Medientechnik

Vektorgraphik und SVG

Michael Granitzer

Professur für Medieninformatik

Universität Passau

Kapitel Medientechnik: V

I. Medientechnik - Vektorgrafik

- Vektorgrafik Allgemein
- Codierung am Beispiel Scalable Vector Graphics (SVG)

Vektorgrafik Allgemein

Lernziel

Unterthemen

- Beschreibung Vektorgraphik/Unterschied zu Rastergrafiken
- Koordinatensystem, Punkte, Geraden
- Überblick Bezier-Kurven und Splines
- Rendering-Pipeline
  - Szenegraph und Koordinatensysteme
  - Clipping
  - Rasterisierung

Grundlegende Beschreibung von 2D Vektorgraphiken

Grundlegende Beschreibung von 2D Vektorgraphiken

Bilder vs. Grafiken

Digitales Bild besteht aus N Zeilen und je M Bildpunkten Anwendungsbereiche:

- Pixeln bzw. Picture Elements (Auflösung, Farbtiefe etc.)
- Bild kann aus der realen Welt kommen oder virtuell sein
- Wie beschreiben wir digital erstellte Grafiken?

(Vektor-)Grafiken: durch grafische Primitive und ihre Attribute spezifiziert

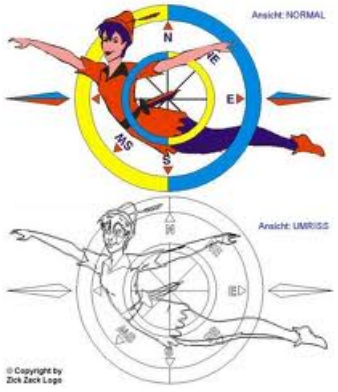
- Primitive 2D Objekte: Linien, Rechtecke, Kreise, Ellipsen, Texte
- 2D Formate: SVG,PostScript, Windoes Metafile, CorelDraw, PDF ...
- Primitive 3D Objekte: Polyeder, Kugeln, ...
- Formate 3D: VRML, X3D
- Attribute: Stil der Linie, Breite, Farbe etc

Grundlegende Beschreibung von 2D Vektorgraphiken

Vektorgrafiken

Definition 1 (Vektorgrafik)

Als Vektorgrafiken bezeichnet man **mathematisch und programmatisch** definierte Zeichenanweisungen in einem Koordinatensystem, aus welchen Rastergrafiken generiert, gespeichert und präsentiert werden können.



Bildquelle zz-logo.de

Eigenschaften

- Vektorgrafiken können einfach und exakt geometrisch transformiert werden
  - Rotation, Skalierung, Verschiebung
  - Separierung einzelner Bildelemente (Gruppierung, Ebenen)
  - Änderung der Attribute einzelner Elemente (Farbe einer Fläche etc)
  - Im Grunde ein perfektes mathematisches Modell, welches jedoch in ein Rasterbild überführt werden muss

Grundlegende Beschreibung von 2D Vektorgraphiken

Bilder vs. Grafiken

Unterschiede zu Rasterbild

- Erstellungsformat - Editor, Vektorgrafik-Programm
- Speicherformat - XML, Properitär (nicht in binären Daten)
- Ausgabeformat - Rendering
- Falls keine Konvertierung in ein Standardformat, sind je Editorensystem eigene Previewer nötig
- Für einen Ausdruck sind (fast) immer Konvertierungen nötig (z.B. nach Postscript)

Grundlegende Beschreibung von 2D Vektorgrafiken

Grundlegende Beschreibung von 2D Vektorgrafiken

Koordinatensystem

Koordinatensystem: Grundlage jeder 2D Vektorgrafik ist ein zweidimensionaler Vektorraum

