8. Vektorgrafik

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

Geänderte Gliederung!

1. Grundbegriffe
2. Digitale Codierung und Übertragung
3. Zeichen und Schrift
4. Signalverarbeitung
5. Ton und Klang
6. Licht, Farbe und Bilder
7. Bewegtbilder
8. Vektorgrafik
9. Web-Dokumente
10. Interaktive Web-Inhalte

Vektor-Grafikformate für das Web

- Nachteile von Bitmap-basierten Bildern:
  - Grosse Dateien; Kompression führt zu Verlusten
  - Maximale Auflösung unveränderlich festgelegt
  - Hyperlinks in Bildern (image maps) schwierig zu realisieren
  - Animation und Interaktion nicht möglich
  - Trennung von Inhalt und Präsentation nicht möglich
    - » Im Gegensatz z.B. zu HTML+CSS

- Vektorgrafik:
  - Bild beschrieben durch seine grafischen Objekte

- Anwendungsbereiche für Vektorgrafik:
  - Technische Zeichnungen, Illustrationen
  - Logos, Icons

Rendering

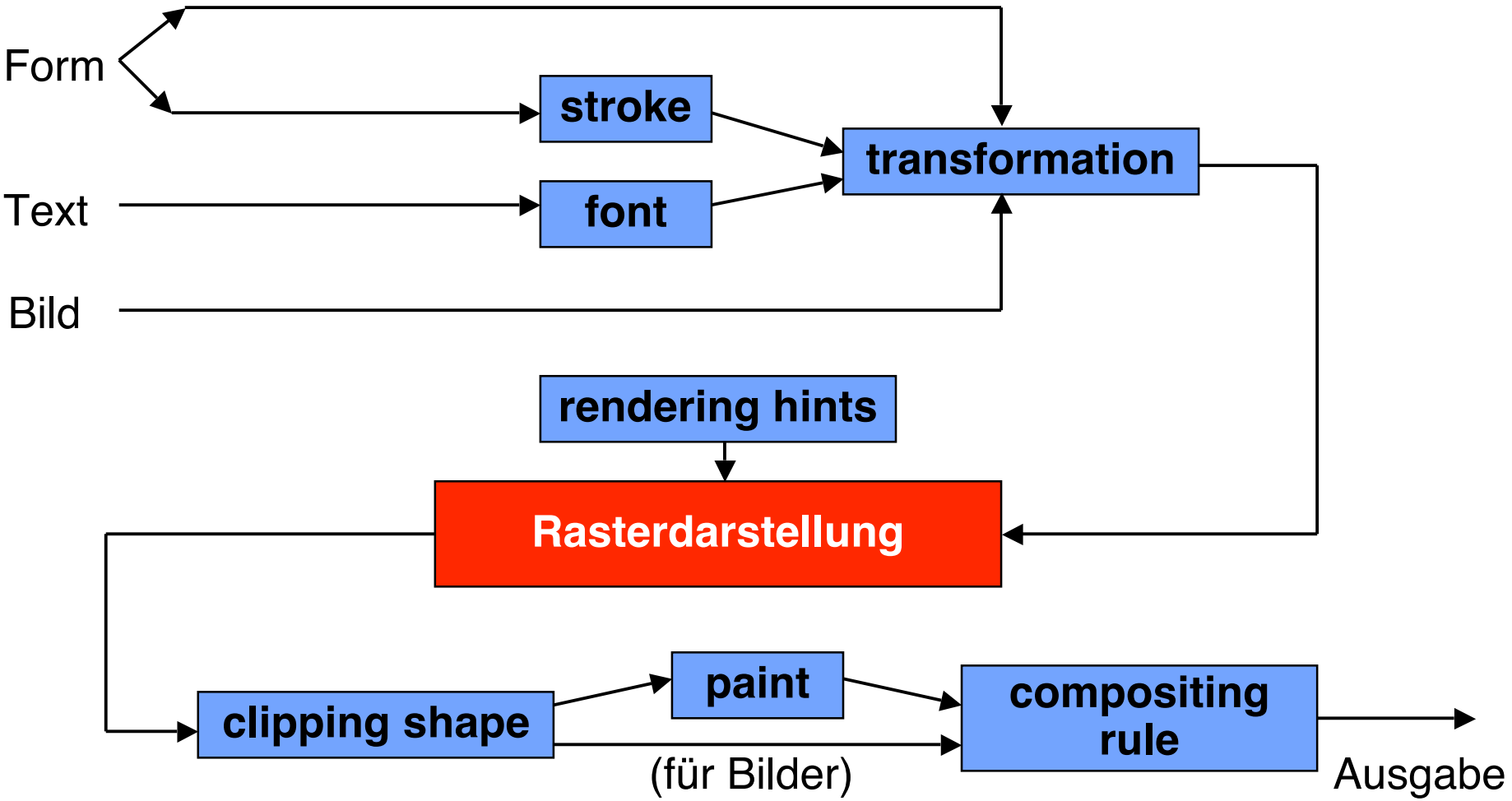
- *Rendering* ist die Umrechnung einer darzustellenden Information in ein Format, das auf einem Ausgabegerät in einer dem Menschen angemessener Form dargestellt werden kann.
- Rendering bei zweidimensionaler (2D-)Grafik:
  - Gegeben eine Ansammlung von Formen, Text und Bildern mit Zusatzinformation (z.B. über Position, Farbe etc.)
  - Ergebnis: Belegung der einzelnen Pixel auf einem Bildschirm oder Drucker

