6. Mediendokumente

Weiterführende Literatur:
J. David Eisenberg: SVG Essentials, O'Reilly 2002

Vektor-Grafikformate für das Web

- Nachteile von Bitmap-basierten Bildern:
  - Grosse Dateien; Kompression führt zu Verlusten
  - Maximale Auflösung unverränderlich festgelegt
  - Hyperlinks in Bildern (image maps) schwierig zu realisieren
  - Animation und Interaktion nicht möglich
  - Trennung von Inhalt und Präsentation nicht möglich
    - » vgl. XML + CSS
- Vektorgrafik:
  - Bild beschrieben durch seine grafischen Objekte
- Anwendungsbereiche für Vektorgrafik:
  - Technische Zeichnungen, Illustrationen
  - Logos, Icons

Scalable Vector Graphics (SVG): Geschichte

- Erstes weit verbreitetes Vektorgrafikformat im Web:
  - CGM (Computer Graphics Metafile): ISO-Standard seit 1987
- 1997: Suche nach einem Vektorgrafikformat durch das W3C
  - Nachteile von CGM: Nicht kompatibel mit CSS-Stilen, keine Links
- 1998: Ausschreibung durch das W3C für XML-basierte Sprache für Vektorgrafik, vier Einreichungen:
  - Web Schematics (abgeleitet von troff pic)
  - Precision Graphics Markup Language (PGML) (PostScript-orientiert)
  - Vector Markup Language (VML) (PowerPoint-orientiert)
  - DrawML
- 2001: W3C Recommendation SVG
  - Elemente aus allen Vorschlägen
  - Stark beeinflusst von PGML
  - Starker industrieller Befürworter von SVG: Adobe

Grundstruktur einer SVG-Datei

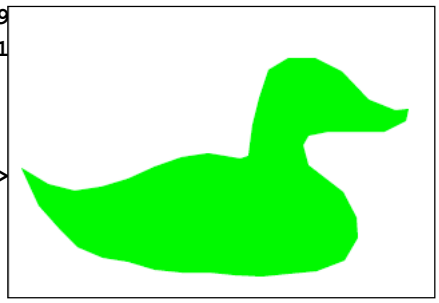
```
<?xml version="1.0" encoding="ISO-8859-1"
  standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904
  /DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">

  ... SVG-Inhalte ...

</svg>
```

Eine erste SVG-Grafik

```
<svg width="320" height="220">
  <rect width="320" height="220" fill="white"
  stroke="black"/>
    <g transform="translate(10 10)">
      <g stroke="none" fill="lime">
        <path d="M 0 112 L 20 124 L 40 129 L 60 126 L 80 120
           L 100 111 L 120 104 L 140 101 L 164 105 L 170 103
           L 173 80 L 178 60 L 185 39 L 200 30 L 220 30
           L 260 61 L 280 69 L 29
           L 250 85 L 230 85 L 21
           L 228 120 L 241 130 L
           L 221 189 L 200 191 L
           L 120 190 L 100 188 L
           L 30 159 L 13 140 Z"/>
        </g>
    </g>
</svg>
```

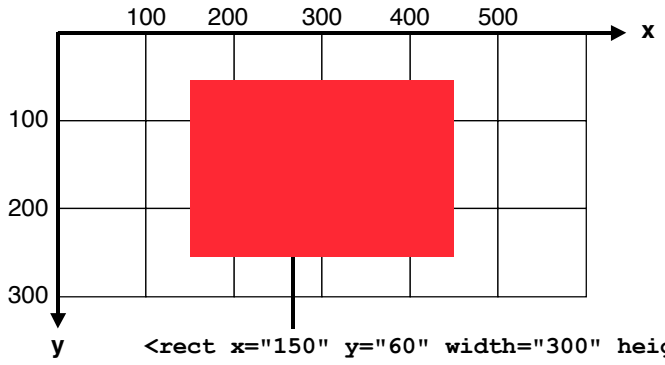


Software zur Darstellung von SVG

- Übersicht unter:
  - http://www.w3.org/Graphics/SVG/SVG-Implementations
- Eigenständige SVG-Viewer
  - für alle Plattformen, meist in Java geschrieben, teilweise frei
- SVG-Viewer für mobile Geräte
  - z.B. für PalmOS, PocketPC, Symbian
- Plug-In für Web-Browser
  - Adobe SVGViewer Plugin
    - » für alle gängigen Plattformen und Browser
- Grafiksoftware mit SVG-Export
  - z.B. Adobe Illustrator, CorelDraw, JViews

