein Symbol, das mit einem Ende oder vielen Enden eines **path**-Elements, **line**-Elements, **polyline**-Elements oder **polygon**-Elements verbunden ist. Im etwas verständlicherem Sinne ist ein Marker eine Pfeilspitze, die entweder an einem oder an vielen Enden eines Pfades befestigt werden kann.

Um einem der oben genannten Elemente einen Marker zuzuordnen, können Sie diesen Elementen die Attribute

- ***marker-start*** - Pfeilspitze am Beginn des Pfades,
- ***marker-end*** - Pfeilspitze am Ende des Pfades, und
- ***marker-mid*** - alle anderen Enden des Pfades (außer Beginn und Ende)

zuordnen. Mögliche Werte für diese Attribute sind entweder eine *URI*, d.h. eine Referenz auf einen, im **defs**-Container, definierten Marker oder *none*.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="600" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das marker-Element</title>
<desc>Pfeilspitzen definieren</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;}
      polyline {fill:none; stroke:black; stroke-width:1px;}
      polygon {fill:none; stroke:black; stroke-width:1px;}
    ]]>
    </style>

    <marker id="pf1"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth"
      markerWidth="15" markerHeight="15"
      orient="auto">
      <path d="M 0,0 l 10,5 l -10,5 z" />
    </marker>

    <marker id="pf1a"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth"
      markerWidth="15" markerHeight="15"
      orient="160">
      <path d="M 0,0 l 10,5 l -10,5 z" />
    </marker>
    <marker id="pf2"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerWidth="50" markerHeight="50"
      orient="auto" >
      <path d="M 0,0 C 0,4 6,5 10,5 C 6,5 0,6 0,10 z"
        fill="red" stroke="black" stroke-width=".5" />
    </marker>
    <marker id="pf3"
      viewBox="0 0 20 10" refX="10" refY="5"
      markerWidth="30" markerHeight="30"
      orient="auto">
      <path d="M 0,0 L 20,5 L 0,10 z" />
    </marker>
  </defs>
```

```
  <polyline points="20,20 50,50 100,50 200,100"
    marker-end="url(#pf1)" />
  <polyline points="20,70 50,100 100,100 200,150"
    marker-mid="url(#pf1)" />
  <polyline points="20,120 50,150 100,150 200,200"
    marker-start="url(#pf1)" />
  <polyline points="20,170 50,200 100,200 200,250"
    marker-start="url(#pf1)"
    marker-end="url(#pf1)" />
  <polyline points="20,220 50,250 100,250 200,300"
    marker-start="url(#pf1)"
    marker-mid="url(#pf1)"
    marker-end="url(#pf1)" />

  <polyline points="300,120 400,80 400,100 460,80 500,80"
    marker-start="url(#pf1a)"
    marker-end="url(#pf2)" />
  <polyline points="350,150
                    400,150
                    450,200
                    450,250
                    400,300
                    350,300
                    300,250
                    300,200"
    marker-start="url(#pf3)"
    marker-end="url(#pf2)" />

  <polygon points="480,300 550,330 480,360"
    marker-start="url(#pf2)"
    marker-end="url(#pf3)" />
</svg>
```

Marker werden mit dem **marker**-Element im Definitionsbereich der SVG Grafik platziert und mit einer eindeutigen ID versehen. Innerhalb des **marker**-Elements können Sie durch einen beliebigen Pfad eine Pfeilspitze definieren. Im obigen Beispiel wurde für alle Marker ein **path**-Element gewählt - es hätte aber auch eine der 6 SVG Grundformen sein können.

Für das **marker**-Element gibt es einige wichtige Attribute, deren Bedeutung nachfolgend erläutert wird.

Das Attribut ***viewBox***. Mit diesem Attribut legen Sie im weitesten Sinn den sichtbaren Bereich des Markers fest (auf dieses Attribut wird an anderer Stelle in diesem Tutorial noch eingegangen). Das Attribut viewBox erwartet 4 Werte, die nacheinander die x-Koordinate, die y-Koordinate, die Breite und die Höhe des rechteckigen Anzeigebereichs angeben. Wählen Sie am Besten immer einen viewBox-Bereich, der den Marker genau umfasst.

Die Attribute ***refX*** und ***refY***. Mit Hilfe dieser Attribute bestimmen Sie den genauen Koordinatenpunkt innerhalb eines Markers, der auf dem Pfadende aufsetzt. Sie können so die genaue Position des Markers in Bezug zum Pfadende bestimmen. Die Voreinstellung dieser Attribute ist *0*. Das hat zur Folge, dass der Marker mit seiner linken, oberen Ecke an das Pfadende stößt, was selten gewollt sein wird ;-). Die Verwendung dieser Attribute ist daher empfehlenswert.

Das Attribut ***markerUnits***. Dieses Attribut bestimmt welche Maßeinheit für den Marker verwendet werden soll. Dabei sind *strokeWidth* und *userSpaceOnUse* zulässige Werte für dieses Attribut. Bei Verwendung von *strokeWidth* passt sich die Größe der Pfeilspitze an die Liniendicke des Pfades - der den Marker referenziert - an. Bei Verwendung von *userSpaceOnUse* geschieht dies nicht, d.h. der Marker wird immer in der gleichen Größe dargestellt.

Die Attribute ***markerWidth*** und ***markerHeight***. Durch diese Attribute können Sie die dargestellte Größe des Markers verändern. Maßeinheit ist die durch ***markerUnits*** festgelegte. Voreinstellung ist *3*.

Das Attribut ***orient***. Mit diesem Attribut legen Sie die Orientierung der Pfeilspitze fest. In der Einstellung *auto* übernimmt der user agent diese Aufgabe. Um die Orientierung selbst festzulegen weisen Sie diesem

Attribut einen Winkel zu. Im obigen Beispiel wurde dies für den Marker pf1a praktiziert. Wenn Sie dieses Attribut nicht verwenden, wird die Voreinstellung *0* verwendet, d.h. die Pfeilspitze wird nicht rotiert, sondern immer genau so dargestellt, wie sie definiert wurde.

## 10.10 Tipp zur Erstellung von Pfaden

Pfade per Hand mittels einem Text-Editor zu erzeugen, ist ein Unterfangen, dass bei einfachen Pfaden das Vorstellungsvermögen in Bezug auf das Koordinatensystem schult. Bei komplexeren Pfaden kann ich jedoch nur vom "Handkodieren" abraten.

Meine Empfehlung: Komplexe Pfade mit Hilfe eines SVG WYSIWYG-Editors wie z.B. Sodipodi zu erzeugen. Dies hat folgende Vorteile:

- es ist effektiver.
- es ist komfortabler.
- es schont die Nerven ;-).

.. und natürlich können Sie auch die komplette Grafik mit einem WYSIWYG-Editor erzeugen.

Nichtsdestotrotz verleiht Ihnen die Kenntnis um den verwendeten Quellcode immer die Möglichkeit, Ihre Grafik per Hand "feinzutunen". Dies kann z.B. notwendig sein, falls der verwendete Editor bestimmte Funktionalitäten (noch) nicht unterstützt, wenn Sie Objekte animieren möchten, wenn Sie Erweiterungssprachen wie ECMAScript verwenden wollen, wenn Sie Grafiken dynamisch erzeugen wollen, oder, oder ...

# 11 Painting

In den vorangegangenen Beispielen dieses Tutorials wurden Elemente hauptsächlich mit den Attributen oder Eigenschaften ***fill*** und/oder ***stroke*** formatiert. SVG stellt eine Reihe weiterer Möglichkeiten bereit, die Darstellung von Objekte zu beeinflussen.

Dazu gehören Attribute,

- welche die Darstellung von Füllungen und Randlinien näher bestimmen:Attribute der Attributgruppe ***PresentationsAttributes-FillStroke***,
- welche die Sichtbarkeit der Objekte definieren und die Qualität der Darstellung beeinflussen:Attribute der Attributgruppe ***PresentationAttributes-Graphics***
- und Attribute, mit deren Hilfe Sie die Farbdarstellung manipulieren können:Attribute der Attributgruppe ***PresentationAttributes-Color***.

Die Bedeutung und Verwendung der wichtigsten Attribute aus den oben genannten Attributgruppen wird in den folgenden Unterkapiteln erläutert.

## 11.1 Eigenschaften von Füllungen und Randlinien

***fill*** Voreinstellung: *black*. Das Attribut ***fill*** legt die Füllfarbe für ein Objekt fest. Es erwartet eine Farbangabe als Wertzuweisung. Gültige Farbangaben sind entweder ein spezielles Farbwort, eine hexadezimale Farbangabe mit vorangstellter Raute oder einer dezimale Farbangabe der Form: rgb(0,0,0). Der Wert *none* bewirkt, dass das Objekt keine Füllfarbe erhält.
***fill-opacity*** Voreinstellung: *1*. Mit dem Attribut ***fill-opacity*** können Sie die Durchsichtigkeit (Transparenz) einer Füllfarbe bestimmen. Zulässige Werte liegen zwischen *0* und *1*. *0* bedeutet völlige Tranzparenz, *1* bedeutet keine Transparenz. Beachten Sie: Wenn Sie die Transparenz eines Objekts verändern vermischen sich die übereinanderliegenden Farben.
***fill-rule*** Voreinstellung: *nonzero*. Mit der Eigenschaft ***fill-rule*** können Sie bestimmen wo sich die zu füllende Fläche (das innere) des Objekts befindet. Diese Festlegung kann von Bedeutung sein, wenn sich der Pfad einer Form überschneidet. Mögliche Werte sind *nonzero* und *evenodd*. Im Beispiel dieses Unterkapitels sind die Unterschiede in der Darstellung bei Verwendung der beiden Werte anschaulich dargestellt.
***stroke*** Voreinstellung: *none*. Mit dem Attribut ***stroke*** legen Sie eine Farbe für die Randlinie fest. Zulässige Werte sind gültige Farbangaben oder der Wert *none*, wenn keine Randlinie dargestellt werden soll.
***stroke-dasharray*** Voreinstellung: *none*. Durch die Verwendung des Attributs ***stroke-dasharray*** können Sie eine gestrichelte Linie erzeugen. Als Wertzuweisung erwartet dieses Attribut eine gerade Anzahl durch Kommata getrennte Zahlen. Dabei entspricht jede ungerade Zahl einer Strichlänge und jede gerade Zahl der Länge des Zwischenraums. Um ein gleichmäßig gestrichelte Linie zu erzeugen, reicht es aus nur 2 Zahlen anzugeben. Die Gesamt-Länge der Linie sollte nach Möglichkeit der Summe aller Zahlenwerte entsprechen.
***stroke-dashoffset*** Voreinstellung: *0*. Mit dem Attribut ***stroke-dashoffset*** haben Sie die Möglichkeit den Beginn der Strichelung einer, mit ***stroke-dasharray*** definierten Linie, um eine angegebenen Länge zu verschieben. Das Attribut akzeptiert eine Längenangabe als Wertzuweisung.
***stroke-linecap*** Voreinstellung: *butt*. Durch das Attribut stroke-linecap bestimmen Sie die exakte Form der Linienenden. Dabei sind *butt* (abschließend rechteckig), *round* (abgerundet) und *square* (über das Linienende hinausgezogen, rechteckig) zulässige Werte.
***stroke-linejoin*** Voreinstellung: *miter*. Mit dem Attributs ***stroke-linejoin*** können Sie die Darstellung eines Eckpunktes bestimmen, wenn zwei Linien in einem bestimmten Winkel aufeinandertreffen. Der Wert *miter* bewirkt einen spitzen Winkel, der Wert *bevel* einen abgeflachte Winkel und durch den Wert *round* wird der Eckpunkt gerundet dargestellt.
***stroke-miterlimit*** Voreinstellung: *4*. Das Attribut ***stroke-miterlimit*** kann nur verwendet werden, wenn ***stroke-linejoin*** die Einstellung *miter* verwendet (Voreinstellung). Es erwartet eine Zahl als Wertzuweisung, die der user agent als Faktor versteht. Dieser Faktor wird mit der, im Objekt verwendeten Linienbreite, multipliziert. Das so enstandene Produkt legt die maximale Länge des spitzen Winkels beim aufeinandertreffen der Linien fest. Wenn die Länge des spitzen Winkels diesen Maximalwert übersteigt, ignoriert der user agent die Anweisung *miter* und stellt den Winkel abgeflacht (*bevel*) dar. Durch Verwendung eines höheren Wertes als 4 für das Attribut stroke-miterlimit erreichen Sie auch in solchen Fällen die Darstellung eines spitzen Winkels.
***stroke-opacity*** Voreinstellung: *1*. Mit dem Attribut ***stroke-opacity*** können Sie die Durchsichtigkeit (Transparenz) einer Randlinie bestimmen. Zulässige Werte liegen zwischen *0* und *1*. *0* bedeutet völlige Tranzparenz, *1* bedeutet keine Transparenz.
***stroke-width*** Voreinstellung: *1*. Mit dem Attribut ***stroke-width*** legen Sie die Dicke der Randlinie fest. Das Attribut erwartet eine Längenangabe als Wertzuweisung.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="750" height="480"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>PresentationAttributes-FillStroke</title>
<desc>Anwendungsbeispiele</desc>
<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:Verdana,sans-serif; font-size:13px;
```

```
        font-weight:bold;}
    rect {fill:limegreen;}
    circle {fill:yellow; stroke:darkgray; stroke-width:10px;}
    circle.d {fill:none; stroke-width:1px;}
    path {fill:limegreen; stroke:black;}
    line {fill:none; stroke:limegreen; stroke-width:20px;}
    line.d {fill:none; stroke:limegreen; stroke-width:2px;}
    polyline {fill:none; stroke:limegreen; stroke-width:10px;}
  ]]>
  </style>
</defs>

<!-- Durchsichtigkeit -->
<!-- die im Hintergrund liegenden Rechtecke -->
  <rect x="20" y="30" width="120" height="60" />
  <rect x="160" y="30" width="120" height="60" />
  <rect x="300" y="30" width="120" height="60" />
  <rect x="440" y="30" width="120" height="60" />
  <rect x="580" y="30" width="120" height="60" />
<!-- die teiltransparenten Kreise -->
  <circle cx="80" cy="60" r="50" />
  <circle cx="220" cy="60" r="50"
    opacity=".2"/>
  <circle cx="360" cy="60" r="50"
    style="opacity:.8" />
  <circle cx="500" cy="60" r="50"
    fill-opacity=".5" />
  <circle cx="640" cy="60" r="50"
    stroke-opacity=".5" />

<!-- wo ist innen? wo ist aussen? .. einer Form  -->
  <path d="M 20,160 h 120 l -80,100 l 20,-80 l 20,80 z"
    style="fill:none" />
  <path d="M 240,160 h 120 l -80,100 l 20,-80 l 20,80 z" />
  <path d="M 460,160 h 120 l -80,100 l 20,-80 l 20,80 z"
    fill-rule="evenodd" />

<!-- Linienenden -->
  <line x1="30" y1="400" x2="30" y2="300"/>
  <line x1="80" y1="400" x2="80" y2="300"
    stroke-linecap="round" />
  <line x1="130" y1="400" x2="130" y2="300"
    stroke-linecap="square" />

<!-- Linienwinkel -->
  <g transform="rotate(-20,400,350) translate(-30)">
    <polyline points="220,320 350,350 220,380" />
    <polyline points="300,320 430,350 300,380"
      stroke-miterlimit="5" />
    <polyline points="380,320 510,350 380,380"
      stroke-linejoin="bevel" />
    <polyline points="460,320 590,350 460,380"
      stroke-linejoin="round" />
  </g>

<!-- gestrichelte Linien -->
  <line class="d" x1="690" y1="460" x2="590" y2="265"
    stroke-dasharray="8,6" />
  <line class="d" x1="720" y1="460" x2="620" y2="265"
    stroke-dasharray="8,6" stroke-dashoffset="8" />

<!-- Texte, Hilfslinien, Hilfsformen -->
  <text x="22" y="130">normal</text>
  <text x="162" y="130">opacity .2</text>
```

```
  <text x="305" y="130">opacity .8</text>
  <text x="445" y="130">fill-opacity .5</text>
  <text x="582" y="130">stroke-opacity .5</text>
  <text x="110" y="230">die Form</text>
  <text x="330" y="230">fill-rule nonzero</text>
  <text x="550" y="230">fill-rule evenodd</text>
  <line x1="10" y1="300" x2="163" y2="300"
    style="stroke:black; stroke-width:1;" />
  <line x1="10" y1="400" x2="163" y2="400"
    style="stroke:black; stroke-width:1;" />
  <text x="20" y="430">stroke-linecap</text>
  <text x="15" y="340">butt</text>
  <text x="65" y="360">round</text>
  <text x="115" y="380">square</text>
  <text x="220" y="320">stroke-linejoin</text>
  <text x="230" y="440">miter (aber Darstellung bevel)</text>
  <text x="306" y="413">miter durch stroke-miterlimit 5</text>
  <text x="382" y="386">bevel</text>
  <text x="458" y="359">round</text>
  <text x="560" y="360">stroke-dasharray 8,6</text>
  <text x="595" y="380">stroke-dashoffset 8</text>
  <line x1="720" y1="454" x2="720" y2="382"
    style="stroke:black; stroke-width:1px;" />
  <circle class="d" cx="690" cy="460" r="6"/>
  <circle class="d" cx="720" cy="460" r="6" />
</svg>
```

## 11.2 Eigenschaften zur Darstellung der Objekte

***clip-path*** Voreinstellung: *none*. Mit dem Attribut ***clip-path*** können Sie ein zuvor definiertes **clip-path**-Element referenzieren (siehe Masken). Als Wert aktzeptiert dieses Attribut eine gültige *URL* oder *none*.

***clip-rule*** Voreinstellung: *nonzero*. Mit der Eigenschaft ***clip-rule*** können Sie bestimmen welches die innere Fläche des zuvor mit ***clip-path*** festgelegten Pfades ist. Diese Festlegung ist von Bedeutung, wenn sich der Pfad überschneidet. Mögliche Werte sind *nonzero* und *evenodd*.

***cursor*** Voreinstellung: *auto*. Durch Verwendung dieses Attributes können Sie die Anzeige des Mauszeigers an Ihre Wünsche anpassen. Einige mögliche Werte sind: *crosshair* (kreuz), *default* (Standard Mauszeiger des Systems), *pointer* (Mauszeiger für eine Verweis), *busy* (beschäftigt) oder *help* (hilfesymbol). und es gibt noch weitere vordefinierte Werte. Außerdem akzeptiert das Attribut ***cursor*** noch eine *URL* als Wertzuweisung. Sie können sich Ihren Mauszeiger also selbst malen :-). Der Adobe SVG-Viewer 3.0 unterstützt dieses Attribut aber leider noch nicht :-(.

***display*** Voreinstellung: *inline*. Das Attribut ***display*** entspricht der gleichnamigen CSS 2 Eigenschaft. Es akzeptiert eine Reihe von Werten, wobei die wichtigsten *inline* (die Voreinstellung) und *none* sind. Der Wert *none* bewirkt, das das entsprechende Element und all seine Kind-Elemente nicht am Bildschirm dargestellt werden. Wenn Sie also einem Container-Element (z.B. g) das Attribut ***display*** mit dem Wert *none* zuordnen, wird keines der Elemente innerhalb des Containers angezeigt. Hierdurch unterscheidet sich die Verwendung von ***display*** vom Gebrauch des Attributs ***visibility*** (siehe unten).

***filter*** Voreinstellung: *none*. Mit dem Attribut ***filter*** können Sie einem Element einen zuvor definierten Filter (siehe Filtereffekte) zuordnen. Das Attribut akzeptiert eine *URL* oder *none* als Wert.

***image-rendering*** Voreinstellung: *auto*. Durch das Attribut ***image-rendering*** können Sie das Verhalten des user agents bei der Dastellung des **image**-Elements beeinflussen. Mögliche Werte sind *auto*, *optimizeSpeed* und *optimizeQuality*. Der Wert *auto* errechnet automatisch einen günstigen Mittelwert zwischen Qualität und Geschwindigkeit der Anzeige, wo bei die Qualität etwas höher gewichtet wird. Mit *optimizeSpeed* wird die Qualität verschlechtert und die Geschwindigkeit der Anzeige erhöht. Der Wert *optimizeQuality* bewirkt das Gegenteil - besser Qualität, langsamere Darstellung.

***mask*** Voreinstellung: *none*. Mit dem Attribut mask können Sie einem Element eine zuvor, durch das mask-Element definierte Maske zuordnen (siehe Masken) zuordnen. Das Attribut akzeptiert eine *URL* oder *none* als Wert.

***opacity*** Voreinstellung: *1*. Mit dem Attribut ***opacity*** können Sie die Durchsichtigkeit (Transparenz) eines gesamten Objekts (Füllfarbe und Randlinie) bestimmen. Zulässige Werte liegen zwischen *0* und *1*. *0* bedeutet völlige Tranzparenz, *1* bedeutet keine Transparenz.

***pointer-events*** Voreinstellung: *visiblePainted*. Wenn Sie ein Element zum Ziel von Maus-aktionen machen (z.B. durch die Definition als Hyperlink), können Sie mit dem Attribut pointer-events festlegen, wann genau das Ereignis ausgelöst werden soll. Im weitesten Sinn heißt das: über welche Fläche des Elements der Mauszeiger bewegt werden muß, um das Ereignis auszulösen. Mögliche Werte sind *visiblePainted* (sichtbare Füllfarbe und/oder sichtbare Randlinie), *visibleFill* (sichtbare Füllfarbe), *visibleStroke* (sichtbare Randlinie), *visible* (das gesamte sichtbare Element), *painted* (Füllfarbe und/oder Randlinie), *fill* (Füllfarbe), *stroke* (Randlinie), *all* (das gesamte Element) und *none* (nix).

***shape-rendering*** Voreinstellung: *auto*. Durch das Attribut ***shape-rendering*** können Sie das Verhalten des user agents bei der Dastellung SVG-eigener Formen, wie Pfade oder Grundformen, beeinflussen. Mögliche Werte sind *auto*, *optimizeSpeed* und *optimizeQuality*. Der Wert *auto* errechnet automatisch einen günstigen Mittelwert zwischen Qualität und Geschwindigkeit der Anzeige, wo bei die Qualität etwas höher gewichtet wird. Mit *optimizeSpeed* wird die Qualität verschlechtert und die Geschwindigkeit der Anzeige erhöht. Der Wert *optimizeQuality* bewirkt das Gegenteil - besser Qualität, langsamere Darstellung.

***text-rendering*** Voreinstellung: *auto*. Durch das Attribut ***text-rendering*** können Sie das Verhalten des user agents bei der Dastellung des **text**-Elements beeinflussen. Mögliche Werte sind *auto*, *optimizeSpeed* und *optimizeQuality*. Der Wert *auto* errechnet automatisch einen günstigen Mittelwert zwischen Qualität und Geschwindigkeit der Anzeige, wo bei die Qualität etwas höher gewichtet wird. Mit *optimizeSpeed* wird die Qualität verschlechtert und die Geschwindigkeit der Anzeige erhöht. Der Wert *optimizeQuality* bewirkt das Gegenteil - besser Qualität, langsamere Darstellung.

***visibility*** Voreinstellung: *visible*. Wenn Sie einzelne Elemente nicht darstellen wollen, können Sie diesen das Attribut ***visibility*** mit dem Wert *hidden* zuordnen. Der zweite mögliche Wert für dieses Attribut ist *visible*: normale Anzeige des Elements. Im Gegensatz zu ***display*** vererbt sich die Eigenschaft von

***visibility*** nicht auf mögliche Kind-Elemente (siehe oben), wenn diese durch *visible* als sichtbar definiert werden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="600" height="400"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>PresentationAttributes-Graphics</title>
<desc>Anwendungsbeispiele</desc>
<defs>
  <style type="text/css">
  <![CDATA[
    text {font-family:Verdana,sans-serif; font-size:14px;}
    ellipse {fill:lightslategray;}
    circle {fill:goldenrod; opacity:.5;}
    line {fill:none; stroke:black; stroke-dasharray:8,4;}
  ]]>
  </style>
</defs>

<!-- viermal die gleiche Gruppe: sichtbar und unsichtbar -->
  <g>
    <ellipse cx="80" cy="60" rx="70" ry="20" visibility="visible" />
    <circle cx="80" cy="50" r="30" />
  </g>
  <g visibility="hidden">
    <ellipse cx="80" cy="160" rx="70" ry="20" />
    <circle cx="80" cy="150" r="30" visibility="visible" />
  </g>
  <g visibility="hidden">
    <ellipse cx="80" cy="260" rx="70" ry="20" visibility="visible" />
    <circle cx="80" cy="250" r="30" visibility="visible" />
  </g>
  <g display="none">
    <ellipse cx="80" cy="360" rx="70" ry="20" visibility="visible" />
    <circle cx="80" cy="350" r="30" visibility="visible" />
  </g>

<!-- texte und hilflinien -->
  <text x="560" y="30" writing-mode="tb"
    style="font-size:30px; font-weight:bold;" >
    visibility und display
  </text>
  <text x="170" y="35">Gruppe: -
  <tspan x="190" dy="20">Ellipse: - </tspan>
  <tspan x="210" dy="20">Kreis: - </tspan>
  </text>
  <text x="170" y="135">Gruppe: visibility hidden
  <tspan x="190" dy="20">Ellipse: - </tspan>
  <tspan x="210" dy="20">Kreis: visibility visible</tspan>
  </text>
  <text x="170" y="235">Gruppe: visibility hidden
  <tspan x="190" dy="20">Ellipse: visibility visible</tspan>
  <tspan x="210" dy="20">Kreis: visibility visible</tspan>
  </text>
  <text x="170" y="335">Gruppe: display none
  <tspan x="190" dy="20">Ellipse: visibility visible</tspan>
  <tspan x="210" dy="20">Kreis: visibility visible</tspan>
  </text>
  <line x1="20" y1="100" x2="500" y2="100" />
  <line x1="20" y1="200" x2="500" y2="200" />
```

```
  <line x1="20" y1="300" x2="500" y2="300" />
</svg>
```

## 11.3 Eigenschaften zur Farbdarstellung

***color*** Voreinstellung: vom user agent abhängig. Das Attribut ***color*** entspricht dem gleichnamigen Attribut aus der CSS 2 Spezifikation. Es wird zur Zeit jedoch nicht unterstützt. Verwenden Sie stattdessen die SVG-eigenen Eigenschaften ***fill*** und/oder ***stroke***.

***color-interpolation*** Voreinstellung: *sRGB*. Mit Hilfe des Attributs ***color-interpolation*** können Sie das Verhalten des user agents bei der Farbberechnung beeinflussen. Mögliche Werte sind: *sRGB* (Farbberechnung nach dem sRGB-Farbmodell), *auto* (der user agent entscheidet) und *linearRGB* (Farbberechnung nach dem linearen RGB-Farbmodell).

***color-rendering*** Voreinstellung: *auto*. Durch das Attribut ***color-rendering*** können Sie das Verhalten des user agents bei der Dastellung von Farbe oder Farbverläufen beeinflussen. Mögliche Werte sind *auto*, *optimizeSpeed* und *optimizeQuality*. Der Wert *auto* errechnet automatisch einen günstigen Mittelwert zwischen Qualität und Geschwindigkeit der Anzeige, wo bei die Qualität etwas höher gewichtet wird. Mit *optimizeSpeed* wird die Qualität verschlechtert und die Geschwindigkeit der Anzeige erhöht. Der Wert *optimizeQuality* bewirkt das Gegenteil - besser Qualität, langsamere Darstellung.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="600" height="420"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>PresentationAttributes-Color</title>
<desc>Interpolation und Rendering - sehen Sie den Unterschied?</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:14px; }
      rect {fill:url(#g01);}
    ]]>
    </style>
    <linearGradient id="g01"
      x1="0%" y1="0%" x2="0%" y2="100%">
      <stop offset="1%" stop-color="blue" />
      <stop offset="99%" stop-color="yellow" />
    </linearGradient>
  </defs>

  <!-- 3 verschiedene Interpolationsarten -->
  <rect x="170" y="20" width="120" height="160" />
  <rect x="310" y="20" width="120" height="160"
    color-interpolation="auto" />
  <rect x="450" y="20" width="120" height="160"
    color-interpolation="linearRGB" />

  <!--3 verschiedene Rendering-Einstellungen -->
  <rect x="170" y="220" width="120" height="160" />
  <rect x="310" y="220" width="120" height="160"
    color-rendering="optimizeSpeed" />
  <rect x="450" y="220" width="120" height="160"
    color-rendering="optimizeQuality" />

  <text x="10" y="40" style="font-weight:bold;">color-interpolation</text>
  <text x="175" y="200">sRGB</text>
  <text x="315" y="200">auto</text>
  <text x="455" y="200">linearRGB</text>
  <text x="10" y="240" style="font-weight:bold;">color-rendering</text>
  <text x="175" y="400">auto</text>
  <text x="315" y="400">optimizeSpeed</text>
  <text x="455" y="400">optimizeQuality</text>
```

```
</svg>
```

Im obigen Beispiel wird ein Rechteck, das mit einen Farbverlauf (das nächste Kapitel: Verläufe und Muster) gefüllt ist, jeweils mit 3 verschiedenen Interpolationsarten und 3 unterschiedlichen Rendering-Einstellung dargestellt. Falls Sie keinen Unterschied in der Darstellung entdecken können - der Autor auch nicht :-).

## 12 Verläufe und Muster

In SVG Dokumenten gibt es 3 unterschiedliche Darstellungsmöglichkeiten für Füllungen mit ***fill*** oder Randlinien mit ***stroke***:

- einfarbig (Paint)
- als Farbverlauf (Gradient)
- durch ein Muster ausgefüllt (Pattern)

Im letzten Kapitel wurde das Painting behandelt, in diesem Kapitel folgen die Gradienten und die Pattern - die Farbverläufe und Muster.

Unter einem Farbverlauf versteht man den Übergang von einer Farbe in eine andere entlang eines festgelegten Vektors. Dabei wird zwischen geraden Gradienten und kreisförmigen Gradienten unterschieden. Farbverläufe werden durch Verwendung der jeweiligen Container-Elemente **linearGradient** und **radialGradient** im defs-Bereich einer SVG Grafik definiert.

Unter einem Muster versteht man eine immer währende Wiederholung einer Musterform. So kann ein Objekt mit einem gleichartigen Muster gefüllt werden. Das Container-Element **pattern** zur Festlegung eines Musters, wird ebenfalls im defs-Container platziert.

Um die Verläufe oder Muster referenzieren zu können, müssen Sie diesen jeweils das Attribut ***id*** mit einem eindeutigen Bezeichner als Wert zuweisen. Die so erzeugten Verläufe oder Muster werden dann von Elementen und Objekten durch das Attribut ***fill*** oder ***stroke*** referenziert, z.B.: <circle cx="50" cy="50" r="20" fill="url(#bezeichner-id)" />

## 12.1 Lineare Verläufe - das linearGradient-Element

Lineare Farbverläufe werden durch das Element **linearGradient** im defs-Container definiert.In diesem Element werden mit Hilfe der Attribute ***x1***, ***y1***, ***x2*** und ***y2***, 2 Koordinatenpunkte angegeben, welche die Länge und die Richtung des Verlaufsvektors bestimmen.Die 4 Werte sind relativer Natur (0% bis 100% oder 0 bis 1) und beziehen sich auf das Element, das den Verlauf referenziert.

Das Element **linearGradient** ist selbst ein Container-Element und sollte mindestens 2 Kind-Elemente **stop** enthalten. Es sind jedoch auch mehrere Kind-Element möglich. In jedem **stop**-Element, legen Sie durch Verwendung des Attributs ***stop-color*** genau eine Verlaufsfarbe fest und bestimmen den Beginn dieser Farbe auf dem Verlaufsvektor durch Verwendung des Attributs ***offset***. Der Wert des Attributs ***offset*** ist ebenfalls relativer Natur, bezieht sich aber nicht auf das referenzierende Element, sondern auf den Verlaufsvektor. Dies ist von Bedeutung, falls der Verlaufsvektor in der Länge nicht der Ausdehnung des Elements entspricht, also kürzer als 100% ist.

Der Abstand zwischen den jeweils nachfolgenden ***offset***-Werten definiert einen Abschnitt auf dem Verlaufsvektor. In diesem Bereich wird der Farbübergang der beiden Farben berechnet. Dabei kehrt sich die Richtung des Verlaufsvektors für den nachfolgenden Punkt um.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="510" height="540"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Farbverläufe</title>
<desc>Funktionsweise von Farbverläufen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:14px;
           font-weight:bold;}
      circle {fill:black; stroke:black;}
      line {fill:none; stroke:black; stroke-width:2;
           stroke-dasharray:10,5;}
    ]]>
    </style>
    <marker id="marker1"
      viewBox="0 0 10 10" refX="10" refY="5"
      markerUnits="strokeWidth"
      markerWidth="5" markerHeight="5"
      orient="auto">
      <path d="M 0,0 l 10,5 l -10,5 z"  />
    </marker>

<!-- das Rechteck 'rechteck' -->
    <rect id="rechteck" x="60" y="80" width="250" height="150" />
<!-- die Definition des linearen Verlaufs 'lgr1' -->
    <linearGradient id="lgr1"
      x1="0" y1="0" x2="1" y2="0">
      <stop offset="0" stop-color="limegreen" />
      <stop offset="1" stop-color="yellow" />
    </linearGradient>
  </defs>

<!-- 2 Instanzen von 'rechteck' mit dem Verlauf lgr1 gefüllt -->
  <use xlink:href="#rechteck" fill="url(#lgr1)" />
  <use xlink:href="#rechteck" y="250" fill="url(#lgr1)" />

<!-- Texte, Hilfslinien, Punkte -->
  <text x="30" y="30" style="fill:green">
    &lt;linearGradient x1=&quot;0&quot; y1=&quot;0&quot;
    x2=&quot;1&quot; y2=&quot;0&quot;&gt;
  </text>
```

```
  <text x="50" y="290" style="fill:green">
   &lt;stop offset=&quot;0&quot; stop-color=&quot;limegreen&quot; /&gt;
   <tspan x="50" dy="16">
    &lt;stop offset=&quot;1&quot; stop-color=&quot;yellow&quot; /&gt;
   </tspan>
  </text>
  <text x="30" y="520" style="fill:green">
   &lt;/linearGradient&gt;
  </text>
  <circle cx="60" cy="80" r="3" />
  <circle cx="310" cy="80" r="3" />
  <circle cx="310" cy="230" r="3" />
  <circle cx="60" cy="230" r="3" />
  <circle cx="60" cy="390" r="3" />
  <circle cx="310" cy="390" r="3" />
  <circle cx="60" cy="450" r="3" />
  <circle cx="310" cy="460" r="3" />
  <line x1="60" y1="155" x2="310" y2="155" marker-end="url(#marker1)" />
  <line x1="60" y1="390" x2="310" y2="390" marker-end="url(#marker1)" />
  <line x1="60" y1="450" x2="310" y2="450" marker-end="url(#marker1)" />
  <line x1="310" y1="460" x2="60" y2="460" marker-end="url(#marker1)" />
  <text x="30" y="70">(0,0) oder (0%,0%)</text>
  <text x="280" y="70">(1,0) oder (100%,0%)</text>
  <text x="280" y="250">(1,1) oder (100%,100%)</text>
  <text x="30" y="250">(0,1) oder (0%,100%)</text>
  <text x="80" y="125">Verlaufsvektor x1,y1 x2,y2
  <tspan x="80" dy="16">0,0 1,0</tspan>
  </text>
  <text x="80" y="175">oder
  <tspan x="80" dy="16">0%,0% 100%,0%</tspan>
  </text>
  <text x="65" y="410">limegreen</text>
  <text x="255" y="410">yellow</text>
  <text x="12" y="380">offset 0 oder 0%</text>
  <text x="262" y="380">offset 1 oder 100%</text>
  <text x="80" y="440">Berechung des Gradienten</text>
</svg>
```

Die Länge und die Ausrichtung der Verlaufsvektoren können Sie mit den Attributen ***x1***, ***y1***, ***x2*** und ***y2*** also beliebig festlegen. Somit sind also auch kurze oder diagonale Farbverläufe möglich.
Durch Verwendung von mehr als 2 **stop**-Elementen werden auch mehrere Farbverläufe erzeugt.
Beachten Sie, dass der relative Wert eines ***offset***-Attributs immer größer sein muß, als der Wert des vorangegangenen ***offset***-Attributs. Sind zwei aufeinanderfolgende Werte gleich, oder ist der nachfolgende Wert kleiner als der vorhergende, findet der Übergang zwischen den Farben ohne Farbverlauf statt.
Im nachfolgenden Beispiel finden Sie ein Dutzend lineare Gradienten. Machen Sie sich einen kleinen Eindruck über die verschiedensten Möglichkeiten der Erzeugung und Darstellung von Farbverläufen in SVG.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="660" height="480"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Das Element linearGradient</title>
<desc>Verschiedene Beispiele für linearer Farbverläufe</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:40px;
            font-stretch:ultra-expanded;}
    ]]>
    </style>
```

```
<rect id="rechteck" x="20" y="20" width="180" height="80" />

<linearGradient id="lgr1"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset="0" stop-color="blue" />
  <stop offset="1" stop-color="yellow" />
</linearGradient>
<linearGradient id="lgr2"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset=".2" stop-color="blue" />
  <stop offset=".8" stop-color="yellow" />
</linearGradient>
<linearGradient id="lgr3"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset=".4" stop-color="blue" />
  <stop offset=".6" stop-color="yellow" />
</linearGradient>
<linearGradient id="lgr4"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset=".5" stop-color="blue" />
  <stop offset=".5" stop-color="yellow" />
</linearGradient>

<linearGradient id="lgr5"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset="0" stop-color="red" />
  <stop offset="1" stop-color="limegreen" />
</linearGradient>
<linearGradient id="lgr6"
  x1="0" y1="0" x2="0" y2="1">
  <stop offset="0" stop-color="red" />
  <stop offset="1" stop-color="limegreen" />
</linearGradient>
<linearGradient id="lgr7"
  x1="0" y1="0" x2="1" y2="1">
  <stop offset="0" stop-color="red" />
  <stop offset="1" stop-color="limegreen" />
</linearGradient>
<linearGradient id="lgr8"
  x1="0" y1="1" x2="1" y2="0">
  <stop offset="0" stop-color="red" />
  <stop offset="1" stop-color="limegreen" />
</linearGradient>

<linearGradient id="lgr9"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset="0" stop-color="blue" />
  <stop offset=".25" stop-color="yellow" />
  <stop offset=".5" stop-color="red" />
  <stop offset=".75" stop-color="green" />
  <stop offset="1" stop-color="blue" />
</linearGradient>
<linearGradient id="lgr10"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset="0" stop-color="blue" />
  <stop offset=".2" stop-color="yellow" />
  <stop offset=".8" stop-color="yellow" />
  <stop offset="1" stop-color="blue" />
</linearGradient>
<linearGradient id="lgr11"
  x1="0" y1="0" x2="1" y2="0">
  <stop offset="0" stop-color="blue" />
  <stop offset=".1" stop-color="yellow" />
  <stop offset=".2" stop-color="white" />
```

```
    <stop offset=".9" stop-color="blue" />
   </linearGradient>
   <linearGradient id="lgr12"
    x1="0" y1="0" x2="1" y2="1">
    <stop offset="0" stop-color="white" />
    <stop offset=".3" stop-color="yellow" />
    <stop offset=".5" stop-color="blue" />
    <stop offset=".7" stop-color="yellow" />
    <stop offset="1" stop-color="white" />
   </linearGradient>
  </defs>

  <use xlink:href="#rechteck" fill="url(#lgr1)" />
  <use xlink:href="#rechteck" y="100" fill="url(#lgr2)" />
  <use xlink:href="#rechteck" y="200" fill="url(#lgr3)" />
  <use xlink:href="#rechteck" y="300" fill="url(#lgr4)" />

  <use xlink:href="#rechteck" x="220" fill="url(#lgr5)" />
  <use xlink:href="#rechteck" x="220" y="100" fill="url(#lgr6)" />
  <use xlink:href="#rechteck" x="220" y="200" fill="url(#lgr7)" />
  <use xlink:href="#rechteck" x="220" y="300" fill="url(#lgr8)" />

  <use xlink:href="#rechteck" x="440" fill="url(#lgr9)" />
  <use xlink:href="#rechteck" x="440" y="100" fill="url(#lgr10)" />
  <use xlink:href="#rechteck" x="440" y="200" fill="url(#lgr11)" />
  <use xlink:href="#rechteck" x="440" y="300" fill="url(#lgr12)" />

  <text x="50" y="465" style="fill:url(#lgr9);">
  linearGradient&amp;apos;s
  </text>
</svg>
```

## 12.2 Radiale Verläufe - das radialGradient-Element

Radiale Farbverläufe werden durch das Element **radialGradient** im defs-Container definiert.Durch die Attribute ***cx*** und ***cy***, die eine relative Angabe (von 0 bis 1 oder von 0% bis 100%) erwarten, können Sie den Mittelpunkt des Kreisverlaufs festlegen. Das Attribut ***r*** definiert den Radius bzw. die Ausdehnung des Kreisverlaufs.

Normalerweise wird an der Stelle des Mittelpunktes die höchste Farbintensität der festgelegen Verlaufsfarbe erzeugt. Durch die Verwendung der Attribute ***fx*** und ***fy*** können Sie die höchste Farbintensität aber auch an einer anderen Position innerhalb des Verlaufs platzieren. Im folgenden Beispiel werden die Verläufe der mittleren Spalte mit Hilfe dieser Attribute erzeugt.

Wenn Sie keines der oben genannten Attribute verwenden, wird für jedes der Attribute der voreingestellte Wert 0.5 verwendet.

Das Element **radialGradient** ist, wie auch das Element **linearGradient** selbst ein Container-Element und sollte mindestens 2 Kind-Elemente **stop** enthalten. In jedem **stop**-Element, legen Sie durch Verwendung des Attributs ***stop-color*** genau eine Verlaufsfarbe fest und bestimmen den Beginn dieser Farbe auf den Verlaufsvektoren durch Verwendung des Attributs ***offset***. Bei radialen Gradienten verlaufen mehrere Vektoren vom Mittelpunkt des Gradienten sternförmig nach außen. Der Wert des Attributs ***offset*** ist ebenfalls relativer Natur, bezieht sich aber nicht auf das referenzierende Element, sondern auf die Verlaufsvektoren. Dies ist von Bedeutung, falls die Verlaufsvektoren in der Länge nicht der Ausdehnung des Elements entsprechen, also kürzer als 100% sind.