- *Grafikprimitive (graphics primitives):* Formen, Text, Bilder
- *Zeichenfläche (drawing surface):* Ansammlung von Pixeln
- *Rendering Engine:* Programm zur Rendering-Umrechnung

Rendering-Parameter

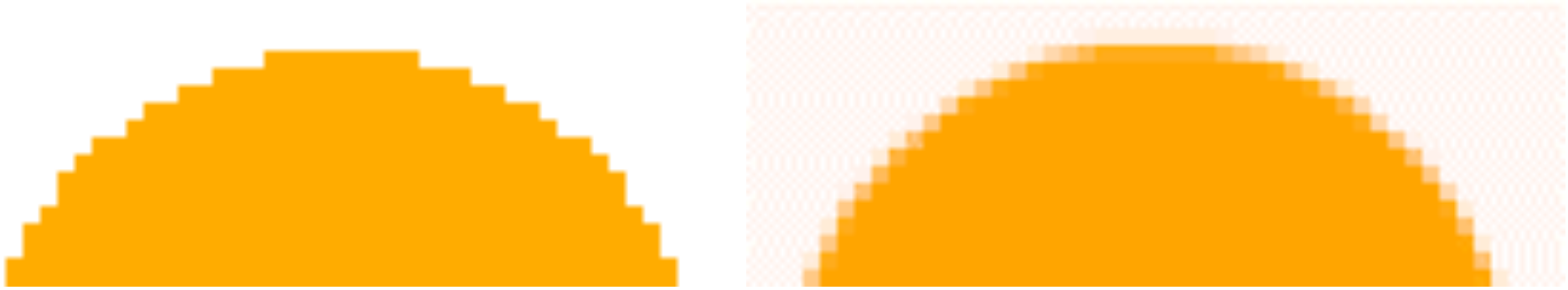
- Jedes primitive Grafikobjekt hat eigene Parameter, die die Darstellung beeinflussen:
  - Form *(shape)*: Ecken, Platzierung etc.
  - Text: Textinhalt
  - Bild *(image)*: Bildinhalt
- Weitere Parameter werden erst in der Rendering Engine festgelegt und beeinflussen ebenfalls die Darstellung:
  - Füllung *(paint):* Wie werden die Pixel für Formen, Linien und Text gefärbt?
  - Strich *(stroke):* Wie werden Linien gezeichnet (Stärke, Strichelung etc.)?
  - Schrift *(font):* Wie wird Text dargestellt (Schriftart, Schriftschnitt etc.)?
  - Transformation: Z.B. Verschieben, drehen, dehnen
  - Überlagerung *(compositing):* Kombination mit anderen Bildern (z.B. Hintergrund)
  - Zuschnitt *(clipping):* Bestimmung eines darzustellenden Ausschnitts
  - *Rendering hints:* Spezialtechniken zur Darstellungsoptimierung