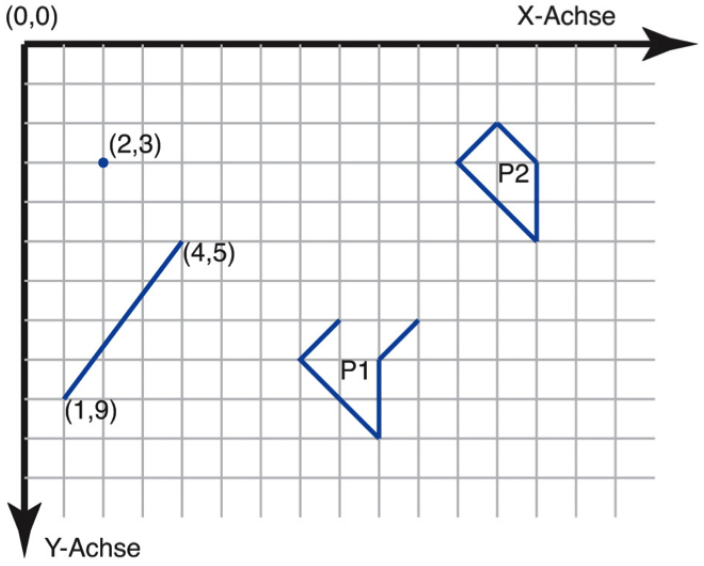
- X/Horizontal- und Y/Vertikalachse
- i.A. gleicher Abstand auf beiden Achsen

Elemente

- Punkt beschrieben durch X/Y Koordinate
- Gerade beschrieben durch Start- und Endpunkt
- Polygon aus mehreren Geraden
  - Geschlossenes Polygon (Fläche)
  - Offenes Polygon
- Kreis - Radius und Mittelpunkt

Grundlegende Beschreibung von 2D Vektorgrafiken

Koordinatensystem



Bildquelle [1]

Canvas: Die vom Koordinatensystem aufgespannte Zeichenfläche wird auch Canvas bezeichnet

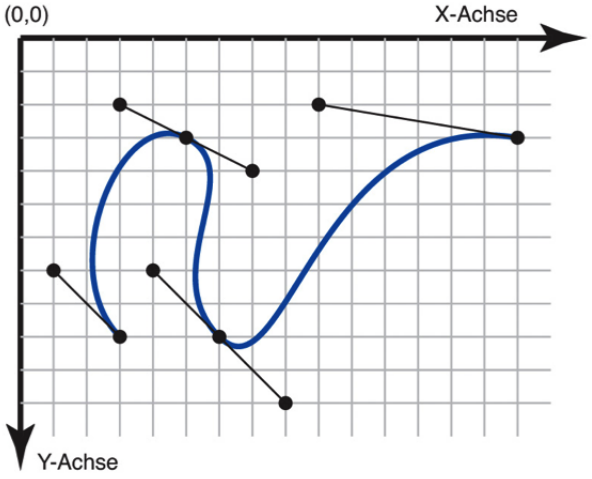
- Je nach Format liegt der Ursprung des Koordinatensystems woanders
- Java/SVG/Web: Links Oben; PostScript-Standard Links unten

Grundlegende Beschreibung von 2D Vektorgrafiken

Kurven Formen im Koordinatensystem

Wie zeichnet man Kurven?

Interpolationskurven oder Splines werden durch **Kontroll- oder Stützpunkte** beschrieben



Bildquelle [1]

- Eine Spline Kurve $n - ten$ Grades ist stückweise aus Polynomen maximal $n - ten$ Grades zusammen gesetzt
- Angabe der Randbedingungen für jedes Stück (1.-3. Grad der Ableitung) bestehend aus **Steigung, Krümmung und Krümmungsänderung**
- Randpunkte eralten somit die Glätte und Stetigkeit
- Angabe in Grafikprogrammen duch **Kontrollinien**
  - Richtung = Steigung
  - Länge = Steifigkeit/Krümmung

Grundlegende Beschreibung von 2D Vektorgrafiken

Kurven Formen im Koordinatensystem

Eine **Bezier-Kurven** $n - ten$ Grades ist eine spezielle Art der Interpolationskurve welche durch $n + 1$ Kontrollpunkte beschrieben wird.