Eine weitere Möglichkeit zur Darstellung radialer Verläufe bietet das Attribut ***spreadMethod***. Dieses Attribut akzeptiert 3 Werte. Der Wert *pan* ( die Voreinstellung) bewirkt, dass der füllbare Raum des Elements mit der zuletzt angegebenen Farbe ausgefüllt wird. Der Wert *reflect* wiederholt den Verlauf in wechselnder Reihenfolge (start-end-end-start-start-...), der Wert *repeat* wiederholt den Verlauf in gleichbleibender Reihenfolge (start-end-start-end-start-...).

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="660" height="480"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Das Element radialGradient</title>
<desc>Verschiedene Beispiele für radiale Farbverläufe</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:40px;
           font-stretch:ultra-expanded;}
    ]]>
    </style>
    <rect id="rechteck" x="20" y="20" width="180" height="80" />

    <radialGradient id="rgr1"
      cx=".5" cy=".5" r=".5">
      <stop offset="0" stop-color="blue" />
      <stop offset="1" stop-color="yellow" />
    </radialGradient>
    <radialGradient id="rgr2"
      cx=".9" cy=".5" r=".5">
      <stop offset="0" stop-color="blue" />
      <stop offset="1" stop-color="yellow" />
    </radialGradient>
    <radialGradient id="rgr3"
      cx=".3" cy=".7" r=".7">
      <stop offset="0" stop-color="blue" />
      <stop offset="1" stop-color="yellow" />
    </radialGradient>
    <radialGradient id="rgr4"
      cx=".5" cy=".5" r=".5">
      <stop offset="0" stop-color="blue" />
```

```
    <stop offset=".5" stop-color="yellow" />
    <stop offset="1" stop-color="blue" />
  </radialGradient>

  <radialGradient id="rgr5"
    cx=".5" cy=".5" r=".5" fx="1" fy=".5">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>
  <radialGradient id="rgr6"
    cx=".5" cy=".5" r=".5" fx=".5" fy="1">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>
  <radialGradient id="rgr7"
    cx=".8" cy=".3" r=".9" fx=".8" fy=".3">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>
  <radialGradient id="rgr8"
    cx=".5" cy="1" r=".9" fx=".3" fy="0">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>

  <radialGradient id="rgr9"
    cx=".5" cy=".5" r=".5">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>
  <radialGradient id="rgr10" spreadMethod="repeat"
    cx=".5" cy=".5" r=".2">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>
  <radialGradient id="rgr11"
    cx=".5" cy=".5" r=".5" spreadMethod="reflect">
    <stop offset="0" stop-color="blue" />
    <stop offset="1" stop-color="yellow" />
  </radialGradient>
  <radialGradient id="rgr12"
    cx=".5" cy=".5" r=".5">
    <stop offset=".3" stop-color="blue" />
    <stop offset=".7" stop-color="yellow" />
  </radialGradient>

  <radialGradient id="rgr_text"
    cx=".5" cy=".5" r=".8" fx=".5" fy="1">
    <stop offset="0" stop-color="red" />
    <stop offset=".25" stop-color="yellow" />
    <stop offset=".5" stop-color="blue" />
    <stop offset=".75" stop-color="yellow" />
    <stop offset="1" stop-color="red" />
  </radialGradient>
</defs>

<use xlink:href="#rechteck" fill="url(#rgr1)" />
<use xlink:href="#rechteck" y="100" fill="url(#rgr2)" />
<use xlink:href="#rechteck" y="200" fill="url(#rgr3)" />
<use xlink:href="#rechteck" y="300" fill="url(#rgr4)" />

<use xlink:href="#rechteck" x="220" fill="url(#rgr5)" />
<use xlink:href="#rechteck" x="220" y="100" fill="url(#rgr6)" />
<use xlink:href="#rechteck" x="220" y="200" fill="url(#rgr7)" />
```

```
  <use xlink:href="#rechteck" x="220" y="300" fill="url(#rgr8)" />

  <use xlink:href="#rechteck" x="440" fill="url(#rgr9)" />
  <use xlink:href="#rechteck" x="440" y="100" fill="url(#rgr10)" />
  <use xlink:href="#rechteck" x="440" y="200" fill="url(#rgr11)" />
  <use xlink:href="#rechteck" x="440" y="300" fill="url(#rgr12)" />

  <text x="18" y="455" style="fill:url(#rgr_text);font-weight:bold;">
  radialGradient&amp;apos;s
  </text>
</svg>
```

Sie können Gradienten auch transformieren. Dazu müssen Sie allerdings das besondere Attribut ***gradientTransform*** im Element **linearGradient** oder im Element **radialGradient** verwenden. ***gradientTransform*** ist in der gleichen Art und Weise zu verwenden wie das ***transform***-Attribut. Sie können also *translate*, *rotate*, *scale*, *skewX* und *skewY* inclusive den jeweiligen Zahlwerten (Koordinaten, Radien, ...) als Wert verwenden.

Um die Durchsichtbarkeit von Verlausfarben zu ändern gibt es ebenfalls ein besonderes Attribut: ***stop-opacity***. ***stop-opacity*** wird im Element **stop** einer Verlaufsfarbe zugeordnet. Es akzeptiert einen Wert zwischen 0 und 1, wie alle anderen Attribute, die eine Durchsichtbarkeit einstellen (***opacity***, ***stroke-opacity***, ***fill-opacity***, ..).

## 12.3 Muster - das pattern-Element

Ein Muster ist die Wiederholung eines besonderen Symbols, das aus beliebigen SVG-Objekten (Grundformen, Pfade, Animationen, etc.) bestehen kann. Dieses "Mustersymbol" - Pattern - kann man sich als eine einzelne Kachel vorstellen, mit der Elemente gekachelt werden können.

Die Möglichkeit Muster zu realisieren stellt das **pattern**-Element zur Verfügung. Dieses Element ist lediglich ein Container-Element, das im **defs**-Container platziert werden sollte.

Mit dem Element **pattern** definieren Sie ein besonderes Symbol - eben ein Muster. Daher können Sie innerhalb des **pattern**-Containers alle SVG-Elemente zur Erzeugung von geometrischen Formen, Pfade und sogar Animationen verwenden.

Mit Hilfe der Attribute ***width*** und ***height*** legen Sie die Ausmaße des Symbols fest - also die Größe der einzelnen Kachel.

Die Attribute ***x*** und ***y*** legen fest, an welchem Punkt die erste Kachel platziert wird. Falls sie ***x*** und/oder ***y*** nicht verwenden, wird die Voreinstellung *0* verwendet. Größere Werte als die Breite und/oder die Höhe des gesamten Musters machen hier keinen Sinn, da "in alle Richtungen gekachelt" wird.

Durch die zwei möglichen Werte *objectBoundingBox* (Voreinstellung) und *userSpaceOnUse* des Attributs ***patternUnits*** können Sie festlegen, wie die Werte von ***x***, ***y***, ***width*** und ***height*** interpretiert werden sollen. In der Voreinstellung *objectBoundingBox* werden relative Werte (0 bis 1 oder 0% bis 100%) erwartet, die sich auf das Koordinatensystem des zu füllenden Elements beziehen. Durch Verwendung von *userSpaceOnUse* werden die Werte als absolute Werte im Koordinatensystem des zu füllenden Elements interpretiert (siehe folgendes Beispiel).

Elemente, die ein Muster als Füllung erhalten sollen, müssen es mit Hilfe einer eindeutigen ID referenzieren können. Deshalb müssen Sie dem **pattern**-Element in jedem Fall durch das ***id***-Attribut einen eindeutigen Namen zuordnen.

Über diese ID wird das Muster dann als Wert für die Eigenschaft ***fill*** von einem Element oder Objekt referenziert: fill:url(#pattern-id) oder fill="url(#pattern-id)".

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Das pattern-Element</title>
<desc>Muster definieren und verwenden</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:38px;
            font-weight:bold; stroke:black;}
    ]]>
    </style>

    <pattern id="muster"
      patternUnits="userSpaceOnUse"
      width="10" height="10"
      x="0" y="0">
      <line x1="5" y1="0" x2="5" y2="10"
        stroke="lightgray" />
      <line x1="0" y1="5" x2="10" y2="5"
        stroke="lightgray" />
    </pattern>
    <pattern id="muster_a"
      patternUnits="userSpaceOnUse"
      width="10" height="10"
      x="0" y="0">
```

```
    <rect x="0" y="0" width="10" height="10"
      fill="white" />
    <circle cx="5" cy="5" r="5"
      fill="lightgray" stroke="red" />
  </pattern>
  <pattern id="muster_b"
    patternUnits="userSpaceOnUse"
    width="10" height="10"
    x="5" y="5">
    <rect x="0" y="0" width="10" height="10"
      fill="white" />
    <circle cx="5" cy="5" r="5"
      fill="lightgray" stroke="red" />
  </pattern>
  <pattern id="muster_c"
    patternUnits="userSpaceOnUse"
    width="6" height="6"
    x="0" y="0">
    <rect x="0" y="0" width="6" height="6"
      fill="limegreen" />
    <circle cx="3" cy="3" r="2.2"
      fill="black" />
  </pattern>
 </defs>

 <rect x="10" y="10" width="180" height="180"
  fill="url(#muster)" />
 <text x="90" y="180"
  style="fill:url(#muster_a); font-size:220px;">!</text>
 <text x="20" y="180"
  style="fill:url(#muster_b); font-size:220px;">!</text>
 <text x="20" y="140"
  transform="rotate(-20,20,140)"
  style="fill:url(#muster_c);">MUSTER</text>
</svg>
```

Zur Transformation von Mustern benötigt man - wie bei Verläufen - ein besonderes Attribut, das im Element **pattern** platziert wird. Bei Mustern heißt es ***patternTransform***. ***patternTransform*** aktzeptiert die gleichen Werte und ist daher genauso anzuwenden wie die Attribute ***transform*** oder ***gradientTransform***.

# 13 Animationen

Mit SVG-Elementen können Sie Animationen - also bewegte Bilder - verwirklichen. SVG bietet Elemente mit denen Sie zeitgesteuerte Abläufe in Ihre Grafik einbauen können. Die Animationen können durch verschiedenste Ereignisse gestartet werden, wie z.B. durch das Laden der Grafik, durch den Mausklick eines Users, oder ähnlichem.

Mit den Animations-Elementen von SVG können Sie Attributwerte sichtbar verändern, Objekte an einem Pfad entlang bewegen, Farben sichtbar verändern oder Transformation innerhalb einer bestimmten Zeit ablaufen lassen.

Diese Animations-Elemente sowie deren Kind-Elemente und Attribute, von denen einige aus dem Sprachumfang der W3C Auszeichnungssprache SMIL stammen, werden in den Unterkapiteln dieses Kapitels vorgestellt:

- **animate**,
- **set**,
- **animateMotion**,
- **animateColor** und
- **animateTransform**.

Leider unterstützen noch nicht alle Browser SVG Animationen. So kann momentan (2003) nur der MS IExplorer in Zusammenarbeit mit dem Adobe SVG Viewer Animationen darstellen. Allerdings wird es bestimmt in absehbarer Zeit möglich sein, Animationen auch unter Mozilla, Netscape, Opera oder Konquerer darzustellen (auch unter Linux). Ein Besuch auf den Heimatseiten der Browser gibt Aufschluß über den Status Quo.

## 13.1 Animationen erstellen - animate

Durch die Verwendung des Elements **animate** ist es möglich genau einen Wert eines Attributs oder einer Eigenschaft innerhalb eines bestimmten Zeitablaufs zu verändern.

Das Element, dessen Attributwert geändert werden soll, kann vom **animate**-Element durch das Attribut ***xlink:href*** referenziert werden. Das setzt voraus, dass das zu animierende Element oder Objekt mittels des ***id***-Attributs eindeutig identifizierbar ist.

Eine zweite Möglichkeit besteht darin, das animate-Element innerhalb des zu animierenden Elements zu platzieren. Dazu werden die entsprechenden Elemente mit öffnendem und schließendem Tag, und nicht wie sonst üblich als leeres Element, verwendet (z.B. statt <rect /> verwenden Sie jetzt <rect> .. Bereich für animate .. </rect>).

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Das animate-Element</title>
<desc>Attribut- oder Eigenschaftswerte
zeitgesteuert mit dem animate-Element animieren</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:36px;
            font-weight:bold; fill:limegreen; stroke:black;}
      ellipse {stroke:black;}
    ]]>
    </style>
  </defs>

  <ellipse id="elli" cx="100" cy="100" rx="48" ry="90"
    style="fill:limegreen;" />
  <animate xlink:href="#elli"
    attributeType="XML" attributeName="rx"
    begin="0s" dur="6s"
    from="48" to="198"
    fill="freeze" />

  <ellipse cx="200" cy="100" rx="48" ry="90"
    style="fill:whitesmoke;">
    <animate
      attributeType="XML" attributeName="rx"
      begin="3s" dur="6s"
      from="48" to="198"
      fill="freeze"  />
  </ellipse>

  <text id="tx" x="20" y="115">SVG animate</text>
  <animate xlink:href="#tx"
    attributeName="opacity"
    begin="0s" dur="9s"
    from="0" to="1"
    fill="freeze" />
</svg>
```

Im obigen Beispiel werden 2 Ellipsen durch die Veränderung ihrer Eigenschaft ***rx*** (ihrem Radius zu x-Achse) und ein Text durch Veränderung seiner Eigenschaft ***opacity*** (seiner Durchsichtigkeit) animiert. Beachten Sie, dass die Eigenschaft ***opacity*** im **text**-Element nicht ausdrücklich gesetzt ist - das ist auch nicht notwendig.

Die erste, grüne Ellipse wird über Ihre ID vom **animate**-Element referenziert. Die zweite, "weißrauch"-ige Ellipse beinhaltet das **animate**-Element als Kind-Element. Der Text wird wieder über seine ID referenziert.

Über die Attribute ***attributeName*** und ***attributeType*** wird die zu änderende Eigenschaft ausgewählt. Mit Hilfe des Attributs ***begin*** legen Sie den Startpunkt der Animation fest, das Attribut ***dur*** (duration) bestimmt die Dauer der Animation. Um die Werte der Eingenschaft zu verändern werden die Attribute ***from*** und ***to*** verwendet, die den Anfangswert und den Endwert für die Eigenschaft festlegen. Das Attribut ***fill*** hat in Animationselementen eine besondere Bedeutung. Hiermit wird festgelegt, ob am Ende der Animation das Element mit dem Anfangswert oder dem Endwert für die Eigenschaft angezeigt werden soll.

Da für alle Animationselemente von SVG nahezu die selben Attribute oder Eigenschaften zu verwendet werden können, ist eine detaillierte Übersicht der wichtigsten Attribute, ihrer Bedeutung und Verwendung, ihrer möglichen Werte und dem jeweiligen voreingestellten Wert in den folgenden 4 Kapiteln zu finden.

## 13.2 Attribute zur Zielbestimmung - animAttributeAttrs

***attributeType*** Voreinstellung: *auto*. Mit dem Attribut ***attributeType*** geben Sie den Typ der Eigenschaft bzw. des Attributs an: der voreingestellte Wert *auto* bewirkt, dass der user agent selbst ermittelt ob es sich um eine CSS-Eigenschaft oder um ein XML-Attribut (SVG ist eine XML-basierte Sprache !) handelt. Die beiden anderen möglichen Werte für ***attributeType*** sind daher *CSS* und *XML*.

***attributeName*** Voreinstellung: keine. Mit der Eigenschaft ***attributName*** legen Sie die Eigenschaft bzw. das Attribut im Zielelement fest, die verändert werden soll. Möglicher Wert ist eine der im Zielelement akzeptierten Eigenschaften bzw. Attribute.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200"
 xmlns:xlink="http://www.w3.org/1999/xlink">
<title>animAttributeAttrs</title>
<desc>Attribute/Eigenschaften für die Zielbestimmung</desc>
 <defs>
  <symbol id="smilie">
   <desc>ein Symbol-Smilie</desc>
   <circle cx="20" cy="20" r="15"
    fill="yellow" stroke="black" />
   <circle cx="15" cy="15" r="2"
    fill="black" stroke="black" />
   <circle cx="25" cy="15" r="2"
    fill="black" stroke="black" />
   <line x1="20" y1="18" x2="20" y2="23"
    stroke="black" stroke-width="2" />
   <path id="mund" d="M 13 30 A 5 3 0 0 1 27 30"
    stroke="black" fill="none" stroke-width="2" />
  </symbol>
 </defs>

 <use xlink:href="#smilie" transform="scale(5)" />
 <animate xlink:href="#mund"
  attributeType="XML" attributeName="d"
  begin="1" dur="4"
  from="M 13 30 A 5 3 0 0 1 27 30"
  to="M 13 26 A 5 3 0 0 0 27 26"
  fill="freeze" />
</svg>
```

Die Animation im obigen Beispiel wird durch die Veränderung der Pfadwerte eines Pfades "mund" erzeugt. Die y-Koordinaten der Start- und Endpunkte dieser elliptischen Bogenkurve werden verringert. Außerdem wird der Wert für die Zeichenrichtung auf 0 geändert, d.h. die Kurve wird jetzt entgegen dem Uhrzeigersinn gezeichnet.

Dieser Pfad mit der ID "mund" wird durch die Referenzierung im Attribut ***xlink:href***, das im Element **animate** gesetzt wird, zum Zielelement der Animation.

Durch ***attributeName*** wird das Attribut ***d*** im Zielelement ausgewählt. Mit ***attributeType*** wird die Art dieser Eigenschaft angegeben: *XML*. (***d*** ist keine CSS-Eigenschaft, sondern gehört zum Sprachumfang von SVG. SVG basiert wiederum auf XML.) Es ist jedoch meist nicht notwendig dieses Attribut zu verwenden, da die Voreinstellung *auto* den Attributtyp eigenständig ermittelt.

Die Animation wird nach 1 Sekunde gestartet und läuft 4 Sekunden, wobei das Endbild der Animation am Bildschirm stehen bleibt .. Näheres dazu in den folgenden Unterkapiteln.

## 13.3 Attribute zur Zeitsteuerung - animTimingAttrs

***begin*** Voreinstellung: keine. Mit diesem Attribute legen Sie den Beginn der Animation fest. Möglicher Wert ist eine *Zeitangabe* folgender Syntax:

1. hh:mm:ss.ms - Stunden:Minuten:Sekunden.Millisekunden. Dabei müssen alle Angabe zweistellig erfolgen. Sie können eine Zeitangabe dieser Form auch ohne Angabe der Stunden verwenden, also als mm:ss.ms - das spart Zeit :-). Ein Beispiel: *03:15.10* als Wert für ***begin*** würde die Animation 3 Minuten 15 Sekunden und 10 Millisekunden nach Laden der Grafik starten.
2. Zahl und optional Zeiteinheit - Bei dieser möglichen Zeitangabe können Sie eine Kommazahl gefolgt von einer Zeiteinheit verwenden. Mögliche Zeiteinheiten sind *h* (Stunde), *min* (Minute), *s* (Sekunde) und *ms* (Millisekunde). Die voreingestellte Zeiteinheit ist s (Sekunde), d.h. wenn Sie keine Zeiteinheit angeben wird Sekunde als Zeiteinheit verwendet.

Ein weiterer möglicher Wert ist ein Ereignis - ein *Event*, wie zum Beispiel das Überfahren des Elementes mit der Maus oder ein Klick auf das Element. Folgenden Events sind möglich:

- click - Klick auf das Element
- activate - Aktivierung des Elements durch beliebigen Zeiger (Mausklick, Taste, ..)
- mousedown - Drücken der linken Maustaste
- mouseup - Loslassen der linken Maustaste
- mouseover - Den Bereich des Elements mit dem Mauszeiger betreten
- mousemove - Bewegen der Maus über dem Bereich des Elements
- mouseout - Den Bereich des Elements mit dem Mauszeiger verlassen
- keypress/keydown - Taste gedrückt
- keyup - Taste losgelassen
- focusin - Das Element bekommt den Fokus
- focusout - Das Element verliert den Fokus

Da diese Events einen globalen Charakter haben, ist es empfehlenswert die ID des Elements, das den Event auslösen soll, gefolgt von einem Punkt vor den Event zu stellen, z.B.: *elementid.click* für das click-Ereignis.

***dur*** Voreinstellung: keine. Mit diesem Attribute legen Sie die Dauer der Animation fest und sollte immer verwendet werden. Möglicher Wert ist eine zulässige *Zeitangabe*.

***end*** Voreinstellung: keine. Durch die Verwendung dieses Attributs können Sie den Zeitpunkt für das Ende der Animation bestimmen, also die Animation vorzeitig beenden. Möglicher Wert ist eine zulässige *Zeitangabe* oder ein *Event*. Falls Sie nicht ein Event zum vorzeitigen Abbruch verwenden, achten Sie darauf für ***end*** eine spätere Zeitangabe zu wählen als für ***begin*** festgelegt ist.

***repeatCount*** Voreinstellung: keine. Wenn Sie die Animation wiederholt abspielen möchten, ist dieses Attribute das geeignete. Es akzeptiert entweder eine Ganzzahl größer 0, die Animation wird dann so oft wiederholt wie durch die Ganzzahl angegeben, oder den Wert *indefinite*. Dabei wird die Animation "unendlich" wiederholt, d.h. eigentlich wird sie nur solange wiederholt, bis sie irgendein Event stoppt, wie z.B. das Laden einer anderen Seite.

***repeatDur*** Voreinstellung: keine. Mit diesem Attribut legen Sie eine Zeitdauer für die Wiederholung der Animation fest. Es akzeptiert eine *Zeitangabe* oder *indefinite* (unendlich) als zulässige Wertangabe.

***fill*** Voreinstellung: *remove*. Das Attribut ***fill*** hat im Zusammenhang mit Animationselementen eine andere Bedeutung als ,als bisher gewohnt zum Füllen von Elementen. Sie können mit diesem ***fill***-Attribut festlegen, ob nach Ablauf der Animation das Endbild der Animation oder der Anfangszustand des Elements dargestellt werden soll. Durch Verwendung des Wertes *freeze* bleibt das Endbild der Animation auf dem Bildschirm sichtbar, mit *remove* wird der Anfangszustand des animierten Elements wieder angezeigt.

***min*** Voreinstellung: *0*. Mit diesem Attribut können Sie festlegen, wie lange die Animation innerhalb der mit ***dur*** festgelgten Dauer, mindestens abgespielt wird. Als Wert wird eine Zeitangabe akzeptiert. Falls Sie die Animation durch Events enden lassen (z.B. durch Mausklick) kann dieses Attribut sinnvoll sein um eine Mindestspieldauer festzulegen. Möglicherweise wird dieses Attribut noch nicht unterstützt.

***max*** Voreinstellung: *linear*. Mit diesem Attribut können Sie festlegen, wie lange die Animation innerhalb der mit ***dur*** festgelgten Dauer, höchstens (maximal) abgespielt wird. Als Wert wird eine Zeitangabe akzeptiert. Falls Sie längere Animation (z.B. durch Wiederholungen) mittels Events enden lassen

(z.B. durch Mausklick) kann dieses Attribut sinnvoll sein um eine maximale Spieldauer festzulegen. Möglicherweise wird dieses Attribut noch nicht unterstützt.

***restart*** Voreinstellung: *always*. Durch das Attribut restart können Sie das Abspielverhalten der Animation bei Wiederholungen mit Hilfe von drei Werten festlegen. Der voreingestellte Wert *always* bewirkt, dass die Animation grundsätzlich jederzeit neu gestartet werden kann. Der Wert *whenNotActive* legt fest, dass die Animation nicht neu gestartet werden kann, falls sie gerade läuft. Der letzte zulässige Wert *never* sorgt dafür, dass die Animation nie neu gestartet werden kann, also genau 1x abgespielt wird.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="410" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>animTimingAttrs</title>
<desc>Attribute für die Zeitsteuerung von Animationen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:24px;
            font-weight:bold; fill:limegreen;}
      rect {fill:url(#verlauf);}
    ]]>
    </style>
    <radialGradient id="verlauf"
      cx=".5" cy=".5" r=".5">
      <stop offset="0" stop-color="limegreen" />
      <stop offset=".8" stop-color="white" />
    </radialGradient>
  </defs>

<!-- Hintergrundanimation -->
<!-- bg -->
  <rect id="bg"
    x="0" y="0" width="410" height="200" />
  <animate xlink:href="#bg"
    attributeName="x"
    begin="0" dur="6"
    from="410" to="-410"
    repeatDur="15" />

<!-- Beginn der interaktiven Animationen -->
<!-- txt1 -->
  <text id="txt1" x="20" y="40"
    style="stroke:black;">
    click - jederzeit!
  </text>
  <animate xlink:href="#txt1"
    attributeName="opacity"
    begin="txt1.click" dur="3"
    from="1" to="0" />

<!-- txt2 -->
  <text id="txt2" x="20" y="80"
    style="fill:red">
    auch click - whenNotActive!
  </text>
  <animate xlink:href="#txt2"
    attributeName="opacity"
    begin="txt2.click" dur="3"
    from="1" to="0"
    restart="whenNotActive" />
```

```
<!-- txt3 -->
  <text id="txt3" x="20" y="120"
    style="fill:blue">
    2 x animate click!
  </text>
  <animate xlink:href="#txt3"
    attributeName="rotate"
    begin="txt3.click" dur="3"
    from="0" to="360"
    repeatCount="3"
    restart="whenNotActive" />
  <animate xlink:href="#txt3"
    attributeName="font-size"
    begin="txt3.click" dur="3"
    from="24" to="30"
    fill="freeze" />

<!-- txt3 -->
  <text id="txt4" x="20" y="170"
    style="fill:limegreen; font-size:44px; stroke:black;">
    mouseover!        mouseout!
  </text>
  <animate xlink:href="#txt4"
    attributeName="x"
    begin="txt4.mouseover" dur="1"
    from="20" to="-392"
    fill="freeze" />
  <animate xlink:href="#txt4"
    attributeName="x"
    begin="txt4.mouseout" dur="1"
    from="-392" to="20"
    fill="freeze" />
</svg>
```

Im obigen Beispiel sind 5 Animationen dargestellt, die durch unterschiedliche Verwendung der zeitsteuernden Attribute verschiedenste Eigenschaften aufweisen. Nachfolgend eine Kurzbeschreibung dieser Animationen.

- **Animiertes Element (id): bg**
- Animiertes Attribut: x
- Beginn: 0 Sekunden nach Laden der Grafik
- Dauer der Animation: 6 Sekunden
- Gesamtdauer der Animation: 15 Sekunden
- Anzeige nach Abspielen der Animation: Orginalbild
- **Animiertes Element (id): txt1**
- Animiertes Attribut: opacity
- Beginn: Beim Anklicken des Elements txt1
- Dauer der Animation: 3 Sekunden
- Gesamtdauer der Animation: 3 Sekunden
- Anzeige nach Abspielen der Animation: Orginalbild
- **Animiertes Element (id): txt2**
- Animiertes Attribut: opacity
- Beginn: Beim Anklicken des Elements txt2, nur wenn die Animation nicht gerade abgespielt wird.
- Dauer der Animation: 3 Sekunden
- Gesamtdauer der Animation: 3 Sekunden
- Anzeige nach Abspielen der Animation: Orginalbild
- **Animiertes Element (id): txt3**
- Animiertes Attribut 1: rotate
- Beginn: Beim Anklicken des Elements txt3, nur wenn die Animation nicht gerade abgespielt wird.
- Dauer der Animation: 3 Sekunden

- Gesamtdauer der Animation: 9 Sekunden (3 Abläufe)
- Anzeige nach Abspielen der Animation: Orginalbild
- ---
- Animiertes Attribut 2: font-size
- Beginn: Beim Anklicken des Elements txt3,
- Dauer der Animation: 3 Sekunden
- Gesamtdauer der Animation: 3 Sekunden
- Anzeige nach Abspielen der Animation: letztes Bild der Animation
- **Animiertes Element (id): txt4**
- Beginn 1: Beim Betreten des Elementbereichs von txt4.
- Animiertes Attribut: x
- Dauer der Animation: 1 Sekunde
- Gesamtdauer der Animation: 1 Sekunde
- Anzeige nach Abspielen der Animation: letztes Bild der Animation
- ---
- Beginn 2: Beim Verlassen des Elementbereichs von txt4.
- Animiertes Attribut: x
- Dauer der Animation: 1 Sekunde
- Gesamtdauer der Animation: 1 Sekunde
- Anzeige nach Abspielen der Animation: letztes Bild der Animation

## 13.4 Attribute zur Wertänderung - animValueAttrs

***from*** Voreinstellung: keine. Mit diesem Attribut legen Sie den Startwert für die Eigenschaft des animierten Elements fest. Akzeptierte Werte sind abhängig von der entsprechenden Eigenschaft.
***to*** Voreinstellung: keine. Mit diesem Attribut legen Sie den Endwert für die Eigenschaft des animierten Elements fest. Akzeptierte Werte sind abhängig von der entsprechenden Eigenschaft.
***by*** Voreinstellung: keine. Mit diesem Attribut können Sie bei numerischen Eigenschaften einen additiven Endwert angeben. Der im Attribut ***by*** verwendete Wert wird mit dem im Attribut ***from*** festgelegten Wert addiert (from-Wert + by-Wert = to-wert). ***by*** darf nicht mit dem Attribut ***to*** gemeinsam verwendet werden, da es in diesem Falle ignoriert wird.
***values*** Voreinstellung: keine. Durch Verwendung des Attributs ***values*** können Sie beginnned mit dem Startwert und abschließend mit dem Endwert, eine Liste von Zwischenwerten angeben, die bei der Berechnung der Animation berücksichtigt werden. Alle Werte werden untereinander durch Semikolon getrennt und sind in ihrer Art abhängig von der animierten Eigenschaft. In den meisten Fällen sind es Längenangaben. Verwenden Sie ***values*** anstelle der Attribute ***from***, ***to*** und ***by***. Bei gleichzeitiger Verwendung werden die Angaben von ***values*** interpretiert, wogegen die anderen Angaben nicht beachtet werden. Um zusätzlich auch die Zeitsequenzen zwischen den einzelnen Übergangswerten festzulegen, können Sie das Attribut ***keyTimes*** verwenden.
***calcMode*** Voreinstellung: *linear*. Ausnahme! Voreinstellung für das Element **animateMotion**: *paced*. Durch das Attribut ***calcMode*** können Sie die Art und Weise, wie der user agent die Animation berechnen soll, einstellen. Dazu können Sie 4 Werte verwenden:

1. Die Verwendung des voreingestellten Wertes *linear* führt dazu, dass der zeitliche Übergang zwischen den einzelnen Werten gleichmäßig lang ist, und daher der Übergang fließend dargestellt wird.
2. Der Wert *discrete* bewirkt eine ähnliche Berechnung der Animation mit dem Unterschied, dass der Übergang sprunghaft dargestellt wird.
3. Durch den Wert *paced* erreichen Sie, dass die Veränderung zwischen den Werten gleichmäßig berechnet wird, dadurch ist unter Umständen der zeitliche Übergang zwischen den Werten unterschiedlich lang. Der Übergang wird fließend dargestellt.
4. Wenn Sie den Wert *spline* verwenden, findet die Berechnung der Übergänge anhand einer kubischen Bezierkurve statt, die sie mit Hilfe des Attributes ***keySplines*** festlegen müssen.

Interressant wird die Verwendung dieses Attributs in Zusammenhang mit dem Attribut ***values***.
***keyTimes*** Voreinstellung: keine. Dieses Attribut ist nur in Zusammenhang mit der Verwendung von ***values*** möglich. Durch ***keyTimes*** können Sie die zeitlichen Übergänge zwischen den einzelnen Werten durch Angabe von relativen Werten zwischen 0 und 1, die sich auf die Gesamtabspieldauer der Animation beziehen, festlegen. Dabei bezieht sich jeder Wert aus der, durch Semikolon getrennten Liste von relativen Angaben, auf die gleichplatzierten Werte der Liste von Werten im ***values***-Attribut. Daher muß die Anzahl der Werte in beiden Attributen übereinstimmen.
***keySplines*** Voreinstellung: keine. Dieses Attribut hat nur dann Gültigkeit, wenn zuvor das Attribut ***calcMode*** mit dem Wert *spline* verwendet wurde. Hier können Sie die Werte für die Kontrollpunkte der kubische Bezierkurven eintragen, die zur Berechnung der Animation zwischen den einzelnen Animationswerten (values) verwendet wird. Diese Funktionalität ist etwas für ganz besondere Ansprüche, daher verweise ich für weitere Informationen auf die SVG-Recommendation des W3C.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="600" height="300"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>animValueAttrs</title>
<desc>Attribute/Eigenschaften für die Werteveränderung
im Zielelement einer Animation</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;
           fill:black; text-rendering:optimizeSpeed;}
      rect {fill:limegreen; stroke:black;}
      line {stroke:black;}
```

```
    ]]>
    </style>
  </defs>

<!-- kasten 1 -->
  <g>
    <text x="25" y="36">
      kasten 1:
      &amp;#160;&amp;#160;&amp;#160;
      from 15 to 565
    </text>
    <line x1="20" y1="40" x2="580" y2="40" />
  </g>
  <rect id="kasten1" x="15" y="45" width="20" height="10" />
  <animate xlink:href="#kasten1"
    attributeName="x"
    begin="go.click" dur="10"
    from="15" to="565"
    restart="whenNotActive"
    fill="freeze" />

<!-- kasten 2 -->
  <g>
    <text x="25" y="76">
      kasten 2:
      &amp;#160;&amp;#160;&amp;#160;
      from 15 by 550
    </text>
    <line x1="20" y1="80" x2="580" y2="80" />
  </g>
  <rect id="kasten2" x="15" y="85" width="20" height="10" />
  <animate xlink:href="#kasten2"
    attributeName="x"
    begin="go.click" dur="10"
    from="15" by="550"
    restart="whenNotActive"
    fill="freeze" />

<!-- kasten 3 -->
  <g>
    <text x="25" y="116">
      kasten 3:
      &amp;#160;&amp;#160;&amp;#160;
      values 15;565;15
    </text>
    <line x1="20" y1="120" x2="580" y2="120" />
  </g>
  <rect id="kasten3" x="15" y="125" width="20" height="10" />
  <animate xlink:href="#kasten3"
    attributeName="x"
    begin="go.click" dur="10"
    values="15;565;15"
    restart="whenNotActive"
    fill="freeze" />

<!-- kasten 4 -->
  <g>
    <text x="25" y="156">
      kasten 4:
      &amp;#160;&amp;#160;&amp;#160;
      values 15;565;15 keyTimes 0;.8;1
    </text>
    <line x1="20" y1="160" x2="580" y2="160" />
  </g>
```

```
  <rect id="kasten4" x="15" y="165" width="20" height="10" />
  <animate xlink:href="#kasten4"
    attributeName="x"
    begin="go.click" dur="10"
    values="15;565;15"
    keyTimes="0;.8;1"
    restart="whenNotActive"
    fill="freeze" />

<!-- kasten 5 -->
  <g>
    <text x="25" y="196">
      kasten 5:
      &amp;#160;&amp;#160;&amp;#160;
      values 15;165;505;565;505;165;15
      calcMode discrete
    </text>
    <line x1="20" y1="200" x2="580" y2="200" />
  </g>
  <rect id="kasten5" x="15" y="205" width="20" height="10" />
  <animate xlink:href="#kasten5"
    attributeName="x"
    begin="go.click" dur="10"
    values="15;165;505;565;505;165;15"
    calcMode="discrete"
    restart="whenNotActive"
    fill="freeze" />

<!-- START/RESTART-Button -->
  <g id="go">
    <rect x="15" y="255" width="60" height="20" />
    <text id="rs" x="18" y="270"
      opacity="0">
      RESTART
    </text>
    <text id="s" x="27" y="270">
      START
    </text>
  </g>
  <animate xlink:href="#s"
    attributeName="opacity"
    begin="go.click" dur="10"
    values="1;.9;.1;0"
    keyTimes="0;.4;.6;1"
    restart="never"
    fill="freeze" />
  <animate xlink:href="#rs"
    attributeName="opacity"
    begin="go.click" dur="10"
    values="0;.1;.9;1"
    keyTimes="0;.4;.6;1"
    restart="never"
    fill="freeze" />
</svg>
```

Die 6 Animationen im obigen Beispiel verdeutlichen den Gebrauch der Attribute ***by***, ***values***, ***keyTimes*** und ***calcMode***. Alle Animationen werden durch das Event *click* auf das Objekt "go", den START/ RESTART Button, ausgelöst und haben jeweils eine Gesamtspieldauer von 10 Sekunden, was mittels des Attributs ***dur*** definiert ist. Die Animation der Kästen kann nach Beendigung erneut gestartet werden, die Animation des Button kann nicht wieder gestartet werden (außer nach einem erneuten Laden der Seite). Hierfür ist die Verwendung von ***restart*** mit den Werten *whenNotActive* bzw. *never* verantwortlich. Durch Verwendung des speziellen Attributes ***fill*** und dem Wert *freeze*, ist für alle Animationen das letzte Bild der Animation als Endbild festgelegt.

Die Animation **kasten 1** erhält den absoluten Startwert (15) durch die Verwendung von ***begin*** und ihren absoluten Endwert (565) durch das Attribut ***to***.

Die Animation **kasten 2** erhält den absoluten Startwert (15) ebenfalls durch ***begin***, der Endwert wird jedoch durch die Verwendung von ***by*** festgelegt. Der für ***by*** angegebene Wert (550) wird relativ zum Startwert interpretiert. Das bedeutet, dass dieser Wert zum Startwert addiert wird und auf diese Art den absoluten Endpunkt bestimmt: (Startwert) 15 + (by-Wert) 550 = (Endwert) 565

Für die Animation **kasten 3** wird das Attribut ***values*** verwendet, um die Werte für die Zieleigenschaft festzulegen. Mit diesem Attribut ist es möglich, mehrere Werte anzugeben. Dabei ist der erste Wert, der durch Semikolon getrennten Liste, der Startwert (15) und der letzte Wert der Liste der Endwert (15). Dazwischen können beliebig viele Zwischenwerte angegeben werden. In diesem Beispiel ist es allerdings nur einer (565). So ist es möglich den Kasten an der Linie hin und her wanderen zu lassen.

Die Animation **kasten 4** ist bis auf die zusätzliche Verwendung des Attributs ***keyTimes*** mit der Animation kasten 3 identisch. Das Attribut ***keyTimes*** hat als Wert eine Liste von relativen Werten, die sich auf die Gesamtabspieldauer (10 Sekunden) der Animation beziehen. Daher werden die einzelnen Werte entweder von 0 bis 1 oder von 0% bis 100% angegeben. Die Anzahl der Werte muß mit der Anzahl der Werte von ***values*** übereinstimmen, da sich jeder einzelne Wert von ***keyTimes*** auf den gleichplatzierten Wert von ***values*** bezieht. Im Beispiel ist dem Attribut ***values*** eine Liste von 3 Werten (15;565;15) zugeordnet, daher hat auch das Attribut ***keyTimes*** 3 Werte in der seiner Liste (0;.8;1). Der erste Wert dieser Liste ist immer die Startzeit 0 (Beginn der Gesamtspieldauer), der letzte Wert dieser Liste immer der Endwert 1 (Ende der Gesamtspieldauer). Der Wert oder die Werte dazwischen gibt/geben eine relative Zeitdauer an. In unserem Beispiel ist es nur ein Wert (.8 oder 80%). Die Animation beginnt also bei 0 Sekunden: der Kasten befindet sich auf der x-Koordinate (15). Nach 8 Zehnteln oder 80% der Gesamtzeit, also nach 8 Sekunden, hat der Kasten die Koordinate (565) erreicht. In der restlichen Zeit, also den letzten 2 Sekunden, bewegt sich der Kasten wieder zum Endpunkt der Animation hin (15), den er am Ende der Animation erreicht hat.

Die Animation **kasten 5** zeigt die Verwendung von ***calcMode*** und dem Wert *discrete*. Die in ***values*** angegebenen Werte werden nicht fließend, sondern sprunghaft erreicht. Der Kasten hüpft von Wert zu Wert.

Die letzte Animation **go** ändert die Button-Beschriftung nur beim ersten click. Dazu wird die Eigenschaft ***opacity*** in zwei Text-Elementen geändert (von 1 auf 0 und von 0 auf 1). Die Werteveränderung wird durch ***values*** und ***keyTimes*** näher bestimmt.

## 13.5 Weitere Attribute - animAdditionAttrs

***additive*** Voreinstellung: *replace*. Durch die Verwendung dieses Attributs mit dem, außer *replace* einzig möglichen Wert *sum*, haben Sie die Möglichkeit den Startwert der Animation additiv zu bestimmen. Der Wert, der für den Startwert im Attribut ***from*** (oder in ***values***) angegeben wird, ist normalerweise ein absoluter Wert. Wenn Sie allerdings ***additive*** mit dem Wert *sum* verwenden, wird der für ***from*** angegebene Wert, dem orginalen Wert der Eigenschaft oder des Attribus im Zielelement hinzuaddiert. Der Startwert wird also wie folgt berechnet: Wert des Attributs im Zielelement + from-Wert.