Koordinaten

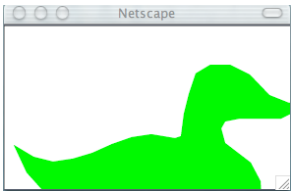
- Grafik entsteht auf einer unbegrenzt grossen Leinwand *(canvas)*
- Punkte werden mit x- und y-Koordinaten beschrieben
- In der Computergrafik verläuft die y-Achse von Oben nach Unten!
- Wie beim Malen: Neue Elemente überdecken vorhandene Elemente



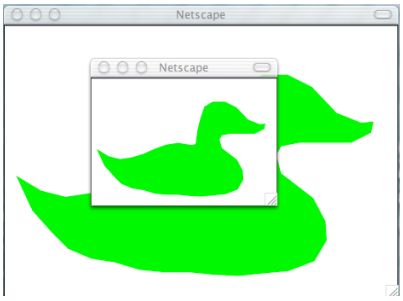

```
<rect x="150" y="60" width="300" height="200">
```


Skalierbarkeit mittels "ViewBox"

- Größenangabe durch Höhe und Breite:
  ```
  <svg width="320" height="220">
  ```
  - Absolute Grösse in Pixel
  - Grafik wird bei Verkleinerung des Fensters abgeschnitten



- Größenangabe durch Sichtfenster *(viewBox)*:
  ```
  <svg viewBox="0 0 320 220">
  ```
  - Anforderung eines rechteckigen sichtbaren Bereichs (*x-oben-links y-oben-links x-unten-rechts y-unten-rechts*)
  - Grafik wird bei Verkleinerung /Vergrösserung des Fensters skaliert (variable Abbildung der Bildpixel auf Darstellungspixel)



Konzepte: Detailinformationen zum Rendering

- Für die präzise Darstellung eines Grafikelements sind Zusatzinformationen notwendig:
  - Farbe, Strichstärke, Linienstil (z.B. gestrichelt), ...
- Häufig in Zeichenprogrammen und Grafik-Softwareschnittstellen:
  - Werkzeug-Metapher: Zuerst Werkzeug wählen, dann Aktion ausführen
  - z.B. Pinselstärke = 2pt, Pinselfarbe = rot
  - Einstellungen bleiben bis zur Umstellung erhalten ("aktuelles Werkzeug")
  - Relativ empfindlich gegen Code-Änderungen (Umstellungen)
- Deklarativer Ansatz:
  - Individuelle Attributwerte zu einzelnen Objekten
- Klassifizierender Ansatz:
  - Objekte werden abstrakt klassifiziert
  - Attributwerte werden den Klassen zugeordnet
  - Sehr änderungsfreundlich, geeignet z.B. für technische Zeichnungen (CAD)

Rendering-Attribute in SVG

- Darstellung *(rendering)* eines grafischen Objekts kann mit Attributen beeinflusst werden, z.B.:
  - **`fill`** Füllfarbe
  - **`opacity`** Transparenz
  - **`stroke`** Linienfarbe
  - **`stroke-width`** Linienstärke
  - **`stroke-linecap`** Form von Linienenden
  - **`font-family`** Schriftfamilie
  - **`font-size`** Schriftgrösse
- Angabe der Attribute auf mehreren Wegen möglich:
  - Direkt als Attributwert
  - Über ein **`style`**-Attribut in CSS2-Syntax
  - Über ein CSS2-Stylesheet
- Frage: Gehört bei einem Bild die Farbe eines Elements zum Inhalt oder zur Darstellung?