Rendering-Pipeline



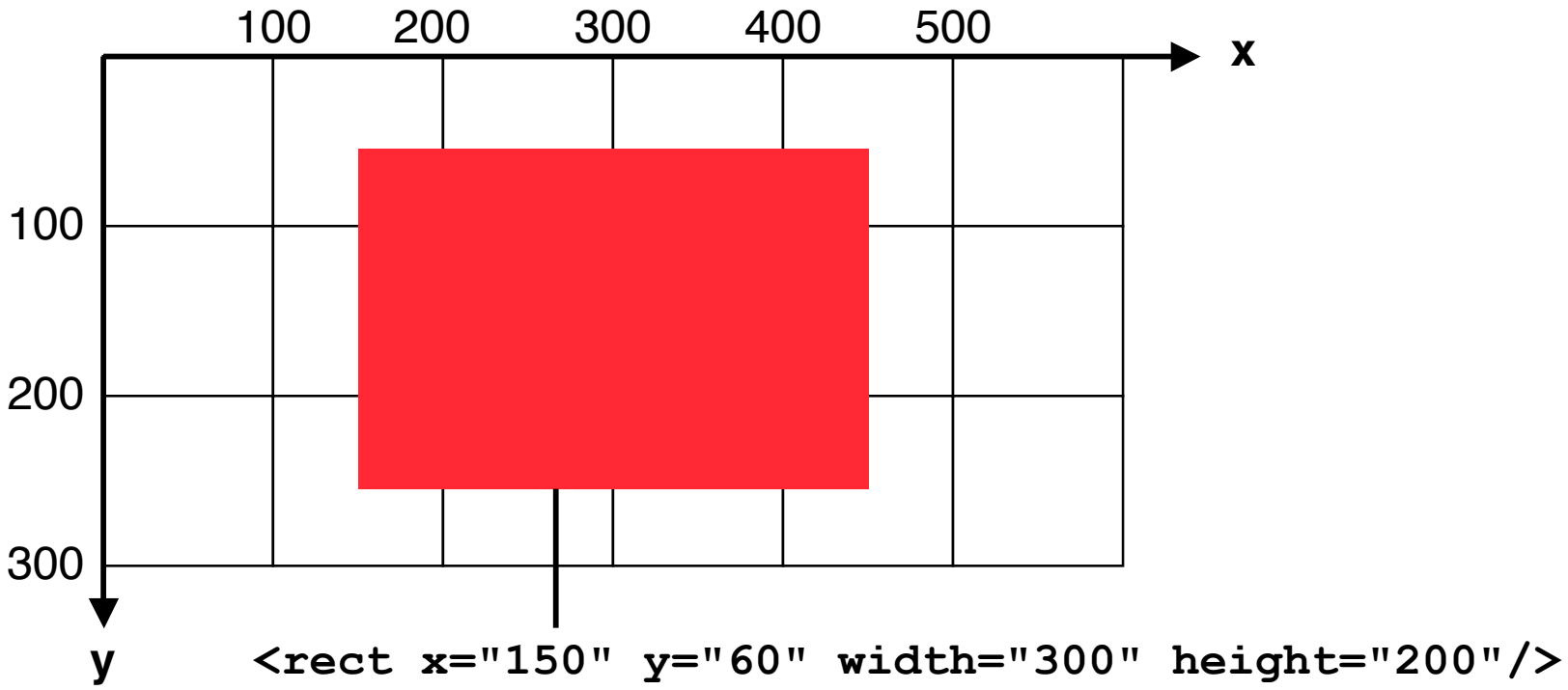
Anti-Aliasing

- Unzureichende Auflösung bei der Wiedergabe erzeugt Artefakte
  - z.B. Treppeneffekte, verschwundene Öffnungen
  - Anwendungsfall des Abtasttheorems...
- Anti-Aliasing-Technik für Farbübergänge und Kanten:
  - Bild einer höheren Auflösung wird künstlich erzeugt
  - Jedes neue (kleine) Pixel wird mit einer Mischfarbe nach Anteil an den beiden beteiligten Flächen belegt
  - Benutzung des Alpha-Kanals, wenn verfügbar (Alphawert = Anteil des Hintergrunds am Pixel)
  - Effekt: Kantenglättung



Koordinaten

- Grafik entsteht auf einer unbegrenzt grossen Leinwand *(canvas)*
- Punkte werden mit x- und y-Koordinaten beschrieben
  - y-Achse bei 2D-Computergrafik nach ***unten***!
- Einfachste Compositing-Regel: Neue Elemente überdecken vorhandene



8. Vektorgrafik

8.1 Basisbegriffe für 2D-Computergrafik

8.2 2D-Vektorgrafik mit SVG

8.3 Ausblick: 3D-Computergrafik mit VRML

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

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 Markup-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
- 2001: "SVG Mobile" als Grafikstandard für Mobiltelefone akzeptiert
- 2003: SVG Version 1.1 verabschiedet, seit 2005 Entwurf für Version 1.2

Grundstruktur einer SVG-Datei

- SVG-Syntax gehört zur Familie der **XML**-Sprachen
  - Mehr zu XML in der nächsten Vorlesung
- Grundidee: Syntax sehr ähnlich zu HTML
- Spezieller Vorspann, dann Hauptelement **<svg>**

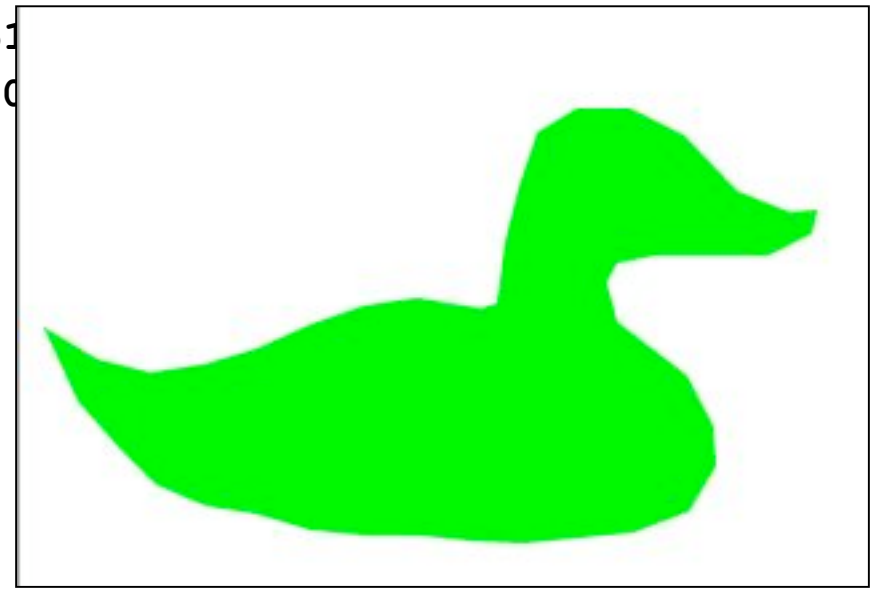
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904
  /DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">