***accumulate*** Voreinstellung: *none*. Das Attribut ***accumulate*** akzeptiert ebenfalls nur 2 mögliche Werte. Die Voreinstellung *none* und den Wert *sum*. Das Attribut wird nur in Verbindung mit ***repeatCount*** oder ***repeatDur*** interpretiert. Bei Verwendung des Werts *sum* wird der mit ***from*** festgelegte Startwert für jeden weiteren Durchlauf der Animation mit dem aktuellen Endwert addiert. Der Wert dieser Summe ist dann der Startwert den jeweils nächsten Duchlaufs.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="460" height="300"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>animAdditonAttrs</title>
<desc>Weitere Attribute/Eigenschaften für Animationen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           fill:black;}
      circle {stroke:black;}
      line {stroke:black;}
    ]]>
    </style>
  </defs>
<!-- Animationen k4 bis k1 -->
  <circle id="k4" cx="260" cy="200" r="20"
    style="fill:limegreen;" />
  <animate xlink:href="#k4"
    attributeName="r"
    begin="k4.click" dur="4"
    from="20" to="50"
    restart="whenNotActive"
    fill="freeze"
    repeatCount="5"
    accumulate="sum" />

  <circle id="k3" cx="260" cy="80" r="20"
    style="fill:darkgoldenrod;"  />
  <animate xlink:href="#k3"
    attributeName="r"
    begin="k3.click" dur="4"
    from="0" to="10"
    restart="whenNotActive"
    fill="freeze"
    repeatCount="5"
    accumulate="sum" />

  <circle id="k2" cx="80" cy="200" r="20"
    style="fill:dimgray;"  />
  <animate xlink:href="#k2"
    attributeName="r"
    begin="k2.click" dur="4"
    from="20" to="50"
```

```
    restart="whenNotActive"
    fill="freeze"
    additive="sum" />

  <circle id="k1" cx="80" cy="80" r="20"
    style="fill:dodgerblue;"  />
  <animate xlink:href="#k1"
    attributeName="r"
    begin="k1.click" dur="4"
    from="20" to="50"
    restart="whenNotActive"
    fill="freeze" />

<!-- die Texte -->
  <text x="80" y="50"
    style="font-weight:bold;">k1
  </text>
  <text x="80" y="170"
    style="font-weight:bold;">k2
  </text>
  <text x="260" y="50"
    style="font-weight:bold;">k3
  </text>
  <text x="260" y="170"
    style="font-weight:bold;">k4
  </text>
  <text x="110" y="60">from 20
    <tspan x="110" dy="18">to 50</tspan>
  </text>
  <text x="110" y="180">from 20
    <tspan x="110" dy="18">to 50</tspan>
    <tspan x="110" dy="18">additive sum</tspan>
  </text>
  <text x="290" y="60">from 0
    <tspan x="290" dy="18">to 10</tspan>
    <tspan x="290" dy="18">repeatCount 5</tspan>
    <tspan x="290" dy="18">accumulate sum</tspan>
  </text>
  <text x="290" y="180">from 20
    <tspan x="290" dy="18">to 50</tspan>
    <tspan x="290" dy="18">repeatCount 5</tspan>
    <tspan x="290" dy="18">accumulate sum</tspan>
  </text>
  <text x="15" y="270">
    Klicken Sie in die Kreise, um die Animation zu starten.
  </text>
</svg>
```

Im obigen Beispiel sind 4 Kreise zu sehen, die jeweils auf Mausklick animiert werden. Die dabei berechneten Werte werden nachfolgendend für jede Animation aufgeführt.

Animation **k1**.

- Startwert des Radius: from **20**
- Endwert des Radius: to **50**

Animation **k2** additive sum.

- Startwert des Radius: from 20 + r 20 = **40**
- Endwert des Radius: to 50 + r 20 = **70**

Animation **k3** accumulate sum.

- Anzahl der Durchläufe: repeatCount 5
- Durchlauf 1:
- Startwert des Radius: from **0**

- Endwert des Radius: to 10
- Durchlauf 2:
- Startwert des Radius: from 0 + aktueller Endwert 10 = 10
- Endwert des Radius: to 10 + aktueller Endwert 10 = 20
- Durchlauf 3:
- Startwert des Radius: from 0 + aktueller Endwert 20 = 20
- Endwert des Radius: to 10 + aktueller Endwert 20 = 30
- Durchlauf 4:
- Startwert des Radius: from 0 + aktueller Endwert 30 = 30
- Endwert des Radius: to 10 + aktueller Endwert 30 = 40
- Durchlauf 5:
- Startwert des Radius: from 0 + aktueller Endwert 40 = 40
- Endwert des Radius: to 10 + aktueller Endwert 40 = **50**

Animation **k4** accumulate sum.

- Anzahl der Durchläufe: repeatCount 5
- Durchlauf 1:
- Startwert des Radius: from **20**
- Endwert des Radius: to 50
- Durchlauf 2:
- Startwert des Radius: from 20 + aktueller Endwert 50 = 70
- Endwert des Radius: to 50 + aktueller Endwert 50 = 100
- Durchlauf 3:
- Startwert des Radius: from 20 + aktueller Endwert 100 = 120
- Endwert des Radius: to 50 + aktueller Endwert 100 = 150
- Durchlauf 4:
- Startwert des Radius: from 20 + aktueller Endwert 150 = 170
- Endwert des Radius: to 50 + aktueller Endwert 150 = 200
- Durchlauf 5:
- Startwert des Radius: from 20 + aktueller Endwert 200 = 220
- Endwert des Radius: to 50 + aktueller Endwert 200 = **250**

## 13.6 Attributwerte verändern - set

Mit dem Element **set** können Sie Eigenschafts- bzw. Attributwerte in einem Zielelement für eine bestimmte Zeit oder für "unendlich" verändern. Außer den Attributen der Gruppen animValueAttrs (Kapitel 12.4) und animAdditionAttrs (Kapitel 12.5) können alle Attribute für Animationselemente verwendet werden.

Als zusätzliches Attribut wird im **set**-Element ein Attribut ***to*** verwendet, das aber nichts mit dem gleichnamigen Attribut aus der Attributgruppe animValueAttrs zu tun hat. Im **set**-Element legen Sie mit dem Attribut ***to*** den neuen Wert für das Zielelement fest!

Der Beginn der Veränderung wird mit dem Attribut ***begin*** festgelegt. Die Dauer der Veränderung mit dem Attribut ***dur***. Wenn Sie für ***dur*** den Wert *indefinite* angeben, haben Sie den Zielwert dauerhaft verändert. Diesen Effekt können Sie auch durch Verwendung von ***fill*** mit dem Wert *freeze* erreichen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="300" height="300"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das set-Element</title>
<desc>Zuweisung von bestimmenten Werten
für ein Attribut für eine gewisse Zeit (auch dauerhaft)</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:24px;
            font-weight:bold; fill:black;}
      circle {fill:url(#verlauf); stroke-width:5px;}
    ]]>
    </style>
    <radialGradient id="verlauf"
      cx=".5" cy=".5" r=".5">
      <stop offset="0" stop-color="limegreen" />
      <stop offset="1" stop-color="white" />
    </radialGradient>
  </defs>

<!-- 5 Kreise, deren Radius temporär verändert wird -->
  <circle id="k1" cx="40" cy="40" r="20"
    style="stroke:gray" />
  <set xlink:href="#k1"
    attributeName="r"
    begin="1" dur="3"
    to="40" />
  <circle id="k2" cx="210" cy="80" r="70"
    style="stroke:black;" />
  <set xlink:href="#k2"
    attributeName="r"
    begin="00:01.50" dur="2"
    to="30" />
  <circle id="k3" cx="140" cy="160" r="40"
    style="stroke:red;" />
  <set xlink:href="#k3"
    attributeName="r"
    begin="2" dur="2"
    to="20" />
  <circle id="k4" cx="60" cy="190" r="10"
    style="stroke:blue;" />
  <set xlink:href="#k4"
    attributeName="r"
    begin="3" dur="3"
    to="90" />
```

```
  <circle id="k5" cx="230" cy="200" r="30"
    style="stroke:limegreen;" />
  <set xlink:href="#k5"
    attributeName="r"
    begin="00:02.5" dur="3"
    to="50" />

<!-- dauerhafte Veränderung der Durchsichtigkeit des Textes -->
  <text id="ende" x="40" y="280"
    style="opacity:0;">ENDE</text>
  <set xlink:href="#ende"
    attributeName="opacity"
    begin="00:06.5" dur="indefinite"
    to="1" />
</svg>
```

## 13.7 Bewegung entlang eines Pfades - animateMotion

Um Objekte zu bewegen, können sie mit Hilfe von **animate** deren Koordinatenwerte verändern oder Sie können mittels **animateTransform** Verschiebungen animiert darstellen (siehe Kapitel 13.9).

Durch das Element **animateMotion** haben sie zusätzlich die Möglichkeit Objekte entlang eines Pfades zu bewegen. Diesem Element stehen die Attribute aller Atttributgruppen für Animationen zur Verfügung, sowie zusätzlich die Attribute path und rotate. Außerdem kann **animateMotion** ein Kind-Element **mpath** zugeordnet werden.

Mittels ***path*** wird der Pfad festgelegt, an welchem das Objekt bewegt werden soll. Die Werte für ***path*** sind von der gleichen Art wie die Werte des ***d***-Attributs im **path**-Element.

Mit rotate können Sie die Drehung des Objekts während der Bewegung entlang des Pfades einstellen. Dieses Attribut akzeptiert 3 mögliche Werte bzw. Wertangaben: *auto*, *auto-reverse* oder eine Winkelangabe.

1. *auto* bewirkt, dass das Objekt dem Verlauf des Pfades entsprechend rotiert wird. Bei einem Wellenverlauf würde ein Objekt demnach immer entsprechend dem Verlauf gedreht.
2. *auto-reverse* entspricht dem Wert *auto* bis auf den Umstand, das das Objekt "auf dem Kopf steht", also grundsätzlich erst um 180 Grad gedreht wird.
3. Durch eine Winkelangabe legen Sie den Drehungswinkel definitiv fest, d.h. der Winkel wird während der gesamten Pfandwanderung nicht verändert. Wenn Sie den Wert *0* verwenden, bleibt das Objekt im Orginalzustand, während es am Pfad entlang bewegt wird.

Eine weitere Möglichkeit einen Pfad als Animationspfad festzulegen, bietet das leere Kind-Element **mpath**. Über das Attribut ***xlink:href*** können Sie hier ein vorher definierten Pfad über dessen ID referenzieren. Sie sollten das Kind-Element **mpath** logischerweise nicht gemeinsam mit dem Attribut **path** in einem **animateMotion**-Element verwenden.

Beispiel Quellcode

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="280"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das animateMotion-Element</title>
<desc>Bewegung entlang eines Pfades 1</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:20px;
           fill:black;}
      rect {fill:limegreen; stroke:white; stroke-width:2px; opacity:.8}
      path {fill:none; stroke:black; stroke-width:2px;}
    ]]>
    </style>
    <linearGradient id="verlauf"
      x1="0" y1="0" x2="1" y2="1">
      <stop offset=".4" stop-color="white" />
      <stop offset=".6" stop-color="limegreen" />
    </linearGradient>
    <path id="p" d="M 30 80 c 50 -50 150 30 200 20 s 50 -30 100 -20" />
  </defs>

<!-- Hintergrund und Anzeige der Animationspfade  -->
  <rect x="1" y="1" width="398" height="298"
    style="fill:url(#verlauf); stroke:none; opacity:1;" />
  <use xlink:href="#p" />
  <use xlink:href="#p" transform="translate(0,60)" />
  <use xlink:href="#p" transform="translate(0,120)" />

<!-- die drei Animationen -->
  <rect id="re1" x="0" y="0" width="50" height="20"
    visibility="hidden" />
```

```
  <animateMotion xlink:href="#re1"
    begin="go.click" dur="6"
    rotate="auto"
    path="M 30 80 c 50 -50 150 30 200 20 s 50 -30 100 -20"
    fill="freeze" />
  <set xlink:href="#re1"
    attributeName="visibility"
    begin="go.click" dur="6"
    to="visible" />

  <rect id="re2" x="0" y="0" width="50" height="20"
    visibility="hidden" />
  <animateMotion xlink:href="#re2"
    begin="go.click" dur="6"
    rotate="auto-reverse"
    path="M 30 140 c 50 -50 150 30 200 20 s 50 -30 100 -20"
    fill="freeze"  />
  <set xlink:href="#re2"
    attributeName="visibility"
    begin="go.click" dur="6"
    to="visible" />

  <rect id="re3" x="0" y="0" width="50" height="20"
    visibility="hidden" />
  <animateMotion xlink:href="#re3"
    begin="go.click" dur="6"
    rotate="0"
    path="M 30 200 c 50 -50 150 30 200 20 s 50 -30 100 -20"
    fill="freeze"  />
  <set xlink:href="#re3"
    attributeName="visibility"
    begin="go.click" dur="6"
    to="visible" />

<!-- Text  -->
  <text x="20" y="60">rotate auto</text>
  <text x="20" y="120">rotate auto-reverse</text>
  <text x="20" y="180">rotate 0</text>
  <rect id="go" x="20" y="240" width="50" height="20"
    style="fill:limegreen; stroke:white; opacity:1;" />
  <text x="80" y="258" style="fill:white;">
    &lt;&lt; START
  </text>
</svg>
```

**Beachten Sie:** Der Startpunkt des Elements oder des Objekts, welches animiert wird, wird als relativ zum Startpunkt des Animationspfades interpretiert. Der Startpunkt der Animation ist somit immer: **x,y-Wert des Elements + x,y-Wert des Pfads = x,y-Wert des Elements beim Start der Animation.** Man kann diesen Umstand auch auf folgende Art beschreiben: Der eigentliche Nullpunkt des Koordinatensystems, in dem sich das zu animierenende Element befindet stimmt immer mit dem Anfangspunkt des Animationspfades überein.

Daher haben die animierten Rechtecke im obigen Beispiel ihren Anfangspunkt bei 0,0. Der Anfangspunkt der Animation fällt auf diese Weise mit dem Anfangspunkt des Animationspfades zusammen, welcher durch das Attribut ***path*** im jeweiligen **animateMotion**-Element festgelegt ist.

Die Ränder der Rechtecke wären normalerweise vor dem Start der Animation innerhalb der Grafik sichtbar. Und leider können die Elemente nicht im **defs**-Bereich definiert werden, da sie dann nicht direkt durch **animateMotion** verwendet werden können. Es muß hierfür zuerst eine Instanz des Elements mit **use** in die Grafik eingebunden werden, deren Ränder dann ebenfalls am Bildschirm zu sehen wären.

Im obigen Beispiel werden zur Lösung dieses Problems die zu animierenden Elemente nur während des Abspielzeitraums sichtbar gemacht. Dazu wird durch das Element **set** die Eigenschaft ***visibility*** in den jeweiligen Ziel-Rechtecken für die Dauer der Animation auf *visible* (sichtbar) eingestellt.

Eine weitere Möglichkeit zeigt das folgende Beispiel ..

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="220"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das animateMotion-Element</title>
<desc>Bewegung entlang eines Pfades 2</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:14px;
            font-weight:bold; fill:black;}
      path {fill:none; stroke:black;}
      polygon {fill:seagreen; stroke:black;}
      circle {fill:black;}
    ]]>
    </style>
    <path id="welle1" d="M 0 0 q 50 -50, 100 0
                         t 100 0
                         t 100 0
                         t 100 0" />
    <g id="boot">
      <polygon points="0 0,
                       -20 0,
                       -30 -15,
                       -2 -15,
                       -2 -35,
                       -2 -35,
                       2 -15,
                       35 -15,
                       20 0" />
      <circle id="startpunkt" cx="0" cy="0" r="2" />
      <text x="-21" y="-2.8"
        style="font-size:14px;
               text-rendering:optimizeSpeed;
               fill:white;">
        sailor
      </text>
    </g>
  </defs>

<!-- Anzeige des Animationspfads -->
  <use xlink:href="#welle1" x="40" y="100" />

<!-- Animation -->
  <use id="go" xlink:href="#boot" x="40" y="100" />
  <animateMotion xlink:href="#boot"
    begin="go.click" dur="6"
    rotate="auto">
      <mpath xlink:href="#welle1" />
  </animateMotion>

<!-- Texte, Linie und Punkt -->
  <text x="35" y="210" style="fill:seagreen;">
    Klicken Sie zum Start der Animation auf das Boot.
  </text>
  <text x="10" y="30">Im defs-Bereich:</text>
  <text x="145" y="30">Anfangspunkt des Boots: 0,0
    <tspan x="145" dy="16">
```

```
    Anfangspunkt des Pfades: M 0,0
   </tspan>
  </text>
  <line x1="40" y1="100" x2="40" y2="150" stroke="black"/>
  <text x="35" y="170">Anfangspunkt der use-Instanz des Boots: 40,100
   <tspan x="35" dy="16">
    Anfangspunkt des Pfades: 40,100
   </tspan>
  </text>
  <circle cx="40" cy="100" r="2" />
</svg>
```

Im vorangegangenen Beispiel sind sowohl der Animationspfad, wie auch das zu animierende Objekt innerhalb des **defs**-Containers definiert. Sie werden also nicht direkt am Bildschirm angezeigt.

Der Punkt des Objekts, der genau auf dem Nullpunkt des Koordinatensystems liegt (quasi der Nullpunkt des Objekts), befindet sich genau in der Mitte der unteren Linie des Polygons (in der Grafik als kleiner, schwarzer Punkt gekennzeichnet). Der Startpunkt des Animationspfades ist mit *0,0* festgelegt, d.h. er bezieht sich in diesem Fall auf den Punkt ebenfalls auf den Nullpunkt des Koordinatensystems (0,0).

Das zu animierende Objekt "boot" wird innerhalb der Grafik durch das **use**-Element eingebunden. Dabei legen die Attribute ***x*** und ***y*** innerhalb dieses Elements die Platzierung dieser Instanz des Objekts innerhalb der Grafik fest (40,100). Beachten Sie: Es handelt sich hier um eine Instanz des Objekts, die auch eine eingene ID "go" besitzt. Es ist also nicht das Objekt selbst.

Durch das Attribut ***xlink:href*** im **animateMotion**-Element wird das Objekt "boot" referenziert, und nicht (!) die Instanz "go". Dadurch wird zuerst die Animation berechnet und dann auf die Instanz angewandt, so dass das Objekt direkt am Animationspfad entlang verläuft.

Im folgenden und letzten Beispiel dieses Kapitels wird **animateMotion** nicht mit Hilfe eines Pfades definiert, sondern durch Attribute aus der Gruppe animValuesAttrs: ***from***, ***to*** oder ***by*** und ***values***.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="220"
 xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das animateMotion-Element</title>
<desc>Bewegung entlang eines Pfades 3</desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana,sans-serif; font-size:14px;
        font-weight:bold; fill:black;}
   polyline {fill:none; stroke:blue; stroke-width:2px;
           stroke-dasharray:2,2;}
   circle {fill:red;}
   polygon {fill:seagreen; stroke:black;}
  ]]>
  </style>
  <g id="boot">
   <polygon points="0 0,
                    -20 0,
                    -30 -15,
                    -2 -15,
                    -2 -35,
                    -2 -35,
                    2 -15,
                    35 -15,
                    20 0" />
   <circle id="startpunkt" cx="0" cy="0" r="2" />
   <text x="-21" y="-2.8"
```

```
      style="font-size:14px;
           text-rendering:optimizeSpeed;
           fill:white;">
      sailor
    </text>
  </g>
 </defs>
<!-- Hilfslinien zur Anzeige der Animationspfade -->
 <polyline points="-60 60, 540 60" />
 <polyline points="-60 120,
                140 120,
                240 180,
                340 120,
                540 120" />
 <polyline points="-60 180,
                140 180
                240 120
                340 180
                500 180" />

<!-- Animationen mit animValueAttrs: from,to und values -->
 <use id="b1" xlink:href="#boot" x="40" y="60" />
 <animateMotion xlink:href="#b1"
  begin="0" dur="6"
  rotate="auto"
  from="-100,0" to="500,0"
  repeatCount="5" />

 <use id="b2" xlink:href="#boot" x="40" y="120" />
 <animateMotion xlink:href="#b2"
  begin="0" dur="6"
  rotate="0"
  values="-100,0;100,0;200,60;300,0;500,0"
  repeatCount="5" />

 <use id="b3" xlink:href="#boot" x="40" y="180" />
 <animateMotion xlink:href="#b3"
  begin="0" dur="3"
  rotate="0"
  values="-100,0;100,0;200,-60;300,0;500,0"
  repeatCount="10" />
</svg>
```

In der ersten Animation des obigen Beispiels wird der Animationspfad durch die Attribute ***from*** und ***to*** definiert. Beide Attribute erhalten als Wert jeweils eine x,y-Koordinate, die relativ zum animierenden Objekt interpretiert wird. Diese Koordinaten bestimmen den Start- und den Endpunkt einer Linie, die als Animationspfad verwendet wird.

Die beiden folgenden Animationen verwenden das Attribut ***values*** zur Definition des Animationspfades (in Form eine Polylinie). Der Wert von ***values*** ist bei der Verwendung mit **animateMotion** eine durch Semikolon getrennte Liste mit x,y-Koordinaten, die den Verlauf des Aninmationspfads festlegen.

Beachten Sie: das **animateMotion**-Element referenziert in diesem Beispiel die mit **use** erzeugten Instanzen des Objekts "b1", "b2" und "b3" und nicht, wie im zweiten Beispiel dieses Kapitels, das Objekt "boot" selbst. In diesem Fall würden sich die Anweisungen der drei **animateMotion**-Elemente nämlich auf das Objekt beziehen und somit gleichzeitig auf alle drei (!) Instanzen des Objekts. .. und deshalb würden daraufhin nur die Anweisungen des zuletzt platzierten **animateMotion**-Elements auf alle (!) Instanzen des Objekts ausgeführt.

## 13.8 Animierte Farbveränderungen - animateColor

Um eine Farbänderung animiert darzustellen, stellt SVG das Element **animateColor** zur Verfügung. In diesem Element können Sie alle Attribute aus den Attributgruppen für Animationen verwenden.

Übrigens: Eine animierte Farbänderung ist mit dem **animate**-Element ebenfalls möglich.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="100" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das animateColor-Element</title>
<desc>Animierte Farbänderungen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:14px;
           font-weight:bold; fill:black;}
    ]]>
    </style>
  </defs>
<!-- die Ampel -->
  <rect id="kiste" x="20" y="20" width="60" height="160" fill="#333333" />
  <circle id="k1" cx="50" cy="50" r="20"
    style="fill:red; stroke:#eeeeee;" />
  <circle id="k2" cx="50" cy="100" r="20"
    style="fill:#333333; stroke:#eeeeee;" />
  <circle id="k3" cx="50" cy="150" r="20"
    style="fill:#333333; stroke:#eeeeee;" />

<!-- die Animation: Gesamtspieldauer 12 Sekunden -->
<!-- 2 mal mit animateColor -->
  <animateColor xlink:href="#k1"
    attributeName="fill"
    begin="2" dur="2"
    from="red" to="#333333"
    fill="freeze" />
  <animateColor xlink:href="#k2"
    attributeName="fill"
    begin="3" dur="3"
    values="#333333;yellow;#333333"
    fill="freeze" />
<!-- 1 mal mit animate -->
  <animate xlink:href="#k3"
    attributeName="fill"
    begin="5" dur="2"
    from="#333333" to="limegreen"
    fill="freeze" />
<!-- 2 mal set: zurück zum Anfangszustand -->
  <set xlink:href="#k1"
    attributeName="fill"
    begin="12" dur="indefinite"
    to="red" />
  <set xlink:href="#k3"
    attributeName="fill"
    begin="12" dur="indefinite"
    to="#333333" />
</svg>
```

## 13.9 Animierte Transformationen - animateTransform

Auch für animierte Transformationen gibt es das spezielle Element **animateTransform**. Zusätzlich zu allen Attributen aus den Attributgruppen für Animationen muß diesem Element noch das Attribute ***type*** zugeordnet werden.

Mit dem Attribut ***type*** geben Sie die Art der Transformation an: *translate*, *scale*, *rotate*, *skewX* und *skewY* sind aktzeptierte Werte für dieses Attribut. Der Wert *matrix* ist für dieses Attribut **nicht** zulässig.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="200"
 xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das animateTransform-Element</title>
<desc>Animierte Transformationen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:44px;
            font-weight:bold; fill:blue; stroke:black;}
      rect {fill:yellow; stroke:blue; stroke-width:2px; opacity:0;}
      circle {fill:#eeeeee; stroke:none;}
    ]]>
    </style>
  </defs>
<!-- Hintergrund circle und animiertes Hintergrund rect -->
  <circle id="bg" cx="250" cy="100" r="100" />
  <rect x="10" y="57" width="480" height="80">
    <animate attributeName="opacity"
      begin="9" dur="3"
      from="0" to=".5"
      fill="freeze" />
  </rect>

<!-- 4 Animimierte Transformationen hintereinander
     auf ein Text-Element angewandt -->
  <text x="20" y="113">animateTransform
    <animateTransform
      attributeName="transform" type="scale"
      begin="0" dur="3"
      from="0" to="1" />
    <animateTransform
      attributeName="transform" type="rotate"
      begin="3" dur="2"
      from="0,250,120" to="-360,250,120" />
    <animateTransform
      attributeName="transform" type="translate"
      begin="5" dur="2"
      from="0" to="500" />
    <animateTransform
      attributeName="transform" type="translate"
      begin="7" dur="2"
      from="-420" to="0" />
  </text>
</svg>
```

Im vorangegangenen Beispiel wurden die Animationen hintereinander abgespielt. Sie können animierte Transformationen allerdings auch gleichzeitig abspielen lassen, wenn Sie das Attribut ***additive*** mit dem Wert *sum* für jedes einzelne **animateTransform**-Element verwenden. Dank für diesen Tipp an Christian.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="200"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>das animateTransform-Element</title>
<desc>Animierte Transformationen</desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:44px;
            font-weight:bold; fill:blue; stroke:black;}
      rect {fill:yellow; stroke:blue; stroke-width:2px; opacity:0;}
      circle {fill:#eeeeee; stroke:none;}
    ]]>
    </style>
  </defs>
<!-- Hintergrund circle und animiertes Hintergrund rect -->
  <circle id="bg" cx="250" cy="100" r="100" />
  <rect x="10" y="57" width="480" height="80">
    <animate attributeName="opacity"
      begin="1" dur="4"
      from="0" to=".5"
      fill="freeze" />
  </rect>

<!-- 3 Animimierte Transformationen gleichzeitig
     auf ein Text-Element angewandt -->
  <text x="125" y="113">gleichzeitig
    <animateTransform
      attributeName="transform" type="scale"
      begin="0" dur="5"
      from="0" to="1"
      additive="sum" />
    <animateTransform
      attributeName="transform" type="rotate"
      begin="0" dur="5"
      from="0,250,120" to="-360,250,120"
      additive="sum" />
    <animateTransform
      attributeName="transform" type="translate"
      begin="0" dur="5"
      from="100,100" to="0,0"
      additive="sum" />
  </text>
</svg>
```

## 14 Filtereffekte

In SVG Dokumenten müssen Sie auf Filtereffekte nicht verzichten. Mit Hilfe von insgesamt 20 verschiedenen Elementen (sogenannten primitiven Filtern) können Sie Filtereffekte wie Unschärfe, Farbveränderungen oder Beleuchtung auf SVG Elemente und Objekte anwenden.

Filter werden immer in einem, durch das **filter**-Element festgelegten Container definiert, der wiederum immer im **defs**-Container eines SVG Dokuments platziert werden muß. Dieser **filter**-Container ist zur Aufnahme der primitiven Filter bestimmt, welche die entsprechenden Effekte erzeugen.

Damit ein so festgeleger Filter auf ein Objekt angewendet werden kann, wird dem Objekt das Attribut oder die Eigenschaft ***filter*** zugewiesen. Zulässiger Wert für dieses Attribut ist eine Referenz auf den Filter in Form einer gültigen URL.

Die verschiedenen primitiven Filter können kombiniert werden. Einige dieser Filter sind ohne die vorherige Verwendung anderer Filter sogar uneffektiv, d.h. mache Filter sollten kombiniert werden.

Im folgenden Beispiel wird im **defs**-Bereich der Grafik ein **filter**-Container mit der ID *f1* definiert. Dieser Container enthält lediglich einen primitiven Filter **feGaussianBlur** (Gausscher Weichzeichner). Der Filter wird dann innerhalb der Grafik von einen **g**-Element (gruppiertes Objekt aus Rechteck und Text) durch das Attribut ***filter*** referenziert und daraufhin auf dieses Objekt angewandt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="260px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Anwendung eines Filters</title>
  <desc>Das filter-Element und das filter-Attribut</desc>
  <defs>
    <filter id="f1">
      <feGaussianBlur in="SourceGraphic" stdDeviation="3" />
    </filter>
  </defs>

  <g filter="url(#f1)">
    <rect x="40" y="40" width="320" height="180" rx="15"
      style="fill:#33cc33; stroke:black; stroke-width:3px;" />
    <text x="42" y="140"
      style="font-family:verdana; font-size:37px; font-weight:bold;">
      feGaussianBlur
    </text>
  </g>
</svg>
```

## 14.1 Attribute für das filter-Element

Sie können den rechteckigen Wirkungsbereich eines Filters begrenzen, d.h. Sie können den Filtereffekt nicht nur auf das gesamte Element oder Objekt, sondern auch auf rechteckige Teilbereiche des Elements oder des Objekts anwenden. Durch die Attribute ***x*** und ***y*** legen Sie die Koordinaten der linken, oberen Ecke dieses Bereichs fest. Durch die Attribute ***width*** und ***height*** bestimmen Sie die Ausmaße des rechteckigen Bereichs.

Ob Sie relative oder absolute Maßangaben als Werte für diese Attribute verwenden müssen, regelt das Attribut ***filterUnits***. Bei Gebrauch des voreingestellten Wertes *boundingBox* müssen für ***x***, ***y***, ***width*** und ***height*** relative Werte angegeben werden. Dabei können diese Werte kleiner als 0 bzw. 0% oder größer als 1 bzw. 100% gewählt werden. Da der Wirkungsbereich macher Filter größer ist als das referenzierende Objekt, sind folgende Werte voreingestellt: ***x***: -10%, ***y***: -10%, ***width***: 120%, ***height***: 120%.

Der zweite mögliche Wert für ***filterUnits*** ist *userSpaceOnUse*. Wenn Sie diesen Wert verwenden erwarten die Attribute ***x***, ***y***, ***width*** und ***height*** absolute Angaben.

Auch für jedes primitive Filterelement kann durch die Attribute ***x***, ***y***, ***width*** und ***height*** eine Größenangabe definiert werden.

Um auch für diese Attribute den Zugriff auf das Koordinatensystem des referenzierenden Elements oder Objekts festzulegen, können Sie das Attribut ***primitiveUnits*** im **filter**-Element verwenden. Auch dieses Attribut kann die Werte *boundingBox* und *userSpaceOnUse* annehmen. Beachten Sie: Voreingestellter Wert für ***primitiveUnits*** ist *userSpaceOnUse* (im Gegensatz (!) zu ***filterUnits***).

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="330px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Filter</title>
  <desc>
    Attribute des filter-Elements
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:48px;
            font-weight:bold; }
    ]]>
    </style>
<!-- Filterregion so groß wie Textelement -->
    <filter id="f1"
      x="0" y="0" width="1" height="1">
      <feGaussianBlur in="SourceGraphic" stdDeviation="3" />
    </filter>

<!-- Filterregion größer als Textelement -->
    <filter id="f2"
      x="-20%" y="-30%" width="150%" height="150%">
      <feGaussianBlur in="SourceGraphic" stdDeviation="3" />
    </filter>

<!-- Filterregion in der Breite beschränkt -->
    <filter id="f3"
      x="53%" y="-20%" width="100%" height="150%">
      <feGaussianBlur in="SourceGraphic" stdDeviation="3" />
    </filter>

<!-- Filterregion in der Breite beschränkt -->
    <filter id="f4" filterUnits="userSpaceOnUse"
      x="20" y="180" width="100" height="70">
```

```
    <feGaussianBlur in="SourceGraphic" stdDeviation="3" />
   </filter>

<!-- Filterregion größer als Textelement -->
<!-- Region des Weichzeichners in der Breite beschränkt -->
   <filter id="f5" x="-20%" y="-30%" width="150%" height="150%">
    <feGaussianBlur x="-20" y="-20" width="200" height="350"
     in="SourceGraphic" stdDeviation="3" result="blur_out" />
    <feOffset in="blur_out" dx="3" dy="-3" result="offset_out" />
    <feMerge>
     <feMergeNode in="offset_out" />
     <feMergeNode in="SourceGraphic" />
    </feMerge>
   </filter>
  </defs>

  <text filter="url(#f1)" x="35" y="60">
   Filterregion
  </text>
  <text filter="url(#f2)" x="35" y="120">
   Filterregion
  </text>
  <text filter="url(#f3)" x="35" y="180">
   Filterregion
  </text>
  <text filter="url(#f4)" x="35" y="240">
   FilterRegion
  </text>
  <text filter="url(#f5)" x="35" y="300">
   Filterregion
  </text>
</svg>
```

Im obigen Beispiel sind 5 verschiedene Filter definiert, die jeweils auf ein Textelement angewendet werden. Für alle Filter wurde **feGaussianBlur** (Weichzeichner) als primitiver Filter verwendet.

Im letzten Filter "f5" wurde zusätzlich noch die Filtertypen **feOffset** (Bildverschiebung) und **feMerge** (mehrere Bilder kombinieren) mit Kind-Elementen **feMergeNode** (Bestimmung der Einzelbilder) benutzt. Nähere Informationen zu den einzelnen Filtertypen finden Sie in den folgenden Kapiteln.

Verwendung der Attribute für das **filter**-Element im Filter "f1": Das Attribut ***filterUnits*** wurde nicht verwendet, daher gilt die Voreinstellung *boundingBox* für diesen Filter. Die zu verwendenden Werte für die Attribute ***x***, ***y***, ***width*** und ***height*** sind deshalb relativer Natur. Durch Verwendung dieser Attribute wurde die Größe der Filterregion so bestimmt, dass sie der Größe des referenzierenden Textelements entspricht. Da der Filtertyp ***feGaussianBlur*** mehr Raum benötigt, wird der Filtereffekt leicht beschnitten dargestellt.

Verwendung der Attribute für das **filter**-Element im Filter "f2": Die zu verwendenden Werte für die Attribute ***x***, ***y***, ***width*** und ***height*** sind auch in diesem Filter relativer Natur. Die Filterregion ragt an allen Seiten über das referenzierenden Textelement hinaus. So wird der Filtereffekt korrekt dargestellt.

Verwendung der Attribute für das **filter**-Element im Filter "f3": Die zu verwendenden Werte für die Attribute ***x***, ***y***, ***width*** und ***height*** sind auch in diesem Filter relativer Natur. Die Filterregion überlagert das referenzierenden Textelement nur im hinteren Bereich. Die Ausgabe (der Output) des Filters ist deshalb lediglich dieser Teil des Textelements. Der Rest des Elements wird nicht am Bildschirm dargestellt.

Verwendung der Attribute für das **filter**-Element im Filter "f4": Das Attribut ***filterUnits*** wurde mit der Einstellung *userSpaceOnUse* verwendet. Die zu verwendenden Werte für die Attribute ***x***, ***y***, ***width*** und ***height*** sind deshalb absolut (!). Die Filterregion überlagert das referenzierenden Textelement nur im vorderen Bereich. Die Ausgabe (der Output) des Filters ist deshalb auch hier lediglich dieser Teil des Textelements.

Verwendung der Attribute für das **filter**-Element im Filter "f5": Das Attribut ***filterUnits*** wurde in diesem Filter wiederum nicht verwendet, daher gilt die Voreinstellung *boundingBox* für diesen Filter. Die zu verwendenden Werte für die Attribute ***x***, ***y***, ***width*** und ***height*** im filter-Element sind also relativer Natur. Sie

werden so gewählt, dass die Filterregion größer ist, als das Textelement. Auch das Attribut ***primitiveUnits*** wurde in diesem Filter nicht verwendet, daher gilt die Voreinstellung *userSpaceOnUse* für die primitiven Filter, die dieser **filter**-Container beinhaltet. Die zu verwendenden Werte für die Attribute ***x***, ***y***, ***width*** und ***height*** innerhalb des verwendeten primitiven Filters **feGaussianBlur** sind deshalb absolut (zur Region des Eltern-Elements) angegeben. Dieser Filter wird in der Größe auf den vorderen Berich des Textelements beschränkt. Die derart beschränkte Ausgabe (Output des Weichzeichners), wird vom zweiten primitiven Filter **feOffset** als Input-Bild verwendet und erzeugt einen weiteren Output. Durch die Verwendung von **feMerge** wird dieser letzte Output mit dem originalen Bild des Textelements (*SourceGraphic*) kombiniert.

## 14.2 Alle primitiven Filter - Übersicht

Jeder primitive Filter verwendet entweder 0, 1 oder mehrere Input-Bilder und erzeugt genau ein Output-Bild.

Als Input-Bilder können SVG-Elemente oder Objekte, deren Alphakanal, deren Hintergrund-Bild oder Output-Bilder vorangegangener Filteroperationen verwendet werden.

**feConvolveMatrix** Bewirkt eine Veränderung der Bildpunkt-Matrix. Dabei werden Bildpunkte des Input-Bildes mit benachbarten Bildpunkten vermischt. Dadurch eignet sich dieser Filter für eine große Anzahl von Effekten, wie: verwischen, Kanten scharfzeichen, scharfzeichnen oder Prägeeffekte.

**feDisplacementMap** Verwendet Bildpunkte eines zweiten Input-Bildes um ein erstes Input-Bild räumlich zu verschieben.

**feGaussianBlur** Der sogenannte "Gaussche Weichzeichner" verwischt ein Input-Bild.

**feOffset** Verschiebt ein Input-Bild durch einen festzulegenden Vektor.

**feMorphology** Verdickt oder verdünnt ein Input-Bild.

**feBlend** Setzt ein erstes und ein zweites Input-Bild durch pixelweise Vermischung (Alpha-Blending) zu einem Bild zusammen. Dabei sind mehrere Effekte wie unter anderm multiplizieren, verdunkeln oder aufhellen möglich.

**feComposite** Kombiniert ein erstes und ein zweites Input-Bild zu einem einzigen Bild. Dabei werden die sogenannten Porter-Duff-Operationen verwendet.

**feMerge** Kombiniert mehrere Input-Bilder zu einem einzigen Bild. Dabei wird die Porter-Duff-Operation *over* verwendet.

**feMergeNode** Kind-Element von **feMerge** zur Definition der einzelenen Input-Bilder.

**feColorMatrix** Wendet eine Matrix-Transformation auf die RGBA Farb- und Alphawerte jedes Pixels einer Input-Grafik an und erzeugt auf diese Weise ein Bild mit veränderten RGBA Farb- und Alphawerten. Es wird also eine Farbveränderung erzeugt.

**feComponentTransfer** Legt für jedes Pixel einer einzelnen Farbe oder des Alphakanals einen neuen Wert fest. Dieser neue Wert wird durch eine festgelegte Funktion berechnet. Auf diese Weise sind Veränderungen der Helligkeit, des Kontrast oder der Farbbalance eines Input-Bildes möglich.

**feDiffuseLighting** Beleuchtet ein Input-Bild mit diffusem Licht. Verwendet den Alpha-Kanal des Input-Bildes als Relief.

**feSpecularLighting** Beleutet ein Input-Bild mit reflektierendem Licht. Verwendet den Alpha-Kanal des Input-Bildes als Relief.

**feDistanceLight** Kind-Element von **feSpecularLighting** und **feSpecularLighting** zur Plazierung der Lichtquelle.

**fePointLight** Kind-Element von **feSpecularLighting** und **feSpecularLighting** zur Plazierung der Lichtquelle.

**feSpotLight** Kind-Element von **feSpecularLighting** und **feSpecularLighting** zur Plazierung der Lichtquelle. Außerdem kann das Ziel des Lichts definiert werden.

**feFlood** Füllt das Zielrechteck mit Farbe. Benötigt dabei kein Input-Bild.

**feImage** Bietet die Möglichkeit eine externe Bitmap-Grafik oder ein svg-Fragment als Input-Bild für nachfolgende primitive Filter einzubinden.

**feTile** Füllt das Zielrechteck mit einem Muster. Als Muster wird das Input-Bild verwendet.

**feTurbulence** Erzeugt eine Turbulenz oder ein Fraktal im Zielrechteck. Benötigt dabei kein Input-Bild.

## 14.3 Primitive Filter - filter_primitive_attributes

***x*** Voreinstellung: siehe unten. Definiert die x-Koordinate der linken, oberen Ecke des rechteckigen Wirkungsbereichs eines einzelnen, primitiven Filters.

***y*** Voreinstellung: siehe unten. Definiert die y-Koordinate der linken, oberen Ecke des rechteckigen Wirkungsbereichs eines einzelnen, primitiven Filters.

***width*** Voreinstellung: siehe unten. Legt die Breite des rechteckigen Wirkungsbereichs eines einzelnen, primitiven Filters fest.

***height*** Voreinstellung: siehe unten. Legt die Höhe des rechteckigen Wirkungsbereichs eines einzelnen, primitiven Filters fest.

Die Voreinstellung der Attribute ***x***, ***y***, ***width*** und ***height***, die die Ausmaße des Unterbereichs für einzelene primitive Filter festlegen, variiert bei jeder Anwendung. Die Werte werden immer entsprechend den Ausmaßen des Zielelements festgelegt.

***result*** Voreinstellung: keine. Mit Hilfe dieses Attributs können Sie dem Output-Bild des primitiven Filters einen Namen geben. Über diesen Namen kann daraufhin das erzeugte Output-Bild auch von einem nicht direkt nachfolgenden primitiven Filter im gleichen **filter**-Container als Input-Bild verwendet werden. Wenn Sie dieses Attribut nicht verwenden, ist der Output nur für das direkt folgende primitive Filter-Element im gleichen **filter**-Element zugänglich.

## 14.4 Primitive Filter - filter_primitive_attributes_with_in

Die Attributgruppe filter_primitive_attributes_with_in enthält alle Attribute der Attributgruppe filter_primitive_attributes und das Attribut ***in*** mit dessen Hilfe Sie das Input-Bild für den primitiven Filter bestimmen können.

***in*** Voreinstellung: *SourceGraphic*. Legt das Input-Bild für den primitiven Filter fest. Möglicher Wert für dieses Attribut ist entweder eines von 6 festgelegten Schlüsselworten:*SourceGraphic*, *SourceAlpha*, *BackgroundImage*, *BackgroundAlpha*, *FillPaint* oder *StrokePaint*, oder der Wert eines vorangegangenen ***result***-Attributs innerhalb desselben **filter**-Elements. Wenn Sie das Attribut ***in*** nicht verwenden, wird der Output des direkt vorangegangenen primitiven Filter-Elements als Input verwendet. Wenn ein solches primitives Filter-Element nicht existiert, d.h. wenn der primitive Filter der erstplatzierte innerhalb eines **filter**-Elements ist, und nur dann, wird die Voreinstellung *SourceGraphic* verwendet.