Beispiel: SVG-Grafik mit Stylesheet

```
<?xml-stylesheet type="text/css" href="renderstyle.css" ?>
<svg viewBox="0 0 300 300">
  <rect width="300" height="300"/>
  <rect class="type1" x="100" y="100" width="100" height="100"/>
  <rect class="type2" x="50" y="50" width="100" height="100"/>
</svg>
```

SVG-Datei

```
rect {stroke:black; fill:white}
rect.type1 {stroke:none; fill:red}
rect.type2 {stroke:black; stroke-width:6; fill:green}

.heavy {stroke:black; stroke-width:10}
```

renderstyle.css

Konzept: "Virtueller Zeichenstift"

- In fast allen Softwareschnittstellen und Ablageformaten für Vektorgrafik:
  - Konzept einer "aktuellen Position"
  - Metapher eines 2-dimensional beweglichen Zeichenwerkzeugs
- Typische Kommandos in der Zeichenstift-Metapher:
  - "move to"
  - Gehe dx, dy Einheiten nach rechts, unten
- Vorteile:
  - Leicht zu verstehen
  - Wenige Grundprimitive für fast alle grafischen Formen
  - Dominierend in Computergrafik-Standards
- Nachteile:
  - Wenig änderungsfreundlich, da abhängig von globaler, impliziter Information
  - Erfordert manchmal konzeptionell unnötige Bewegungen (z.B. "zurück zur Anfangsposition")

Pfade

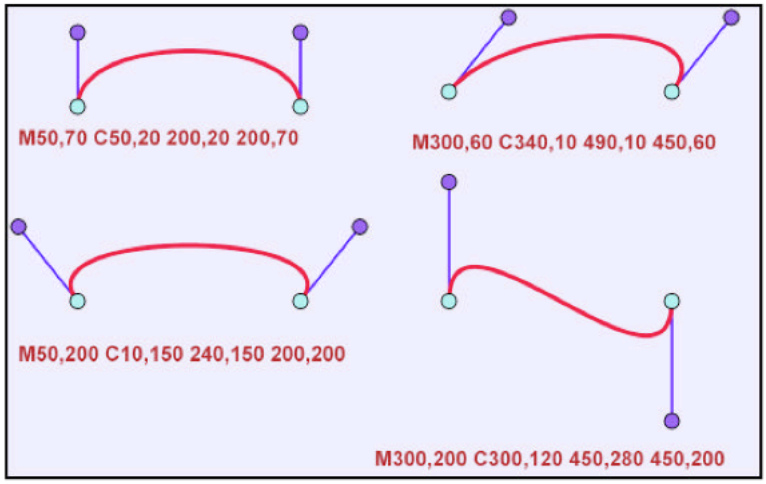
- *Pfad* bedeutet eine Folge von Kommandos zum Zeichnen einer (offenen oder geschlossenen) Kontur
- Viele andere SVG-Tags (z.B. **`<rect>`**) sind Abkürzungen für Pfade
- Pfad-Syntax ist extrem knapp gehalten, um Speicherplatz bei der Übertragung zu sparen
  - Zusätzlich dürfen SVG-Dateien auch (verlustfrei) komprimiert vorliegen (gzip)
- Pfad
  - besteht aus einer Folge (auch einelementig) von Pfadsegmenten
- Pfadsegment
  - Folge von Kommandos, bei denen das erste eine neue "aktuelle Position" bestimmt ("M" = Move to)
- Beispiel (ein Dreieck):

```
<path d="M 0 0 L 100 0 L 50 100 Z">
```


Pfad-Kommandos (Auswahl)

Kommando	Wirkung	Parameter
M	Startpunkt festlegen	x, y
L	Gerade Linie zum angegebenen Punkt	x, y
H	Horizontale Linie bis x	x
V	Vertikale Linie bis y	y
Z	Gerade Linie zurück zum Startpunkt	--
C	Kubische Bezier-Kurve	c1x, x1y, c2x, c2y, x, y