- Entwickelt von Bezier und Casteljau bei Renault bzw. Citroen zur Formgebung bei Autos
- Der Kurvenlauf lässt sich mit dem Algoritmus von Casteljau ermitteln

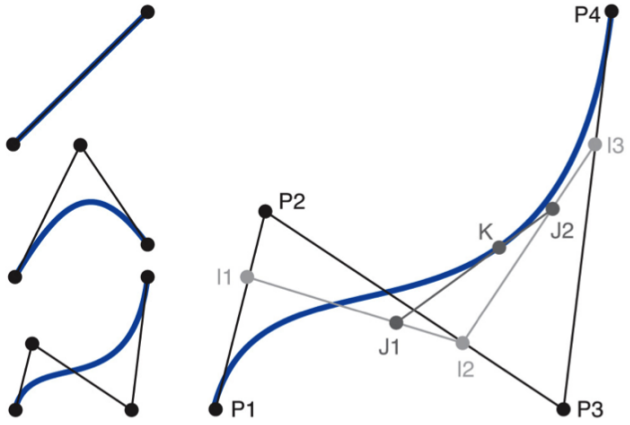


Abbildung 7.3: Links von oben nach unten Bézier-Kurven ersten, zweiten und dritten Grades, rechts eine Darstellung des Algorithmus von Casteljau

Bildquelle [1]

Skizze des Algorithmus:

1. Gegeben: $n + 1$ Kontrollpunkte P einer Berzier Kurve $n - ten$ Grades
2. Initalisiere Laufparameter $t \in [0 : 1]$ mit einem kleinen Wert
3. Setze $P' = P$
4. Teile die durch P' definierten Geraden im Verhältnis t
5. Verwende die Teilungspunkte als neue Kontrollpunkte P'
6. Wenn $|P'| > 1$ gehe zu 4
7. Wenn $|P'| == 1$ zeichne den Punkte an Position P', erhöhe t
8. Solange $t < 1$ gehe zu 3.

Grundlegende Beschreibung von 2D Vektorgrafiken

Kurven Formen im Koordinatensystem

Konstruktionsbeispiel f. Kurve erster Ordnung

http://en.wikipedia.org/wiki/B

Grundlegende Beschreibung von 2D Vektorgrafiken

Kurven Formen im Koordinatensystem

Konstruktionsbeispiele für Kurve 2. Ordnung

http://en.wikipedia.org/wiki/B

Grundlegende Beschreibung von 2D Vektorgrafiken

Kurven Formen im Koordinatensystem

Konstruktionsbeispiele für Kurve 3. Ordnung

http://en.wikipedia.org/wiki/B

Grundlegende Beschreibung von 2D Vektorgrafiken

Geometrische Transformationen

Wie bei den Bildoperationen, können Punkte im Koordinatensystem geometrisch transformiert werden.

- Translation
- Rotation
- Skalierung
- Scherung

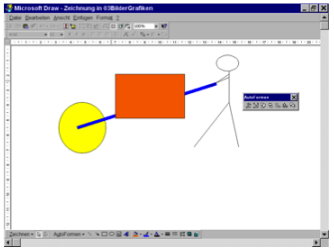
Sie VO Einheit Bildoperationen

Grundlegende Beschreibung von 2D Vektorgrafiken

Formate und Beispiel

Formate:

- Vektor-Grafik-Formate
- PostScript (.ps, .eps). PDF
- Windows Metafile (*.wmf, *.emf)
- Corel Draw (*.cdr)
- ScalableVector Graphics (*.svg)
- VRML (3D)
- u.a.m.



vgl. z.b. Grafikerstellung mit MS Draw

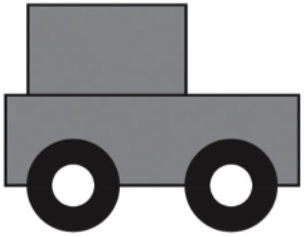
- Einzelobjekt zusammengesetzt aus verschiedenen Grundprimitiven
- Objektorientierte-Sichtweise
- Gruppierungen müssen möglich sein
- Typische Größe 10-100 KByte

Rendering

Rendering

Rendering

Wie erfolgt das Zeichnen/Darstellen einer Vektorgrafik am Ausgabegerät?



Bildquelle [1]

Definition 2 (Rendering)

Als **Rendering** bezeichnet man die Darstellung/das Zeichen von Vektorgrafiken auf Repräsentationsmedien (meist Bildschirm).