  ... 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 290 68 L 288 77 L 272 85
             L 250 85 L 230 85 L 215 88 L 211 95 L 215 110
             L 228 120 L 241 130 L 251
             L 221 189 L 200 191 L 180
             L 120 190 L 100 188 L 80
             L 30 159 L 13 140 Z"/>
        </g>
    </g>
</svg>
```



Software zur Darstellung von SVG

- Übersicht unter:
  http://www.svgi.org/

- Eigenständige SVG-Viewer
  - für alle Plattformen, meist in Java geschrieben, teilweise frei
- Plug-In für Web-Browser
  - Adobe SVGViewer Plugin
    - » für alle gängigen Plattformen und Browser
  - Direkte Unterstützung in Browsern: Mozilla Firefox ab V. 1.5, Opera ab V. 8
- SVG-Viewer für mobile Geräte
  - z.B. für PalmOS, PocketPC, Symbian
- Erzeugen von SVG:
  - Grafiksoftware mit SVG-Export, z.B. Adobe Illustrator, CorelDraw, JViews
  - Spezielle Editoren, z.B. Inkscape, SVGDraw, Renesis Studio
  - XML-Editoren

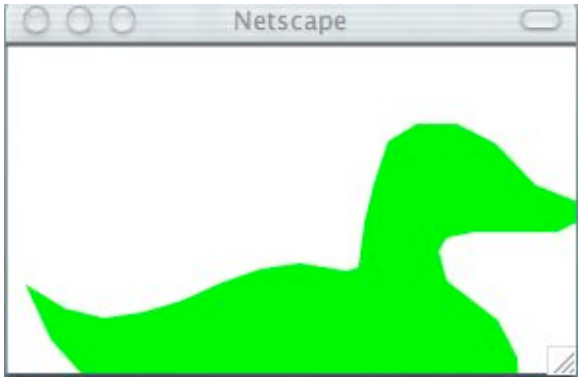
Skalierbarkeit mittels "ViewBox"

- Größenangabe durch Höhe und Breite:

  `<svg width="320" height="220">`

  – Absolute Grösse in Pixel

  – Grafik wird bei Verkleinerung des Fensters abgeschnitten