*SourceGraphic*: Der voreingestellte Wert des Attributs ***in***. Bezeichnet das Bild des Elements oder Objekts, das den Filter referenziert. Also das Originalbild bzw. das "QuellBild".

*SourceAlpha*: Bezeichnet den Alphakanal des Originalbildes. Der Alphakanal definiert alle transparenten bzw. semitransparenten Pixel in einer Grafik. Das Bild des Alphakanal beschreibt also die Transparenz der Grafik.

*BackgroundImage*: Bezeichnet das Bild, das den Hintergrund des referenzierenden Elements bzw. Objekts darstellt. Um ein Hintergrundbild als Input verwenden zu können, muß innerhalb eines entsprechenden Container-Elements (g oder svg) mit Hilfe des Attributs ***enable-background*** und dem Wert *new* ausdrücklich die Erzeugung von Hintergrundbildern aktiviert werden. Nur dann gehören alle Kind-Elemente des Containers zum möglichen Hintergrundbild. Das eigentliche Hintergrundbild ist der Ausschnitt des möglchen Hintergrundbildes, der vom referenzierenden Element überdeckt wird. Beachten Sie: Das Container-Element muß ein Vorfahr des referenzierenden Elements sein. Beachten Sie: Das referenzierende Element gehört nicht zum Hintergrundbild.

*BackgroundAlpha*: Bezeichnet den Alphakanal des Bildes, das den Hintergrund des referenzierenden Elements bzw. Objekts darstellt. Auch hier muß die Erzeugung von Hintergrundbildern durch Verwendung von ***enable-background*** und dem Wert *new* ausdrücklich definiert werden.

*FillPaint*: Bezeichnet den Inhalt der Eigenschaft ***fill*** im referenzierden Element. Normalerweise ist dieser Input (dieses Bild) lichtundurchlässig. Wenn der Inhalt von ***fill*** allerdings ein Verlauf oder Muster ist, dann besitzt das Bild einen eigenen Alphakanal. Es ist an einigen Stellen lichtdurchlässig (transparent oder semi-transparent).

*StrokePaint*: Bezeichnet den Inhalt der Eigenschaft ***stroke*** im referenzierden Element. Normalerweise beinhaltet auch dieses Bild keine Transparenz, also keinen Alphakanal. Wenn der Inhalt von ***fill*** allerdings ein Verlauf oder Muster ist, dann ist es an einigen Stellen transparent oder semi-transparent, hat also einen Alphakanal.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="600px" height="460px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>filter_primitive_attributes_with_in</title>
  <desc>
    Attribute für primitive Filter.
    Das Attribut in und seine Schlüsselworte.
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:20px;
            font-weight:bold;}
      ellipse {fill:url(#v1); stroke:red; stroke-width:2px;}
      line {fill:none; stroke:black;}
    ]]>
    </style>
    <linearGradient id="v1"
      x1="0" y1="0" x2="1" y2="0">
      <stop offset=".2" stop-color="limegreen" />
```

```
      <stop offset="1" stop-color="black" />
    </linearGradient>
<!-- Vier Weichzeichner mit verschiedenen inputs -->
    <filter id="f1">
      <feGaussianBlur in="SourceGraphic" stdDeviation="3" result="out1" />
    </filter>
    <filter id="f2">
      <feGaussianBlur in="SourceAlpha" stdDeviation="3" result="out1" />
    </filter>
    <filter id="f3">
      <feGaussianBlur in="FillPaint" stdDeviation="3" result="out1" />
    </filter>
    <filter id="f4">
      <feGaussianBlur in="StrokePaint" stdDeviation="3" result="out1" />
    </filter>
  </defs>

<!-- die referenzierenden Elemente -->
  <ellipse cx="120" cy="90" rx="80" ry="30" />
  <ellipse cx="120" cy="170" rx="80" ry="30" filter="url(#f1)" />
  <ellipse cx="120" cy="250" rx="80" ry="30" filter="url(#f2)" />
  <ellipse cx="120" cy="330" rx="80" ry="30" filter="url(#f3)" />
  <ellipse cx="120" cy="410" rx="80" ry="30" filter="url(#f4)" />

<!-- Text und Hilfslinien -->
  <line x1="230" y1="90" x2="380" y2="90" />
  <text x="400" y="97">Ohne Filter</text>
  <line x1="230" y1="170" x2="380" y2="170" />
  <text x="400" y="177">SourceGraphic</text>
  <line x1="230" y1="250" x2="380" y2="250" />
  <text x="400" y="257">SourceAlpha</text>
  <line x1="230" y1="330" x2="380" y2="330" />
  <text x="400" y="337">FillPaint</text>
  <line x1="230" y1="410" x2="380" y2="410" />
  <text x="400" y="417">FillStroke</text>
  <text x="15" y="40"
    style="font-size:24px;">
    feGaussianBlur mit verschiedenen Input&amp;apos;s
  </text>
</svg>
```

Im obigen Beispiel wird der Filter **feGaussianBlur** auf 4 Ellipsen angewendet. Dabei wird für jede Ellipse ein anderer Filter verwendet, denn in jedem der 4 Filter sorgt ein anderes Schlüsselwort für einen unterschiedlichen Input. Nacheinander werden die Schlüsselworte *SourceGraphic*, *SourceAlpha*, *FillPaint* und *StrokePaint* als Wert für das Attribut ***in*** verwendet. Da der Output von **feGaussianBlur** das Bild lediglich weichzeichnet, ist gut zu erkennen, welche Bildpunkte durch die verschiedenen Schlüsselworte des Attributs ***in*** verwendet bzw. gefiltert werden.
Das folgende Beispiel zeigt die Verwendung der Schlüsselworte *BackgroundImage* und *BackgroundAlpha*.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="340px" height="500px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>filter_primitive_attributes_with_in</title>
  <desc>
    Attribute für primitive Filter. Das Attribut in und
    die Schlüsselworte BackgroundImage und BackgroundAlpha.
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
```

```
      text {font-family:Verdana,sans-serif; font-size:20px;
           font-weight:bold; fill:white;}
      rect {fill:blue; stroke:black;}
    ]]>
    </style>
    <filter id="f1">
      <feColorMatrix in="BackgroundImage"
        type="hueRotate" values="-50" result="out1" />
      <feGaussianBlur in="out1" stdDeviation="10" result="out2" />
    </filter>
    <filter id="f2">
      <feColorMatrix in="BackgroundImage"
        type="hueRotate" values="-50" result="out1" />
      <feGaussianBlur in="out1" stdDeviation="10" result="out2" />
      <feMerge>
        <feMergeNode in="out2" />
        <feMergeNode in="SourceGraphic" />
      </feMerge>
    </filter>
    <filter id="f3">
      <feColorMatrix in="BackgroundAlpha"
        type="hueRotate" values="-50" result="out1" />
      <feGaussianBlur in="out1" stdDeviation="10" result="out2" />
      <feMerge>
        <feMergeNode in="out2" />
        <feMergeNode in="SourceGraphic" />
      </feMerge>
    </filter>
  </defs>

<!-- Grafik ohne Filter -->
  <g enable-background="new" transform="translate(10,10)" >
    <rect x="10" y="10" width="300" height="100" />
    <g>
      <rect x="50" y="30" width="220" height="60"
        style="fill:none; stroke:black; stroke-width:2px;" />
      <text x="57" y="68">BackgroundImage</text>
    </g>
  </g>

<!-- BackgroundImage-Filter ohne SourceGraphic -->
  <g enable-background="new" transform="translate(10,130)">
    <rect x="10" y="10" width="300" height="100" />
    <g filter="url(#f1)">
      <rect x="50" y="30" width="220" height="60"
        style="fill:none;" />
      <text x="57" y="68">BackgroundImage</text>
    </g>
  </g>

<!-- BackgroundImage-Filter mit SourceGraphic -->
  <g enable-background="new" transform="translate(10,250)">
    <rect x="10" y="10" width="300" height="100" />
    <g filter="url(#f2)">
      <rect x="50" y="30" width="220" height="60"
        style="fill:none;" />
      <text x="57" y="68">BackgroundImage</text>
    </g>
  </g>

<!-- BackgroundAlpha-Filter mit SourceGraphic -->
  <g enable-background="new" transform="translate(10,370)">
    <rect x="10" y="10" width="300" height="100" />
    <g filter="url(#f3)">
```

```
    <rect x="50" y="30" width="220" height="60"
      style="fill:none;" />
    <text x="57" y="68">BackgroundAlpha</text>
   </g>
  </g>
</svg>
```

Im obigen Beispiel sind drei Filter *f1*, *f2* und *f3* definiert, die mit Hilfe des primitiven Filters **feColorMatrix** eine Farbveränderung des Inputs erzeugen und das Ergebnis dieses Filters dann durch **feGaussianBlur** weichzeichnen. Die Filter *f2* und *f3* beinhalten außerdem noch den primitiven Filter **feMerge** mit seinen Kindelementen **feMergeNode**, welcher den Output der ersten beiden primitiven Filter mit dem Quellbild zusammenfügt.

Weiterhin sind, durch Verwendung des **g**-Elements drei Gruppen, die jeweils identisch aufgebaut sind, im Beispiel festgelegt. Jede dieser Gruppen besteht aus einem blau gefüllten Rechteck und einer Unter-Gruppe, die ein transparentes Rechteck und einen Text enthält. Die Filter werden jeweils von dem **g**-Element referenziert, welches die Untergruppen bildet.

Die erste Gruppe wird ohne Filter dargestellt.

In der zweiten Gruppe wird der Filter *f1* eingebunden, der als Input *BackgroundImage* verwendet. Der Bereich, den das referenzierende Objekt auf dem Hintergrundbild einnimmt wird dadurch verändert. Das referenzierende Objekt selbst, die Untergruppe, wird nicht am Bildschirm dargestellt. Beachten Sie: damit dieser Filter auf das Objekt angewendet werden kann, wird im äußeren **g**-Element das Attribut ***enable-background*** mit dem Wert *new* verwendet. Dieses bewirkt, dass vom user agent überhaupt ein Hintergrundbild erzeugt wird. Da die Erzeugung von Hintergrundbildern eine äußerst ressourcen-intensive Angelegenheit darstellt, ist dies nicht das Standardverhalten des user-agents.

In der dritten Gruppe wird der Filter *f2* referenziert, der auch das referenzierende Objekt, durch Verwendung von **feMerge**, darstellt.

In der letzten Gruppe wird der Filter *f3* verwendet. Dieser Filter benutzt das Alphabild des Hintergrundbildes als Input, das in diesem Fall nur aus einem schwarzen Rechteck besteht. Außerdem wird auch hier das referenzierende Objekt angezeigt.

Anmerkung des Autors: Der Adobe SVG Viewer 3.0 hat noch Probleme bei der korrekten Interpretation von Hintergrundbildern. Die Beispielgrafik der W3C SVG Recommendation zu diesem Thema wurde nicht gemäß der Empfehlung dargestellt.

## 14.5 Primitive Filter - PresentationAttributs-FilterPrimitives

***color-interpolation-filters*** Voreinstellung: *linearRGB*. Alle primitiven Filter verwenden als Input-Bild ein RGBA Bild. Dieser Farbraum wird durch das Attribut ***color-interpolation-filters*** bzw. dessen voreingestelltem Wert *linearRGB* festgelegt. Für andere Farboperationen, wie z.B. Farbverläufe, gilt der Farbraum der innerhalb der SVG Grafik durch das Attribut ***color-interpolation*** definiert ist: das ist in der Voreinstellung *sRGB*. Wenn Sie also auf ein Element, dessen Füllung durch einen anderen Farbraum definiert ist (das z.B. mit einem Farbverlauf gefüllt ist), einen der farbverändernden Filter **feColorMatrix** oder **feComponentTransfer** anwenden wollen, sollten Sie die Werte dieser beiden Attribute anpassen. Nur so ist gewährleistet, dass der gewünschen Effekt korrekt dargestellt wird.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="760px" height="200px"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>PresentationAttributs-FilterPrimitives</title>
 <desc>
  Attribute für primitive Filter 1
 </desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana,sans-serif; font-size:48px;
        font-weight:bold; }
  ]]>
  </style>
<!-- der Verlauf -->
  <linearGradient id="v1"
   x1="0" y1="0" x2="1" y2="0">
   <stop offset=".2" stop-color="limegreen" />
   <stop offset="1" stop-color="black" />
  </linearGradient>

<!-- Filter: linearRGB -->
  <filter id="f1">
   <feColorMatrix in="SourceGraphic"
    type="hueRotate" values="90" />
  </filter>

<!-- Filter: sRGB -->
  <filter id="f2" color-interpolation-filters="sRGB">
   <feColorMatrix in="SourceGraphic"
    type="hueRotate" values="90"
    color-interpolation-filters="sRGB" />
  </filter>
 </defs>

 <text x="30" y="70" fill="url(#v1)"
  filter="url(#f1)">
  color-interpolation-filters
  <tspan x="330" dy="20"
   style="font-size:16px; font-stretch:ultra-expanded;">
   linearRGB
  </tspan>
 </text>
 <text x="30" y="150" fill="url(#v1)"
  filter="url(#f2)">
  color-interpolation-filters
  <tspan x="330" dy="20"
   style="font-size:16px; font-stretch:ultra-expanded;">
```

```
    sRGB
   </tspan>
  </text>
</svg>
```

Die obige Beispielgrafik enthält 2 Textelemente, die mit einem Verlauf gefüllt sind. Auf diese beiden **text**-Elemente wird jeweils eine Farbveränderung durch die Filter *f1* und *f2* angewandt. Beide Filter enthalten den primitiven Filter **feColorMatrix** der eine Farbveränderung bewirkt.

Im Filter *f1* werden die unterschiedlichen Farbräume von Verlauf und Filteroperation nicht angepasst. Im Filter *f2* wird durch Verwendung des Attributs ***color-interpolation-filters*** der Farbraum *sRGB* als gültiger Farbraum definiert.

Das Attribut ***color-interpolation-filters*** kann übigens auch im **filter**-Element verwendet werden.

## 14.6 feConvolveMatrix

Der primitive Filter **feConvolveMatrix** bewirkt eine Veränderung der Bildpunkt-Matrix. Dabei werden Bildpunkte des Input-Bildes mit benachbarten Bildpunkten vermischt. Dadurch eignet sich dieser Filter für eine große Anzahl von Effekten, wie: verwischen, Kanten scharfzeichen, scharfzeichnen oder Prägeeffekte.

Mit Hilfe des Attributs ***order*** müssen Sie festlegen, welche und wieviele benachbarte Bildpunkte in die Berechnung einbezogen werden. Zu diesem Zweck definieren Sie eine sogenannte Kernel-Matrix, die aus einer festgelegten Anzahl von Punkten besteht, welche in Spalten und Zeilen geordnet sind. Das Attribut ***order*** aktzeptiert 2 durch Komma getrennte Ganzzahlen als Wert. Dabei bestimmt die erste Ganzahl die Anzahl der Spalten der Kernel-Matrix und die zweite Ganzahl die Anzahl der Zeilen der Kernel-Matrix. Wenn Sie nur eine Ganzahl als Wert für ***order*** verwenden, gibt diese sowohl die Anzahl der Spalten als auch die Anzahl der Zeilen der Kernel-Matrix an. Der voreingestellte Wert für ***order*** ist *3*, d.h. eine Kernel-Matrix, deren insgesamt 9 Bildpunkte in 3 Zeilen a 3 Reihen angeordnet sind. Die so festgelegte Kernel-Matrix wird mittig auf jeden Bildpunkt der Bild-Matrix gelegt. Auf diese Art sind alle benachbarten Bildpunkte, für jeden neu zu berechnenden Bildpunkt eindeutig bestimmt.

Mit dem Attribut ***kernelMatrix*** müssen Sie für jeden einzelnen Bildpunkt der Kernel-Matrix einen Zahlwert bestimmen, der bei der Neuberechnung als Faktor verwendet wird. Dem Attribut ***kernelMatrix*** muß als Wert also eine, durch Leerzeichen und/oder Komma getrennte Liste aus Zahlen zugeordnet werden. Dabei muß die Anzahl der Zahlen mit der Anzahl der Bildpunkte der Kernel-Matrix übereinstimmen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feConvolveMatrix</title>
  <desc>
    Beispiele für die Verwendung von feConvolveMatrix
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feConvolveMatrix -->
    <filter id="f1">
      <feConvolveMatrix order="3"
        kernelMatrix="1 1 1 1 1 1 1 1 1" />
    </filter>
    <filter id="f2">
      <feConvolveMatrix order="3"
        kernelMatrix="1 0 0 0 0 0 0 0 1" />
```

```
    </filter>
    <filter id="f3">
      <feConvolveMatrix order="10,1"
        kernelMatrix="1 1 1 1 1 1 1 1 1 1" />
    </filter>
    <filter id="f4">
      <feConvolveMatrix order="1,10"
        kernelMatrix="1 1 1 1 1 1 1 1 1 1" />
    </filter>
<!-- zusätzlich mit feBlend -->
    <filter id="f5">
      <feConvolveMatrix order="10,1"
        kernelMatrix="1 1 1 1 1 1 1 1 1 1" result="out1" />
      <feBlend in="SourceGraphic" in2="out1" mode="multiply" />
    </filter>
    <filter id="f6">
      <feConvolveMatrix order="10,1"
        kernelMatrix="0 0 0 0 0 1 1 1 1 1" result="out1" />
      <feBlend in="SourceGraphic" in2="out1" mode="multiply" />
    </filter>
  </defs>

<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
    filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
    filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
    filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
    filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
    filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feConvolveMatrix</text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Die Filter *f1* und *f2* im obigen Beispiel, verwenden den Standardwert *3* für das Attribut ***order***. Daher sind jeweils genau 9 Werte für das Attribut ***kernelMatrix*** angegeben. Die Kernel-Matrix die im Filter *f3* festgelegt wird besteht nur aus einer Zeile mit 10 Spalten, die Kernel-Matrix im Filter *f4* aus 10 Zeilen mit nur einer Spalte. In beiden Fällen sind hierdurch genau 10 Punkte definiert. Daher sind jeweils genau 10 Werte für das Attribut ***kernelMatrix*** angegeben. Bei den Filtern *f5* und *f6* wird zusätzlich **feBlend** verwendet, um das Quellbild (Orginalbild) mit dem Output von **feConvolveMatrix** zu verschmelzen.

Für den primitiven Filter **feConvolveMatrix** stehen noch weitere Attribute zur Verfügung. Mit Hilfe der Attribute ***targetX*** und ***targetY*** können Sie einen beliebigen Punkt der Kernel-Matrix bestimmen, der auf den orginalen Bildpunkt gelegt wird. Im Normalfall ist das der Mittelpunkt der Kernel-Matrix. ***targetX*** akzeptiert einen Spaltenwert und ***targetY*** einen Zeilenwert, wobei beide Werte natürlich innerhalb der durch ***order*** angegeben Spalten- und Zeilenwerte der Kernel-Matrix liegen müssen.

Durch das Attribut ***divisor*** können Sie einen bei der Berechnung des Outputs verwendeten Divisior (Teiler) verändern. Wenn Sie dieses Attribut nicht verwenden, wird die Summe aller Bildpunkte der Kernel-Matrix als Divisor verwendet.

An den Rändern des Bildes kann es vorkommen, dass die Kernel-Matrix teilweise über das Bild lappt, so das einzelne Punkte der Kernel-Matrix nicht mehr auf Bildpunkten zu liegen kommen. Um festlegen zu können, wie solche Fälle gehandhabt werden, verwenden Sie das Attribut ***edgeMode***. ***edgeMode*** akzeptiert drei Werte: *duplicate*, *wrap* und *none*. Voreingestellter Wert ist *duplicate*, wodurch den

Werten, die im Leerraum liegen, die äußersten Werte am entsprechenden Rand zugewiesen werden. Bei Verwendung von *wrap* werden die aüßersten Werte des gegenüberliegenden Randes verwendet. Bei *none* wird den Werten 0 zugewiesen.

Falls Sie nicht möchten, dass auch die Alphawerte des Bildes neu berechnet werden, steht Ihnen zu diesem Zweck das Attribut ***preserveAlpha*** zur Verfügung. Dieses Attribut akzeptiert 2 mögliche Werte: den voreingestellten Wert *false* (die Alphawerte des Bildes werden neu berechnet) und der Wert *true* (die Alphawerte des Bildes werden nicht neu berechnet).

## 14.7 feDisplacementMap

Der primitive Filter **feDisplacementMap** verwendet Bildpunkte eines zweiten Input-Bildes um ein erstes Input-Bild räumlich zu verschieben.

Das zweite Input-Bild wird durch das Attribut ***in2*** bestimmt. Dabei kann dieses Attribut den gleichen Wert wie das Attribut ***in*** annehmen.

Wenn Sie nichts anderes festlegen, arbeitet der Filter standardmäßig mit den Alphawerten des zweiten Input-Bildes. Durch die Attribute ***xChannelSelector*** und ***yChannelSelector*** können Sie aber auch dafür sorgen, dass die Werte der Farbkanäle für die Verschiebung entlang der x- bwz. y-Achse verwendet werden. Daher sind die akzeptierten Werte für dieses Attribut: *A* (der voreingestellte Wert Alpha), *R* (rot), *G* (grün) und *B* (blau).

Den Verschiebungsfaktor legen Sie durch das Attribut ***scale*** fest. Voreingestellter Wert für ***scale*** ist *0*. Beachten Sie: Beim Wert *0* hat die Filteroperation keinen Effekt auf die Grafik. Weisen Sie diesem Attribut daher immer einen numerischen Wert ungleich *0* zu.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feDisplacementMap</title>
  <desc>
    Beispiele für die Verwendung von feDisplacementMap
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feDisplacementMap -->
    <filter id="f1">
      <feDisplacementMap in="SourceGraphic" in2="SourceGraphic"
        scale="1" />
    </filter>
    <filter id="f2">
      <feDisplacementMap in="SourceGraphic" in2="SourceGraphic"
        scale="2" />
    </filter>
    <filter id="f3">
      <feDisplacementMap in="SourceGraphic" in2="SourceGraphic"
        xChannelSelector="R" yChannelSelector="A" scale="1.3" />
    </filter>
    <filter id="f4">
```

```
    <feDisplacementMap in="SourceGraphic" in2="SourceGraphic"
      xChannelSelector="B" yChannelSelector="R" scale="2" />
   </filter>
<!-- zusätzlich mit feTurbulence -->
   <filter id="f5">
    <feTurbulence baseFrequency=".5" type="fractalNoise" result="out1" />
    <feDisplacementMap in="SourceGraphic" in2="out1"
      scale="5" />
   </filter>
   <filter id="f6">
    <feTurbulence baseFrequency=".5" type="turbulence" result="out1" />
    <feDisplacementMap in="SourceGraphic" in2="out1"
      scale="5" />
   </filter>
  </defs>

<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
    filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
    filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
    filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
    filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
    filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feDisplacementMap</text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Die ersten beiden Filter *f1* und *f2* des obigen Beispiels verwenden *SourceGraphic* für beide Input-Bilder des primitiven Filters **feDisplacementMap**. Da die Attribute ***xChannelSelector*** und/oder ***yChannelSelector*** nicht gesetzt sind, werden die Alphawerte des zweiten Input-Bildes *SourceGraphic* zur Berechnung der räumlichen Verschiebung verwendet. Man hätte also ebenso den Wert *SourceAlpha* für das Attribut ***in2*** verwenden können. Die Filter *f1* und *f2* unterscheiden sich lediglich durch einen unterschiedlichen Skalierungsfaktor.

Bei den Filteren *f3* und *f4* werden die Farbkanäle für die Verschiebungswerte durch die Verwendung der Attribute ***xChannelSelector*** und ***yChannelSelector*** verändert.

In den letzten beiden Filtern *f5* und *f6* wird als zweites Input-Bild der Output des primitiven Filters **feTurbulence** verwendet. Dieser primitive Filter erzeugt entweder ein Fraktal oder eine Turbulenz und füllt dann die Region des referenzierenden Elements komplett mit dem Output-Bild auf. Der nachfolgende primitive Filter **feDisplacementMap** verwendet dann die Alphawerte dieses Outputs um eine Verschiebung zu berechnen. Das so erzeugte Bild, d.h. der Output von **DisplacementMap** wird dann am Bildschirm dargestellt.

## 14.8 feGaussianBlur

Der primitive Filter **feGaussianBlur**, der sogenannte "Gaussche Weichzeichner", verwischt ein Input-Bild. Es wird also eine Unschärfe erzeugt.

Durch das Attribut ***stdDeviation*** geben Sie die Stärke oder Ausdehnung der Unschärfe in Form einer Zahl an. Wenn Sie zwei Werte für ***stdDeviation*** verwenden, bezieht sich der erste Wert auf die Ausdehnung der Unschärfe entlang der x-Achse und der zweite Wert auf die Ausdehnung der Unschärfe entlang der y-Achse. Beachten Sie: Wenn Sie hohe Werte für ***stdDeviation*** verwenden, sollten Sie den Filterbereich entsprechend groß festlegen, da der Effekt von **feGaussianBlur** viel Raum benötigt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feGaussianBlur</title>
  <desc>
    Beispiele für die Verwendung von feGaussianBlur
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feGaussianBlur -->
    <filter id="f1">
      <feGaussianBlur in="SourceGraphic" stdDeviation=".5" />
    </filter>
    <filter id="f2">
      <feGaussianBlur in="SourceGraphic" stdDeviation="1" />
    </filter>
    <filter id="f3" x="-20%" y="-20%" width="150%" height="150%">
      <feGaussianBlur in="SourceGraphic" stdDeviation="1.5,.5" />
    </filter>
    <filter id="f4" x="-20%" y="-20%" width="150%" height="150%">
      <feGaussianBlur in="SourceGraphic" stdDeviation=".5,1.5" />
    </filter>
<!-- zusätzlich mit feBlend, feOffset und feMerge -->
    <filter id="f5" x="-20%" y="-20%" width="150%" height="150%">
      <feGaussianBlur in="SourceGraphic" stdDeviation="5" result="out1" />
      <feBlend in="SourceGraphic" in2="out1" mode="darken" />
    </filter>
    <filter id="f6" x="-20%" y="-20%" width="150%" height="150%">
      <feGaussianBlur in="SourceAlpha" stdDeviation="2" result="out1" />
      <feOffset in="out1" dx="2" dy="-2" result="out2" />
```

```
    <feMerge>
      <feMergeNode in="out2" />
      <feMergeNode in="SourceGraphic" />
    </feMerge>
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
  filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
  filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
  filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
  filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
  filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
  filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feGaussianBlur</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Innerhalb der Filter *f1* bis *f4* werden im obigen Beispiel verschiedene Werte für das Attribut ***stdDeviation*** definiert. Auf diese Weise werden unterschiedliche Unschärfen des Quellbildes erzeugt.

Im Filter *f5* wird zusätzich noch der primitive Filter **feBlend** verwendet um den Output von **feGaussianBlur** mit dem Quellbild zu verschmelzen. Das Objekt *smilie* hat jetzt quasi einen Hof. Beachten Sie, dass die Größe des Filterbereichs durch die Attribute ***x***, ***y*** ***width*** und ***height*** im **filter**-Element entsprechend angepasst wurde, damit der Unschärfe-Effekt komplett dargestellt wird. Wenn der Filterbereich zu klein ist, wird der Effekt teilweise abgeschnitten.

Der Filter *f6* erzeugt einen Schatten des referenzierende Objekts. Dazu wird zuerst eine Unschärfe auf den Alphakanal des Bildes erzeugt. Dann wird dieser (schwarze) Output, also der Schatten, mit Hilfe von **feOffset** verschoben. Zuletzt wird das Quellbild durch **feMerge** mit dem Schatten vereinigt.

## 14.9 feOffset

Der primitive Filter **feOffset** verschiebt ein Input-Bild durch einen festzulegenden Vektor.

Die Verschiebung wird mit Hilfe der Attribute ***dx*** (Verschiebung entlang der x-Achse) und ***dy*** (Verschiebung entlang der y-Achse) bestimmt, die als Wert eine Längenangabe erwarten. Voreingestellter Wert für beide Attribute ist *0*, d.h. wenn Sie diese Attribute nicht verwenden, zeigt **feOffset** keinen sichtbaren Effekt.

Der primitive Filter **feOffset** ist ein wichtiger Bestandteil für die Erzeugung von Schatteneffekten.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Der primitive Filter feOffset</title>
 <desc>
  Beispiele für die Verwendung von feOffset
 </desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana,sans-serif; font-size:16px;
        font-weight:bold;}
  ]]>
  </style>
  <symbol id="smilie">
   <desc>ein lachendes Smilie</desc>
   <circle id="gesicht" cx="20" cy="20" r="15"
    fill="yellow" stroke="black" />
   <circle id="auge-links" cx="15" cy="15" r="2"
    fill="black" stroke="black" />
   <circle id="auge-rechts" cx="25" cy="15" r="2"
    fill="black" stroke="black" />
   <line id="nase" x1="20" y1="18" x2="20" y2="23"
    stroke="black" stroke-width="2" />
   <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
    stroke="black" fill="none" stroke-width="2" />
  </symbol>

<!-- 6 Filter mit feOffset -->
  <filter id="f1">
   <feOffset in="SourceGraphic" dx="-5" dy="-5" />
  </filter>
  <filter id="f2">
   <feOffset in="SourceGraphic" dx="5" dy="5" />
  </filter>
  <filter id="f3">
   <feOffset in="SourceAlpha" dx="-5" />
  </filter>
  <filter id="f4">
   <feGaussianBlur in="SourceAlpha" stdDeviation="2" result="out1" />
   <feOffset in="out1" dx="-3" dy="2" />
  </filter>
<!-- zusätzlich mit feGaussianBlur und feMerge -->
  <filter id="f5" x="-25%" y="-20%" width="150%" height="150%">
   <feGaussianBlur in="SourceAlpha" stdDeviation="2" result="out1" />
   <feOffset in="out1" dx="-3" dy="2" result="out2" />
   <feMerge>
    <feMergeNode in="out2" />
    <feMergeNode in="SourceGraphic" />
```

```
    </feMerge>
   </filter>
   <filter id="f6" x="-.3" y="-.5" width="1.9" height="1.9">
    <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="out1" />
    <feOffset in="out1" dx="4" dy="-4" result="out2" />
    <feMerge>
     <feMergeNode in="out2" />
     <feMergeNode in="SourceGraphic" />
    </feMerge>
   </filter>
  </defs>

<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
   filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
   filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
   filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
   filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
   filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
   filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feOffset</text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Im obigen Beispiel bewirken die Filter *f1* und *f2* eine Verschiebung des Quellbildes. Der Effekt von **feOffset** benötigt in der Regel mehr Raum, als das Ausmaß des Orginalbildes. Der Effekt wird im Beispiel also abgeschnitten dargestellt, da der Bereich des jeweiligen Filters nicht angepasst wurde.

Im Filtern *f3* wird **feOffset** auf den Alphakanal des Quellbildes angewandt. Im Filter *f4* wird das Alphabild zuvor noch durch die Verwendung des primitiven Filters **feGaussianBlur** weichgezeichnet. Diese Technik ist die Grundlage zur Erzeugung von Schatteneffekten.

Um einen Schatteneffekt zu erzeugen müssen Sie also zuerst der weichgezeichnete Alphakanal des Quellbildes etwas verschieben und diesen Output dann mit dem Quellbild vermengen (überblenden). In den Filtern *f5* und *f6* wird daher zuletzt **feMerge** verwendet. Dieser Filter überblendet mehrere Input-Bilder in der Reihenfolge Ihres Vorkommens innerhalb der ***in***-Atteribute seiner Kind-Elemente **feMergeNode**.

## 14.10 feMorphology

Der primitive Filter **feMorphology** verdickt oder verdünnt ein Input-Bild.

Mit dem Attribut ***operator*** geben Sie an, ob Sie das Bild verdünnen oder verdicken möchten. Daher akzeptiert ***operator*** zwei Werte: der voreingestellte Wert *erode* verdünnt das Bild, der Wert *dilate* verdickt das Bild.

Das Ausmaß der Verdünnung oder Verdickung legen Sie durch das Attribut ***radius*** fest, das eine oder zwei Zahlen als Wert akzeptiert. Wenn Sie zwei, durch Komma getrennte, Zahlen verwenden bezieht sich die erste Zahl auf den x-Radius und die zweite Zahl auf den y-Radius. Wenn Sie dem Attribut ***radius*** nur eine Zahl als Wert zuweisen bezieht sich dieser Wert auf beide Radien.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feMorphology</title>
  <desc>
    Beispiele für die Verwendung von feMorphology
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feMorphology -->
    <filter id="f1">
      <feMorphology operator="erode" radius="1" />
    </filter>
    <filter id="f2">
      <feMorphology operator="dilate" radius="1" />
    </filter>
    <filter id="f3">
      <feMorphology radius=".5" />
    </filter>
    <filter id="f4">
      <feMorphology operator="dilate" radius=".5" />
    </filter>
<!-- zusätzlich mit feGaussianBlur, feOffset,
        feTurbulence, feDisplacementMap und feMerge -->
    <filter id="f5"
      x="-.3" y="-.5" width="1.9" height="1.9">
      <feGaussianBlur in="SourceAlpha"
        stdDeviation="2"
        result="out1" />
```

```
    <feOffset in="out1"
      dx="4" dy="-.5"
      result="out2" />
    <feTurbulence in="SourceGraphic"
      baseFrequency=".7" type="turbulence"
      result="out3" />
    <feDisplacementMap in="SourceGraphic" in2="out3"
      scale="5"
      result="out4" />
    <feMorphology in="out4"
      operator="erode" radius=".5"
      result="out5" />
    <feMerge>
      <feMergeNode in="out2" />
      <feMergeNode in="out5" />
    </feMerge>
  </filter>
  <filter id="f6"
    x="-.3" y="-.5" width="1.9" height="1.9">
    <feGaussianBlur in="SourceAlpha" stdDeviation="2" />
    <feOffset dx="4" dy="-.5"
      result="out2" />
    <feTurbulence in="SourceGraphic" baseFrequency=".7" type="turbulence"
      result="out3" />
    <feDisplacementMap in="SourceGraphic" in2="out3"
      scale="5" />
    <feMorphology operator="dilate" radius=".5"
      result="out5" />
    <feMerge>
      <feMergeNode in="out2" />
      <feMergeNode in="out5" />
    </feMerge>
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
   filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
   filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
   filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
   filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
   filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
   filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feMorphology</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Die Filter *f1* bis *f4* zeigen unterschiedliche Effekte des primitiven Filters **feMorphology**. Die Filter *f1* und *f3* verwenden den Wert *erode* für das Attribut ***operator*** (Voreinstellung), das Bild wird also verdickt dargestellt. Die Filter *f2* und *f4* verwenden den Wert *dilate*, das Bild wird also verdünnt dargestellt.

Die Filter *f5* und *f6* sind bis auf den Wert für das Attribut ***operator*** im primitiven Filter **feMorphology** identisch. Für *f5* wurde *erode* und für *f6* wurde *dilate* verwendet.

Beachten Sie: Im Filter *f6* wird auf alle, nicht unbedingt notwendigen Attribute ***in*** und ***result*** für die einzelnen Filteroperationen verzichtet. Das Attribut ***in*** verwendet als voreingestellten Wert den Output des

direkt vorangegangen primitiven Filters im selben **filter**-Element. Wenn ein solcher Output nicht existiert (weil der primitive Filter der erstplatzierte ist) verwendet ***in*** den Wert *SourceGraphic*. Das Attribut ***result*** muß nur dann ausdrücklich festgelegt werden, wenn ein nicht direkt folgender primitiver Filter im selben **filter**-Element diesen Output als Input-Bild verwenden soll.

Funktionsweise der Filter *f5* und *f6*: Durch den primitiven Filter **feGaussianBlur** wird zuerst das Alphabild weichgezeichnet, danach wird es mit Hilfe von **feOffset** verschoben. Der so erzeugte Schatten ist das Resultat *out2*. Nun wird durch **feTurbulence** eine Turbulenz im Bereich des Quellbildes erzeugt, die von der folgenden Filteroperation **feDisplacementMap** als zweites Input-Bild verwendet wird. Das daraus resultierende Output-Bild wird hiernach durch **feMorphology** verdickt bzw. verdünnt. Das Resultat dieser 3 aufeinanderfolgenden Filter trägt die Bezeichnung *out5*. Zuletzt wird **feMerge** mit seinen Kindern **feMergeNode** verwendet, um die Outputs *out2* und *out5* zu vermischen.

## 14.11 feBlend

Der primitive Filter **feBlend** setzt ein erstes und ein zweites Input-Bild durch pixelweise Vermischung (Alpha-Blending) zu einem Bild zusammen. Dabei sind mehrere Effekte wie unter anderm multiplizieren, verdunkeln oder aufhellen möglich.

Das zweite Input-Bild wird mit dem Attribut ***in2*** festgelegt.

Durch das Attribut ***mode*** können Sie die Vermischung (Überblendung) der beiden Input-Bilder näher bestimmen. Dazu stehen diesem Attribut 5 Werte zur Verfügung, welche die Art der Berechnung festlegen: *normal* (voreingestellter Wert), *screen*, *darken*, *lighten* und *multiply*.

Das Output-Bild, das der primitive Filter **feBlend** bei Verwendung des Wertes *normal* für das Attribut ***mode*** erzeugt, könnten Sie ebenfalls durch den Gebrauch folgender Filter erzeugen lassen: **feComposite** mit dem Wert *over* für das Attribut ***operator*** (siehe dort) oder **feMerge** und **feMergeNode**.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feBlend</title>
  <desc>
    Beispiele für die Verwendung von feBlend
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 5 Filter mit feBlend und feTurbulence -->
    <filter id="f1">
      <feTurbulence in="SourceGraphic"
        baseFrequency=".2" type="fractalNoise"
        result="out1" />
      <feBlend in="SourceGraphic" in2="out1"
        mode="normal" />
    </filter>
    <filter id="f2">
      <feTurbulence in="SourceGraphic"
        baseFrequency=".2" type="fractalNoise"
        result="out1" />
      <feBlend in="SourceGraphic" in2="out1"
        mode="screen" />
    </filter>
    <filter id="f3">
```

```
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feBlend in="SourceGraphic" in2="out1"
      mode="darken" />
  </filter>
  <filter id="f4">
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feBlend in="SourceGraphic" in2="out1"
      mode="lighten" />
  </filter>
  <filter id="f5">
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feBlend in="SourceGraphic" in2="out1"
      mode="multiply" />
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
  filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
  filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
  filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
  filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
  filter="url(#f5)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feBlend
 <tspan dy="-1.5"
  style="font-size:12px; text-rendering:optimizeSpeed; font-weight:normal;">
  + feTurbulence</tspan>
 </text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Das obige Beispiel zeigt die Darstellung der 5 verschiedenen Werte des ***mode***-Attributs.

Als zweites Input-Bild wurde für alle Filter *f1* bis *f5* ein direkt zuvor durch **feTurbulence** erzeugtes Fraktal bestimmt. Mit diesem Fraktal wird das Quellbild durch **feBlend** vermischt. Dabei werden nacheinander die Werte *normal*, *screen*, *darken*, *lighten* und *multiply* für das ***mode***-Attribut verwendet.

## 14.12 feComposite

Der primitive Filter **feComposite** kombiniert ein erstes und ein zweites Input-Bild zu einem einzigen Bild. Dabei werden die sogenannten Porter-Duff-Operationen verwendet. Dieser primitive Filter ist sinnvoll um den Output von Beleuchtungsfiltern wie **feSpecularLighting** oder **feDiffuseLighting** mit mit dem Quellbild zu vermischen.

Das zweite Input-Bild wird mit dem Attribut ***in2*** festgelegt.

Durch einen von insgesamt 6 Werten für das Attribut ***operator*** bestimmen Sie die Porter-Duff-Operation, die für die Vermischung der beiden Input-Bilder verwendet wird. Mögliche Werte für ***operator*** sind: *over* (der voreingestellte Wert), *in*, *out*, *atop*, *xor* und *arithmetic*.

Wenn Sie den Wert *arithmetic* für das ***operator***-Attribut verwenden, müssen Sie zusätzlich vier Konstanten angeben, die bei der Berechnung des Output als Faktoren verwendet werden. Diese vier Konstanten sind einzeln durch die Attribute ***k1***, ***k2***, ***k3*** und ***k4*** festzulegen. Die Voreinstellung für jedes dieser vier Attribute ist der Wert *0*.

Das Output-Bild, das der primitive Filter **feComposite** bei Verwendung des Wertes *over* für das Attribut ***operator*** erzeugt, könnten Sie ebenfalls durch den Gebrauch folgender Filter erzeugen lassen: **feBlend** mit dem Wert *normal* für das Attribut ***mode*** (siehe dort) oder **feMerge** und **feMergeNode**.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feComposite</title>
  <desc>
    Beispiele für die Verwendung von feComposite
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feComposite -->
    <filter id="f1">
      <feTurbulence in="SourceGraphic"
        baseFrequency=".2" type="fractalNoise"
        result="out1" />
      <feComposite in="SourceGraphic" in2="out1"
        operator="over" />
    </filter>
    <filter id="f2">
      <feTurbulence in="SourceGraphic"
```

```
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feComposite in="SourceGraphic" in2="out1"
      operator="in" />
  </filter>
  <filter id="f3">
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feComposite in="SourceGraphic" in2="out1"
      operator="out" />
  </filter>
  <filter id="f4">
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feComposite in="SourceGraphic" in2="out1"
      operator="atop" />
  </filter>
  <filter id="f5">
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feComposite in="SourceGraphic" in2="out1"
      operator="xor" />
  </filter>
  <filter id="f6">
    <feTurbulence in="SourceGraphic"
      baseFrequency=".2" type="fractalNoise"
      result="out1" />
    <feComposite in="SourceGraphic" in2="out1"
      operator="arithmetic" k1="2" k2="0" k3="0" k4="0" />
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
  filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
  filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
  filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
  filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
  filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
  filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feComposite</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Das obige Beispiel zeigt die Darstellung der 6 verschiedenen Werte des ***operator***-Attributs.

Als zweites Input-Bild wurde für alle Filter *f1* bis *f5* ein direkt zuvor durch **feTurbulence** erzeugtes Fraktal bestimmt. Mit diesem Fraktal wird das Quellbild durch **feComposite** vermischt. Dabei werden nacheinander die Werte *over*, *in*, *out*, *atop*, *xor* und *arithmetic* für das ***operator***-Attribut verwendet.

Im folgenden Beispiel werden die primitiven Filter **feGaussianBlur** und **feOffset** verwendet um einen Schatten zu erzeugen. Hiernach wird mit den primitiven Filtern **feSpecularLighting** und **fePointLight** eine

Lichtquelle erzeugt und durch **feComposite** mit dem Quellbild vermischt. Zuletzt wird das beleuchtete Quellbild mit dem Schatten vermischt - ebenfalls mit Hilfe von **feComposite**.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feComposite</title>
  <desc>
    Beispiele für die Verwendung von feComposite
    in Verbindung mit Beleutungsfiltern
  </desc>
  <defs>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- Filter mit feComposite, Schatten und Beleuchtung -->
    <filter id="f1" x="-.3" y="-.3" width="1.5" height="1.5">
      <feGaussianBlur in="SourceAlpha" stdDeviation="2"
        result="out1" />
      <feOffset in="out1" dx="2" dy="2" result="out2" />
      <feSpecularLighting in="out1"
        surfaceScale="3" specularExponent="15"
        result="out3">
        <fePointLight x="-150" y="-150" z="100" />
      </feSpecularLighting>
      <feComposite in="SourceGraphic" in2="out3" operator="arithmetic"
        k1="0" k2="1" k3="1" k4="0" result="out4" />
      <feComposite in="out4" in2="out2" operator="over" />
    </filter>
  </defs>

<!-- die Instanz des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(0) scale(9)"
    filter="url(#f1)" />
</svg>
```

## 14.13 feMerge

Der primitive Filter **feMerge** kombiniert mit Hilfe seiner Kind-Elemente **feMergeNode** mehrere Input-Bilder zu einem einzigen Bild. Dabei wird die Porter-Duff-Operation *over* verwendet. Die einzelnen Bilder werden also wie Layer übereinandergelegt, wobei das zuerst angegebene Bild zuunterst und das zuletzt angegebene Bild zuoberst angezeigt wird.