Rendering kann als Digitalisierungsprozess von einem idealen, mathematischen Modell in dessen Darstellung angesehen werden. D.h. es können die gleichen Effekte wie bei der Digitalisierung von Licht auftreten.

Unterschiede zwischen 2D und 3D Modellen. Wir betrachten nur 2D Rendering.

Rendering

Rendering Pipeline - Transformation Bild-/Weltkoordinaten

Die zu zeichnenden Elemente sind in dem sogenannten **Szenengraph** definiert

Der Szenegraph ist ein gerichteter-azyklischer Graph von darzustellenden geometrischen Objekten und kann im einfachsten Fall als Baum aufgefasst werden.

- Blatt-Knoten definieren primitive geometrische Objekte (Linien, Polygone)
- Inner Knoten definieren 2D Transformationen auf abhängige Objekte
- Gerichtete Kanten spezifizieren Zusammensetzung von einfacheren Objekten/Primitives zu komplexeren Objekten
- Eigenschaften können vererbt werden (z.B. Farbe eines Objektes)
- Objekte im Szenengraphen sind anhand eines Objekt-lokalen Bezugskoordinatensystems (Objektkoordinaten) definiert (z.B. Mittelpunt des Rades ist Punkt 0,0)
- Objektkoordinaten müssen beim darstellen der Szene in das Weltkoordinatensystem überführt werden (Position des Rades in der Szene)
- Das Weltkoordinatensystem ist das Bezugskoordinatensystem der Szene in dem alle relevanten Objekte (inkl. Kamera, Lichter etc.) positioniert werden