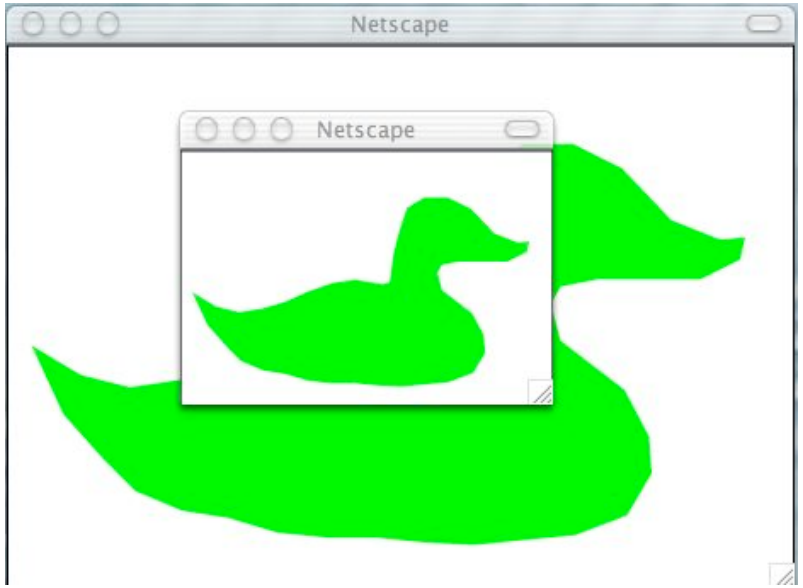


- Größenangabe durch Sichtfenster *(viewBox)*:

  `<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)



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 zu x, y (absolute Position) oder um dx, dy Einheiten nach rechts, unten (relative Position)
- Vorteile:
  - Leicht zu verstehen
  - Wenige Grundprimitive für fast alle grafischen Formen
  - Dominierend in Computergrafik-Standards
- Nachteil:
  - Abschnitte sind keine Einzelobjekte

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 werden (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", "L" = "Line to")
- Beispiel (ein Dreieck):

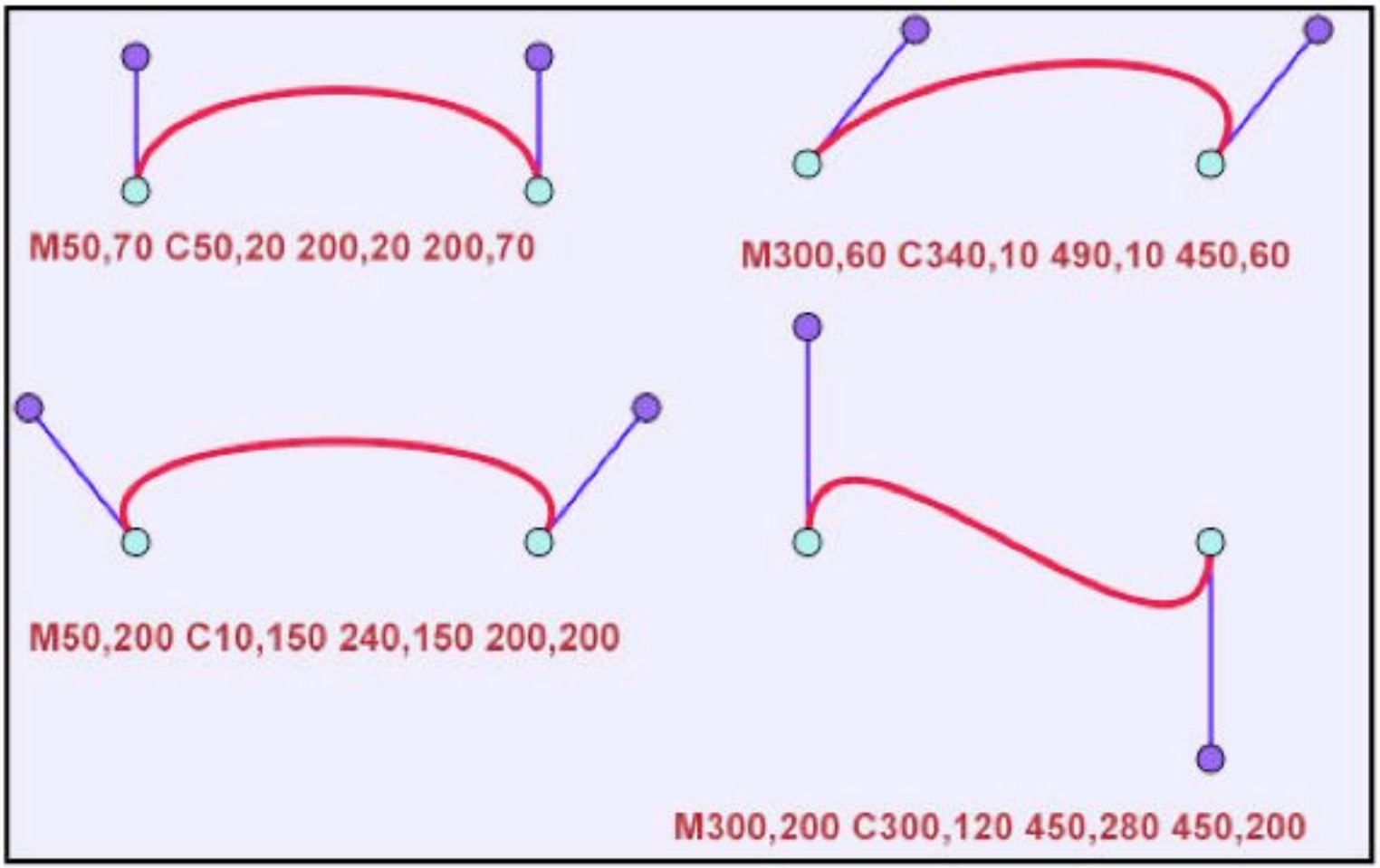
```
<path d="M 0 0 L 100 0 L 50 100 Z">
```

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	--
Q	Quadratische Bezier-Kurve	cx, cy, x, y
C	Kubische Bezier-Kurve	c1x, c1y, c2x, c2y, x, y
A	Elliptischer Kurvenbogen	...

Kleinbuchstaben-Versionen der Kommandos:
relative statt absolute Koordinaten

Kubische Bezier-Kurven in SVG

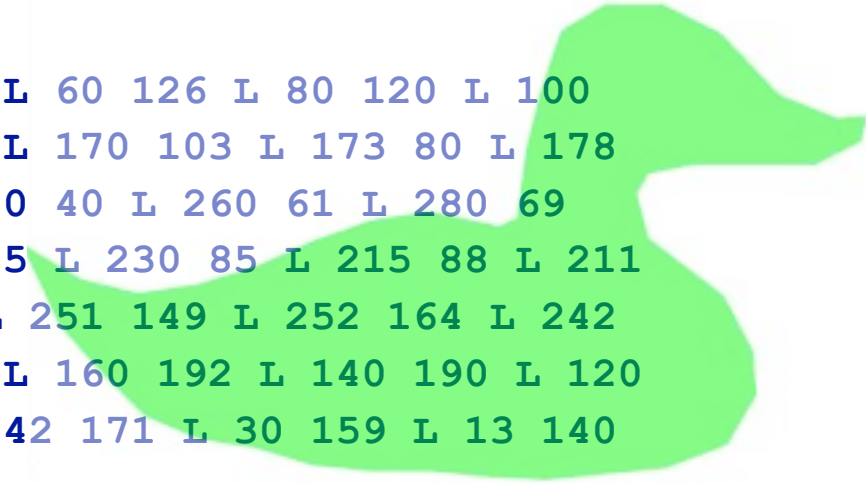


Aus: D.Duce, I.Herman, B.Hopgood: SVG Tutorial

Beispiele für Pfade

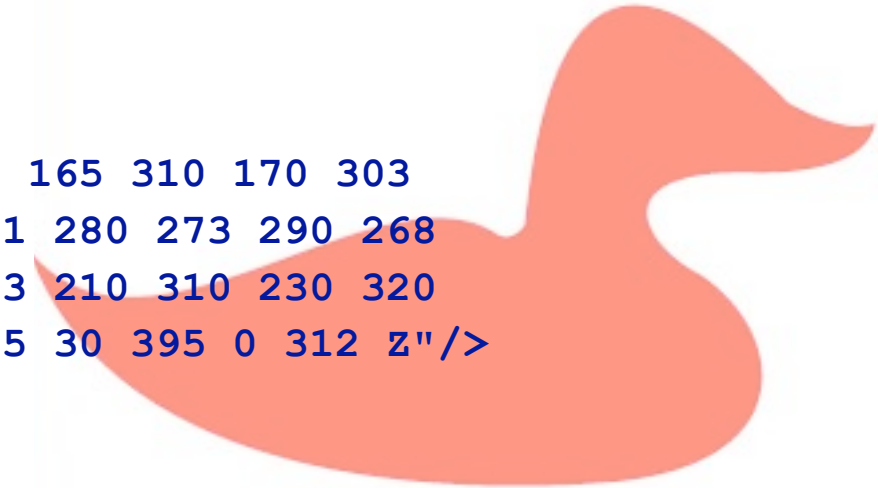
- Entenumriss mit Linien (43 Punkte):