Der Output von primitiven Filtern ist oftmals nur ein Zwischenprodukt, das nur mit anderen Outputs bzw. anderen Zwischenprodukten vereint, das gewünschte Ergebnis liefert. Durch die Verwendung von **feMerge** und **feMergeNode** können Sie beliebig viele Outputs zu einem Bild zusammenfügen. Dies unterscheidet ***feMerge*** von **feBlend** und **feComposite**, die nur zwei Input-Bilder vermischen können.

Das Element **feMerge** dient als Container-Element. Innerhalb dieses Containers können Sie beliebig viele Kind-Elemente **feMergeNode** platzieren.

Über das Attribut ***in***, das jedem einzelnen Kind-Element **feMergeNode** zugeordnet wird, legen Sie die einzelnen Input-Bilder fest, die dann zu einem einzigen Bild zusammengefügt werden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feMerge</title>
  <desc>
    Ein Beispiel für die Verwendung von feMerge
    und feMergeNode
  </desc>
  <defs>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- Filter mit feMerge und feMergeNode -->
    <filter id="f1" x="-.3" y="-.3" width="1.8" height="1.8">
<!-- out1: primitive Filter zur Erzeugung eines Hintergrunds  -->
      <feTurbulence in="SourceGraphic"
        baseFrequency=".2" type="fractalNoise"
        result="tmp1" />
      <feColorMatrix in="tmp1"
        type="saturate" values=".1"
        result="out1" />
<!-- out2: primitive Filter zur Erzeugung eines Schattens -->
      <feGaussianBlur in="SourceAlpha"
        stdDeviation="3"
        result="tmp2" />
      <feOffset in="tmp2"
        dx="4" dy="2"
        result="out2" />
<!-- out3: primitiver Filter zur "Veränderung" des Smilie -->
      <feTurbulence in="SourceGraphic"
        baseFrequency=".5" type="turbulence"
```

```
      result="tmp3" />
    <feDisplacementMap in="SourceGraphic" in2="tmp3"
      result="tmp3" />
    <feMorphology in="tmp3"
      operator="dilate" radius=".5"
      result="out3" />
<!-- Vermengung der Output-Bilder out1 bis out3 (bedeutet:
    Aufeinanderlegen der erzeugenten Layer out1 bis out3
    in der angegebenen Reihenfolge). -->
    <feMerge>
      <feMergeNode in="out1" />
      <feMergeNode in="out2" />
      <feMergeNode in="out3" />
    </feMerge>
  </filter>
 </defs>

<!-- die Instanz des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(15,20) scale(8)"
   filter="url(#f1)" />
</svg>
```

Im einzigen Filter *f1* des obigen Beispiels werden **feMerge** und **feMergeNode** verwendet, um drei Output-Bilder zu vermengen. Diese drei Output-Bilder, die auch als Layer bezeichnet werden können, werden also übereinandergelegt. Dabei überdeckt der jeweils nachfolgende Layer den zuvor angegebenen, d.h. die Reihenfolge der Layer innerhalb des **feMerge**-Containers ist von großer Bedeutung.

Der erste Layer *out1* wird durch den primitiven Filter **feTurbulence** in Verbindung mit **feColorMatrix** erzeugt. **feTurbulence** füllt den Canvas des Zielelements vollständig mit einem Fraktal. In diesem Output-Bild wird danach durch **feColorMatrix** die Farbsättigung vermindert. Es wird also ein farbverminderters Hintergrundbild erzeugt.

Der zweite Layer *out2* beinhaltet einen durch **feGaussianBlur** und **feOffset** einen Fallschatten (drop shadow).

Im dritten und letzten Layer *out3* wird durch **feTurbulence** eine Turbulenz erzeugt, die dem folgenden **feDisplacementMap** zur Verschiebung der Pixel des Quellbildes dienen. Das daraus resultierende leicht gesprenkelte Bild des Smilies wird zuletzt durch die Verwendung von **feMorphology** verdünnt.

Zu guter Letzt werden die drei erzeugen Layer *out1* bis *out3* durch **feMerge** und **feMergeNode** übereinandergelegt.

## 14.14 feColorMatrix

Der primitive Filter **feColorMatrix** wendet eine Matrix-Transformation auf die RGBA Farb- und Alphawerte jedes Pixels einer Input-Grafik an und erzeugt auf diese Weise ein Bild mit veränderten RGBA Farb- und Alphawerten. Es wird also eine Farbveränderung erzeugt.

Durch das Attribut ***type*** legen Sie die Art der Matrixoperation, d.h. die Art der Farbveränderung fest. Der voreingestellte Wert für ***type*** ist *matrix*. Der Wert *matrix* benötigt die Angabe einer kompletten 5 x 4 Matrix, d.h. 20 Werte für die 20 neu zu berechnenden Bildpunkte dieser Matrix, die auf jeden Punkt des Orginalbildes gelegt wird. Alle weiteren möglichen Werte für das Attribut ***type*** sind Vereinfachungen dieser grundlegenden Matrixoperation, die es Ihnen erlauben, allgemein gebräuchliche Farboperationen auf weniger komplizierte Art zu erzeugen. Alle möglichen Werte für ***type***:

- *matix* (selbst festzulegende Matrixoperation),
- *saturate* (Sättigung),
- *hueRotate* (Farbtonänderung) und
- *luminanceToAlpha* (Aufhellung des Alphakanals).

Wenn Sie *luminanceToAlpha* verwenden, sollten Sie den so erzeugten Output zusätzlich durch **feComposite** mit dem Quellbild vermengen um eine Aufhellung des Originalbildes (und nicht des Alphakanals) zu erzeugen.

Mit Ausnahme des Wertes *luminanceToAlpha* benötigen alle übrigen Werte zusätzliche Angaben. Diese Angaben werden mit Hilfe des Attributes ***values*** vorgenommen. Der Wert für das Attribut ***values*** ist also direkt abhängig vom verwendeten Wert des Attributes ***type***.

Wenn Sie den Wert *matrix* für das Attribut ***type*** verwenden, erhält ***values*** folgenden Wert: eine Liste von 20 Matrix-Werten (zwischen 0 und 1). Als Beispiel eine zur Originalmatrix identische Matrix: ***type***: *matrix* ***values***: *1 0 0 0 0, 0 1 0 0 0, 0 0 1 0 0, 0 0 0 1 0* Dieser Wert ist in diesem Fall auch der voreingestellte Wert für ***values***.

Wenn Sie den Wert *saturate* verwenden, erhält ***values*** folgenden Wert: genau eine Zahl zwischen 0 und 1. Ein Beispiel: ***type***: *saturate* ***values***: *.3* Der Wert *1* ist in diesem Fall der voreingestellte Wert für ***values***.

Wenn Sie den Wert *hueRotate* verwenden, erhält ***values*** folgenden Wert: eine Ganzahl die eine Gradzahl darstellt (von 0 bis 360). Ein Beispiel: ***type***: *hueRotate* ***values***: *90* Der Wert *0* ist in diesem Fall der voreingestellte Wert für ***values***.

Wenn Sie den Wert *luminanceToAlpha* für das Attribut ***type*** verwenden, benötigen Sie das Attribut ***values*** nicht.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feColorMatrix</title>
  <desc>
    Beispiele für die Verwendung von feColorMatrix
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
```

```
      fill="black" stroke="black" />
    <circle id="auge-rechts" cx="25" cy="15" r="2"
      fill="black" stroke="black" />
    <line id="nase" x1="20" y1="18" x2="20" y2="23"
      stroke="black" stroke-width="2" />
    <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
      stroke="black" fill="none" stroke-width="2" />
  </symbol>

<!-- 6 Filter mit feColorMatrix -->
  <filter id="f1">
    <feColorMatrix type="saturate" values=".3" />
  </filter>
  <filter id="f2">
    <feColorMatrix type="saturate" values=".7" />
  </filter>
  <filter id="f3">
    <feColorMatrix type="hueRotate" values="90" />
  </filter>
  <filter id="f4">
    <feColorMatrix type="hueRotate" values="-90" />
  </filter>
  <filter id="f5">
    <feColorMatrix type="luminanceToAlpha" result="tmp1" />
    <feComposite in="SourceGraphic" in2="tmp1"
      operator="in" />
  </filter>
  <filter id="f6">
    <feColorMatrix type="matrix"
      values="1 0 0 0 0, 0 1 1 1 0, 0 1 1 0 0, 0 0 0 1 0" />
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
  filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
  filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
  filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
  filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
  filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
  filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feColorMatrix</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Das obige Beispiel zeigt die Verwendung alle Arten von Farbmanipulation durch den primitiven Filter **feColorMatrix**.

Die Filter *f1* und *f2* zeigen Farbveränderungen des Typs *saturate* und die Filter *f3* und *f4* Farbveränderungen des Typs *hueRotate*. Bei dem Filter *f5* wird der Typ *luminanceToAlpha* verwendet. Zusätzlich wird der Filter **feComposite** angewandt, um den aufgehellten Alphakanal mit dem Originalbild zu verschmelzen. Der Filter *f6* zeigt eine Farbveränderung des Typs *matrix*.

## 14.15 feComponentTransfer

Der primitive Filter **feComponentTransfer** legt für jeden Bildpunkt einer einzelnen Fabe oder des Alphakanals einen neuen Wert fest. Dieser neue Wert wird für jeden Farbkanal und für den Alphakanal durch eine festgelegte mathematische Funktion berechnet. Auf diese Weise sind Veränderungen der Helligkeit, des Kontrast oder der Farbbalance eines Input-Bildes möglich.
**feComponentTransfer** ist ein Container-Element. Um den Bildpunkten aus jedem Farbkanal und dem Alphakanal einen neuen Wert zuzuordnen, müssen Sie die Kind-Elemente von **feComponentTransfer** verwenden:

- **feFuncR** für den roten Farbkanal
- **feFuncG** für den grünen Farbkanal
- **feFuncB** für den blauen Farbkanal
- **feFuncA** für den Alphakanal

Dabei müssen Sie nicht alle Kind-Elemente verwenden. Wenn Sie z.B. den Alphakanal nicht verändern wollen, brauchen Sie das Element **feFuncA** nicht. Jedem dieser Kind-Elemente sind die Attribute der Attributgruppe **component_transfer_function_attributes** zugeordnet. Diese Attribute werden im folgenden erläutert.
***type*** Voreinstellung: keine. Mit diesem Attribut legen Sie die Funktion fest, die bei Berechnung des neuen Farbwertes angewendet wird. Der für ***type*** verwendete Wert legt die Anwendbarkeit der weiteren Attribute fest. Nachfolgend ein Liste der möglichen Werte, ihrer Bedeutung und der daraufhin anzuwendenden Attribute.

- *identity* - übernimmt exakt den alten Wert des Bildpunkts.
- *table* - wendet als Funktion eine Tabellenfunktion (lineare Interpolation) an, deren Werte durch das Attribut ***tableValues*** bestimmt werden müssen. Dieser Wert eignet sich gut für Farbveränderungen.
- *discrete* - wendet eine Treppenfunktion (step function) an, deren Werte ebenfalls durch das Attribut ***tableValues*** bestimmt werden müssen. Dieser Wert eignet sich für Farbveränderungen.
- *linear* - wendet eine lineare Funktion an, deren Steigung durch das Attribut ***slope*** und das Attribut ***intercept*** bestimmt werden muß. Dieser Wert eignet sich gut für Veränderungen der Helligkeit.
- *gamma* - wendet eine Exponentialfunktion an, die durch die Attribute ***amplitude***, ***exponent*** und ***offset*** bestimmt werden muß. Dieser Wert eignet sich gut für Kontrasteinstellungen.

***tableValues*** Voreinstellung: leer. Zu verwenden bei den Funktionstypen *table* und *discrete*. Dieses Attribut akzeptiert als Wert eine durch Leerzeichen und/oder getrennt Liste von Zahlen, wie z.B. *0 1 0 0*, die als Tabellenwerte für die mathematische Funktion verwendet werden. Eine leere Liste bewirkt eine identische Ausgabe des Bildpunkts.
***slope*** Voreinstellung: *1*. Zu verwenden beim Funktionstyp *linear*. Durch dieses Attribut legen Sie die Steigung bzw. einen Faktor für die lineare Funktion fest. Als Wert ist eine Zahl zu verwenden.
***intercept*** Voreinstellung: *0*. Zu verwenden beim Funktionstyp *linear*. Durch dieses Attribut legen Sie eine Verschiebungwert bzw. einen Summanden für die lineare Funktion fest. Als Wert ist eine Zahl zu verwenden.
***amplitude*** Voreinstellung: *1*. Zu verwenden beim Funktionstyp *gamma*. Durch dieses Attribut bestimmen Sie die Wellenlänge bzw. einen Faktor für die Exponentialfunktion (gamma function). Als Wert ist eine Zahl zu verwenden.
***exponent*** Voreinstellung: *1*. Zu verwenden beim Funktionstyp *gamma*. Durch dieses Attribut bestimmen Sie den Exponenten für die Exponentialfunktion (gamma function). Als Wert ist eine Zahl zu verwenden.
***offset*** Voreinstellung: *0*. Zu verwenden beim Funktionstyp *gamma*. Durch dieses Attribut bestimmen Sie eine Verschiebungswert bzw. einen Summanden für die Exponentialfunktion (gamma function). Als Wert ist eine Zahl zu verwenden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feComponentTransfer</title>
  <desc>
    Beispiele für die Verwendung von feComponentTransfer
  </desc>
  <defs>
```

```
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <linearGradient id="verlauf"
      x1="0" y1="0" x2="1" y2="0">
      <stop offset="0" stop-color="gray" />
      <stop offset="1" stop-color="yellow" />
    </linearGradient>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="url(#verlauf)" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 5 Filter mit feComponentTransfer -->
    <filter id="f1">
      <feComponentTransfer>
        <feFuncR type="table" tableValues="1 1 0 0" />
        <feFuncG type="table" tableValues="0 1 1 0" />
        <feFuncB type="table" tableValues="0 0 1 1" />
      </feComponentTransfer>
    </filter>
    <filter id="f2">
      <feComponentTransfer>
        <feFuncR type="discrete" tableValues="1 1 0 0" />
        <feFuncG type="discrete" tableValues="0 1 1 0" />
        <feFuncB type="discrete" tableValues="0 0 1 1" />
      </feComponentTransfer>
    </filter>
    <filter id="f3">
      <feComponentTransfer>
        <feFuncR type="linear" slope=".3" intercept="0" />
        <feFuncG type="linear" slope=".3" intercept="0"  />
        <feFuncB type="linear" slope=".3" intercept="0"  />
      </feComponentTransfer>
    </filter>
    <filter id="f4">
      <feComponentTransfer>
        <feFuncR type="linear" slope="2" intercept="0" />
        <feFuncG type="linear" slope="2" intercept="0"  />
        <feFuncB type="linear" slope="2" intercept="0"  />
        <feFuncA type="linear" slope="1" intercept=".1"  />
      </feComponentTransfer>
    </filter>
    <filter id="f5">
      <feComponentTransfer>
        <feFuncR type="gamma" amplitude="2" exponent="3" offset="0" />
        <feFuncG type="gamma" amplitude="2" exponent="3" offset="0"  />
        <feFuncB type="gamma" amplitude="2" exponent="3" offset="0"  />
      </feComponentTransfer>
    </filter>
    <filter id="f6">
      <feComponentTransfer>
```

```
      <feFuncR type="gamma" amplitude="6" exponent="3" offset="0" />
      <feFuncG type="gamma" amplitude="6" exponent="3" offset="0"  />
      <feFuncB type="gamma" amplitude="6" exponent="3" offset="0"  />
    </feComponentTransfer>
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
  filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
  filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
  filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
  filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
  filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
  filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feComponentTransfer</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Die obige Grafik zeigt Beispiele der verschiedenen Funktionsarten von **feComponentTransfer**, außer für *identity*, da bei Verwendung dieses Wertes für das Attribut ***type*** innerhalb der einzelnen Kind-Elemente, eine identische Ausgabe erzeugt wird. Da die Effekte dieses primitiven Filters besser an einem Farbverlauf demonstriert werden können, ist das *smilie* in diesem Beispiel mit einem solchen gefüllt.

Der Filter *f1* verwendet für die Farbkanäle R, G und B den Funktionstyp *table* als Wert für ***type***, daher wird den jeweiligen Kind-Elementen außerdem das Attribut ***tableValues*** mit eine Liste von Zahlwerten zugeordnet. Die Farbbalance wird verändert.

Der Filter *f2* verwendet für die Farbkanäle R, G und B den Funktionstyp *discrete* als Wert für ***type***, daher wird den jeweiligen Kind-Elementen außerdem das Attribut ***tableValues*** mit einer Liste von Zahlwerten zugeordnet. Die Farbbalance wird stufenweise verändert. Die Farbübergänge gehen verloren.

Der Filter *f3* verwendet für die Farbkanäle R, G und B den Funktionstyp *linear* als Wert für ***type***, daher werden den jeweiligen Kind-Elementen außerdem die Attribute ***scope*** und ***intercept*** zugeordnet. Die Helligkeit des Bildes wird nach dunkler verändert.

Der Filter *f4* verwendet für die Farbkanäle R, G, B und den Alphakanl ebenfalls den Funktionstyp *linear* als Wert für ***type***. Auch in diesem Fall werden den jeweiligen Kind-Elementen außerdem die Attribute ***scope*** und ***intercept*** zugeordnet. Die Helligkeit des Bildes wird nach heller verändert. Durch Einbeziehung des Alphakanals wird ein hellgrauer Hintergrund erzeugt.

Die Filter *f5* und *f6* verwenden für die Farbkanäle R, G und B den Funktionstyp *gamma* als Wert für ***type***, daher werden den jeweiligen Kind-Elementen außerdem die Attribute ***amplitude***, ***exponent*** und *offset* zugeordnet. Der Kontrast des Bildes wird verändert. Die Leuchtkraft nimmt zu.

Durch verschiedenste Attributwerte innerhalb des primitiven Filters feComponentTransfer bzw. seiner Kind-Elemente können für alle durch type bestimmten Funktionsarten die verblüffensten Resultate erzielt werden. Test it! :-)

## 14.16 feDistantLight, fePointLight, feSpotLight

Die primitiven Filter für Beleuchtungseffekte **feDiffuseLighting** und **feSpecularLighting**, die in den folgenden Unterkapiteln behandelt werden, verwenden zur Bestimmung der eigentlichen Lichtquelle genau eines der folgenden primitiven Filter-Elemente zur Definition einer Lichtquelle und deren Position als Kind-Element:

- **feDistantLight**
- **fePointLight** oder
- **feSpotLight**

Bei Verwendung des primitiven Filters **feDistantLight** können Sie die die Position der Lichtquelle durch zwei Attribute bestimmen: ***azimuth*** und ***elevation***. Stellen Sie sich vor Ihre Grafik läge flach auf Ihrer Arbeisfläche - und nun stülpen Sie gedanklich eine durchsichtige Halbkugel über Ihre Grafik. Ausgehend von diesem Modell bestimmt ***azimuth*** die Position der Lichtquelle auf der Grundlinie der Halbkugel (der Kreislinie um die Grafik). Als Wert ist ein Winkelmaß (0 bis 360) anzugeben. So bedeutet z.B. der Wert *0*, dass die Lichtquelle rechts auf der Grundlinie positioniert wird, der Wert *90* bedeutet das die Lichtquelle vorne positioniert wird, bei einem Wert von *180* ist sie links, u.s.w. Mit dem Attribut ***elevation*** bestimmen Sie die Position der Lichtquelle auf der Halbkugel. Als Wert ist ebenfalls ein Winkelmaß anzugeben. Dabei bedeutet z.B. der Wert *0*, dass die Lichtquelle auf der Grundlinie bleibt, der Wert *90* bedeutet den höchsten Punkt der Halbkugel oberhalb der Grafik, d.h. das Licht fällt senkrecht auf die Grafik. Bei einem Wert zwischen *180* und *360* wird die Grafik von unten beleuchtet.

Der primitive Filter **fePointLight** benötigt zu Angabe seiner Position die Attribute ***x***, ***y*** und ***z***. Durch diese Attribute legen Sie eine Koordinate innerhalb eines 3-dimensionalen Raumes fest, mit der die Postion der Lichtquelle bestimmt wird. Mit ***x*** wird die Position auf der x-Achse definiert, mit ***y*** die Position auf der y-Achse und mit ***z*** die Position auf der z-Achse. Positive Werte für ***z*** bedeuten, dass der Punkt oberhalb der Grafik positioniert ist, negative Werte für ***z*** bedeuten, dass die Grafik von unten beleuchtet wird.

Auch die Positionierung der Lichtquelle bei **feSpotLight** wird durch die Attribute ***x***, ***y*** und ***z*** vorgenommen. Zusätlich stehen diesem Element noch die Attribute ***pointsAtX***, ***pointsAtY*** und ***pointsAtZ*** zur Verfügung. Durch diese Attribute legen Sie einen zweiten Punkt im 3-dimensionalen Koordinatensystem fest, auf den die Lichtquelle zielt. Die Streuung (den Fokus) der Lichtquelle können Sie mit dem Attribut ***specularExponent*** beeinflussen. Es akzeptiert eine Zahl als Wert. Je höher diese Zahl desto größer die Streuung. Voreingestellter Wert ist *1*. Um das Ausmaß des Lichtkegels im Zielbereich des Lichts einzuschränken (normalerweiser erfolgt keine Einschränkung), steht das Attribut ***limitingConeAngle*** zur Verfügung. Dieses Attribut akzeptiert ein Winkelmaß als Wert. Sie geben den Winkel zwischen der Lichtachse, die ihren Ursprung am Punkt der Lichtquelle hat und durch den Zielpunkt des Lichts verläuft, und einer zweiten Achse an, die ihren Ursprung ebenfalls am Punkt der Lichtquelle hat.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="390px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Die primitiven Filter
    feDistantLight, fePointLight, feSpotLight</title>
  <desc>
    Beispiele für die Verwendung von feDistantLight, fePointLight
    und feSpotLight zur Definition von Lichtquellen
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
```

```
      fill="black" stroke="black" />
    <circle id="auge-rechts" cx="25" cy="15" r="2"
      fill="black" stroke="black" />
    <line id="nase" x1="20" y1="18" x2="20" y2="23"
      stroke="black" stroke-width="2" />
    <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
      stroke="black" fill="none" stroke-width="2" />
  </symbol>

<!-- 2 Filter mit feDistantLight -->
  <filter id="f1">
    <feDiffuseLighting in="SourceAlpha" result="out1">
      <feDistantLight azimuth="0" elevation="45" />
    </feDiffuseLighting>
  </filter>
  <filter id="f2">
    <feDiffuseLighting in="SourceAlpha" result="out1">
      <feDistantLight azimuth="180" elevation="45" />
    </feDiffuseLighting>
    <feComposite in="SourceGraphic" in2="out1"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0" />
  </filter>
<!-- 2 Filter mit fePointLight -->
  <filter id="f3">
    <feDiffuseLighting in="SourceAlpha" result="out1">
      <fePointLight x="100" y="100" z="100" />
    </feDiffuseLighting>
    <feComposite in="SourceGraphic" in2="out1"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0" />
  </filter>
  <filter id="f4">
    <feDiffuseLighting in="SourceAlpha" result="out1">
      <fePointLight x="-100" y="-100" z="100" />
    </feDiffuseLighting>
    <feComposite in="SourceGraphic" in2="out1"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0" />
  </filter>
<!-- 2 Filter mit feSpotLight -->
  <filter id="f5">
    <feDiffuseLighting in="SourceAlpha" result="out1">
      <feSpotLight x="-100" y="-100" z="50"
        pointsAtX="350" pointsAtY="350" pointsAtZ="-10"
        specularExponent="1" />
    </feDiffuseLighting>
    <feComposite in="SourceGraphic" in2="out1"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0" />
  </filter>
  <filter id="f6">
    <feDiffuseLighting in="SourceAlpha" result="out1">
      <feSpotLight x="-100" y="-100" z="50"
        pointsAtX="350" pointsAtY="350" pointsAtZ="-10"
        specularExponent=".1"
        limitingConeAngle="13" />
    </feDiffuseLighting>
    <feComposite in="SourceGraphic" in2="out1"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0" />
  </filter>
 </defs>
```

```
<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
    filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
    filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
    filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
    filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
    filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feDistantLight
    <tspan x="145" dy="120">fePointLight</tspan>
    <tspan x="145" dy="120">feSpotLight</tspan>
    </text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Im obigen Beispiel sind 6 Beleuchtungsfilter *f1* bis *f6* durch das Element **feDiffuseLighting** definiert. Der Output von **feDiffuseLighting** wird in den Filtern *f2* bis *f6* zusätzlich durch den primitiven Filter **feComposite** mit dem Originalbild verschmolzen.

Die Filter *f1* und *f2* verwenden als Lichtquelle das Element **feDistantLight**, dabei kommt das Licht im Filter *f1* von rechts und im Filter *f2* einmal von links. Dies wurde durch die verschiedenen Werte von ***azimuth*** bewirkt. Der Winkel des Lichteinfalls wird für beide Lichtquellen durch das Attribut ***elevation*** auf 45° Grad festgelegt. Der Filter *f1* zeigt nur die Ausgabe des Beleuchtungsfilters, da **feComposite** nicht verwendet wird.

In den Filtern *f3* und *f4* wird die Lichquelle **fePointLight** verwendet. Die unterschiedlichen Werte von ***x*** und ***y*** lassen das Licht im Filter *f3* von links oben und das Licht im Filter *f4* von rechts unten in die Grafik fliessen. Der Wert *100* für das Attribut ***z*** in beiden Lichtquellen bewirkt, dass die Grafik von oben beleuchtet wird.

Die letzten beiden Beleuchtungfilter in *f5* und *f6* verwenden **feSpotLight** als Lichtquelle. Im Filter *f6* wird zusätzlich zur Positionierung der Lichtquelle und der Definition des Zielpunkts durch die Attribute ***pointsAtX***, ***pointsAtY*** und ***pointsAtZ*** der Fokus der Lichtquelle durch ***specularExponent*** und das Ausmaß des Lichtkegels durch ***limitingConeAngle*** verändert.

## 14.17 feDiffuseLighting

Der primitive Filter **feDiffuseLighting** beleuchtet ein Input-Bild mit diffusem Licht und verwendet dabei den Alpha-Kanal des Input-Bildes als Oberflächenrelief. Sie können die Relieftiefe und die Intensität des Lichts durch Attribute beeinflussen.
Die eigentliche Lichtquelle bzw. die Position dieser Lichtquelle wird durch eins von drei unterschiedlichen Kind-Elementen **feDistantLight**, **fePointLight** oder **feSpotLight** festgelegt (siehe letztes Unterkapitel). Dabei können Sie genau eine Lichtquelle, d.h. genau ein Kind-Element innerhalb des **feDiffuseLighting**-Containers verwenden. Dieses gilt auch für den Beleuchtungsfilter **feSpecularLighting** (siehe nächstes Unterkapitel).
Durch das Attribut ***surfaceScale*** können Sie die Relieftiefe verändern, d.h. Sie können festlegen, dass die Unterschiede der Alphawerte verstärkt (Werte größer 1) oder vermindert werden (Werte kleiner 1) und das Relief so eine ausgeprägtere bzw. flachere Struktur erhält. Voreingestellter Wert für dieses Attribut ist *1*.
Die Intensität des Lichts wird durch das Attribut ***diffuseConstant*** festgelegt. Eigentlich bestimmen Sie hier eine Konstante, die bei der Berechung der neuen Farbwerte nach dem Phong-Lichtmodell für das Input-Bild verwendet wird. Gültiger Wert ist eine Ganzzahl größer 0. Voreingestellter Wert ist *1*.
**feDiffuseLighting** kann außerdem das Attribut ***lighting-color*** zugeordnet werden. Dieses Attribut erwartet eine Farbangabe als Wert, und bestimmt die Farbe des Lichts. Voreingestellter Wert ist *white*.
Um den Output eines Beleuchtungsfilters mit dem Originalbild zu verschmelzen (was notwendig ist), müssen Sie den primitiven Filter **feComposite** mit dem ***operator***-Wert *arithmetic* verwenden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feDiffuseLighting</title>
  <desc>
    Beispiele für die Verwendung von feDiffuseLighting
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feDiffuseLighting -->
    <filter id="f1">
      <feDiffuseLighting in="SourceAlpha"
        result="out1">
        <feDistantLight azimuth="225" elevation="45" />
      </feDiffuseLighting>
      <feComposite in="SourceGraphic" in2="out1"
        operator="arithmetic"
        k1="0" k2="1" k3="1" k4="0" />
    </filter>
```

```
<filter id="f2">
  <feDiffuseLighting in="SourceAlpha"
    result="out1">
    <feDistantLight azimuth="225" elevation="5" />
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="out1"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0" />
</filter>
<filter id="f3">
  <feGaussianBlur in="SourceAlpha"
    stdDeviation="2"
    result="out1" />
  <feOffset in="out1"
    dx="2" dy="2"
    result="out2" />
  <feDiffuseLighting in="out1"
    result="out3">
    <feDistantLight azimuth="225" elevation="5" />
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="out3"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0"
    result="out4" />
  <feComposite in="out4" in2="out2"
    operator="over" />
</filter>
<filter id="f4">
  <feGaussianBlur in="SourceAlpha"
    stdDeviation="2"
    result="out1" />
  <feOffset in="out1"
    dx="2" dy="2"
    result="out2" />
  <feDiffuseLighting in="out1"
    surfaceScale=".5"
    diffuseConstant="3"
    result="out3">
    <feDistantLight azimuth="225" elevation="5" />
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="out3"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0"
    result="out4" />
  <feComposite in="out4" in2="out2"
    operator="over" />
</filter>
<filter id="f5">
  <feGaussianBlur in="SourceAlpha"
    stdDeviation="2"
    result="out1" />
  <feOffset in="out1"
    dx="2" dy="2"
    result="out2" />
  <feDiffuseLighting in="out1"
    surfaceScale="2"
    diffuseConstant="1"
    result="out3">
    <feDistantLight azimuth="225" elevation="5" />
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="out3"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0"
    result="out4" />
```

```
    <feComposite in="out4" in2="out2"
      operator="over" />
  </filter>
  <filter id="f6">
    <feGaussianBlur in="SourceAlpha"
      stdDeviation="2"
      result="out1" />
    <feOffset in="out1"

      dx="2" dy="2"
      result="out2" />
    <feDiffuseLighting in="out1"
      surfaceScale="2"
      diffuseConstant="1"
      lighting-color="yellow"
      result="out3">
      <feDistantLight azimuth="225" elevation="5" />
    </feDiffuseLighting>
    <feComposite in="SourceGraphic" in2="out3"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0"
      result="out4" />
    <feComposite in="out4" in2="out2"
      operator="over" />
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
   filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
   filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
   filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
   filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
   filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
   filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feDiffuseLighting</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Alle Filter *f1* bis *f6* im obigen Beispiel verwenden den primitiven Filter **feDiffuseLighting** mit einer Lichtquelle, die durch den primitven Filter **feDistantLight** definiert wird.

In den Filtern *f1* und *f2* werden keine zusätzlichen Attribute von **feDiffuseLighting** verwendet, sondern lediglich die Lichtquelle **feDistantLight** verändert. Der Winkel des Lichteinfalls wird durch das Attribut ***elevation*** vermindert.

Da zu einem Lichteinfall auch ein Schatten gehört, wird für allen folgenden Filter *f3* bis *f6* durch **feGaussianBlur** und **feOffset** ein solcher erzeugt. Selbstverständlich passend zum Einfall des Lichts ;-).

Innerhalb der Filter *f4* bis *f6* werden die unterschiedlichen Outputs durch Verwendung der Attribute ***surfaceScale*** und ***diffuseConstant*** erzeugt, die dem primitiven Filter **feDiffuseLighting** zu Veränderung der Relieftiefe und der Lichtintensität zur Verfügung stehen. Im Filter *f6* sorgt das Attribut ***lighting-color*** außerdem dafür, dass eine gelbe Lichtquelle verwendet wird (Voreinstellung ist weiß).

## 14.18 feSpecularLighting

Der primitive Filter **feSpecularLighting** beleutet ein Input-Bild mit reflektierendem Licht und verwendet dabei den Alpha-Kanal des Input-Bildes als Relief. Sie können die Relieftiefe, die Intensität und den Glanz des Lichts durch Attribute beeinflussen.
Die eigentliche Lichtquelle bzw. die Position dieser Lichtquelle wird durch eins von drei unterschiedlichen Kind-Elementen **feDistantLight**, **fePointLight** oder **feSpotLight** festgelegt (siehe vorletztes Unterkapitel). Dabei können Sie genau eine Lichtquelle, d.h. genau ein Kind-Element innerhalb des **feSpecularLighting**-Containers verwenden. Dieses gilt auch für den Beleuchtungsfilter **feDiffuseLighting** (siehe letztes Unterkapitel).
Durch das Attribut ***surfaceScale*** können Sie die Relieftiefe verändern, d.h. Sie können festlegen, dass die Unterschiede der Alphawerte verstärkt (Werte größer 1) oder vermindert werden (Werte kleiner 1) und das Relief so ausgeprägtere bzw. flachere Struktur erhält. Voreingestellter Wert für dieses Attribut ist *1*.
Die Intensität des Lichts wird durch das Attribut ***specularConstant*** festgelegt. Eigentlich bestimmen Sie hier eine Konstante, die bei der Berechung der neuen Farbwerte nach dem Phong-Lichtmodell für das Input-Bild verwendet wird. Gültiger Wert ist eine Ganzzahl größer 0. Voreingestellter Wert ist *1*.
Um den Glanz des Output-Bildes festzulegen, steht das Attribut ***specularExponent*** zur Verfügung. Akzeptierter Wert dieses Attributs ist eine Zahl zwischen 1 und 128. Voreingestellter Wert ist *1*.
**feSpecularLighting** kann außerdem das Attribut ***lighting-color*** zugeordnet werden. Dieses Attribut erwartet eine Farbangabe als Wert, und bestimmt die Farbe des Lichts. Voreingestellter Wert ist *white*.
Um den Output eines Beleuchtungsfilters mit dem Originalbild zu verschmelzen (was notwendig ist), müssen Sie den primitiven Filter **feComposite** mit dem ***operator***-Wert *arithmetic* verwenden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feSpecularLighting</title>
  <desc>
    Beispiele für die Verwendung von feSpecularLighting
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feSpecularLighting -->
    <filter id="f1">
      <feSpecularLighting in="SourceAlpha"
        result="out1">
        <feDistantLight azimuth="225" elevation="45" />
      </feSpecularLighting>
      <feComposite in="SourceGraphic" in2="out1"
        operator="arithmetic"
```

```
    k1="0" k2="1" k3="1" k4="0" />
</filter>
<filter id="f2">
  <feSpecularLighting in="SourceAlpha"
    result="out1">
    <feDistantLight azimuth="225" elevation="5" />
  </feSpecularLighting>
  <feComposite in="SourceGraphic" in2="out1"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0" />
</filter>
<filter id="f3">
  <feGaussianBlur in="SourceAlpha"
    stdDeviation="2"
    result="out1" />
  <feOffset in="out1"
    dx="2" dy="2"
    result="out2" />
  <feSpecularLighting in="out1"
    result="out3">
    <feDistantLight azimuth="225" elevation="5" />
  </feSpecularLighting>
  <feComposite in="SourceGraphic" in2="out3"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0"
    result="out4" />
  <feComposite in="out4" in2="out2"
    operator="over" />
</filter>
<filter id="f4" x="-.2" y="-.2" width="1.4" height="1.4">
  <feGaussianBlur in="SourceAlpha"
    stdDeviation="2"
    result="out1" />
  <feOffset in="out1"
    dx="2" dy="2"
    result="out2" />
  <feSpecularLighting in="out1"
    surfaceScale="15"
    result="out3">
    <feDistantLight azimuth="225" elevation="5" />
  </feSpecularLighting>
  <feComposite in="SourceGraphic" in2="out3"
    operator="arithmetic"
    k1="0" k2="1" k3="1" k4="0"
    result="out4" />
  <feComposite in="out4" in2="out2"
    operator="over" />
</filter>
<filter id="f5" x="-.2" y="-.2" width="1.4" height="1.4">
  <feGaussianBlur in="SourceAlpha"
    stdDeviation="2"
    result="out1" />
  <feOffset in="out1"
    dx="2" dy="2"
    result="out2" />
  <feSpecularLighting in="out1"
    surfaceScale="12"
    specularConstant="2"
    specularExponent="16"
    result="out3">
    <feDistantLight azimuth="225" elevation="5" />
  </feSpecularLighting>
  <feComposite in="SourceGraphic" in2="out3"
    operator="arithmetic"
```

```
      k1="0" k2="1" k3="1" k4="0"
      result="out4" />
    <feComposite in="out4" in2="out2"
      operator="over" />
  </filter>
  <filter id="f6" x="-.2" y="-.2" width="1.4" height="1.4">
    <feGaussianBlur in="SourceAlpha"
      stdDeviation="2"
      result="out1" />
    <feOffset in="out1"
      dx="2" dy="-2"
      result="out2" />
    <feSpecularLighting in="out1"
      surfaceScale="16"
      specularExponent="24"
      result="out3">
      <feDistantLight azimuth="135" elevation="5" />
    </feSpecularLighting>
    <feComposite in="SourceGraphic" in2="out3"
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0"
      result="out4" />
    <feComposite in="out4" in2="out2"
      operator="over" />
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
 <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
   filter="url(#f1)" />
 <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
   filter="url(#f2)" />
 <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
   filter="url(#f3)" />
 <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
   filter="url(#f4)" />
 <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
   filter="url(#f5)" />
 <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
   filter="url(#f6)" />

<!-- Text und Hilfslinien -->
 <text x="145" y="130">feSpecularLighting</text>
 <text x="58" y="130">Original</text>
 <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

Alle Filter *f1* bis *f6* im obigen Beispiel verwenden den primitiven Filter **feSpecularLighting** mit einer Lichtquelle, die durch den primitven Filter **feDistantLight** definiert wird.

In den Filtern *f1* und *f2* werden keine zusätzlichen Attribute von **feSpecularLighting** verwendet, sondern lediglich die Lichtquelle **feDistantLight** verändert. Der Winkel des Lichteinfalls wird durch das Attribut ***elevation*** vermindert.

Da zu einem Lichteinfall auch ein Schatten gehört, wird für alle folgenden Filter *f3* bis *f6* durch **feGaussianBlur** und **feOffset** ein solcher erzeugt.

Innerhalb der Filter *f4* bis *f6* werden die unterschiedlichen Outputs durch Verwendung der Attribute ***surfaceScale***, ***specularConstant*** und ***specularExponent*** erzeugt, die dem primitiven Filter **feDiffuseLighting** zu Veränderung der Relieftiefe, der Lichtintensität und dem Glanz des Lichts zur Verfügung stehen. Im Filter *f6* wird außerdem der Einfall der Lichtquelle und der Schattewurf verändert.

## 14.19 feFlood

Der primitive Filter **feFlood** füllt das Zielrechteck mit Farbe. Benötigt dabei kein Input-Bild.

Die Farbangabe nimmt dabei das Attribut ***flood-color*** entgegen. Es akzeptiert aber ebenfalls Referenzen auf Verläufe oder Muster als Wert. Voreingestellter Wert ist *black*.

Um für die gewählte Farbe eine Transparenz festzulegen, können Sie vom Attribut ***flood-opacity*** Gebrauch machen. Wie alle anderen SVG Attribute zur Einstellung von Transparenz, akzeptiert auch ***flood-opacity*** eine Zahl zwischen *0* und *1* als Wert. Voreingestellter Wert ist *1*.

Es ist sinnvoll **feFlood** in Verbindung mit anderen primitiven Filtern einzusetzen, da **feFlood** lediglich den Canvas des referenzierenden Elements mit der angegebenen Farbe und Durchsichtigkeit füllt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Der primitive Filter feFlood</title>
 <desc>
  Beispiele für die Verwendung von feFlood
 </desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana,sans-serif; font-size:16px;
        font-weight:bold;}
  ]]>
  </style>
  <symbol id="smilie">
   <desc>ein lachendes Smilie</desc>
   <circle id="gesicht" cx="20" cy="20" r="15"
    fill="yellow" stroke="black" />
   <circle id="auge-links" cx="15" cy="15" r="2"
    fill="black" stroke="black" />
   <circle id="auge-rechts" cx="25" cy="15" r="2"
    fill="black" stroke="black" />
   <line id="nase" x1="20" y1="18" x2="20" y2="23"
    stroke="black" stroke-width="2" />
   <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
    stroke="black" fill="none" stroke-width="2" />
  </symbol>

<!-- 6 Filter mit feFlood -->
  <filter id="f1">
   <feFlood flood-color="limegreen" />
  </filter>
  <filter id="f2">
   <feFlood flood-color="limegreen" flood-opacity=".5" />
  </filter>
<!-- zusätzlich mit feComposite -->
  <filter id="f3">
   <feFlood flood-color="limegreen"
    result="out1" />
   <feComposite in="SourceGraphic" in2="out1"
    operator="over" />
  </filter>
  <filter id="f4">
   <feFlood flood-color="limegreen"
    result="out1" />
   <feComposite in="SourceGraphic" in2="out1"
```

```
      operator="arithmetic"
      k1="0" k2="1" k3="1" k4="0" />
    </filter>
    <filter id="f5">
      <feFlood flood-color="limegreen"
        result="out1" />
      <feComposite in="out1" in2="SourceGraphic"
        operator="in" />
    </filter>
<!-- zusätzlich mit feComposite und feBlend -->
    <filter id="f6">
      <feFlood flood-color="limegreen"
        result="out1" />
      <feComposite in="out1" in2="SourceGraphic"
        operator="in"
        result="out2" />
      <feBlend in="SourceGraphic" in2="out2"
        mode="darken" />
    </filter>
  </defs>

<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
    filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
    filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
    filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
    filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
    filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feFlood</text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

## 14.20 feImage

Der primitive Filter **feImage** bietet die Möglichkeit eine externe Bitmap-Grafik oder ein svg-Fragment als Input-Bild für nachfolgende primitive Filter einzubinden. **feImage** füllt die Filterregion dieses primitiven Filters durch eine referenzierte Grafik.

Statt des Attributs ***in*** verwendet dieser primitive Filter das Attribut ***xlink:href*** um ein Input-Bild festzulegen. Das Input-Bild wird durch **feImage** auf die Größe des Filters bzw. des referenzierenden Elements skaliert. Das Resultat der Skalierung ist das Output-Bild dieses Filters. Gültiger Wert für ***xlink:href*** ist eine *URL*, die entweder auf eine Grafik im Format JPG, PNG oder SVG oder auf ein beliebiges SVG Fragment zeigt.

Es ist sinnvoll **feImage** in Verbindung mit anderen primitiven Filtern einzusetzen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Der primitive Filter feImage</title>
 <desc>
  Beispiele für die Verwendung von feImage
 </desc>
 <defs>
  <style type="text/css">
  <![CDATA[
   text {font-family:Verdana,sans-serif; font-size:16px;
        font-weight:bold;}
  ]]>
  </style>
  <symbol id="smilie">
   <desc>ein lachendes Smilie</desc>
   <circle id="gesicht" cx="20" cy="20" r="15"
    fill="yellow" stroke="black" />
   <circle id="auge-links" cx="15" cy="15" r="2"
    fill="black" stroke="black" />
   <circle id="auge-rechts" cx="25" cy="15" r="2"
    fill="black" stroke="black" />
   <line id="nase" x1="20" y1="18" x2="20" y2="23"
    stroke="black" stroke-width="2" />
   <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
    stroke="black" fill="none" stroke-width="2" />
  </symbol>

<!-- 2 Filter mit feImage -->
  <filter id="f1" x=".075" y=".075" width=".95" height=".95">
   <feImage xlink:href="strand.jpg"
    result="out1" />
  </filter>
<!-- zusätzlich mit feComposite und feBlend -->
  <filter id="f2">
   <feImage xlink:href="strand.jpg"
    result="out1" />
   <feComposite in="out1" in2="SourceGraphic"
    operator="in"
    result="out2" />
   <feBlend in="SourceGraphic" in2="out2"
    mode="darken" />
  </filter>
 </defs>

<!-- die Instanzen des Symbols "smilie" -->
 <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
```

```
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(5.4)"
    filter="url(#f2)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feImage</text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

## 14.21 feTile

Der primitive Filter **feTile** füllt das Zielrechteck mit einem Muster. Als Muster wird das Input-Bild verwendet.
Damit das Input-Bild von **feTile** überhaupt als Muster verwendet werden kann, muß es kleiner sein als das Zielrechteck, d.h der Canvas des referenzierenden Elements. Aus diesem Grund ist das entsprechende Input-Bild meist das Output-Bild eines vorangegangenen primitiven Filters, der durch die Attribute ***x***, ***y***, ***width*** und ***height*** in seinen Ausmassen beschränkt ist.
In den Filtern des folgenden Beispiels wird der, in seinen Ausmassen beschränkte primitive Filter **feGaussianBlur** vorangestellt, um ein Input-Bild für **feTile** zu erzeugen. Damit **feGaussianBlur** ansonsten keinen Effekt zeigt, wird seinem Attribut ***stdDeviation*** in einigen Filtern der Wert *0* zugeordnet.
Es ist daher besonders sinnvoll **feTile** in Verbindung mit anderen primitiven Filtern einzusetzen ;-).

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feTile</title>
  <desc>
    Beispiele für die Verwendung von feTile
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
           font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feTile
     und feGaussianBlur ohne Weichzeichnung -->
    <filter id="f1">
      <feGaussianBlur in="SourceGraphic"
        x="0" y="0" width="20" height="20"
        stdDeviation="0"
        result="out1" />
      <feTile in="out1"
        result="out2" />
    </filter>
    <filter id="f2">
      <feGaussianBlur in="SourceGraphic"
        x="20" y="0" width="20" height="20"
        stdDeviation="0"
        result="out1" />
      <feTile in="out1"
        result="out2" />
    </filter>
    <filter id="f3">
```

```
      <feGaussianBlur in="SourceGraphic"
        x="11" y="6" width="18" height="28"
        stdDeviation=".5"
        result="out1" />
      <feTile in="out1"
        result="out2" />
    </filter>
    <filter id="f4">
      <feGaussianBlur in="SourceGraphic"
        x="6" y="11" width="28" height="18"
        stdDeviation=".5"
        result="out1" />
      <feTile in="out1"
        result="out2" />
    </filter>
    <filter id="f5">
      <feGaussianBlur in="SourceGraphic"
        x="16" y="16" width="8" height="8"
        stdDeviation="0"
        result="out1" />
      <feTile in="out1"
        result="out2" />
    </filter>
<!-- zusätzlich mit feGaussianBlur,  feOffset,
     feSpecularLighting, feComposite,
     und feMerge inclusiv Kind-Elemente -->
    <filter id="f6">
      <feGaussianBlur in="SourceGraphic"
        x="16" y="16" width="8" height="8"
        stdDeviation="2"
        result="out1" />
      <feTile in="out1"
        result="out2" />
      <feGaussianBlur in="SourceAlpha"
        stdDeviation="2"
        result="out3" />
      <feOffset in="out3"
        dx="2" dy="2"
        result="out4" />
      <feSpecularLighting in="out3"
        surfaceScale="2"
        specularExponent="16"
        result="out5">
        <feSpotLight x="-100" y="-100" z="100" />
      </feSpecularLighting>
      <feComposite in="SourceGraphic" in2="out5"
        operator="arithmetic"
        k1="0" k2="1" k3="1" k4="0"
        result="out6" />
      <feMerge>
        <feMergeNode in="out2" />
        <feMergeNode in="out4" />
        <feMergeNode in="out6" />
      </feMerge>
    </filter>
  </defs>

<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
    filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
```

```
    filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
    filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
    filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
    filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feTile</text>
  <text x="58" y="130">Original</text>
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

## 14.22 feTurbulence

Der primitive Filter **feTurbulence** erzeugt eine Turbulenz oder ein Fraktal im Zielrechteck. Benötigt dabei kein Input-Bild.
Wenn Sie den primitiven Filter ***feTurbulence*** verwenden, wird das Output-Bild durch die Perlin-Turbulenz-Funktion erzeugt. Diese berechnet turbulente Strukturen, wie z.B. Wolken oder Marmorierungen, durch welche das Zielrechteck (die Filterregion) komplett ausgefüllt wird. Daher hat das Input-Bild keinen Einfluss auf auf den Filter.
Das Attribut ***type*** bietet Ihnen die Möglichkeit durch den (voreingestellten) Wert *turbulence* eine Turbulenz oder durch den Wert *fractalNoise* ein Fraktal erzeugen zu lassen. Wenn Sie dieses Attribut nicht verwenden wird eine Turbulenz erzeugt.
Mit ***baseFrequency*** legen Sie einen oder zwei, durch Komma getrennte, Frequenz-Parameter fest, die von der Perlin-Turbulenz-Funktion bei der Berechnung verwendet werden. Wenn Sie zwei Zahlen als Wert verwenden, legt die erste Zahl die Frequenz für die x-Richtung und die zweite Zahl die Frequenz für die y-Richtung fest. Wenn Sie nur eine Zahl als Wert verwenden, wird diese sowohl für die horizontale als auch für die vertikale Frequenz verwendet. Voreingestellter Wert für ***baseFrequency*** ist *0*.
Durch das Attribut ***numOctaves*** können Sie die erzeugten Strukturen verfeinern. Das Attribut akzeptiert eine Ganzzahl als Wert, wobei ein höherer Wert eine feinere Strukturierung bewirkt. Voreingestellter Wert ist *1*.
Die Perlin-Turbulenz-Funktion verwendet bei der Berechnung der Turbulenz oder des Fraktals eine Zufallszahl, die durch einen Zufallsgenerator erzeugt wird. Die Startzahl dieses Zufallsgenerators läßt sich mit dem Attribut ***seed*** verändern. Voreingestellter Wert für dieses Attribut ist *0*, d.h. die Startzahl des Zufallsgenerators ist standardmäßig 0. ***seed*** erwartet als Wert eine beliebige Zahlangabe.
Das Attribut ***stitchTiles*** kann durch den Wert *stitch* geglättete Übergänge an den Rändern des erzeugten Output-Bildes erzeugen. Falls das Output-Bild als Muster (z.B. von **feTile**) verwendet werden soll, ist dies eine sinnvolle Einstellung. Der zweite mögliche Wert für ***stitchTiles*** und zugleich auch die Voreinstellung für dieses Attribut ist *noStitch*.
Es ist sinnvoll **feTurbulence** in Verbindung mit anderen primitiven Filtern einzusetzen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="380px" height="370px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Der primitive Filter feTurbulence</title>
  <desc>
    Beispiele für die Verwendung von feTurbulence
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
    ]]>
    </style>
    <symbol id="smilie">
      <desc>ein lachendes Smilie</desc>
      <circle id="gesicht" cx="20" cy="20" r="15"
        fill="yellow" stroke="black" />
      <circle id="auge-links" cx="15" cy="15" r="2"
        fill="black" stroke="black" />
      <circle id="auge-rechts" cx="25" cy="15" r="2"
        fill="black" stroke="black" />
      <line id="nase" x1="20" y1="18" x2="20" y2="23"
        stroke="black" stroke-width="2" />
      <path id="mund" d="M 13 26 A 5 3 0 0 0 27 26"
        stroke="black" fill="none" stroke-width="2" />
    </symbol>

<!-- 6 Filter mit feTurbulence -->
```

```
    <filter id="f1">
      <feTurbulence type="turbulence"
        baseFrequency=".2" />
    </filter>
    <filter id="f2">
      <feTurbulence type="fractalNoise"
        baseFrequency=".2" />
    </filter>
    <filter id="f3">

      <feTurbulence type="turbulence"
        baseFrequency=".2,.5"
        numOctaves="5" />
    </filter>
    <filter id="f4">
      <feTurbulence type="fractalNoise"
        baseFrequency=".2,.5"
        numOctaves="5" />
    </filter>
<!-- zusätzlich mit feBlend -->
    <filter id="f5">
      <feTurbulence
        baseFrequency=".5,.1"
        numOctaves="2"
        seed="3"
        result="out1" />
      <feBlend in="SourceGraphic" in2="out1"
        mode="darken"
        result="out2" />
    </filter>
<!-- zusätzlich mit feComposite und feBlend -->
    <filter id="f6">
      <feTurbulence
        baseFrequency=".5,.1"
        numOctaves="2"
        seed="3"
        result="out1" />
      <feComposite in="out1" in2="SourceGraphic"
        operator="in"
        result="out2" />
      <feBlend in="SourceGraphic" in2="out2"
        mode="darken"
        result="out3" />
    </filter>
  </defs>

<!-- die Instanzen des Symbols "smilie" -->
  <use xlink:href="#smilie" transform="translate(20,10) scale(2.7)" />
  <use xlink:href="#smilie" transform="translate(150,10) scale(2.7)"
    filter="url(#f1)" />
  <use xlink:href="#smilie" transform="translate(250,10) scale(2.7)"
    filter="url(#f2)" />
  <use xlink:href="#smilie" transform="translate(150,130) scale(2.7)"
    filter="url(#f3)" />
  <use xlink:href="#smilie" transform="translate(250,130) scale(2.7)"
    filter="url(#f4)" />
  <use xlink:href="#smilie" transform="translate(150,250) scale(2.7)"
    filter="url(#f5)" />
  <use xlink:href="#smilie" transform="translate(250,250) scale(2.7)"
    filter="url(#f6)" />

<!-- Text und Hilfslinien -->
  <text x="145" y="130">feTurbulence</text>
  <text x="58" y="130">Original</text>
```

```
  <line x1="137" y1="40" x2="137" y2="330" stroke="black" />
</svg>
```