Rendering

Rendering Pipeline - Transformation Bild-/Weltkoordinaten

Beispiel eines Szenengraphen

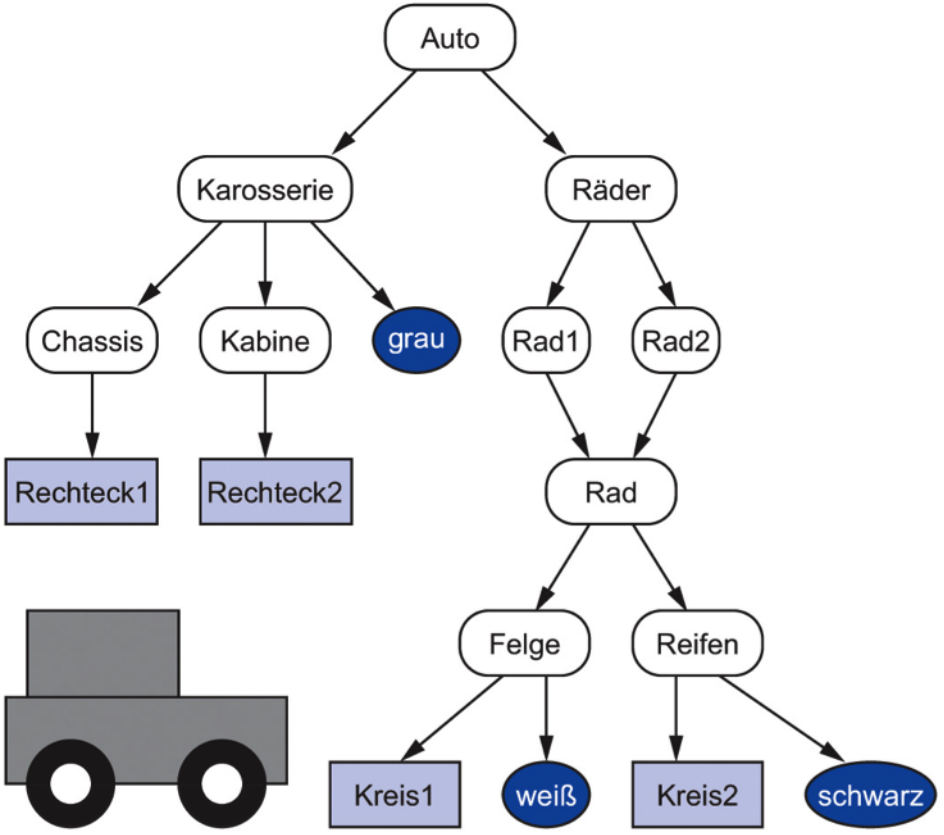


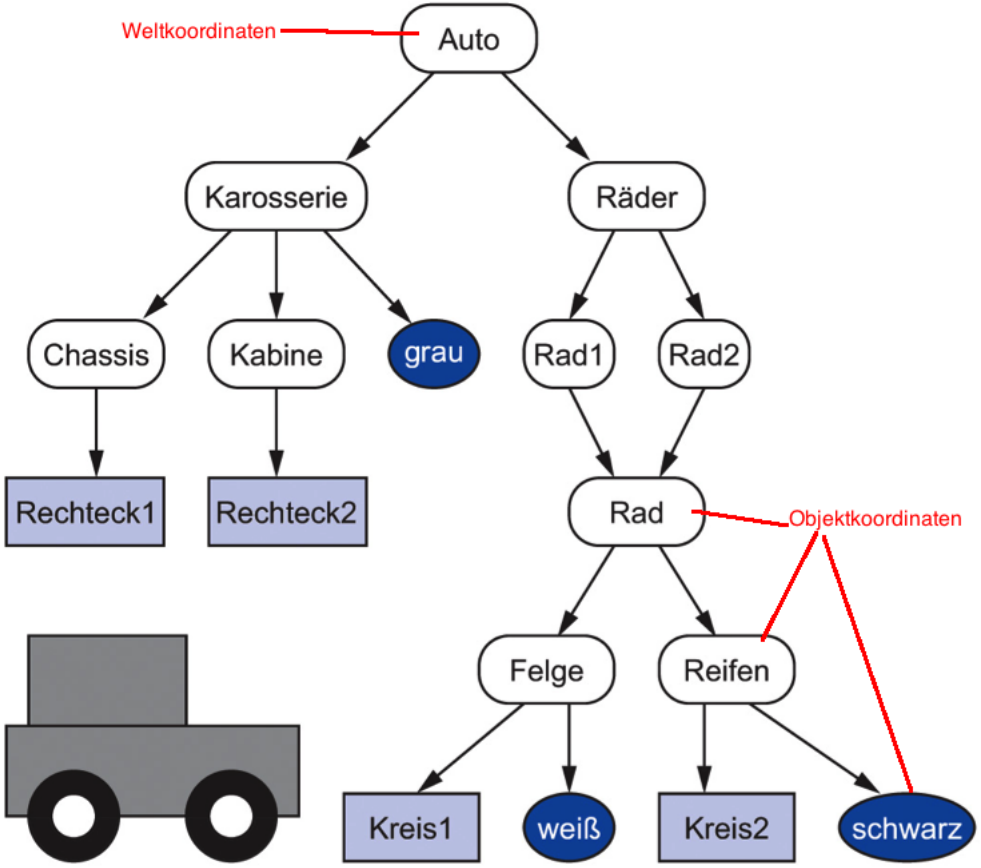
Abbildung 7.6: Szenegraph eines Autos mit zwei identischen Rädern

Bildquelle [1]