```
<path d="M 0 112 L 20 124 L 40 129 L 60 126 L 80 120 L 100
 111 L 120 104 L 140 101 L 164 106 L 170 103 L 173 80 L 178
 60 L 185 39 L 200 30 L 220 30 L 240 40 L 260 61 L 280 69
 L 290 68 L 288 77 L 272 85 L 250 85 L 230 85 L 215 88 L 211
 95 L 215 110 L 228 120 L 241 130 L 251 149 L 252 164 L 242
 181 L 221 189 L 200 191 L 180 193 L 160 192 L 140 190 L 120
 190 L 100 188 L 80 182 L 61 179 L 42 171 L 30 159 L 13 140
 Z"/>
```



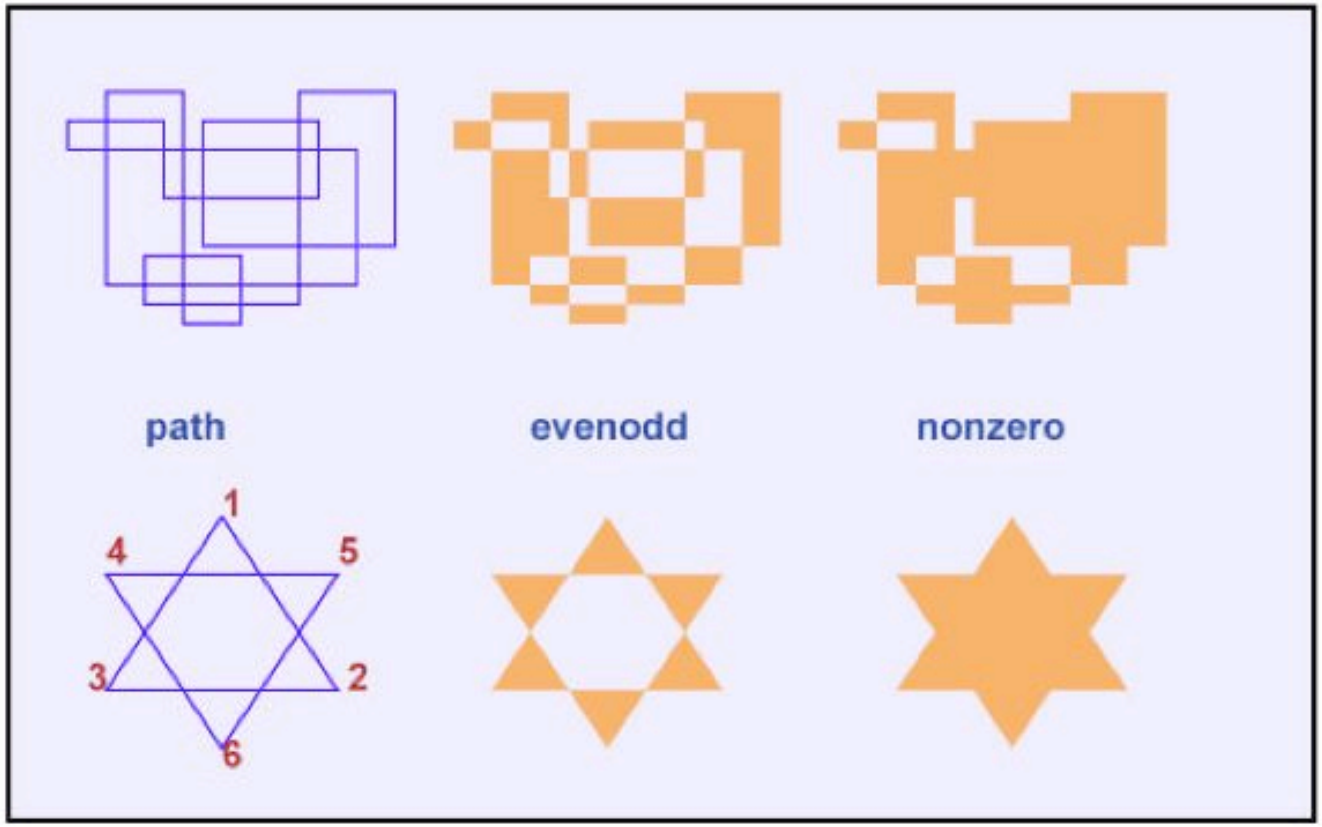
- Entenumriss mit Bezier-Kurven (25 Punkte)