# 15 Masken

Mit Clipping und Masking bezeichnet man die Möglichkeit, eine Schablone mit transparenten (oder halbtransparenten) Aussparungen zu erzeugen, die dann auf ein SVG Objekt gelegt werden kann. Daraus resultierend werden nur die Bereiche des SVG Objekts angezeigt, die unter den transparenten (oder halbtransprarenten) Bereichen der Schablone liegen.

SVG unterstützt folgende Clipping/Masking Eigenschaften:

- **clipping paths** - Clip-Pfade verwenden eine beliebige Kombination aus Pfaden, Texten oder geometrischen Grundformen um eine transparente Region innerhalb eines rechteckigen Canvas zu bestimmen (die Schablone). Wenn ein solcher Clip-Pfad von einem Objekt referenziert wird, dann wird nur der Bereich des Objekts dargestellt, der unter der transparenten Region des Clip-Pfades liegt.
- **masks** - Masken sind Clip-Pfaden sehr ähnlich. Der Unterschied besteht darin, dass in Masken auch halbtransparente Elemente innerhalb der Kombination (der Schablone) verwendet werden können. Wenn eine solche Maske von einem Objekt referenziert wird, dann wird der Bereich der Objekts dargestellt, der unter den transparenten und halbtransparenten Pixeln der Maske liegt.

Der grundlegende Unterschied zwischen Clipping und Masking besteht also darin, dass die Kombination beim Clipping nur aus transparenten Pixeln und/oder nicht transparenten Pixeln besteht, wohingegen beim Masking die Kombination als Bild angesehen wird, bei der jedes Pixel einen eigenen Transparenzwert besitzt. Dabei kann dieser Transparenzwert alle Werte von durchsichtig bis undurchsichtig (von 0 bis 1) erhalten.

## 15.1 Clipping - das clipPath-Element

Clipping wird in SVG mit dem Container-Element **clipPath** im **defs**-Bereich der Grafik realisiert. Das Element **clipPath** darf eine beliebige Anzahl von **path**-Elementen, **text**-Elementen und geometrischen Grundformen (**rect**, **circle**, **ellipse**, **polyline**, **polygon**) als Kind-Elemente enthalten, deren gemeinsame Grundfläche bzw. Silhouette, den transparenten Bereich der Schablone darstellt.

Durch das Attribut ***clipPathUnits*** im **clipPath**-Element wird festgelegt, ob die Koordinaten und Längenangaben innerhalb der Kind-Elemente von **clipPath** die Werte des augenblicklichen Koordinatensystems repräsentieren (Voreinstellung *userSpaceOnUse*) oder ob relative Werte verwendet werden müssen (*boundingBox*).

Innerhalb der Kind-Elemente von **clipPath** können Sie zusätzlich das Attribut ***clip-rule*** verwenden. Dieses Attribut akzeptiert die Werte *nonzero* (Voreinstellung) und *evenodd*. Bei Verwendung von *evenodd* kehren sich der innere und der äußere Bereich des Elements um, d.h. der zuvor transparente Bereich des Elements, ist jetzt nicht-transparent und umgekehrt. Leider wird dieses Attribut vom SVG Viewer 3.0 noch nicht unterstützt.

Die durch **clipPath** erzeugte Schablone, welche durch das Attribut **id** eine eindeutigen Bezeichnung erhalten muß, wird von Objekten durch das Attribut ***clip-path*** referenziert. Dabei erhält ***clip-path*** als Wert die URI der Schablone.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="360px" height="510px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Das clipPath-Element</title>
  <desc>
    Dieses Beispiel für Clipping beinhaltet 2 Clip-Pfade,
    von denen einer animiert wird.
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:36px;
           font-weight:bold; fill:white;}
      rect {fill:none; stroke:white; stroke-dasharray:5,2;}
    ]]>
    </style>

<!-- zwei Clip-Pfade: cp1 und cp2 -->
    <clipPath id="cp1">
      <circle cx="80" cy="110" r="50" />
      <circle cx="180" cy="130" r="80" />
      <circle cx="280" cy="150" r="50" />
    </clipPath>
    <clipPath id="cp2">
      <circle id="clippie" cx="-10" cy="380" r="30" />
    </clipPath>
  </defs>

<!-- schwarzer Hintergrund -->
  <rect x="0" y="0" width="360" height="510"
    style="fill:black; stroke:none;" />

<!-- die Clip-Pfade werden von zwei image-Elementen referenziert -->
  <image x="20" y="10" width="320" height="240" xlink:href="raupen.jpg"
    clip-path="url(#cp1)" />
  <image x="20" y="260" width="320" height="240" xlink:href="raupen.jpg"
    clip-path="url(#cp2)" />
```

```
<!-- der Clip-Pfad cp2 wird animiert -->
  <animate xlink:href="#clippie"
    attributeType="XML" attributeName="cx"
    begin="0" dur="12"
    values="-10;380;-10"
    repeatDur="indefinite" />
  <animate xlink:href="#clippie"
    attributeType="XML" attributeName="r"
    begin="0" dur="12"
    values="10;110;10;110;30"
    repeatDur="indefinite" />

<!-- Hilfslinien: Umrisse der eingebundenen Grafiken -->
  <rect x="20" y="10" width="320" height="240" />
  <rect x="20" y="260" width="320" height="240" />
  <text x="95" y="265">Clipping</text>
</svg>
```

Im obigen Beispiel werden zwei Clip-Pfade *cp1* und *cp2* definiert. *cp1* enthält drei Kreise, *cp2* einen Kreis als Kind-Element. Auf diese Weise wird der sichtbare Bereich (der Ausschnitt) für das referenzierende Element festgelegt.

Diese beiden Clip-Pfade werden jeweils von einem **image**-Element durch das Atttribut ***clip-path*** referenziert. So ist von der Grafik, die vom ersten **image**-Element eingebunden wird lediglich der Ausschnitt sichtbar, der durch die drei Kreise im Clip-Pfad *cp1* festgelegt wurde. Beachten Sie, dass den Kind-Elementen von **clipPath** keine Eigenschaften für Füllung oder Randlinie zugeordnet wird, da diese hier keine Bedeutung haben.

Im zweiten Clip-Pfad *cp2* wird der Wert der x-Koordinate und der Wert des Radius des Kind-Elements **circle** zusätzlich durch zwei **animate**-Elemente verändert.

## 15.2 Masking - das mask-Element

Ein Masking wird durch die Verwendung des Container-Elements **mask** innerhalb des **defs**-Bereich einer SVG Grafik definiert. Das Element **mask** darf, wie auch schon das Element **clip-path**, eine beliebige Anzahl von **path**-Elementen, **text**-Elementen und geometrischen Grundformen (**rect**, **circle**, **ellipse**, **polyline**, **polygon**) als Kind-Elemente enthalten.

Im Unterschied zum **clipPath**-Element, dürfen die Kind-Elemente von **mask** durchaus mit Eigenschaften zur Darstellung definiert werden (z.B. durch Farbverläufe oder Filtereffekte). Auf diese Weise können Sie halbtransparente Schablonen erzeugen. Beachten Sie: Wenn Sie die Elemente der Maske mit der Farbe weiß füllen, erhalten Sie volle Transparenz, wenn Sie als Füllfarbe schwarz verwenden erhalten Sie keine Transparenz.

Dem mask-Element stehen die Attribute ***x***, ***y***, ***width*** und ***height*** zur Verfügung. Damit können Sie die Ausmasse der Maske festlegen. Voreinstellung für ***x*** und ***y*** ist jeweils *-10%*, Voreistellung für ***width*** und ***height*** jeweils *120%*.

Das Attribut ***maskUnits*** regelt wie die Werte von ***x***, ***y***, ***width*** und ***height*** interpretiert werden. In der Voreinstellung *boundingBox* werden die Angaben relativ betrachtet, d.h. es sind relative Werte für ***x***, ***y***, ***width*** und ***height*** zu verwenden, bei Verwendung von *userSpaceOnUse* repräsentieren die Werte das momentan gültige Koordinatensystem.

Durch das Attribut ***maskContentUnits*** im **mask**-Element bestimmen Sie, ob die Koordinaten und Längenangaben innerhalb der Kind-Elemente von **mask** die Werte des augenblicklichen Koordinatensystems repräsentieren (Voreinstellung *userSpaceOnUse*) oder ob relative Werte verwendet werden müssen (*boundingBox*).

Die durch **mask** erzeugte Maske (Schablone), welche durch das Attribut **id** eine eindeutigen Bezeichnung erhalten muß, wird von Objekten durch das Attribut ***mask*** referenziert. Dabei erhält ***mask*** als Wert die URI der Schablone.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="360px" height="510px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Das mask-Element</title>
  <desc>
    Dieses Beispiel für Masking beinhaltet 2 Masken
    mit semi-transparenten Füllungen.
  </desc>
  <defs>
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:36px;
            font-weight:bold; fill:white;}
      rect {fill:none; stroke:white; stroke-dasharray:5,2;}
    ]]>
    </style>

<!-- Verlauf für Maske m1 -->
    <linearGradient id="verlauf"
      x1="0" y1="0" x2="1" y2="0">
      <stop offset="0" stop-color="white" />
      <stop offset=".8" stop-color="black" />
    </linearGradient>
<!-- Weichzeichnungsfilter für Maske m2 -->
    <filter id="filter" x="-.3" y="-.3" width="1.6" height="1.6">
      <feGaussianBlur stdDeviation="15" />
    </filter>

<!-- zwei Masken: m1 und m2 -->
```

```
    <mask id="m1">
      <g fill="url(#verlauf)">
        <circle cx="80" cy="110" r="50" />
        <circle cx="180" cy="130" r="80" />
        <circle cx="280" cy="150" r="50" />
      </g>
    </mask>
    <mask id="m2">
      <circle cx="180" cy="380" r="90"
        style="fill:white;" filter="url(#filter)" />
    </mask>
  </defs>

<!-- schwarzer Hintergrund -->
  <rect x="0" y="0" width="360" height="510"
    style="fill:black; stroke:none;" />

<!-- die Masken werden von zwei image-Elementen referenziert -->
  <image x="20" y="10" width="320" height="240" xlink:href="raupen.jpg"
    mask="url(#m1)" />
  <image x="20" y="260" width="320" height="240" xlink:href="raupen.jpg"
    mask="url(#m2)" />

<!-- Hilfslinien: Umrisse der eingebundenen Grafiken -->
  <rect x="20" y="10" width="320" height="240" />
  <rect x="20" y="260" width="320" height="240" />
  <text x="95" y="265">Masking</text>
</svg>
```

Im obigen Beispiel werden im **defs**-Bereich durch das **mask**-Element zwei Masken *m1* und *m2* definiert. *m1* enthält drei Kreise, *m2* einen Kreis als Kind-Element. Auf diese Weise wird der sichtbaren Bereich (der Ausschnitt) für das referenzierenden Element festgelegt.

Diese beide Masken werden jeweils von einem **image**-Element durch das Atttribut ***mask*** referenziert. So ist von der Grafik, die vom ersten **image**-Element eingebunden wird lediglich der Ausschnitt sichtbar, der durch die drei Kreise der Maske *m1* festgelegt wurde. Dabei wurde jedem der drei Kreise als Füllung ein linearer Farbverlauf von weiß nach schwarz zugeordnet. Dadurch nimmt die Transparenz innerhalb der Kreise von links nach rechts ab.

In der zweiten Maske *m2* wird dem weiß gefüllten **circle**-Element ein Weichzeichnungsfilter zugeordnet. Dadurch werden die Ränder der Maske semi-transparent.

# 16 Scripting

Sie können in SVG Dokumenten Skriptsprachen wie ECMAScript, JavaScript, JScript oder VBScript verwenden, um Dynamik und/oder Interaktivität zu verwirklichen.

ECMAScipt ist ein Standard der ECMA (European Computer Manufactures Association), der es erfolgreich verstanden hat, die verschiedene Skriptsprachen unter einen Hut zu bringen. So sind JavaScript 1.5 und JScript 5.5 nach Angaben der Hersteller vollständig kompatibel zu ECMAScript.

SVG Dokumente verwenden standardmäßig ECMAScript um die Funktionalität von Event-Attributen (wie z.B. ***onclick***, ***onmouseover***, etc.) zu gewährleisten. Dies wird durch den voreingestellen Wert *text/ecmascript* des Attributs ***contentScriptType*** bewirkt, das im **svg**-Wurzel-Element gesetzt ist.

Für die Skriptbeispiele im Verlauf dieses Kapitels wird hauptsächlich JavaScript verwendet. Anmerkung: Dieses Kapitel ist keine Anleitung für JavaScript, sondern eine Erläuterung, wie JavaScript in SVG Dokumenten verwendet werden kann. Dabei werden Sie wichtige JavaScript Objekt-Methoden kennenlernen, die den Zugriff und die Manipulation von SVG Inhalten erlauben. Wenn Sie weitere Informationen zu JavaScript benötigen, empfehle ich SelfHTML 8.0.

## 16.1 Übersicht Event-Attribute

Event-Attribute werden verwendet, um auf Ereignisse reagieren zu können. Wenn ein bestimmtes Ereignis (Event) eintritt, wie z.B. der Klick auf ein Element, können Sie als Reaktion auf dieses Ereignis eine JavaScript-Funktion aufrufen.

SVG unterstützt folgende Event-Attribute:

- ***onclick*** - Klick auf das Element
- ***onactivate*** - Aktivierung des Elements durch beliebigen Zeiger (Mausklick, Taste, ..)
- ***onmousedown*** - Drücken der linken Maustaste
- ***onmouseup*** - Loslassen der linken Maustaste
- ***onmouseover*** - Den Bereich des Elements mit dem Mauszeiger betreten
- ***onmousemove*** - Bewegen der Maus über dem Bereich des Elements
- ***onmouseout*** - Den Bereich des Elements mit dem Mauszeiger verlassen
- ***onkeypress***/***onkeydown*** - Taste gedrückt
- ***onkeyup*** - Taste losgelassen
- ***onfocusin*** - Das Element bekommt den Fokus
- ***onfocusout*** - Das Element verliert den Fokus

## 16.2 Skriptsprachen einbinden - das script-Element

Skriptanweisungen werden innerhalb des **defs**-Container mit dem **script**-Element eingeleitet (ähnlich zu HTML). Mit dem Attribut ***type*** muß der MIME-Typ der verwendeten Skriptsprache angegeben werden. Die Skriptsprache JavaScript wird durch den MIME-Type *text/javascript* eindeutig identifiziert.

Vor der eigentlichen Notierung der Skriptanweisungen muß (genau wie bei der Definiton von Style-Vorschriften mit dem **style**-Element) ein **CDATA**-Bereich definiert werden. Laut XML Spezifikation werden Daten innerhalb eines CDATA-Bereichs von der interpretierenden Anwendungssoftware nicht als XML-Code interpretiert.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="130px" height="62x"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das script-Element</title>
  <desc>
    Javascript und SVG. Skriptanweisungen durch das
    script-Element in SVG Dokumente einbinden.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function meldung(){
        alert("Javascript in SVG");
      }
    ]]>
    </script>

<!-- der Style Bereich -->
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
      rect {fill:#33cc33; stroke:black;}
    ]]>
    </style>
<!-- der button -->
    <g id="button">
      <rect x="0" y="0" width="100" height="30" rx="5" ry="5" />
      <text x="18" y="20">onclick</text>
    </g>
  </defs>

<!-- button-Instanz mit Event-Attribut onclick;
     ruft bei Klick die JavaSriptfunktion meldung() auf -->
  <use xlink:href="#button" x="15" y="15" onclick="meldung();" />
</svg>
```

Im obigen Beispiel wird im **defs**-Bereich ein **script**-Element definiert, das eine JavaScript-Funktion des Namens **meldung()** beinhaltet. Diese Funktion besteht nur aus einer Anweisung: der Ausgabe eines sogenannten ***alert***-Fensters mit etwas Text.

Die durch das **use**-Element eingebundene Instanz des SVG Objekts button, ruft bei einem Klick-Ereignis diese JavaScript-Funktion auf - bewirkt durch das Event-Attribut ***onclick***, dem als Wert der Aufruf der Funktion zugeordnet wird.

Sie können JavaScript-Anweisungen auch in eine externe Datei mit der Dateinamenerweiterung .js auslagern. Diese Datei wird dann durch das Attribut ***xlink:href*** innerhalb des **script**-Elements referenziert.

Das folgende Beispiel ist mit dem vorangegangem identisch. Nur wird hier die Funktion **meldung()** in einer externen Datei namens meldung.js definiert. Diese Datei wird durch das **script**-Element in das SVG Dokument eingebunden, d.h. die Funktion steht innerhalb des Dokumentes zur Verfügung.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="130px" height="62x"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das script-Element</title>
  <desc>
    Javascript und SVG. Skriptanweisungen durch das
    script-Element in SVG Dokumente einbinden.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript" xlink:href="meldung.js" />

<!-- der Style Bereich -->
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:16px;
            font-weight:bold;}
      rect {fill:#33cc33; stroke:black;}
    ]]>
    </style>
<!-- der button -->
    <g id="button">
      <rect x="0" y="0" width="100" height="30" rx="5" ry="5" />
      <text x="18" y="20">onclick</text>
    </g>
  </defs>

<!-- button-Instanz mit Event-Attribut onclick;
     ruft bei Klick die JavaSriptfunktion meldung()
     aus der Datei meldung.js auf -->
  <use xlink:href="#button" x="15" y="15" onclick="meldung();" />
</svg>
```

## 16.3 SVG Inhalte verfügbar machen

Um mit Hilfe einer Skriptsprache auf die Inhalte einer SVG Grafik zugreifen zu können, müssen Sie zuerst dafür sorgen, dass Sie das **Document Object Model (DOM)** für das Dokument verwenden können. Das DOM stellt eine Schnittstelle (API) für Skriptsprachen dar.

Innerhalb der Baustruktur eines SVG Dokuments wird jede Verzweigung als Knoten bezeichnet. Die wichtigsten dieser Knoten (Verzweigungen) repräsentieren Elemente, Attribute und Text. Es gibt allerdings weitere Knoten, wie z.B. Kommentar-Knoten, die hier jedoch vernachlässigt werden.

Das DOM eines SVG Dokuments besteht aus einer Baumstruktur, die zu der des SVG Dokuments identisch ist. In dieser Baumstruktur stehen die die Knoten des SVG Dokuments, als Objekte (im Sinne einer objektorientierten Programmiersprache) für die Skriptsprache zur Verfügung. Die Objekte des DOM repräsentieren also die Knoten des SVG Dokuments, d.h. über diese Objekte kann die Skriptsprache auf die Knoten des SVG Dokuments zugreifen.

Ein lesender wie ein schreibender Zugriff auf die Knoten des SVG Dokuments ist über die Objekte des DOM möglich, da jedes einzelne Objekt spezifische Eigenschaften und Methoden besitzt:

- Eigenschaften, die ein Objekt näher beschreiben. Dies sind in der Regel die gleichen Eigenschaften, die auch den entsprechenden SVG Elementen zugeordnet werden können. Diese Eigenschaften können verändert werden.
- Methoden, welche den Zugriff auf die Knoten des SVG Dokuments ermöglichen. Auf diese Art sind die Eigenschaften und Inhalte der Knoten manipulierbar.

Achtung, diese Beschreibung entspricht nicht mehr dem aktuellen Stand und muss überarbeitet werden:

Um auf die Objekte des DOM zugreifen zu können, müssen Sie immer zuerst ein benanntes Objekt für das SVG Dokument (das document-Objekt), in Form einer Variablen, erzeugen:

**var document = evt.getTarget().getOwnerDocument();**

Das Objekt *evt* repräsentiert das eingetretene Ereignis. Die Methode ***getTarget()*** greift auf den Knoten zu, der dieses Ereignis ausgelöst hat, und liefert ihn zurück. Die Methode ***getOwnerDocument()*** liefert die Wurzel des auslösenden Knotens zurück, also das Objekt, welches das Dokument repräsentiert. Dieses document-Objekt wird letztendlich der Variablen document zugeordnet.

Nun haben Sie über das Objekt mit dem Namen *document* Zugriff auf die gesamte Baumstruktur des DOM bzw. des SVG Dokuments. Anstelle des Namens document könnten Sie auch einen beliebiegen anderern Variablennamen für das document-Objekt verwenden, wie z.B. svgdoc oder jupp :-).

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="130px" height="62px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>das script-Element</title>
  <desc>
    Javascript und SVG. Skriptanweisungen durch das
    script-Element in SVG Dokumente einbinden.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function meldung2(){
        var b1x = document.getElementById('b1').getAttribute('x');
        var b1y = document.getElementById('b1').getAttribute('y');
        alert("Die x,y Koordinate der Buttoninstanz:\n\t("
              + b1x
              + ","
              + b1y
              + ")");
```

```
    }
   ]]>
   </script>

<!-- der Style Bereich -->
   <style type="text/css">
   <![CDATA[
     text {font-family:Verdana,sans-serif; font-size:16px;
          font-weight:bold; fill:black;}
     rect {fill:#33cc33; stroke:black;}
   ]]>
   </style>
<!-- der button -->
   <g id="button">
     <rect x="0" y="0" width="100" height="30" rx="5" ry="5" />
     <text x="18" y="20">onclick</text>
   </g>
 </defs>

<!-- button-Instanz mit Event-Attribut onclick;
     ruft bei Klick die JavaSriptfunktion meldung2() auf -->
 <use id="b1" xlink:href="#button" x="15" y="15"
   onclick="meldung2();" />
</svg>
```

Im obigen Beispiel wird innerhalb der JavaScript-Funktion **meldung2()** durch die Variable document das *document*-Objekt erzeugt. Über das so erzeugte Objekt *document* haben Sie nun Zugriff auf die gesamte Baumstruktur des DOM und dadurch auf alle Knoten des SVG Dokuments.

Die Funktion **meldung2()** verwendet die Methoden ***getElementById()*** und ***getAttribute()***, die in den folgenden Kapiteln erläutert werden, um jeweils den Wert des ***x***-Attributes bzw. des ***y***-Attributes innerhalb des **use**-Elements in den Variablen b1x und b1y zu speichern. Der Inhalt der Variablen, also die x,y-Koordinate der Button-Instanz, wird dann mit etwas Text in einem ***alert***-Fenster ausgegeben.

Die Funktion **meldung2()** startet auch in diesem Beispiel, wenn der Butten angeklickt wird.

## 16.4 Methoden für den Zugriff auf Elemente

Um auf Elementknoten eines SVG Dokuments zuzugreifen können Sie folgende Methoden des document-Objekts verwenden:

- ***getElementById()*** Greift auf das Element mit der entsprechenden ID zu. Diesem Element muß daher mit Hilfe des ***id***-Attributs eine eindeutige *ID* zugeordnet sein, die der Methode als Parameter übergeben wird. Beachten Sie: Wenn Sie, wie in diesem Fall, einer Methode Zeichenketten als Parameter übergeben, müssen Sie diese in einfache oder doppelte Anführungszeichen setzen.
- ***getElementsByTagName()*** Greift auf alle Elemente des entsprechenden Elementnamens zu und liefert eine Liste dieser Elemente zurück. Der Elementname wird als Parameter übergeben. Die Länge dieser Liste, d.h. die Anzahl der gefundenen Elemente kann durch die Methode ***getLength()*** ermittelt werden, die auf eine Element-Liste angewendet, eine Ganzzahl zurückgibt. Auf einzelne Elemente dieser Liste können Sie mit Hilfe der Methode ***item()*** zugreifen. Diese Methode wird ebenfalls auf die Element-Liste angewendet und erwartet als Parameter eine Ganzahl, welche die Position des Elements innerhalb der Liste angibt. Dabei beginnt die Nummerierung innerhalb der Liste bei 0.
- ***documentElement*** Enthält das svg-Element, also das Wurzelelement der Grafik. Dieses Attribut ist sehr nützlich, wenn Sie Knoten oder Elemente manipulieren wollen.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="140px" height="62px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>
    Javascript: Methoden für den Zugriff auf SVG Elemente.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function meldung3(){
        var but = document.getElementById('button');
        var ubut = document.getElementById('usebut');
        var rects = document.getElementsByTagName('rect');
        var zrects = document.getElementsByTagName('rect').getLength();
        var rect = document.getElementsByTagName('rect').item(0);
        alert("document.getElementById('button'):  "
          + but
          + "\ndocument.getElementById('usebut'): - "
          + ubut
          + "\ndocument.getElementsByTagName('rect'): - "
          + rects
          + "\ndocument.getElementsByTagName('rect').getLength(): - "
          + zrects
          + "\ndocument.getElementsByTagName('rect').item(0): - "
          + rect);
      }
    ]]>
    </script>

<!-- Styles und Filter -->
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;
           font-weight:bold;}
      rect {fill:#33cc33; stroke:black;}
    ]]>
    </style>
```

```
    <filter id="filter" x="-.3" y="-.3" width="1.9" height="1.9">
      <feGaussianBlur in="SourceAlpha" stdDeviation="3"
        result="out1" />
      <feOffset dx="2" dy="2"
        result="out2" />
      <feSpecularLighting in="out1"
        surfaceScale="5" specularConstant=".75"
        specularExponent="20" lighting-color="#999999">
        <fePointLight x="-100" y="-100" z="100"/>
      </feSpecularLighting>
      <feComposite in2="SourceAlpha" operator="in"
        result="out3"/>
      <feComposite in="SourceGraphic" in2="out3"
        operator="arithmetic" k1="0" k2="1" k3="1" k4="0"
        result="out4"/>
      <feMerge>
        <feMergeNode in="out2"/>
        <feMergeNode in="out4"/>
      </feMerge>
    </filter>

<!-- der Button -->
    <g id="button" style="filter:url(#filter)">
      <rect x="40" y="70" width="80" height="20"
        rx="5" ry="5" />
      <text x="58" y="84">onclick</text>
    </g>
  </defs>

<!-- die Instanz des Buttons -->
  <use id="usebut" xlink:href="#button" x="-10" y="-50"
    onclick="meldung3();" />
</svg>
```

Wenn Sie auf den Button des obigen Beispiels klicken wird ein ***alert***-Fenster angezeigt, in dem Sie die Ausgabe der in diesem Unterkapitel verwendeten Methoden betrachten können.

Rückgabewert der Methode ***getElementById()*** auf das SVG Objekt button ist folgender String: object SVGGElement - also ein DOM Objekt, das das verwendete **g**-Element repräsentiert. Der Rückgabewert der gleichen Methode auf die, durch das **use**-Element verwendete Instanz des button (usebut), ist daher wie folgt: object SVGUseElement.

Durch die Methode ***getElementsByTagName()*** wird danach auf alle verwendeten rect-Elemente zugegriffen. Der Rückgabewert ist eine Liste von Elementknoten: object NodeList. Die Länge dieser Liste wird anschließend durch die Methode ***getLenght()*** ermittelt und beträgt 1, da im SVG Dokument nur 1 **rect**-Element verwendet wird. Zuletzt wird mit der Methode ***item()*** das erste Element der Liste ermittelt (Index 0). Der Rückgabewert ist logischerweise: object SVGRectElement.

## 16.5 Methoden für den Zugriff auf Knoten

Um folgende Methoden zu verwenden, benötigen Sie zuerst einen Knoten, da die Methoden auf Knoten angewendet werden (strenggenommen sind es Methoden des node-Objekts). Einen Knoten erhalten Sie durch Verwendung der Methoden des document-Objekts: ***getElementById()*** und ***getElementsByTagName()***.