Kubische Bezier-Kurven in SVG



Aus: D.Duce, I.Herman, B.Hopgood: SVG Tutorial

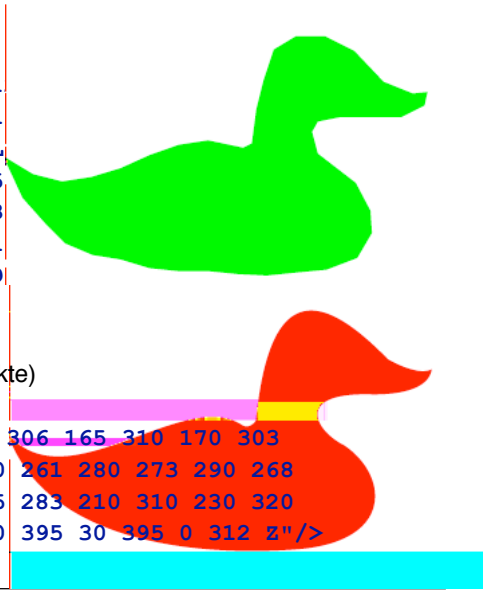
Beispiele für Pfade

- Entenumriss mit Linien (43 Punkte):

```
<path d="M 0 112 L 20 124 L 40 1
 111 L 120 104 L 140 101 L 164 1
 60 L 185 39 L 200 30 L 220 30 L
 L 290 68 L 288 77 L 272 85 L 25
 95 L 215 110 L 228 120 L 241 13
 181 L 221 189 L 200 191 L 180 1
 190 L 100 188 L 80 182 L 61 179
 Z"/>
```

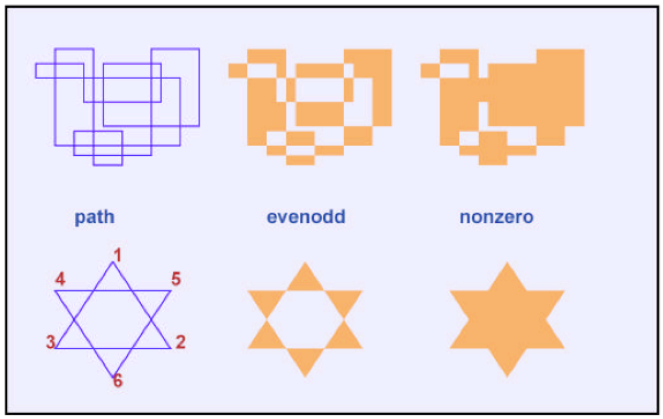
- Entenumriss mit Bezier-Kurven (25 Punkte)

```
<path d="M 0 312
 C 40 360 120 280 160 306 C 160 306 165 310 170 303
 C 180 200 220 220 260 261 C 260 261 280 273 290 268
 C 288 280 272 285 250 285 C 195 283 210 310 230 320
 C 260 340 265 385 200 391 C 150 395 30 395 0 312 Z"/>
```



Füllregeln

- Bei komplexen Pfaden ist nicht mehr klar, welche Flächen mit einer Füllfarbe auszufüllen sind und welche nicht.



Füllregeln (Attribut `fill-rule`)

Idee: Zähle, wie oft Strahlen von einem Punkt in alle Richtungen in das Unendliche den Pfad schneiden

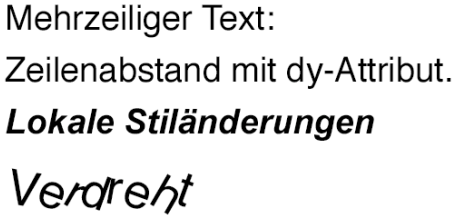
• *nonzero:* Gibt es Schnittpunkte?
• *evenodd:* Heben sich Schnittpunkte gegenseitig auf?

Text

- `<text>`
  - Platzierung von Text auf der Leinwand
  - Koordinaten-Attribute x und y: Linke untere Ecke des ersten Buchstabens
  - Zeichensatz usw. über Attribute oder Stylesheet
- `<tspan>`
  - Untergruppe von Text in einem `<text>`-Element
  - Einheitliche Formatierung (wie `<span>` in HTML)
  - Relative Position zur aktuellen Textposition: Attribute `dx` und `dy`
    - » Typisches Beispiel für "Zeichenstift-Metapher"
- Spezialeffekte
  - Drehen einzelner Buchstaben (`rotate`-Attribut)
  - Text entlang eines beliebigen Pfades (`<textpath>`-Element)

Text in SVG: Beispiel

```
<text x="50" y="20" style="font-size:20pt">
   <tspan x="50" dy="30">Mehrzeiliger Text:</tspan>
   <tspan x="50" dy="30">Zeilenabstand mit
       dy-Attribut.</tspan>
   <tspan x="50" dy="30" style="font-weight:bold;
       font-style:italic">Lokale Stiländerungen</tspan>
</text>
<text x="50" y="150" style="font-size:28">
   <tspan rotate="10 20 30 20 10 20 20">
       Verdreht</tspan>
</text>
```