```
<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 (wo ist „innen“ und „außen“?)



Füllregeln (Attribut `fill-rule`)

***nonzero**:* Gibt es Schnittpunkte?

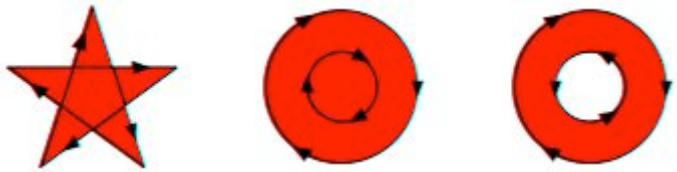
- ***evenodd**:* Heben sich Schnittpunkte gegenseitig auf?

(siehe nächste Folie)

Füllregeln: Evenodd und Nonzero

- Zur Bestimmung, ob ein Punkt „innen“ oder „außen“ liegt:
  - Ziehe einen Strahl vom betrachteten Punkt bis ins unendliche (in beliebiger Richtung)
  - Schnittpunkte des Strahls mit dem Pfad der Form bestimmen „innen“ und „außen“ je nach Füllregel

- Füllregel **evenodd**:
  - Zähle die Anzahl der Schnittpunkte des Strahls mit dem Pfad
  - Bei ungerader Anzahl ist der Punkt „innen“, bei gerader Anzahl ist der Punkt „außen“



- Füllregel **nonzero**:
  - Immer wenn:
    - » Pfad schneidet Strahl von links nach rechts, dann zähle +1
    - » Pfad schneidet Strahl von rechts nach links, dann zähle -1
  - Ist die Summe 0, dann ist der Punkt „außen“, sonst „innen“

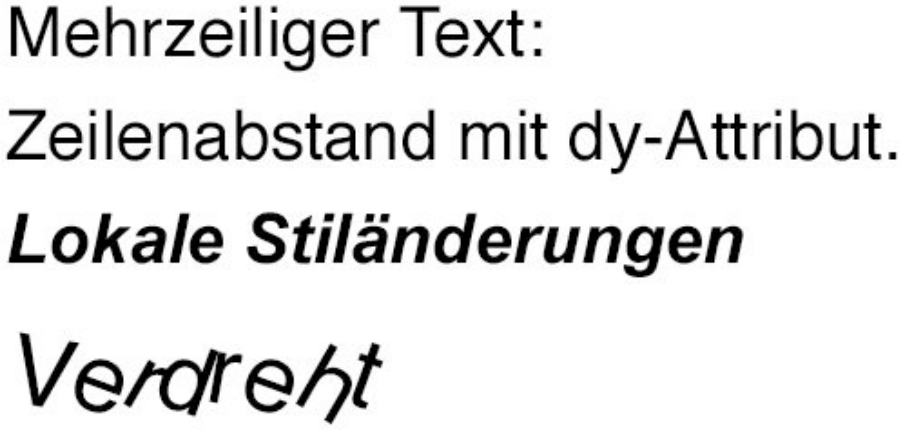
Text

- **<text>**
  - Platzierung von Text auf der Leinwand
  - Koordinaten-Attribute x und y: Linke untere Ecke des ersten Buchstabens
  - Schrift, Größe usw. über Attribute oder Stylesheet
- **<tspan>**
  - Untergruppe von Text in einem **<text>**-Element
  - Einheitliche Formatierung (wie **** 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)
  - (Hinweis: Derzeit (2007) in Firefox nicht implementiert, siehe http://www.mozilla.org/projects/svg/status.html)

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>
```



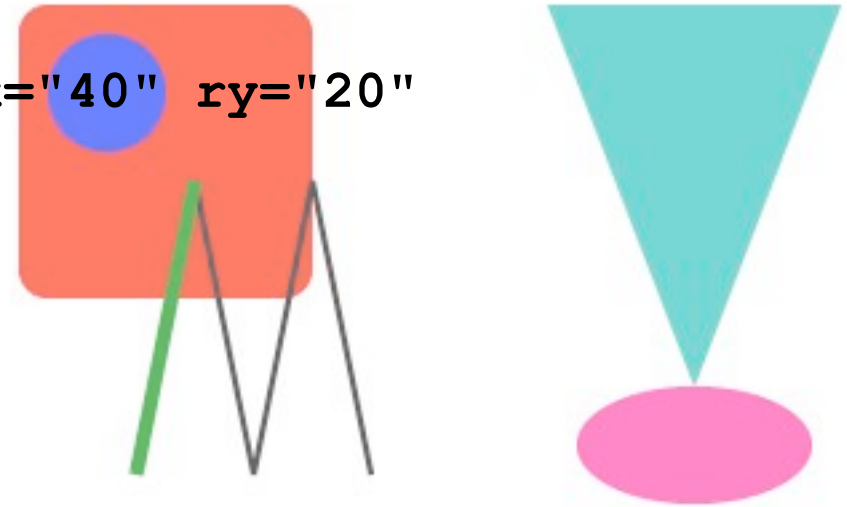
Grundformen von Grafikelementen

- 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, cy: 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 vergrößern/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>
```

Links in SVG und XLink

- Links in SVG funktionieren exakt wie in HTML (anchor tag)
- Beispiel externer Link zu HTML-Dokument:

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

- Die verwendete Syntax (Namensraum **`xlink`**) entspricht dem *XLink*-Standard des W3C für Links in beliebigen XML-Dokumenten.
  - **`http://www.w3.org/1999/xlink`**
- Der Namensraum muss deklariert werden, z.B. so:

```
<svg xmlns=http://www.w3.org/2000/svg
     xmlns:xlink="http://www.w3.org/1999/xlink" … >
```

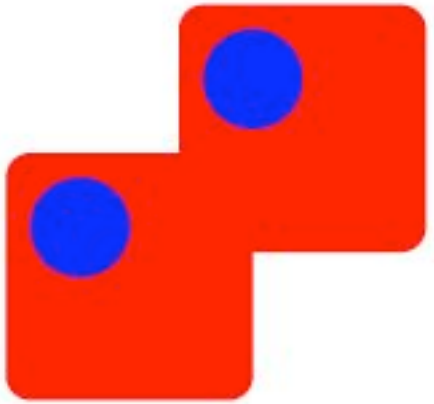
- Details zu Namensräumen siehe nächste Vorlesung!
- Hinweis: Nicht zu verwechseln mit der URI-Syntax (*XPointer*-basiert), z.B. bei Bezug auf Clipping-Pfad