- ***getNodeType()*** Gibt die Art des Knotes als Ganzzahl zurück. Dabei bedeutet die 1 einen Element-Knoten, die 2 einen Attribut-Knoten und die 3 einen Text-Knoten. Es gibt jedoch noch weitere Knotenarten.
- ***getNodeName()*** Gibt den Namen des Knotens zurück (z.B. den Element- oder Attributnamen)
- ***getParentNode()*** Wenn der Knoten ein Eltern-Knoten besitzt, wird es von dieser Methode zurückgegeben.
- ***hasChildNodes()*** Mit Hilfe dieser Methode können Sie überprüfen, ob der Knoten Kind-Knoten besitzt. Im zutreffenden Fall gibt diese Methode true zurück, sonst false. Beachten Sie: auch Attribute oder Text sind Knoten.
- ***getChildNodes()*** Liefert eine Liste der Kind-Knoten zurück.
- ***getFirstChild()*** Liefert den ersten Kind-Knoten zurück. Wenn diese Methode auf ein text-Element angewendet wird, ist der Rückgabewert der Text-Knoten, der die reinen Textdaten enthält.
- ***getNodeValue()*** Liefert den Inhalt des Knotens zurück. Wenn Sie diese Methode auf einen Text-Knoten anwenden, erhalten Sie die reinen Textdaten als Rückgabewert.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="140px" height="62px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>
    Javascript: Methoden für den Zugriff auf die Knoten.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function knotenInfo(){
        var document = evt.getTarget().getOwnerDocument();
        alert("Der Knoten des Text-Elements"
          + "\nKnotentyp: - "
          + document.getElementById('txt').getNodeType()
          + "\nKnotenname: - "
          + document.getElementById('txt').getNodeName()
          + "\nElternelement des Knotens: - "
          + document.getElementById('txt').getParentNode()
          + "\nDer Knoten hat Kind-Elemente: - "
          + document.getElementById('txt').hasChildNodes()
          + "\nKind-Elemente des Knotens: - "
          + document.getElementById('txt').getChildNodes()
          + "\nErstes Kind-Element des Knotens: - "
          + document.getElementById('txt').getFirstChild()
          + "\nKnotentyp des ersten Kind-Elements: - "
          + document.getElementById('txt').getFirstChild().getNodeType()
          + "\nInhalt des ersten Kind-Elements: - "
          + document.getElementById('txt').getFirstChild().getNodeValue()
          + "\n");
      }
    ]]>
    </script>

<!-- Styles und Filter -->
```

```
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;
            font-weight:bold;}
      rect {fill:#33cc33; stroke:black;}
    ]]>
    </style>
    <filter id="filter" x="-.3" y="-.3" width="1.9" height="1.9">
      <feGaussianBlur in="SourceAlpha" stdDeviation="3"
        result="out1" />
      <feOffset dx="2" dy="2"
        result="out2" />
      <feSpecularLighting in="out1"
        surfaceScale="5" specularConstant=".75"
        specularExponent="20" lighting-color="#999999">
        <fePointLight x="-100" y="-100" z="100"/>
      </feSpecularLighting>
      <feComposite in2="SourceAlpha" operator="in"
        result="out3"/>
      <feComposite in="SourceGraphic" in2="out3"
        operator="arithmetic" k1="0" k2="1" k3="1" k4="0"
        result="out4"/>
      <feMerge>
        <feMergeNode in="out2"/>
        <feMergeNode in="out4"/>
      </feMerge>
    </filter>

<!-- der Button -->
    <g id="button" style="filter:url(#filter)">
      <rect id="rechteck" x="40" y="70" width="80" height="20"
        rx="5" ry="5" />
      <text id="txt" x="58" y="84">onclick</text>
    </g>
  </defs>

<!-- die Instanz des Buttons -->
  <use id="usebut" xlink:href="#button" x="-10" y="-50"
    onclick="knotenInfo();" />
</svg>
```

Sämtliche Methoden für den Zugriff auf Knoten, werden im obigen Beispiel auf den Elementknoten des **text**-Element txt angewandt, auf welchen durch die Methode ***getElementById()*** zugegriffen wird. Die Ausgabe erfolgt wieder durch ein ***alert***-Fenster.

Nachfolgend eine kurze Erläuterung der einzelnen Rückgabwerte:

- ***getNodeType()*** liefert die Ausgabe 1: es handelt sich also um einen Elementknoten.
- ***getNodeName()*** ermittelt den Namen des Knotens: den Elementnamen **text**.
- ***getParentNode()*** liefert das Eltern-Element zurück: object SVGGElement - das **g**-Element.
- ***hasChildNodes()*** prüft, ob das text-Element Kind-Elemente hat. Die Anwort ist true. Es sind also Kind-Elemente vorhanden.
- ***getChildNodes()*** erzeugt als Rückgabewert eine Liste aller Kind-Elemente des **text**-Elements: object NodeList.
- ***getFirstChild()*** liefert das erste Kind-Element des text-Elements zurück: den Textknoten - object Text
- ***getNodeType()*** auf diesen Textknoten angewandt liefert zur Bestätigung die Ausgabe 3. Es ist also in der Tat ein Textknoten
- ***getNodeValue()*** ebenfalls auf den Textknoten angewandt, liefert den Inhalt des **text**-Elements, d.h. die reinen Textdaten zurück: onclick.

## 16.6 Methoden für den Zugriff auf Attribute

Für einen direkten Zugriff auf Attribute und Eigenschaften stehen folgende Methoden zur Verfügung:

- ***getAttributes()*** Diese Methode liefert eine Liste der Attributknoten eines Elementknotens zurück. Die Länge dieser Liste, d.h. die Anzahl der gefundenen Attribute kann durch die Methode ***getLength()*** ermittelt werden. Auf einzelne Elemente dieser Liste können Sie mit Hilfe der Methode ***item()*** zugreifen. Siehe auch ***getElementsByTagName()***.
- ***getAttribute()*** Diese Methode liefert den Wert eines bestimmten Attributknotens zurück. Dabei wird der Attributname als Parameter übergeben.
- ***getStyle()*** Rückgabewerte dieser Methode ist eine Liste, die alle Style-Eigenschaften enthält, die durch das Attribut ***style*** für ein Element definiert sind. Die Länge dieser Liste, d.h. die Anzahl der gefundenen Attribute kann durch die Methode ***getLength()*** ermittelt werden. Auf einzelne Elemente dieser Liste können Sie mit Hilfe der Methode ***item()*** zugreifen. Siehe auch ***getElementsByTagName()***. Beachten Sie: Wenn die Style-Eigenschaften in einem externen Style-Sheet definiert sind, ist überhaupt kein Zugriff auf die Eigenschaften möglich. Wenn die Style-Eigenschaften mit Hilfe des **style**-Elements im **defs**-Bereich des SVG Dokuments definiert sind, ist der lesende Zugriff mit ***getStyle()*** nicht möglich. In diesem Fall müssen Sie den Textknoten des **style**-Elements auslesen.
- ***getPropertyValue()*** Lieferte den Wert einer bestimmten Eigenschaft zurück, die innerhalb der Liste aller Style-Eigenschaften gesetzt ist. Daher ist diese Methode auf den Rückgabewert der Methode ***getStyle()*** anzuwenden. Erwartet den Namen der Eigenschaft (Zeichenkette) als Parameter.
- ***getCssText()*** Liefert die komplette Liste aller Style-Eigenschaften bzw. den gesamten Wert (die gesamte Zeichenkette) des ***style***-Attributs zurück. Diese Methode ist ebenfalls auf den Rückgabewert der Methode ***getStyle()*** anzuwenden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="440px" height="80x"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>Javascript: Methoden für den Zugriff auf
    Attribute und Eigenschaften.</desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function attr(){
        var document = evt.getTarget().getOwnerDocument();
        var rechteck = document.getElementById('rechteck');
        var rect_attr = rechteck.getAttributes();
        var rect_attr_anz = rechteck.getAttributes().getLength();
        alert("Alle Attribute des Rechtecks: " + rect_attr +
              "\nAnzahl aller Attribute: " + rect_attr_anz);
        for(var i = 0; i < rect_attr_anz; i++){
          alert("Attribut" + i + " : " +
                rect_attr.item(i).getNodeName() +
                " = " +
                rect_attr.item(i).getNodeValue() +
                "\n");
        }
      }
      function attr2(){
        var document = evt.getTarget().getOwnerDocument();
        var rechteck = document.getElementById('rechteck');
        var r_id = rechteck.getAttribute('id');
        var r_x = rechteck.getAttribute('x');
        var r_y = rechteck.getAttribute('y');
        var r_width = rechteck.getAttribute('width');
        var r_height = rechteck.getAttribute('height');
```

```
      var r_rx = rechteck.getAttribute('rx');
      var r_ry = rechteck.getAttribute('ry');
      var r_style = rechteck.getAttribute('style');
      alert("Alle Attribute des Rechtecks\n" +
            "id = " + r_id + "\n" +
            "x = " + r_x + "\n" +
            "y = " + r_y + "\n" +
            "width = " + r_width + "\n" +
            "height = " + r_height + "\n" +
            "rx = " + r_rx + "\n" +
            "ry = " + r_ry + "\n" +
            "style = " + r_style);
    }
    function prop1(){
      var document = evt.getTarget().getOwnerDocument();
      var rechteck = document.getElementById('rechteck');
      var r_prop = rechteck.getStyle();
      var r_prop_anz = rechteck.getStyle().getLength();
      alert("Alle Eigenschaften des Rechtecks: " + r_prop +
            "\nAnzahl aller Eigenschaften: " + r_prop_anz);
      for(var i = 0; i < r_prop_anz; i++){
        var prop_name = rechteck.getStyle().item(i);
        var prop_wert = rechteck.getStyle().getPropertyValue(prop_name);
        alert("Eigenschaft" + i + " : " +
              prop_name +
              " = " +
              prop_wert);
      }
    }
    function prop2(){
      var document = evt.getTarget().getOwnerDocument();
      var rechteck = document.getElementById('rechteck');
      var r_csstext = rechteck.getStyle().getCssText();
      alert("Der Wert des style-Attributs innerhalb des Rechtecks:\n\t"
            + r_csstext);
    }
  ]]>
  </script>

<!-- Styles und Filter -->
  <style type="text/css">
  <![CDATA[
    text {font-family:Verdana,sans-serif; font-size:12px;
          font-weight:bold;}
    text.normal {font-weight:normal;font-size:14px;}
  ]]>
  </style>
  <filter id="filter" x="-.3" y="-.3" width="1.9" height="1.9">
    <feGaussianBlur in="SourceAlpha" stdDeviation="3"
      result="out1" />
    <feOffset dx="2" dy="2"
      result="out2" />
    <feSpecularLighting in="out1"
      surfaceScale="5" specularConstant=".75"
      specularExponent="20" lighting-color="#999999">
      <fePointLight x="-100" y="-100" z="100"/>
    </feSpecularLighting>
    <feComposite in2="SourceAlpha" operator="in"
      result="out3"/>
    <feComposite in="SourceGraphic" in2="out3"
      operator="arithmetic" k1="0" k2="1" k3="1" k4="0"
      result="out4"/>
    <feMerge>
      <feMergeNode in="out2"/>
```

```
      <feMergeNode in="out4"/>
    </feMerge>
  </filter>

<!-- der Button -->
  <g id="button" style="filter:url(#filter)">
    <rect id="rechteck" x="40" y="70" width="80" height="20"
      rx="5" ry="5"
      style="fill:#33cc33; stroke:black;" />
    <text id="txt" x="58" y="84">onclick</text>
  </g>
 </defs>

<!-- die Instanzen des Buttons -->
 <use id="usebut1" xlink:href="#button" x="-10" y="-50"
  onclick="attr();" />
 <use id="usebut1" xlink:href="#button" x="90" y="-50"
  onclick="attr2();" />
 <use id="usebut1" xlink:href="#button" x="190" y="-50"
  onclick="prop1();" />
 <use id="usebut1" xlink:href="#button" x="290" y="-50"
  onclick="prop2();" />

<!-- Hilfstexte -->
 <text x="35" y="65" class="normal">Attribute 1</text>
 <text x="135" y="65" class="normal">Attribute 2</text>
 <text x="235" y="65" class="normal">Propertys 1</text>
 <text x="335" y="65" class="normal">Propertys 2</text>
</svg>
```

Im obigen Beispiel sind vier Javascript-Funktionen definiert, die jeweils durch einen Klick auf den zugehörigen Button ausgelöst werden. Innerhalb dieser Funktionen wird mit Hilfe der in diesem Unterkapitel vorgestellten Methoden, auf die Attribute und Eigenschaften des **rect**-Elements mit der ID rechteck zugegriffen. Dazu wird in der zweiten Zeile jeder Funktion mit Hilfe der Methode ***getElementById()*** das Objekt rechteck erzeugt, welches das entsprechende **rect**-Element repräsentiert.

In der ersten Funktion **attr()** wird die Methode ***getAttributes()*** verwendet, die eine Liste aller Attribute (object NamedNodeMap) zurückliefert. Danach wird die Länge dieser Liste, d.h. die Anzahl der Elemente, mit ***getLength()*** ermittelt. Die beiden Rückgabewerte werden darauf in einem ***alert***-Fenster angezeigt. Hiernach wird eine for-Schleife verwendet, für die als Endwert die Anzahl der Elemente festgelegt ist. In jedem der (daraus resultierenden) acht Schleifendurchläufe wird zuerst durch Verwendung der Methode ***item()*** und der Methode ***getNodeName()*** der Name des jeweiligen Attributs, und dann ebenfalls durch die Methode ***item()*** und der Methode ***getNodeValue()*** der jeweilige Wert zu diesem Attribut ermittelt. Beide Rückgabewerte werden in jedem Schleifendurchlauf innerhalb eines ***alert***-Fensters angezeigt.

In der zweiten Funktion **attr2()** wird mit Hilfe der Methode ***getAttribut()*** der Wert jedes einzelnen Attributs ermittelt. Alle Werte werden dann, zusammen mit den Attributnamen in einem ***alert***-Fenster ausgegeben.

Innerhalb der dritten Funktion **prop1()** wird durch die Methode ***getStyle()*** zuerst eine Liste mit allen Style-Eigenschaften erzeugt (object CSSStyleDeclaration). Hiernach wird mit ***getLength()*** die Anzahl der Style-Eigenschaften in dieser Liste ermittelt. Beide Werte werden in einem ***alert***-Fenster ausgegeben. Darauf wird wieder eine for-Schleife verwendet, um in jedem Schleifendurchlauf durch die Methode ***item()*** den Namen der Eigenschaft und, ebenfalls mit der Methode ***item()***, aber zusätzlich mit der Methode ***getPropertyValue()***, die als Parameter den jeweiligen Eigenschaftsnamen verwendet, den Wert dieser Eigenschaft zu ermitteln. Beide Rückgabewerte werden in jedem Schleifendurchlauf innerhalb eines ***alert***-Fensters angezeigt.

In der vierten Funktion **prop2()** ermittelt die Methode ***getCssText()*** den Wert des Attributs ***style***, d.h. die gesamte CSS-Zeichenkette. Dieser Wert wird dann in einem ***alert***-Fenster ausgegeben.

## 16.7 Methoden für den Zugriff auf Text

In SVG können Sie mit Hilfe eines **text**-Elements Textdaten in die Grafik einfügen. Diese Textdaten, d.h. der Inhalt des **text**-Elements, stellen einen eigenen Knoten innerhalb der Baumstruktur des SVG Dokuments dar: einen sogenannten Textknoten. Dieser Textknoten ist der erste Kind-Knoten des text-Element-Knotens. Zugriff auf den Textknoten ermöglichen folgende Methoden:

- ***getData()*** Diese Methode liefert den Inhalt eines Textknotens zurück.
- ***getComputedTextLength()*** Mit dieser Methode können Sie die Breite eines Textelements, d.h. der gesamten Zeichkette, in Pixeln ermitteln. Beachten Sie: Diese Methode wird auf den Elementknoten des text-Elements angewandt - und nicht auf den Textknoten.
- ***getSubStringLength()*** Mit dieser Methode können Sie die Breite einer Teilzeichenkette innerhalb eines Textelements in Pixeln ermitteln. Diese Methode wird ebenfalls auf den Elementknoten des text-Elements angewandt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="140px" height="62px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>
    Javascript: Methoden für den Zugriff auf Textknoten.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function txt(){
        var textelement = document.getElementById('text');
        var textlaenge = textelement.getComputedTextLength();
        var textteil_on = textelement.getSubStringLength(0,2);
        var textknoten = textelement.firstChild;
        var textdaten = textknoten.data;
        alert("Länge des Textes - "
              + textlaenge + " Pixel"
              + "\nLänge der Zeichenkette 'on' - "
              + textteil_on + " Pixel"
              + "\nInhalt des Textknotens - "
              + textdaten);
      }
    ]]>
    </script>

<!-- Styles und Filter -->
    <style type="text/css">
    <![CDATA[
      text {font-family:Verdana,sans-serif; font-size:12px;
            font-weight:bold;}
      rect {fill:#33cc33; stroke:black;}
    ]]>
    </style>
    <filter id="filter" x="-.3" y="-.3" width="1.7" height="1.7">
      <feGaussianBlur in="SourceAlpha" stdDeviation="3" result="out1" />
      <feOffset dx="2" dy="2" result="out2" />
      <feSpecularLighting in="out1"
        surfaceScale="5" specularConstant=".75"
        specularExponent="20" lighting-color="#999999">
        <fePointLight x="-100" y="-100" z="100"/>
      </feSpecularLighting>
      <feComposite in2="SourceAlpha" operator="in" result="out3"/>
```

```
    <feComposite in="SourceGraphic" in2="out3" operator="arithmetic"
                 k1="0" k2="1" k3="1" k4="0" result="out4"/>
    <feMerge>
      <feMergeNode in="out2"/>
      <feMergeNode in="out4"/>
    </feMerge>
  </filter>

<!-- der Button -->
  <g id="button" style="filter:url(#filter)">
    <rect x="40" y="70" width="80" height="20"
      rx="5" ry="5" />
    <text id="text" x="58" y="84" width="78" height="18">onclick</text>
  </g>
 </defs>

<!-- die Instanz des Buttons -->
 <use id="usebut" xlink:href="#button" x="-10" y="-50"
  onclick="txt();" />
</svg>
```

Die Funktion **txt()** im obigen Beispiel greift, nach Klick auf den Button, durch die Methode ***getElementById()*** auf das **text**-Element *text* zu, um dann in einem ***alert***-Fenster Informationen zu diesem Element anzuzeigen.

Zuerst wird mit der Methode ***getComputedTextLength()*** die Länge der gesamten Zeichenkette ermittelt. Dann durch die Methode ***getSubStringLength()*** die Länge der Teilzeichenkette on. Beachten Sie: Beide Methoden werden auf den Elementknoten des **text**-Elements angewendet.

Dann wird durch die Methode ***getFirstChild()*** der Textknoten des Elementknotens ermittelt, da der ersten Kind-Knoten eines **text**-Elements immer der Textknoten ist. Auf diesen Textknoten wird zuletzt die Methode ***getData()*** angewendet, welche den Inhalt des Textknotens, d.h. den reinen Textinhalt (die Zeichenkette) ermittelt und zurückliefert.

## 16.8 Methoden zur Manipulation von Attributen/Eigenschaften

Um SVG Inhalte dynamisch verändern zu können, müssen Sie in der Regel zuerst auf Knoten zugreifen. Nachdem also in den letzten vier Kapiteln Methoden für den Zugriff auf verschiedene Knoten behandelt wurden, folgt in den nächsten drei Kapiteln die eigentlich interessantere Thematik: die Manipulation von Inhalten, d.h. das dynamische Ändern von Elementen, Attributen, Werten oder Textdaten.
Es ist natürlich generell möglich, Änderungen durch die Methoden für den Zugriff zu realisieren (den Objekt-Eigenschaften neue Werte zuweisen). Allerdings gibt es spezielle Funktionen zur Änderung, die bevorzugt zu verwenden sind. Die folgenden drei Kapitel stellen diese Methoden kurz vor.
Die folgenden Methoden erlauben die Manipulation von Attributen und Eigenschaften. Sie sind daher auf Elementknoten anzuwenden, da diese Attribute oder Eigenschaften besitzen können.

- ***setAttribute()*** Setzt ein Attribut für ein Elementknoten. Erwartet zwei Parameter: den Namen des Attributs (Zeichenkette) und den entsprechenden Wert (Zahl oder Zeichenkette).
- ***removeAttribute()*** Entfernt ein Attribut aus einem Elementknoten. Erwartet als Parameter den Namen des Attributs (Zeichenkette).
- ***setProperty()*** Setzt ein neues Eigeschaft:Wert-Pärchen innerhalb der Liste aller Style-Eigenschaften. Erwartet zwei Parameter: den Namen der Eigenschaft (Zeichenkette) und den entsprechenden Wert (Zahl oder Zeichenkette).
- ***removeProperty()*** Entfert ein Eigenschaft:Wert-Pärchen aus der Liste aller Style-Eigenschaften. Erwartet den Namen der Eigenschaft als Parameter.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="320px" height="210px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>
    Javascript: Methoden zur Manipulation von Attributen
    und/oder Eigenschaften.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function ohne(){
        var stern_style = document.getElementById('stern').style;
        stern_style.setProperty('fill','none');
      }
      function gelb(){
        var stern_style = document.getElementById('stern').style;
        stern_style.setProperty('fill','yellow');
      }
      function gruen(){
        var stern_style = document.getElementById('stern').style;
        stern_style.setProperty('fill','limegreen');
      }
      function schwatt(){
        var stern_style = document.getElementById('stern').style;
        stern_style.removeProperty('fill');
      }
      function filter(){
        var stern = document.getElementById('stern');
        if (stern.getAttribute('filter') == 'url(#filter)'){
          stern.removeAttribute('filter');
        } else {
          stern.setAttribute('filter','url(#filter)');
        }
      }
    ]]>
    </script>
```

```
<!-- ein Filter -->
    <filter id="filter" x="-.3" y="-.3" width="1.9" height="1.9">
      <feGaussianBlur in="SourceAlpha" stdDeviation="3"
        result="out1" />
      <feOffset dx="2" dy="2"
        result="out2" />
      <feSpecularLighting in="out1"
        surfaceScale="5" specularConstant=".75"
        specularExponent="20" lighting-color="#999999">
        <fePointLight x="-100" y="-100" z="100"/>
      </feSpecularLighting>
      <feComposite in2="SourceAlpha" operator="in"
        result="out3"/>
      <feComposite in="SourceGraphic" in2="out3"
        operator="arithmetic" k1="0" k2="1" k3="1" k4="0"
        result="out4"/>
      <feMerge>
        <feMergeNode in="out2"/>
        <feMergeNode in="out4"/>
      </feMerge>
    </filter>
  </defs>

<!-- ein Stern: Ziel der Dynamik -->
  <polygon id="stern" transform="translate(20,20) scale(4)"
    points="0 20, 15 15, 20 0, 25 15, 40 20, 25 25, 20 40, 15 25"
    style="stroke:black;" />

<!-- 4 Kreise und eine Gruppe als Buttons -->
  <circle cx="240" cy="30" r="10"
    style="fill:white; stroke:black;"
    onclick="ohne()" />
  <circle cx="240" cy="60" r="10"
    style="fill:limegreen; stroke:black;"
    onclick="gruen()" />
  <circle cx="240" cy="90" r="10"
    style="fill:yellow; stroke:black;"
    onclick="gelb()" />
  <circle cx="240" cy="120" r="10"
    style="stroke:black;"
    onclick="schwatt()" />
  <g onclick="filter()">
    <rect x="230" y="144" width="56" height="20"
      style="fill:none; stroke:black;" />
    <text id="txt" x="235" y="160"
      style="font-family:Verdana; font-size:16px; font-weight:bold;">
      Filter
    </text>
  </g>

<!-- Hilfstexte -->
  <text x="235" dy="176"
    style="font-size:11px; font-weight:bold;">
    AN/AUS
  </text>
  <text x="265" y="27"
    style="font-size:11px; font-weight:bold; font-stretch:ultra-expanded;"
    writing-mode="tb">
    Farb-Wahl
  </text>
</svg>
```

Die obige Beispielgrafik zeigt einen schwarzen Stern (ein **polygon**-Element), neben dem vier farbige Kreise und der umrahmte Text Filter platziert sind. Diese fünf Elemente bzw. Objekte dienen als Buttons für die jeweils entsprechenden Funktion, der insgesamt fünf definierten Funktionen. Jede Funktion verändert die Eigenschaften oder Attribute des **polygon**-Elements. So können Sie dem Stern verschiedenen Farben und einen Filter zuordnen.

In der ersten Funktion **ohne()** wird durch die Methoden ***getElementById()*** und ***getStyle()*** zuerst die Liste aller Style-Eigenschaften des **polygon**-Elements ermittelt. Auf diese Liste, die ja ein Objekt darstellt, wird dann die Methode ***setProperty()*** angewendet, die als Parameter zuerst den Namen der zu verändernden Eigenschaft (fill) und dann den gewünschten neuen Wert für diese Eigenschaft (none) enthält. So ist dem Stern nach Klick auf den ersten Kreis (ohne Füllfarbe) keine Füllfarbe mehr zugeordnet.

Die folgenden Funktionen **gelb()** und **gruen()** sind genauso aufgebaut, wie die zuerst definierte Funktion **ohne()**. Die Füllfarbe des **polygon**-Elements ändert sich dadurch, dass der Eigenschaft ***fill*** mit Hilfe der Methode ***setProperty()*** ein neuer Wert zugewiesen wird. In diesem Fall Gelb bzw. Grün.

Bei der Funktion **schwatt()** wird zur Abwechslung ;-) die Methode ***removeProperty()*** verwendet, um den Stern mit der Farbe schwarz zu füllen. Diese Methode entfernt die Eigenschaft ***fill*** aus der Liste der Style-Definitionen für das **polygon**-Element. Da der voreingestellte Wert von ***fill*** (*black*) verwendet wird, wenn diese Eigenschaft (oder das Attribut ***fill***) nicht gesetzt ist, stellt sich der Stern nach dieser Funktion in schwarz dar.

In der letzte Funktion **filter()** wird einen if-Entscheidung verwendet, um dem **polygon**-Element entweder einen definierten Filter zuzuordnen, oder den bereits definierten Filter wieder aus dem Element zu entfernen. In der Bedingung von if wird daher mit der Methode ***getAttribute()*** überprüft, ob dem Attribut ***filter*** innerhalb des **polygon**-Elements der Wert *url(#filter)* zugeordnet ist. Im Ja-Fall wird das Attribut ***filter*** durch die Methode ***removeAttribute()*** entfernt, ansonsten wird es durch die Methode ***setAttribute()*** mit dem Wert *url(#filter)* im **polygon**-Element gesetzt. So können Sie mit Klick auf den Filter-Button den Filtereffekt ein- und ausschalten.

## 16.9 Methoden zur Manipulation von Elementen/Knoten

Mit Hilfe der folgenden Methoden können Sie Elemente bzw. Elementknoten erzeugen oder entfernen. Diese Methoden sind also auf einen Elementknoten anzuwenden.

- ***createElement()*** Erzeugt ein neues Element. Erwartet als Parameter den Namen des Elements.
- ***cloneNode()*** Dupliziert einen Knoten. Erwartet als Parameter entweder true oder false. Wenn Sie true übergeben, werden auch die Kind-Knoten mitkopiert, wenn Sie false verwenden, nur das Element selbst.
- ***createTextNode()*** Erzeugt einen Text-Knoten, d.h. die eigentliche Zeichenkette für ein **text**-Element und explizit nicht das **text**-Element selbst. Erwartet als Parameter eine Zeichenkette. Derart erzeugte Text-Daten können dann als Kind-Element (d.h. als eigentlicher Text-Inhalt) eines **text**-Elements verwendet werden (siehe zweites Beispiel auf dieser Seite).
- ***appendChild()*** Fügt einen duplizierten Knoten in die Baumstruktur des SVG Dokuments ein. Erwartet als Parameter den Namen des duplizierten Knotens und wird auf den entsprechenden Eltern-Elementknoten angewandt.
- ***insertBefore()*** Fügt einen duplizierten Knoten an einer genau festgelegten Stelle in die Baumstruktur des SVG Dokuments ein. Erwartet zwei Parameter: den Namen des duplizierten Knotens und den Namen eines bestehenden Knotens, vor dem der duplizierte Knoten eingefügt wird. Auch diese Methode ist auf den entsprechenden Eltern-Elementknoten anzuwenden.
- ***replaceChild()*** Ersetzt einen bereits bestehenden Elementknoten durch einen zweiten. Erwartet die Namen der beiden Knoten als Parameter. Diese Methode wird ebenfalls auf den entsprechenden Eltern-Elementknoten angewandt.
- ***removeChild()*** Entfernt einen Elementknoten aus der Baumstruktur des SVG Dokuments. Erwartet den Namen des Knotens als Parameter. Diese Methode wird ebenfalls auf den entsprechenden Eltern-Elementknoten angewandt.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="360px" height="240px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>
    Javascript: Methoden zur Manipulation von Knoten.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function sterne(){
        var svgElement = document.documentElement;
        var stern = document.createElement('polygon');
        stern.setAttribute('points',
          '0 20, 15 15, 20 0, 25 15, 40 20, 25 25, 20 40, 15 25');
        stern.setAttribute('style','fill:yellow; stroke:black');
        stern.setAttribute('transform', 'translate(110,60) scale(4)');
        svgElement.appendChild(stern);
        var stern2 = stern.cloneNode(false);
        stern2.setAttribute('transform', 'translate(90,93) scale(1.4)');
        svgElement.appendChild(stern2);
        var stern3 = stern.cloneNode(false);
        stern3.setAttribute('transform', 'translate(235,130) scale(1.4)');
        svgElement.insertBefore(stern3,stern);
        svgElement.removeChild(document.getElementById('go1'));
      }
      function bg(){
        var titel = document.getElementById('titel');
        var svgElement = titel.parentNode;
        var bg = document.createElement('rect');
```

```
      bg.setAttribute('x', 15);
      bg.setAttribute('y', 10);
      bg.setAttribute('width', 330);
      bg.setAttribute('height', 220);
      bg.setAttribute('style', 'fill:url(#gra1); stroke:black');
      svgElement.replaceChild(bg,titel);
      svgElement.removeChild(document.getElementById('go2'));
    }
  ]]>
  </script>

<!-- Styles und Gradients -->
  <style type="text/css">
  <![CDATA[
    text {font-family:Verdana,sans-serif; font-size:24px;
          font-weight:bold;}
  ]]>
  </style>
  <linearGradient id="gra1"
    x1="0" y1="0" x2="0" y2="1">
    <stop offset=".5" stop-color="black" />
    <stop offset="1" stop-color="yellow" />
  </linearGradient>
 </defs>

 <text id="titel" x="20" y="35">Knoten manipulieren</text>
 <g id="go1" onclick="sterne()">
  <rect id="rechteck1" x="20" y="54" width="30" height="20"
    fill="limegreen" stroke="black" />
  <text x="24" y="70" style="font-size:14px;">GO 1</text>
 </g>
 <g id="go2" onclick="bg()">
  <rect id="rechteck2" x="20" y="84" width="30" height="20"
    fill="limegreen" stroke="black" />
  <text x="24" y="100" style="font-size:14px;">GO 2</text>
 </g>
</svg>
```

In der obigen Beispielgrafik sind zwei Funktionen definiert. Die erste Funktion **sterne()** wird durch den Button go1 ausgelöst, die zweite Funktion **bg()** durch den Button go2. Beide Funktionen verwenden die, in diesem Unterkapitel, behandelten Methoden, um Elemente bzw. Knoten des SVG Dokuments zu manipulieren.

In der Funktion **sterne()** wird zuerst mit dem Attribut ***documentElement*** das **svg**-Element ermittelt. Die Methode ***createElement()***, die auf das document-Objekt angewendet wird, erzeugt daraufhin eine neues **polygon**-Element. Für diesen neuen Elementknoten werden mit Hilfe der Methode ***setAttribute()*** die Attribute ***points***, ***style*** und ***transform*** gesetzt. Das so erzeugte und mit Attributen ausgestattete **polygon**-Element (stern) wird dann durch die Methode ***appendChild()*** als Kind-Element des **svg**-Elements in die Grafik eingefügt. Um ein zweites neues Element zu erzeugen, wird anschließend durch die Methode ***cloneNode()*** der zuvor erzeugte stern dupliziert und das Attribut ***transform*** durch die Methode ***setAttribute()*** in diesem Duplikat stern2 mit einem anderen Wert neu gesetzt. Dann wird auch dieses neue Element stern2 durch ***appendChild()*** als Kind-Element des **svg**-Elements in der Grafik plaziert. Beachten Sie: Das Objekt stern2 wird nach stern in die Grafik eingefügt und liegt daher über diesem. Das dritte und letzte neue **polygon**-Element stern3 wird auf die gleiche Art erzeugt wie stern2. Die Plazierung innerhalb der Grafik wird allerdings mit der Methode ***insertBefore()*** vorgenommen. Das Objekt stern3 wird ebenfalls als direktes Kind-Element des **svg**-Elements eingebunden, aber es wird vor dem Objekt stern plaziert und liegt daher unter diesem. Die letzte Aktion der Funktion **sterne()** ist die Entfernung des ersten Buttons go1 durch die Methode ***removeChild()***.

In der Funktion **bg()** wird zuerst das **text**-Element titel ermittelt, das erstplatzierte Element innerhalb der Grafik. Daraufhin wird wieder das **svg**-Element ermittelt. Diesmal allerdings mit Hilfe der Methode ***getParentNode()***, die auf das **text**-Element titel angewendet wird, und das Eltern-Element diese Elements zurückliefert. Das geht natürlich nur, wenn man weiß, dass das entsprechende Eltern-Element das **svg**-

Element ist (wie in diesem Fall). Anschließend wird durch die Methode ***createElement()*** ein **rect**-Element (ein Hintergrundrechteck) erzeugt und diesem, durch fünf-maliges Verwenden der Methode ***setAttribute()***, fünf unterschiedliche Attribute zugeordnet. Dann wird die Methode ***replaceChild()*** verwendet, um das **text**-Element titel aus der Grafik zu entfernen und durch das neue **rect**-Element bg zu ersetzten. Das neue Element wird dabei an der gleichen Stelle plaziert, an der das entfernte Element vorher gestanden hat (in unserem Beispiel an der ersten Stelle in der Grafik). Zuletzt wird auch der zweite Button go2 durch die Methode ***removeChild()*** aus der Grafik entfernt.

Das folgende Beispiel zeigt, wie man unter Verwendung der Methoden ***createElement()*** und ***createTextNode()*** einen neuen Text inclusive Textinhalt in ein SVG-Dokument einfügen kann.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="360" height="90"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Texte erzeugen</title>
<desc>text-Elemente inclusive Inhalt via Scriptsprache erzeugen</desc>
  <defs>
    <style type="text/css"><![CDATA[
      text {font-family:Verdana,sans-serif;}
      text.norm {font-family:Verdana,sans-serif; font-size:20px;}
      text.ext {font-size:25px;
                font-weight:bold;
                fill:red;
                stroke:black;
                font-stretch:extra-expanded;}
    ]]></style>
    <script type="text/ecmascript"><![CDATA[
      function text(){
        var svgElement = document.documentElement;
        var tnode = document.createTextNode('thx for the click');
        var t = document.createElement('text');
        t.setAttribute('x','20');
        t.setAttribute('y','32');
        t.setAttribute('class','ext');
        t.setAttribute('id','text1');
        t.appendChild(tnode);
        svgElement.appendChild(t);
      }
    ]]></script>
  </defs>

  <g onclick="text();">
    <rect x="20" y="50" width="85" height="27" />
    <text x="25" y="70" fill="white" class="norm">clickme</text>
  </g>
</svg>
```

Im Beispiel wird zuerst mit der Methode ***createTextNode()*** eine neuer Text-Knoten erzeugt, der die eigentlichen Zeichendaten enthält.

Dann wird mit der Methode ***createElement()*** ein neues **text**-Element erzeugt und mit Hilfe der Methode ***setAttribute()*** werden einige Attribute für dieses **text**-Element festgelegt. Im direkten Anschluss wird dann durch die Methode ***appendChild()*** der Text-Knoten als Kind-Element des **text**-Elements eingefügt.

Zuletzt wird - ebenfalls mit Hilfe der Methode ***appendChild()*** - der Knoten des **text**-Elements als Kind des **svg**-Elements in das Dokument eingefügt.

## 16.10 Methoden zur Manipulation von Text

Um auf den Text-Knoten zuzugreifen (die eigentlichen Textdaten/-zeichen), wird die Methode ***getFirstChild()*** auf den Knoten eines **text**-Elements verwendet. Den Knoten des **text**-Elements selbst, können Sie durch die Methoden des document-Objekts ***getElementById()*** oder ***getElementsByTagName()*** erzeugen.
Um Texte zu manipulieren, können Sie die nachfolgenden Methoden verwenden, die alle auf einen Text-Knoten angewendet werden (.. wie Sie **text**-Elemente inclusive Textdaten neu erzeugen können ist in Kapitel 16.9 erläutert).

- ***setData()*** Mit Hilfe dieser Methode können Sie den Wert eines Textknotens überschreiben. Als Parameter wird die neue Zeichenkette übergeben.
- ***appendData()*** Diese Methode bietet Ihnen die Möglichkeit eine neue Zeichenkette an eine bestehenden Textknoten anzuhängen. Als Parameter wird die neue Zeichenkette übergeben.
- ***insertData()*** Durch diese Methode können Sie eine neue Zeichenkette an einer beliebigen Position innerhalb des Textknotens einfügen. Als erster von zwei Parametern wird zuerst die Zeichenpostion angegeben an der die neue Zeichenkette platziert werden soll. Als zweiter Parameter wird die neue Zeichenkette übergeben.
- ***deleteData()*** Wenn Sie eine beliebige Zeichenkette aus dem Textknoten entfernen möchten, können Sie diese Methode verwenden. Die Methode erwartet zwei Ganzzahlen als Parameter: das Zeichen, ab dem gelöscht werden soll, dabei beginnt die Numerierung bei 0 und wie viele Zeichen gelöscht werden sollen.
- ***replaceData()*** Verwenden Sie diese Methode, wenn Sie eine beliebige Zeichenkette aus dem Textknoten durch eine andere, neue Zeichenkette ersetzen möchten. Diese Methode erwartet drei Parameter: zwei Ganzzahlen und die neue Zeichenkette. Durch die erste Ganzahl wird das Zeichen festgelegt, ab dem ersetzt werden soll, die zweite Ganzahl legt fest, wie viele Zeichen ersetzt werden sollen. An genau der Stelle wo die Zeichen entnommen wurden, wird die neue Zeichenkette (der dritte Parameter) eingefügt.
- ***substringData()*** Wenn Sie eine beliebige Zeichenkette aus dem Textknoten als Rückgabewert benötigen, können Sie diese Methode verwenden. Die Methode erwartet zwei Ganzzahlen als Parameter: das Zeichen, ab dem extrahiert werden soll, dabei beginnt die Numerierung bei 0 und wie viele Zeichen extrahiert werden sollen. Die so entnommene Zeichenkette kann in einer Variablen aufgefangen werden.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="418px" height="220px"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Javascript und SVG</title>
  <desc>
    Javascript: Methoden zur Manipulation von Textknoten.
  </desc>
  <defs>
<!-- der Script Bereich -->
    <script type="text/javascript">
    <![CDATA[
      function titel_over(){
        var textknoten = document.getElementById('titel').getFirstChild();
        textknoten.setData('Manipulierter Textknoten');
      }
      function titel_out(){
        var textknoten = document.getElementById('titel').getFirstChild();
        textknoten.setData('Textknoten manipulieren');
      }
      function rot(){
        var textknoten = document.getElementById('text_rot').getFirstChild();
        textknoten.appendData(' TOLL!');
      }
      function gruen(){
        var textknoten = document.getElementById('text_gruen').getFirstChild();
```

```
      if (textknoten.substringData(26,1) == ' '){
        textknoten.insertData(26,'(e)');
      }
    }
    function blau(){
      var textknoten = document.getElementById('text_blau').getFirstChild();
      textknoten.deleteData(16,22);
    }
    function schwatt(){
      var textknoten = document.getElementById('text_schwatt').getFirstChild();
      textknoten.replaceData(20,100,'hat diesen Text verändert.');
    }
    function reset(){
      var tk1 = document.getElementById('text_rot').getFirstChild();
      var tk2 = document.getElementById('text_gruen').getFirstChild();
      var tk3 = document.getElementById('text_blau').getFirstChild();
      var tk4 = document.getElementById('text_schwatt').getFirstChild();
      tk1.data = 'Der rote Button verändert diesen Text.';
      tk2.data = 'Der grüne Button verändert diesen Text.';
      tk3.data = 'Der blaue Button verändert diesen Text.';
      tk4.data = 'Der schwarze Button verändert diesen Text.';
    }
  ]]>
  </script>

<!-- Styles -->
  <style type="text/css">
  <![CDATA[
    text {font-family:Verdana,sans-serif; font-size:14px;}
  ]]>
  </style>
 </defs>

<!-- die Texte -->
 <text id="titel" x="20" y="35"
  style="font-size:24px; font-weight:bold;"
  onmouseover="titel_over()"
  onmouseout="titel_out()">
  Textknoten manipulieren
 </text>
 <text id="text_rot" x="50" y="80"
  style="fill:red;">Der rote Button verändert diesen Text.</text>
 <text id="text_gruen" x="50" y="110"
  style="fill:green;">Der grüne Button verändert diesen Text.</text>
 <text id="text_blau"
  x="50" y="140"
  style="fill:blue;">Der blaue Button verändert diesen Text.</text>
 <text id="text_schwatt" x="50" y="170">Der schwarze Button verändert diesen Text.</text>

<!-- die Buttons -->
 <rect x="25" y="67" width="15" height="15"
  style="fill:red;"
  onclick="rot()" />
 <rect x="25" y="97" width="15" height="15"
  style="fill:green;"
  onclick="gruen()" />
 <rect x="25" y="127" width="15" height="15"
  style="fill:blue;"
  onclick="blau()" />
 <rect x="25" y="157" width="15" height="15"
  onclick="schwatt()" />
 <g onclick="reset()">
  <rect x="50" y="185" width="45" height="20"
   style="fill:none; stroke:black;" />
```