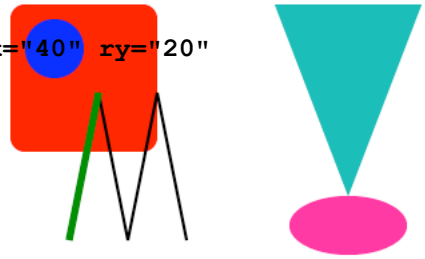
Grundformen von Grafikelementen

- Alle SVG-Grafikelemente sind aus Pfaden und Text ableitbar.
- Zusätzliche häufig verwendete Elemente (Kurzformen):

Elementname	Bedeutung	Attribute
<line>	Linie	x1, y1: Erster Punkt x2, y2: Zweiter Punkt
<polyline>	Folge zusammenhängender Linien	points: Folge von x, y
<polygon>	Polygon	points: Folge von x, y
<rect>	Rechteck	x, y: Linke obere Ecke width: Breite, height: Höhe rx, ry: Radien der Ecken
<circle>	Kreis	cx, cy: Zentrum, r: Radius
<ellipse>	Ellipse	cx, xy: Zentrum rx, ry: Radien

Beispiel: SVG-Grafikelemente

```
<rect x="20" y="20" width="100" height="100" rx="10"
     ry="10" fill="red" stroke="none"/>
<circle cx="50" cy="50" fill="blue" r="20"/>
<polyline points="80,80 100,180 120,80 140,180"
     fill="none" stroke="black" stroke-width="2"/>
<line x1="80" y1="80" x2="60" y2="180" stroke="green"
     stroke-width="5"/>
<polygon points="200,20 300,20 250,150"
     fill="lightseagreen"/>
<ellipse cx="250" cy="170" rx="40" ry="20"
     fill="deeppink"/>
```



Gruppierung und Transformationen

- Gruppe:
  - Grafische Elemente, die eine Einheit bilden und in ihrer relativen Position zueinander erhalten bleiben sollen
  - Sinnvoll,
    - » um einheitliche Attributdefinitionen für die Gruppe festzulegen
    - » um die Gruppe als Gesamteinheit zu verschieben, drehen etc.
  - SVG-Tag **`<g>`**
- Transformationen:
  - Verschieben *(translate)*, drehen *(rotate)*, verzerren *(skew)* oder vergößern/verkleinern *(scale)*
  - Prinzipiell anwendbar auf einzelne Elemente, aber v.a. sinnvoll bei Gruppen
  - SVG-Attribut **`transform`**
    - » Namen für Werte siehe englische Bezeichnungen oben (bei skew zwei Varianten **`skewX`** und **`skewY`**)
    - » jeweils passende Parameter, z.B. **`translate(200, 200)`**

Clipping

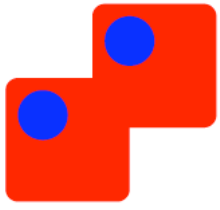
- *Clipping* bedeutet, aus einem Grafikelement einen Teil „auszustanzen“, der einem anderen gegebenen Grafikelement (dem *Clip-Path*) entspricht.
- Clipping in SVG (Beispiel):

```
<clipPath id="myclip">
   <circle cx="250" cy="150" r="150"/>
</clipPath>
<g clip-path="url(#myclip)">
   <rect width="500" height="100"
       x="0" y="0" fill="black"/>
   <rect width="500" height="100"
       x="0" y="100" fill="red"/>
   <rect width="500" height="100"
       x="0" y="200" fill="gold"/>
</g>
```

Symbole und ihre Verwendung

- Man kann in SVGzur wiederholten Verwendung geeignete Symbole definieren (`<symbol>`) und viele Exemplare desselben Symbols erzeugen (`<use>`).
- Beispiel:

```
<symbol id="sym1">
  <rect x="20" y="20" width="100" height="100"
     rx="10" ry="10" fill="red" stroke="none"/>
  <circle cx="50" cy="50" fill="blue" r="20"/>
</symbol>
<use xlink:href="#sym1" x="10" y="10"
<use xlink:href="#sym1" x="80" y="70"
```



Links in SVG und XLink

- Links in SVG funktionieren exakt wie in HTML:

```
<a xlink:href="http://www.mimuc.de">
   <circle cx="50" cy="50" fill="blue" r="20"/>
</a>
```

- Die verwendete Syntax
  (Namespace **`xmlns:xlink="http://www.w3.org/1999/xlink`**)
  entspricht dem *XLink*-Standard des W3C für Links in beliebigen XML-Dokumenten.