Symbole und ihre Verwendung

- Man kann in SVG zur 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="80" y="10"/>
<use xlink:href="#sym1" x="10" y="70"/>
```

- Das **use**-Element benutzt die gleiche XLink-Syntax wie das **a**-Element (Anker)
  - Verweise auf Symbole über die aus HTML bekannte Syntax für Dokumentfragmente **(#xyz)**



Animationen in SVG

- SVG-Objekte können zeitabhängig verändert werden:
  - **`animate, animateTransform, animateMotion, animateColor, ...`**
- Zeitangaben ähnlich wie in SMIL:
  - **`dur, begin, end`**
- Beispiel **`animateTransform:`**
  - **`type`**-Attribut: Art der Transformation (**`rotate, scale, ...`**)
  - **`values`**-Attribut: Wertebereich des zu verändernden Parameters (Startwert, Zwischenwerte, Endwert)

Beispiel: Uhr in SVG


```
<circle cx="100" cy="100" r="80" style="fill:white;stroke:black"/>
  <text x="130" y="46" style="font-size:15">1</text>
  <text x="154" y="71" style="font-size:15">2</text>
  ...
  <text x="92" y="37" style="font-size:15">12</text>

  <g transform="translate(100 100)">
    <g id="hours">
      <line x1="0" y1="0" x2="0" y2="-35" ...>
        <animateTransform attributeName="transform" type="rotate"
            dur="43200s" values="0;360" repeatCount="indefinite"/>
      </line>
    </g>
    <g id="minutes">
      <line x1="0" y1="0" x2="0" y2="-55" ...>
        <animateTransform attributeName="transform" type="rotate"
            dur="3600s" values="0;360" repeatCount="indefinite"/>
      </line>
    </g>
    <g id="seconds">
      <line x1="0" y1="0" x2="0" y2="-75" ...>
        <animateTransform attributeName="transform" type="rotate"
          dur="60s" values="0;360" repeatCount="indefinite"/>
      </line>
    </g>
  </g>
```

Uhr stellen: mit JavaScript!

8. Vektorgrafik

8.1 Basisbegriffe für 2D-Computergrafik

8.2 2D-Vektorgrafik mit SVG

8.3 Ausblick: 3D-Computergrafik mit VRML 

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

3D-Modellierung

- Objekte als Punktwolken im dreidimensionalen Raum
- Grundprinzipien wie bei 2D-Vektorgrafik, jedoch zusätzlich:
  - 2D-Projektion zur Darstellung:
    - » Kamera in 3D-Welt
    - » Perspektive
    - » Verdeckung
  - Oberflächeneigenschaften von Objekten
  - Beleuchtunqsquellen
- Rendering von 3D-Objekten
  - Als Drahtmodell oder Polygonmodell
  - Berechnung von Schattierung abhängig vom Lichteinfall

Virtual Reality Modeling Language VRML

- Beispiel einer Sprache für 3D-Grafikdokumente
- Skriptsprache und Austauschformat zur Beschreibung von 3D-Welten
  - Auf den Einsatz im Internet ausgelegt
  - Vektor-Grafikformat
- VRML hat *keine* XML-Syntax!
  - Nachfolger von VRML: „X3D“ ist XML-Sprache
  - 1997: VRML wird Internationaler Standard ISO-14772
    - » Meist als „VRML 97“ bezeichnet, weitgehend identisch zu VRML 2.0
- Dateiextension:
  - .wrl (wie „world“) und .wrz (= .wrl.gz komprimierte Variante)
- Mäßige praktische Verbreitung
  - Verschiedene proprietäre Formate häufig genutzt

Beispiel einer VRML-Szene

```
#VRML V2.0 utf8
Background { skyColor 1.0 1.0 1.0 }

Shape {

  appearance Appearance {
    material Material {
      emissiveColor 1.0 0 0
    }
  }

  geometry Box {
    size 2.0 2.0 2.0
  }
}
```

box0.wrl

Beispiel: Einfacher Szenegraph

```
Group {
  children [
    Transform {
      children [
        Shape {
          appearance Appearance {
            material Material {
              diffuseColor 1.0 0 0
            }
          }
          geometry Box {
            size 2.0 2.0 2.0
          }
        }
      ]
      translation 2.0 0 0
    }
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0 0 1.0
        }
      }
      geometry Sphere {
        radius 1.0
      }
    }
... (rechte Spalte)
```

```
...
    Transform {
      children [
        Shape {
          appearance Appearance {
            material Material {
              diffuseColor 0 1.0 0
            }
          }
          geometry Box {
            size 2.0 2.0 2.0
          }
        }
      ]
      translation -2.0 0 0
    }
  ]
}

NavigationInfo {
  type "EXAMINE"
}
```

scene0.wrl

Beispiel: Animation in VRML (Würfeldrehung)

```
DEF RotCube Transform {
  children [
    Shape {
      appearance Appearance {
        diffuseColor 0 1.0 0
       }
      geometry Box {
        size 2.0 2.0 2.0
      }
    }
  ]
}

DEF Clock TimeSensor {
  cycleInterval 6.0
  loop TRUE
}

DEF Interpolator OrientationInterpolator {
  key [0.0, 1.0]
  keyValue [
    0 1.0 0 0.00,
    0 1.0 0 3.14
  ]
} ...nächste Spalte
```

```
...
NavigationInfo {
  type "EXAMINE"
}

ROUTE Clock.fraction_changed
  TO Interpolator.set_fraction
ROUTE Interpolator.value_changed
  TO RotCube.set_rotation
```

box1.wrl