```
    <text x="53" y="200">Reset</text>
  </g>
</svg>
```

Das obige Beispiel enthält insgesamt sieben Funktionen. Alle Funktionen verwenden Methoden zur Manipulation eines Textknotens, um den Inhalt eines der fünf **text**-Elemente (also die reinen Textdaten) innerhalb Grafik zu verändern. Zu diesem Zweck wird in allen Funktionen mit Hilfe der Methoden ***getElementById()*** und ***getFirstChild()***, zuerst der entsprechende Textknoten ermittelt.

Die ersten beiden Funktionen **titel_over()** und **titel_out()** verändern den Textknoten des **text**-Elements titel durch die Methode ***setData()***. Diese Methode überschreibt den ursprünglichen Textinhalt duch die Zeichenkette, die als Parameter übergeben wird. Die Funktion **titel_over** wird durch einen mouseover-Event, die Funktion **titel_out** durch eine mouseout-Event ausgelöst, die sich direkt auf das **text**-Element titel beziehen (also wenn Sie mit den Maus den Bereich dieses Elements betreten bzw. wieder verlassen).

Die folgenden vier Funktionen werden durch einen Klick auf die farbigen Rechtecke ausgelöst, die vor jeder weiteren, entsprechend farbigen Zeichenkette in der Grafik plaziert sind.

Die Funktion **rot()** verändert den Textknoten des **text**-Elements text_rot. Mit Hilfe der Methode ***appendData()*** wird bei jedem Klick die als Parameter angegeben Zeichenkette ( TOLL!) an die bestehenden Textdaten angehängt.

Die Funktion **gruen()** verändert den Textknoten des **text**-Elements text_gruen. In dieser Funktion wird zuerst innerhalb einer if-Entscheidung das 26-ste Zeichen der Zeichenkette überprüft. Dies geschieht innerhalb der if-Bedingung durch die Methode ***substringData()***. Nur wenn die Teilzeichenkette, die an der 26-sten Stelle beginnt und genau 1 Zeichen lang ist, ein Leerzeichen enthält wird die Anweisung innerhalb der if-Entscheidung ausgeführt. Hier wird die Zeichenkette durch die Methode ***insertData()*** verändert. Mit Hilfe dieser Methode wird an der 26-sten Stelle der Zeichenkette eine neue Zeichenkette eingefügt, die der Methode als Parameter übergeben wird ((e)). Jetzt ist das 26-ste Zeichen eine runde Klammer.

Die Funktion **blau()** verändert den Textknoten des **text**-Elements text_blau. Mit Hilfe der Methode ***deleteData()*** wird die Teilzeichekette, die an der Position 16 beginnt und 22 Zeichen lang ist, aus der ursprünglichen Zeichenkette entfernt.

Die Funktion **schwatt()** verändert den Textknoten des **text**-Elements text_schwatt. Durch die Methode ***replaceData()*** wird die Teilzeichenkette, die an der Position 20 beginnt und, durch die angegebene Länge 100, auf jeden Fall den Rest der Zeichenkette beinhaltet, durch eine neue Zeichenkette ersetzt. Diese neue Zeichenkette (hat diesen Text verändert) wird der Methode ***replaceData()*** als letzter Parameter übergeben.

Die Funktion **reset()**, die durch einen Klick auf den Button mit der Aufschrift Reset ausgelöst wird, stellt in allen Textknoten durch eine Zuweisung (Eigenschaft = 'Wert') wieder den ursprünglichen Zustand her.

# 17 SVG fonts

Die zuverlässige Darstellung von Schrift ist eine Grundvoraussetzung bei der Erstellung von Grafiken.

Es muß daher auch bei willkürlich gewählten bzw. aussergewöhlichen Schriftarten gewährleistet sein, dass alle SVG darstellenden Programme (user agents) das exakt gleiche Ergebnis am Bildschirm anzeigen, unabhängig ob die in der Grafik verwendete Schriftart auf dem System des Betrachters verfügbar ist oder nicht.

Diese Möglichkeit bieten **SVG fonts**. Allen SVG user agents kann durch ihre Verwendung ein einheitliches Schriftformat zur Verfügung gestellt werden.

Mittels **SVG fonts** können Sie die Schriftzeichen einer vorhandenen oder selbst erstellten Schriftart durch SVG-Elemente grafisch beschreiben. Alle Zeichen liegen dann im SVG Format vor und werden folglich von allen SVG user agents einheitlich dargestellt.

Allerdings sind **SVG fonts** bei sehr kleinen Schriftgrößen nur eingeschränkt verwendbar. Außerdem nimmt der SVG Quellcode, der eine oder sogar mehrere Schriftarten beschreibt, viel Raum innerhalb der Grafik ein und erhöht somit die Dateigröße des SVG Dokuments.

Sie können **SVG fonts** jedoch auch in einem externen SVG Dokument definieren und diese Schriftarten dann aus jedem beliebigen anderen SVG Dokument referenzieren.

Für die effektive Erstellung von **SVG fonts** gibt es Programme, die existierende Schriftarten konvertieren können, wie z.B. der **Font Converter** aus dem Apache SVG Batik Projekt, der aus TrueType fonts **SVG fonts** erzeugt. Eine Erläuterung hierzu finden Sie im Kapitel 17.2 - SVG Fonts mit Apache Batik erzeugen. Sie können allerdings auch jedes einzelne Zeichen einer Schriftart manuell editieren ...

## 17.1 Grundlagen SVG fonts

**SVG fonts** können innerhalb des **defs**-Bereichs einer Grafik mit Hilfe des **font**-Elements und seiner Kind-Elemente definiert werden.

Mit Hilfe des Kind-Elements **font-face** wird die Schrift beschrieben, d.h. sie können hier Eigenschaften und Charakterisktika der Schrift angeben. Einzelne Schriftzeichen werden durch das Kind-Element **glyph** bzw. **missing-glyph** mittels Pfadbeschreibungen festgelegt. Kerning zweier Schriftzeichen kann mittels der Kind-Elemente **hkern** und **vkern** definiert werden.

Die Eigenschaften und Charakteristika von **SVG fonts** sind nahezu identisch mit den Eigenschaften und Charakteristika von Schriften, wie sie in der CSS 2 Spezifikation beschrieben werden. Das SVG Element **font-face** ist gewissermaßen die SVG-Variante der CSS 2 @font-face Regel. Daher sind nahezu alle Möglichkeiten, die mit Hilfe der CSS 2 @font-face Regel in Bezug auf Schriftspezifikation, Schriftcharakteristika und Schriftauswahl vorgenommen werden können, auch mit Hilfe des Elements **font-face** in einem **SVG font** möglich.

Schriftspezifikation, Schriftcharakteristika und Schriftauswahl sind Bestandteil der CSS 2 Spezifikation und haben folgende Bedeutung:

- **Schriftspezifikation** Schriften können nicht lediglich über einen bestimmten Schriftnamen angefordert werden, sondern ebenfalls durch die Festlegung mehrerer Schrifteigenschaften. Diese Eigenschaftsnamen, wie font-family, font-weight, font-style, etc. bilden die Grundlage für den Schriftauswahlmechanismus des Benutzerprogramms.
- **Schriftcharakteristika** Schriftcharakteristika werden herangezogen, um Schriften zu näher zu beschreiben. Mit Hilfe von Eigenschaften können Sie weitere Merkmale der Schrift, wie z.B. metrischen Angaben für Grundlinieneinstellungen, Akzent-höhen, etc. definieren.
- **Schriftauswahl** Die Schriftauswahl wird vom Benutzerprogramm vorgenommen. Dazu wird die **Schriftbeschreibung**, die in SVG durch das Element **font-face** realisiert wird und in der die Schriftspezifikation und die Charakteristika der Schrift durch sogenannte **Schriftdeskriptoren** (Attribute) bestimmt wurden, zur internen Schriftdatenbank hinzugefügt und dann verwendet, um die entsprechenden Schriftdaten auszuwählen.

Metrische Angaben für Schriftzeichen werden in Abhängigkeit eines abstrakten Quadrates angegeben, das ein einzelnes Zeichen einschließt und mit **em square** bezeichnet wird. Dieses **em square** ist nicht lediglich ein Quadrat, sondern vielmehr ein Erstellungs-Raster bestimmter Größe für einzelne Zeichen. übliche Rastergrößen sind 250 (Intellifont), 1000 (Type 1) oder 2048 (TrueType, TrueTypeGX, OpenType). Dabei befindet sich der Nullpunkt dieses Rasters in der linken, unteren Ecke des **em squares**, d.h. die y-Achse verläuft in diesem Raster nicht - wie sonst in SVG üblich - von oben nach unten, sondern von unten nach oben. Durch das Attribut ***units-per-em*** können Sie im **font**-Element und im **font-face**-Element die entsprechende Rastergröße des **em squares** festlegen.

In der W3C CSS 2 Specification finden Sie die normativen Informationen zu dieser Thematik. Weitere empfehlenswerte Seiten: Deutsche Übersetzung der CSS 2 Spezifikation von Stefan Mintert. HTML World - Kapitel Schriftdefinition von Jan Winkler.

Nachfolgend ein SVG Dokument, das die Schriftart Entebbe als **SVG font** beinhaltet. Dieser wurde aus dem gleichnamigen TrueType Font mit Hilfe des Apache Batik Font Converters erzeugt.

Beispiel Quellcode

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" >
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<defs>
<font id="ente" horiz-adv-x="54" >
  <font-face
    font-family="Entebbe"
    units-per-em="1000"
    panose-1="0 0 0 0 0 0 0 0 0 0"
    ascent="935"
```

```
    descent="-292"
    alphabetic="0" />
  <missing-glyph horiz-adv-x="800"
    d="M50 0V800H750V0H50ZM700 50V750H100V50H700Z" />
  <glyph unicode=" " glyph-name="space" horiz-adv-x="290" />
  <glyph unicode="!" glyph-name="exclam" horiz-adv-x="223"
    d="M86 171L83 197V683Q83 719 96 725T117 731Q141 731 145 709L148 683
    V197Q148 161 135 155T114 149Q90 149 86 171ZM76 4Q61 19 61 40Q61 62 75 76
    T112 91Q119 91 140 83T162 41Q162 30 159 22T144 2T111 -10Q92 -10 76 4Z" />
  <glyph unicode="&amp;quot;" glyph-name="quotedbl" horiz-adv-x="354"
    d="M82 581Q74 577 56 573Q61 583 68 599T78 627L79 630Q51 651 51 682
    Q51 687 51 690T60 713T101 732Q110 732 132 723T154 676Q154 641 136 622
    T106 594T82 581ZM232 581Q224 577 206 573Q211 583 218 599T228 627
    L229 630Q201 651 201 682Q201 687 201 690T210 713T251 732Q260 732 282 723
    T304 676Q304 641 286 622T256 594T232 581Z" />
  <glyph unicode="#" glyph-name="numbersign" horiz-adv-x="491"
    d="M119 238H23L36 288H132L167 432H69L81 482H180L239 720H291L232 482
    H327L386 720H438L379 482H469L457 432H366L331 288H423L410 238H318
    L259 0H207L266 238H171L112 0H60L119 238ZM183 288H279L314 432H219L183 288Z" />
  <!-- ... -->
  <!-- ... mehr Zeichendefinitionen folgen hier -->
  <!-- ... -->
</font>
</defs>

<text x="20" y="60"
  style="font-family:Entebbe; font-size:18px; fill:black">
  !SVG fonts!
</text>
</svg>
```

## 17.2 SVG fonts mit Apache Batik erzeugen

Das Apache Batik Projekt stellt ein Programm zur Verfügung, das **SVG fonts** aus TrueType fonts erzeugen kann - den **Font Converter**.

Systemvoraussetzung für das Apache Batik Projekt ist eine Java Laufzeitumgebung. Stellen Sie also sicher, dass auf Ihrem System entweder die aktuelle JRE (Java Runtime Environment) oder J2SDK (Java Development Kit) installiert ist. Nach Installation von Batik (auf Windows oder Linux) kann der **Font Converter** des batik-Projekts (batik-ttf2svg.jar) von der Kommandozeile gestartet werden. Wechseln Sie zu diesem Zweck zuerst in das Installationsverzeichnis von batik.

Als Parameter erwartet das Programm zuerst den Pfad zur ttf-Datei, dann eine eindeutige ID für den SVG font (wird von batik im font-Element gesetzt) und anschließend den Pfad zur Output-Datei (dem SVG Dokument, das den SVG font beinhalten wird). Mit dem Parameter *-testcard* werden zusätzlich alle Zeichen des neuen SVG fonts mit Hilfe von **text**-Elementen in das Dokument eingefügt.

Beispiel für die Erstellung eines **SVG fonts** "Comic Sans MS" aus dem gleichnamigen TrueType font: *> java -jar batik-ttf2svg.jar c:\windows\fonts\comic.ttf -id comic -o comic.svg -testcard* Die Ausgabedatei comic.svg enthält nun den **SVG font** comic (basierend auf der Schriftart Comic Sans MS) und einige **text**-Elemente, die die Schriftzeichen darstellen.

Das folgende Beispiel ist aus Platzgründen reduziert. Um den kompletten Quellcode - wie ihn batik-ttf2svg.jar erzeugt hat - zu betrachten, rufen Sie bitte die Beispielgrafik auf. Über das Kontext-Menü "Quelle anzeigen" (erscheint nach Klick auf die rechte Maustaste) können Sie den kompletten Quellcode betrachten.

Beispiel Quellcode

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" >
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
<defs>
<font id="comic" horiz-adv-x="959">
<font-face font-family="Comic Sans MS"
    units-per-em="2048"
    panose-1="3 15 7 2 3 3 2 2 2 4"
    ascent="2257"
    descent="-597"
    alphabetic="0" />
<missing-glyph horiz-adv-x="1024"
  d="M128 0V1638H896V0H128ZM256 128H768V1510H256V128Z" />
<glyph unicode=" " glyph-name="space" horiz-adv-x="612" />
<glyph unicode="!" glyph-name="exclam" horiz-adv-x="487"
  d="M331 1516V403Q331 314 242 314Q153 314 153 403Q153 444 151 525
  T148 647Q148 792 150 1081T153 1516Q153 1605 242 1605Q331 1605 331 1516Z
  M332 23Q332 -13 304 -39T239 -66Q199 -66 159 -14Q120 35 120 76
  Q120 112 148 138T214 164Q253 164 294 112Q332 63 332 23Z" />
<glyph unicode="&amp;quot;" glyph-name="quotedbl" horiz-adv-x="869"
  d="M294 1116Q294 1099 299 1067T304 1018Q304 981 277 957T212 932
  Q116 932 116 1103Q116 1169 119 1301T122 1499Q122 1538 147 1563
  T211 1588Q250 1588 275 1563T300 1499Q300 1435 297 1308T294 1116Z
  M687 1495Q687 1479 684 1450T681 1408L672 1018Q670 928 583 928
  Q545 928 519 953T494 1016L503 1413Q505 1492 516 1524Q537 1584 598 1584
  Q637 1584 662 1559T687 1495Z" />
<glyph unicode="#" glyph-name="numbersign" horiz-adv-x="1726"
  d="M552 577H1052L1165 992H677L552 577ZM365 577L487 992
  H220Q125 992 125 1092Q125 1168 273 1168H537Q563 1284 616 1484
  Q645 1576 733 1576Q766 1576 787 1550T809 1491Q809 1451 783 1359
  Q739 1200 732 1168H1217Q1251 1289 1310 1491Q1336 1576 1408 1576
  Q1492 1576 1492 1498Q1492 1430 1409 1168H1604Q1706 1168 1706 1093
  Q1706 988 1593 988Q1578 988 1547 990T1500 992H1354L1248 577H1452
  Q1525 577 1555 566Q1607 546 1607 486Q1607 394 1520 394H1198L1108 69
  Q1081 -28 996 -28Q914 -28 914 51Q914 110 947 223Q993 380 996 394
  H496Q466 277 391 50Q362 -21 293 -21Q213 -21 213 58Q213 97 236 171
  Q271 283 306 394H115Q32 394 32 493Q32 532 66 556Q95 577 136 577H365Z" />
```

```
<glyph unicode="$" glyph-name="dollar" horiz-adv-x="1420"
  d="M643 887V1246Q444 1171 444 1014Q444 917 643 887ZM821 676
  V156Q927 192 998 262Q1077 342 1077 435Q1077 609 821 676Z
  M821 1636V1467H825Q921 1467 1044 1434Q1205 1391 1205 1324
  Q1205 1236 1116 1236Q1076 1236 976 1257T821 1278V871
  Q1042 851 1167 726Q1281 610 1281 446Q1281 245 1142 113
  Q1019 -3 821 -40V-306Q821 -345 796 -370T731 -396Q643 -396 643 -308
  V-53Q155 -46 155 200Q155 290 238 290Q285 290 328 253Q392 198 425 181
  Q505 141 643 135V702H635Q480 708 367 778Q227 864 227 1017
  Q227 1154 355 1275Q473 1387 643 1438V1630Q643 1669 669 1695T735 1721
  Q821 1721 821 1636Z" />
<!-- ... -->
<!-- ... mehr Zeichendefinitionen folgen hier -->
<!-- ... -->
</font>
</defs>

<g style="font-family:Comic Sans MS; font-size:18px; fill:black">
<text x="20" y="60">
  !&amp;quot;#$%&amp;amp;&amp;apos;()*+,-./0123456789:;&amp;lt;&amp;gt;?
</text>
<text x="20" y="120">
  @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
</text>
<text x="20" y="180">
  `abcdefghijklmnopqrstuvwxyz|{}~
</text>
<text x="20" y="240">
  &amp;#x80;&amp;#x81;&amp;#x82;&amp;#x83;&amp;#x84;&amp;#x85;&amp;#x86;&amp;#x87;&amp;#x88;&amp;#x89;
  &amp;#x8a;&amp;#x8b;&amp;#x8c;&amp;#x8d;&amp;#x8e;&amp;#x8f;&amp;#x90;&amp;#x91;&amp;#x92;&amp;#x93;
  &amp;#x94;&amp;#x95;&amp;#x96;&amp;#x97;&amp;#x98;&amp;#x99;&amp;#x9a;&amp;#x9b;&amp;#x9c;&amp;#x9d;
  &amp;#x9e;&amp;#x9f;
</text>
<text x="20" y="300">
  &amp;#xa0;&amp;#xa1;&amp;#xa2;&amp;#xa3;&amp;#xa4;&amp;#xa5;&amp;#xa6;&amp;#xa7;&amp;#xa8;&amp;#xa9;
  &amp;#xaa;&amp;#xab;&amp;#xac;&amp;#xad;&amp;#xae;&amp;#xaf;&amp;#xb0;&amp;#xb1;&amp;#xb2;&amp;#xb3;
  &amp;#xb4;&amp;#xb5;&amp;#xb6;&amp;#xb7;&amp;#xb8;&amp;#xb9;&amp;#xba;&amp;#xbb;&amp;#xbc;&amp;#xbd;
  &amp;#xbe;&amp;#xbf;
</text>
<text x="20" y="360">
  &amp;#xc0;&amp;#xc1;&amp;#xc2;&amp;#xc3;&amp;#xc4;&amp;#xc5;&amp;#xc6;&amp;#xc7;&amp;#xc8;&amp;#xc9;
  &amp;#xca;&amp;#xcb;&amp;#xcc;&amp;#xcd;&amp;#xce;&amp;#xcf;&amp;#xd0;&amp;#xd1;&amp;#xd2;&amp;#xd3;
  &amp;#xd4;&amp;#xd5;&amp;#xd6;&amp;#xd7;&amp;#xd8;&amp;#xd9;&amp;#xda;&amp;#xdb;&amp;#xdc;&amp;#xdd;
  &amp;#xde;&amp;#xdf;
</text>
<text x="20" y="420">
  &amp;#xe0;&amp;#xe1;&amp;#xe2;&amp;#xe3;&amp;#xe4;&amp;#xe5;&amp;#xe6;&amp;#xe7;&amp;#xe8;&amp;#xe9;
  &amp;#xea;&amp;#xeb;&amp;#xec;&amp;#xed;&amp;#xee;&amp;#xef;&amp;#xf0;&amp;#xf1;&amp;#xf2;&amp;#xf3;
  &amp;#xf4;&amp;#xf5;&amp;#xf6;&amp;#xf7;&amp;#xf8;&amp;#xf9;&amp;#xfa;&amp;#xfb;&amp;#xfc;&amp;#xfd;
 &amp;#xfe;&amp;#xff;</text>
</g>
</svg>
```

Im defs-Bereich der vom **Font Converter** erzeugten SVG Grafik, finden Sie einen **font**-Bereich, der durch das öffnende und schließende **font**-Element definiert wird.

Innerhalb dieses **font**-Bereichs wird zuerst mit Hilfe des Elements **font-face** die Schriftbeschreibung definiert. Durch Attribute werden Name der Schriftart, die Rastergröße des **em square** und weitere Charakteristika der Schriftart festgelegt. Anschließend werden einzelne Schriftzeichen durch die Elemente **missing-glyph** und **glyph** definiert.

Der so angelegte **SVG font** kann nun von jedem **text**-Element oder **tspan**-Element verwendet werden. Dazu wird das Attribut ***font-family*** in dem jeweiligen **text**-Element verwendet. Als Wert erhält das Attribut die gleiche Angabe, wie das Attribut (der Deskriptor) ***font-family*** im **font-face**-Element.

## 17.3 Externe SVG Fonts

Mit Hilfe des **font-face**-Element können Sie **SVG fonts** referenzieren, die in einem externen SVG Dokument festgelegt wurden. In diesem Fall muß dem Eltern-Element **font** im externen Dokument durch das Attribut ***id*** ein eindeutiger Name zugeordnet werden - wie im folgenden Beispiel.

Beispiel Quellcode

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" >
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<defs>
<font id="ente" horiz-adv-x="54" >
  <font-face
    font-family="Entebbe"
    units-per-em="1000"
    panose-1="0 0 0 0 0 0 0 0 0 0"
    ascent="935"
    descent="-292"
    alphabetic="0">
  </font-face>
  <missing-glyph horiz-adv-x="800"
    d="M50 0V800H750V0H50ZM700 50V750H100V50H700Z" />
  <glyph unicode=" " glyph-name="space" horiz-adv-x="290" />
  <glyph unicode="!" glyph-name="exclam" horiz-adv-x="223"
    d="M86 171L83 197V683Q83 719 96 725T117 731Q141 731 145 709L148 683
    V197Q148 161 135 155T114 149Q90 149 86 171ZM76 4Q61 19 61 40Q61 62 75 76
    T112 91Q119 91 140 83T162 41Q162 30 159 22T144 2T111 -10Q92 -10 76 4Z" />
  <!-- ... -->
  <!-- ... mehr Zeichendefinitionen folgen hier -->
  <!-- ... -->
</font>
</defs>
</svg>
```

Wenn Sie einen externen **SVG font** in einem anderen SVG Dokument verwenden wollen, platzieren Sie das **font-face**-Element direkt im **defs**-Bereich eines SVG Dokuments. Es wird kein **font**-Element als Eltern-Element benötigt.

Das **font-face**-Element muß, selbst wenn es einen externen **SVG Font** referenziert, mindestens das Attribut ***font-family*** enthalten, das den gleichen Wert erhält, wie das ***font-family*** Attribut im externen **font-face**-Element.

Die Referenzierung des externen Fonts können Sie dann durch die folgenden Elemente, die im **font-face**-Bereich platziert werden müssen, und deren Attribute realisieren:

1. Im Element **font-face-name** geben Sie mit dem Attribut ***name*** den Namen der Schriftart an (der Wert aus font-family).
2. Im Element **font-face-src** geben Sie mit dem Attribut ***xlink:href*** die URL zum externen **SVG font** an (in Form dateiname#font-id). Dieses Element entspricht dem Deskriptor src im der CSS 2 @font-face Regel.
3. Im Element **font-face-format**, einem Kind-Element von **font-face-src**, können Sie mit dem Attribut ***string*** das Format des referenzierten Fonts angeben (in unserem Falle natürlich *svg*).

.. und nun das SVG Dokument, das den, im vorangegangenen Beispiel definierten, **SVG font** mit Hilfe des **font-face**-Elements referenziert: Dieses Beispiel funktioniert leider bis dato nur im Batik-Squiggle SVG Browser :-) und noch nicht im Adobe SVG Viewer :-(.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<defs>
<font-face font-family="Entebbe">
  <font-face-src>
    <font-face-name name="Entebbe" />
    <font-face-uri xlink:href="kap17_3.svg#ente">
      <font-face-format string="svg" />
    </font-face-uri>
  </font-face-src>
</font-face>
</defs>

<text x="20" y="60"
  style="font-family:Entebbe;font-size:50px;font-weight:bold;">
  externe !SVG fonts!
  <tspan x="20" dy="25" style="font-size:14px;">
    .. wird vom Adobe SVG Viewer noch nicht unterstützt.
  </tspan>
  <tspan x="20" dy="20" style="font-size:14px;">
    .. aber vom Batik-Squiggle.
  </tspan>
</text>
</svg>
```

Es ist in SVG ebenfalls möglich, mit Hilfe der CSS @font-face Regel innerhalb eines **style**-Bereichs externe **SVG fonts** zu referenzieren. Auch in diesem Fall muß dem Eltern-Element **font** im externen Dokument durch das Attribut ***id*** ein eindeutiger Name zugeordnet werden. Leider funktioniert dieses Feature weder im Adobe SVG Viewer noch im Batik-Squiggle SVG Browser. Nichtsdestotrotz folgt ein Beispiel - das leider nicht funktioniert :-(.

Beispiel Quellcode

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<defs>
<style type="text/css">
<![CDATA[
  @font-face {
    font-family: 'Entebbe';
    src: url('kap17_3.svg#ente') format(svg);
  }
]]>
</style>
</defs>

<text x="20" y="60"
  style="font-family:Entebbe;font-size:50px;font-weight:bold;">
  externe !SVG fonts!
  <tspan x="20" dy="20" style="font-size:14px;">
    .. wird noch nicht unterstützt.
  </tspan>
</text>
</svg>
```

## 17.4 Das font-Element

Das **font**-Element legt den Bereich für den **SVG font** fest und sollte immer im **defs**-Bereich des SVG Dokuments platziert werden. Mit Hilfe des Attributs ***id*** können Sie für das **font**-Element einen eindeutigen Namen festlegen, der benötigt wird, wenn Sie diesen **SVG font** aus einem anderen SVG Dokument referenzieren wollen.

Das **font**-Element muß genau ein Kind-Element **font-face** beinhalten, das die Schriftart auswählt, d.h. die verschiedenen Schriftcharakteristika Ihres **SVG fonts** festlegt.

Weitere Attribute für das font-Element:

***units-per-em*** Voreinstellung: *1000*. Mit diesem Attribut legen Sie die Rastergröße des **em squares** fest. Übliche Rastergrößen und somit mögliche Werte sind 250 (Intellifont), 1000 (Type 1) oder 2048 (TrueType, TrueTypeGX, Open-Type).

***horiz-origin-x*** Voreinstellung: *0*. Die Standard-x-Koordinate im Koordinatensystem des Schriftzeichens, die bei horizontal ausgerichtetem Text für das Zeichen verwendet wird.

***horiz-origin-y*** Voreinstellung: *0*. Die Standard-y-Koordinate im Koordinatensystem des Schriftzeichens, die bei horizontal ausgerichtetem Text für das Zeichen verwendet wird.

***horiz-adv-x*** Voreinstellung: keine. Wenn ein Schriftzeichen bei horizontal ausgerichtetem Text auf den Bildschirm gemalt wurde, muß, bevor ein weiteres Schriftzeichen platziert werden kann, der "virtuelle Cursor" zunächst um eine bestimmt Länge vorgerückt werden. Die Standard-Länge dieses Vorrückens wird durch das Attribut ***horiz-adv-x*** festgelegt.

***vert-adv-x*** Voreinstellung: die Hälfte des bei ***horiz-adv-x*** verwendet Wertes. Die Standard-x-Koordinate im Koordinatensystem des Schriftzeichens, die bei vertikal ausgerichtetem Text für das Zeichen verwendet wird.

***vert-origin-y*** Voreinstellung: Wert des ***ascent***-Attributs im **font-face**-Element. Die Standard-y-Koordinate im Koordinatensystem des Schriftzeichens, die bei vertikal ausgerichtetem Text für das Zeichen verwendet wird.

***vert-adv-y*** Voreinstellung: *1em*. Wenn ein Schriftzeichen bei vertikal ausgerichtetem Text auf den Bildschirm gemalt wurde, muß, bevor ein weiteres Schriftzeichen platziert werden kann, der "virtuelle Cursor" zunächst um eine bestimmt Länge vertikal vorgerückt werden. Die Standard-Länge dieses Vorrückens wird durch das Attribut ***vert-adv-x*** festgelegt.

## 17.5 Beschreibung eines font - das Element font-face

Durch das Element **font-face** müssen Sie Ihrem **SVG font** eine Schriftbeschreibung zuordnen.

Die Schriftbeschreibung wird in SVG durch Attribute realisiert, die den CSS 2 Eigenschaften zum gleichen Thema entsprechen und als Deskriptoren bezeichnet werden. Sie wird der internen Schriftdatenbank hinzugefügt und dann verwendet, um die entsprechenden Schriftdaten auszuwählen.

Schriftdaten sind die abstrakten Daten, die für die Formatierung von Text und die Darstellung der abstrakten Schriftzeichen vom user agent benötigt werden. Anders formuliert sind Schriftdaten also existente Daten-Ressourcen in Form von Schriftartendateien.

Die Schriftbeschreibung enthält also Deskriptoren, die die Schriftdaten näher beschreiben. Mit Hilfe dieser Deskriptoren bzw. Attribute können Sie Werte zur Schriftspezifikation oder zu bestimmtem Charakteristika der Schrift festlegen. Dabei kann die Anzahl der Deskriptoren einer Schriftbeschreibung d.h. die Anzahl der Attribute in einem **font-face**-Element variieren, es muß jedoch mindestens der Schriftname (***font-family***) angegeben werden.

Im folgenden die möglichen Attribute (Deskriptoren) für das **font-face**-Element. Beachten Sie: die meisten Attribute des **font-face**-Elements entsprechen den gleichnamigen CSS 2 Deskriptoren innerhalb einer CSS 2 @font-face Regel. Die möglichen Werte dieser Attribute (Deskriptoren) entsprechen grundsätzlich den möglichen Werten der gleichnamigen CSS 2 Eigenschaften, welche Sie in SVG z.B. innerhalb der Elemente **text** oder **tspan** verwenden können.

***font-family*** Voreinstellung: keine. Deskriptor zu Angabe des Schriftfamiliennamens.

***font-style*** Voreinstellung: *all*. Deskriptor für den Schriftstil.

***font-variant*** Voreinstellung: *normal*. Deskriptor für die Kapitälchen-Variante (kleine Großbuchstaben) einer Schrift.

***font-weight*** Voreinstellung: *all*. Deskriptor für die Schriftgewichtung, d.h. die Dicke der Schrift.

***font-stretch*** Voreinstellung: *normal*. Deskriptor für komprimierte oder gedehnte Schrift.

***font-size*** Voreinstellung: keine. Deskriptor für die Schriftgröße.

***unicode-range*** Voreinstellung: *U+0-10FFFF*. Deskriptor für den Bereich der ISO 10646-Zeichen (UNICODE), welchen die Schrift verwenden kann. Weitere Informationen zu UNICODE finden Sie bei www.unicode.org.

***units-per-em*** Voreinstellung: *1000*. Deskriptor für die Festlegung der Rastergröße des **em squares** fest. Übliche Rastergrößen und somit mögliche Werte sind 250 (Intellifont), 1000 (Type 1) oder 2048 (TrueType, TrueTypeGX, Open-Type). Der Wert dieses Attributs im **font-face**-Element überschreibt einen möglicherweise bereits gesetzten Wert im **font**-Element.

***panose-1*** Voreinstellung: *0 0 0 0 0 0 0 0 0 0*. Deskriptor für die Panose-1-Nummer. Der Wert für dieses Attribut besteht aus 10 dezimalen Integer-Zahlen, die durch Leerzeichnen voneinander getrennt werden. Eine Erläuterung zu Panose-1-Zahlen und Ihrer Bedeutung finden Sie unter www.html-world.de/program/css_pano.htm.

***stemv*** Voreinstellung: keine. Deskriptor für die vertikale Strichbreite der Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***stemh*** Voreinstellung: keine. Deskriptor für die horizontale Strichbreite der Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***slope*** Voreinstellung: keine. Deskriptor für den vertikalen Strichwinkel der Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***cap-height*** Voreinstellung: keine. Deskriptor für die Höhe von Großbuchstaben der Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***x-height*** Voreinstellung: keine. Deskriptor für die Höhe von Kleinbuchstaben der Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***accent-height*** Voreinstellung: Wert des ***ascent***-Attributs. Deskriptor für die maximale Höhe der Schrift mit Berücksichtigung der Akzente. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***ascent*** Voreinstellung: Differenz der Attributwerte von ***units-per-em*** und ***vert-origin-y*** des **font**-Elements. Deskriptor für die maximale Höhe der Schrift ohne Berücksichtigung der Akzente. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***descent*** Voreinstellung: Wert des Attributs ***vert-origin-y*** des **font**-Elements. Deskriptor für die maximale Tiefe der Schrift ohne Berücksichtigung der Akzente. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***widths*** Voreinstellung: keine. Deskriptor um Breiten für die einzelnen Zeichen festzulegen. Mögliche Werte: eine durch Komma getrennte Liste von UNICODE-Bereichswerten gefolgt von Breitenangaben. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***bbox*** Voreinstellung: keine. Deskriptor für die maximale Größe der umschließenden Box der Schrift. Mögliche Werte: 4 Zahlen, wobei die ersten beiden Zahlen den Koordinaten-Wert der linken unteren Ecke und die folgendne zwei Zahlen den Koordinaten-Wert der rechten oberen Ecke der Box festlegen. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***ideographic*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für horizontal ausgerichtete ideographische Schrift (Bilderschrift). Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***alphabetic*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für horizontal ausgerichtete alphabetische Schrift. Entspricht dem CSS 2 ***baseline*** Deskriptor. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***mathematical*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für horizontal ausgerichtete mathematische Schrift. Entspricht dem CSS 2 ***mathline*** Deskriptor. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***hanging*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für horizontal ausgerichtete hängende Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***v-ideographic*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für vertikal ausgerichtete ideographische Schrift (Bilderschrift). Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***v-alphabetic*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für verikal ausgerichtete alphabetische Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***v-mathematical*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für vertikal ausgerichtete mathematische Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***v-hanging*** Voreinstellung: keine. Deskriptor für die Grundlinienposition für vertikal ausgerichtete hängende Schrift. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***underline-position*** Voreinstellung: keine. Deskriptor für die ideale Positionierung eines Unterstrichs. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***underline-thickness*** Voreinstellung: keine. Deskriptor für die ideale Dicke eines Unterstrichs. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***strikethrough-position*** Voreinstellung: keine. Deskriptor für die ideale Positionierung einer durchstreichenden Linie. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***strikethrough-thickness*** Voreinstellung: keine. Deskriptor für die ideale Dicke einer durchstreichenden Linie. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***overline-position*** Voreinstellung: keine. Deskriptor für die ideale Positionierung eines Oberstrichs. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

***overline-thickness*** Voreinstellung: keine. Deskriptor für die ideale Dicke eines Oberstrichs. Um diesen Deskriptor verwenden zu können, muß auch der Deskriptor ***units-per-em*** gesetzt sein.

## 17.6 Zeichen definieren - die Elemente missing-glyph und glyph

Mit dem Element **missing-glyph** wird die Grafik für ein Schriftzeichen festgelegt, das vom user agent verwendet wird, wenn das eigentlich gewünschte Zeichen in den Schriftdaten nicht definiert ist und somit nicht vom user agent dargestellt werden kann. Man könnte das, durch **missing-glyph** festgelegt Zeichen, auch vereinfacht als "Dummy-Schriftzeichen" bezeichnen.

Das **glyph**-Element legt die Grafik für ein Schriftzeichen fest, das dargestellt werden kann.

Die Grafik für ein Schriftzeichen kann innerhalb dieser beiden Elemente durch zwei verschiedenen Methoden bestimmt werden:

1. durch einen einzelnen Pfad als Wert für das Attribut ***d*** im Element **glyph** oder **missing-glyph**
2. oder indem Sie innerhalb eines, durch ein öffnendes und ein schließendes Element festgelegten **glyph**- oder **missing-glyph**-Bereichs, SVG-Elemente für Grundformen oder Pfade verwenden.

Empfehlenswert ist die Verwendung des ***d***-Attributs, da in diesem Fall die Zeichen vom user agent in der gleichen Art und Weise auf den Bildschirm gemalt werden, wie es bei System-Schriftarten der Fall ist.

Allerdings bietet die Verwendung von SVG-Elementen für die Festlegung des Zeiches mehr grafische Möglichkeiten, z.B. könnten Sie für den "i"-Punkt des Zeichens "i" eine andere Farbe definieren als für den Rest des "i"-Zeiches. Falls Sie irgendwelche Längenangaben für das referenzierende **text**-Element gesetzt haben (z.B. mit ***stroke-width***), kann dies jedoch zu überraschenden Anzeigeergebnissen führen.

Für beide Elemente **missing-glyph** und **glyph** können die gleichen Attribute verwendet werden:

***unicode*** Voreinstellung: keine. Als Wert dieses Attributs geben Sie das Unicode-Zeichen an, das mit Ihrem Zeichen korrespondiert. Die Angabe des Unicode-Zeiches kann durch das Zeichen selbst, die dezimale oder die hexadezimale Notation erfolgen. Dieses Attribut sollte immer gesetzt werden.

***glyph-name*** Voreinstellung: keine. Mit Hilfe dieses Attributs können Sie dem **glyph**-Element einen eindeutigen Namen zuweisen. Über diesen Namen kann das Zeichen dann von von den Elementen **hkern** und **vkern** (die das Kerning von Zeichen bestimmen) referenziert werden.

***d*** Voreinstellung: keine. Mit Hilfe dieses Attributs legen Sie den Umriss des Zeichens fest, d.h. sie bestimmen wie das Zeichen gemalt wird. Das Attribut ***d*** entspricht dem gleichnamigen Attribut innerhalb eines SVG **path**-Elements.

***orientation*** Voreinstellung: v,h Mit ***orientation*** können Sie festlegen, ob das Zeichen horizontal und/oder vertikal dargestellt werden kann. Mögliche Werte sind *h* (nur horizontal) und *v* (nur vertikal). Wenn Sie dieses Attribut nicht verwenden sind standardmäßing beide Varianten möglich.

***arabic-form*** Voreinstellung: keine. Mit diesem Attribut können Sie Grundeinstellungen für arabische Schriftzeichen vornehmen. Mögliche Werte sind: *initial*, *medial*, *terminal* und *isolated*.

***lang*** Voreinstellung: keine. Mit ***lang*** legen Sie die Sprache des Zeichens fest. Mögliche Werte sind die gebräuchlichen Sprache-Codes wie z.B. *de*, *en*, *fr*, *it*, undsoweiter.

***horiz-adv-x*** Voreinstellung: Wert des Attributs ***horiz-adv-x*** im **font**-Element. Wenn ein Schriftzeichen bei horizontal ausgerichtetem Text auf den Bildschirm gemalt wurde, muß, bevor ein weiteres Schriftzeichen platziert werden kann, der "virtuelle Cursor" zunächst um eine bestimmt Länge vorgerückt werden. Die Standard-Länge dieses Vorrückens wird durch das Attribut ***horiz-adv-x*** festgelegt.

***vert-origin-x*** Voreinstellung: Wert des Attributs ***vert-origin-x*** im **font**-Element. Die Standard-x-Koordinate im Koordinatensystem des Schriftzeichens, die bei vertikal ausgerichtetem Text für das Zeichen verwendet wird.

***vert-origin-y*** Voreinstellung: Wert des Attributs ***vert-origin-y*** im **font**-Element. Die Standard-y-Koordinate im Koordinatensystem des Schriftzeichens, die bei vertikal ausgerichtetem Text für das Zeichen verwendet wird.

***vert-adv-y*** Voreinstellung: Wert des Attributs ***vert-adv-y*** im **font**-Element. Wenn ein Schriftzeichen bei vertikal ausgerichtetem Text auf den Bildschirm gemalt wurde, muß, bevor ein weiteres Schriftzeichen platziert werden kann, der "virtuelle Cursor" zunächst um eine bestimmt Länge vertikal vorgerückt werden. Die Standard-Länge dieses Vorrückens wird durch das Attribut ***vert-adv-x*** festgelegt.

## 17.7 Kerning - die Elemente hkern und vkern

Wenn zwei Zeichen derselben Schriftart nebeneinander dargestellt werden, kann es bei bestimmten Zeichenpaaren von Vorteil sein, den Abstand zwischen diesen beiden Zeichen zu regulieren, um eine opisch ansprechendere Darstellung dieses Zeichenpaares zu erhalten.

Ein typischen Beispiel ist das Zeichenpärchen "Va". Die Darstellung dieser beiden Zeichen nebeneinander sieht deutlich besser aus, wenn das kleine "a" näher an das große "V" herangerückt wird.

Dieses Justieren zweier bestimmter Zeichen innerhalb von Zeichenpaaren wird als **Kerning** bezeichnet und kann in SVG durch die Elemente **hkern** und **vkern** festgelegt werden. Mit **hkern** können Sie ein Kerning für zwei Zeichen definieren, die horizontal nebeneinander ausgerichtet sind, mit **vkern** können Sie ein Kerning für zwei Zeichen definieren, die vertikal nebeneinander ausgerichtet sind. Dabei ist das erste Zeichen eines Kerning-Paares ist immer das linke bzw. das obere Zeichen.

Die beiden Zeichen innerhalb eines solchen Kerning-Paares können auf zwei verschiedene Arten bestimmt werden:

- Entweder durch eine Unicode-Angabe, die dem Wert des Attributs ***unicode*** im **glyph**-Element entspricht oder
- durch eine Referenz auf den Namen des **glyph**-Elements, der dort durch das Attribut ***glyph-name*** festgelegt wurde.

Für beide Elemente **hkern** und **vkern** können die gleichen Attribute verwendet werden:

***u1*** Voreinstellung: keine. Mit diesem Attribut legen Sie das erste Zeichen eines Kerning-Paares fest. Es erwartet als Wert eine Unicode-Angabe, die dem Wert des Attributs ***unicode*** im **glyph**-Element entspricht.

***g1*** Voreinstellung: keine. Mit diesem Attribut legen Sie ebenfalls das erste Zeichen eines Kerning-Paares fest. Es erwartet aber den mit ***glyph-name*** festgelegten Namen eines **glyph**-Elements

***u2*** Voreinstellung: keine. Gegenpart zu ***u1***. Legt das zweite Zeichen des Kerning-Paares fest.

***g2*** Voreinstellung: keine. Gegenpart zu ***g1***. Legt das zweite Zeichen des Kerning-Paares fest.

***k*** Voreinstellung: keine. Mit diesem Attribut legen Sie einen Betrag fest, um den der Abstand der beiden Zeichen eines Kerning-Paares vermindert werden soll. Der Wert bezieht sich auf das Koordinatensystem des Zeichens. Dieses Attribut muß gesetzt werden.

Weiterhin muß mindestens ein Attribut ***u1*** oder ***g1*** und ein Attribut ***u2*** oder ***g2*** verwendet werden.