Rendering

Rendering Pipeline - Transformation Bild-/Weltkoordinaten

Beispiel eines Szenengraphen

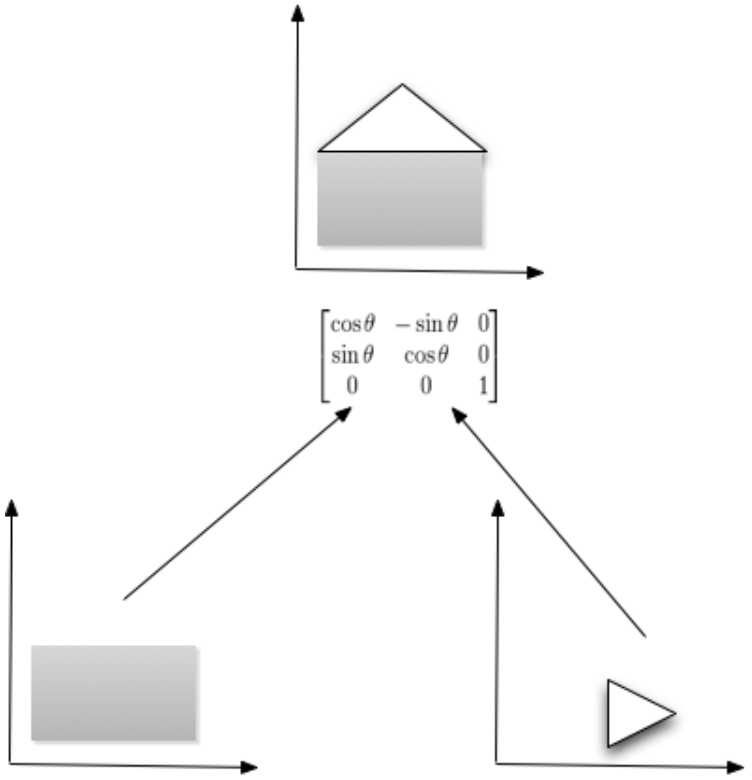


Bildquelle [1]

Rendering

Rendering Pipeline - Transformation Bild-/Weltkoordinaten

Transformation in einem Knoten



Erinnerung: Transformationen sind assoziativ, d.h. eine Kette von Transformationen auf einem Objekt kann als eine Matrix ausgedrückt werden.

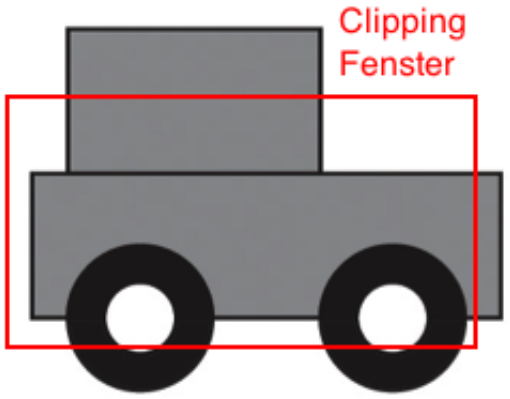
Rendering

Rendering Pipeline - Clipping

Das Weltkoordinatensystem ist unendlich groß. Die Darstellungsfläche ist jedoch endlich.

Zwei Schritte sind notwendig

- Beschneiden des Szenegraphen auf den durch ein Fenster definierten sichtbaren Bereich (Clipping)
- Transformation in Bildschirmkoordinaten



Rendering

Rendering Pipeline - Clipping

Beobachtungen

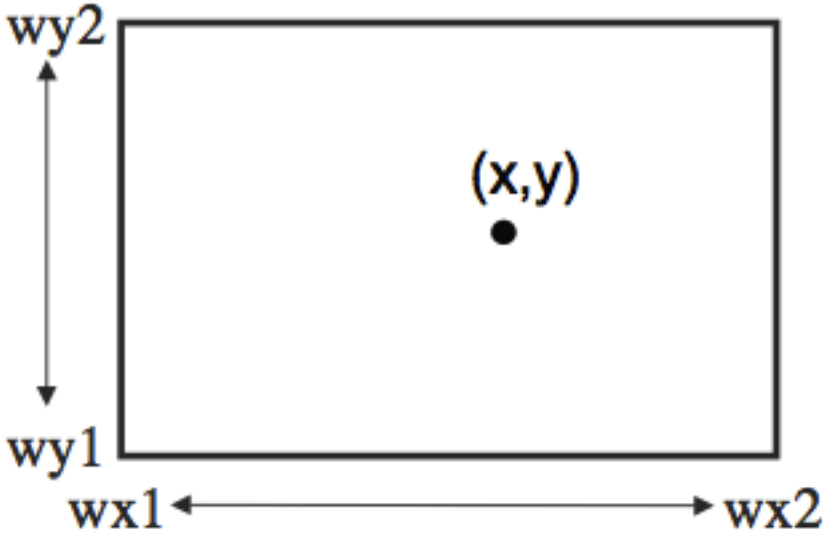
- Die Szene besteht eigentlich nur aus Punkten oder Linien, d.h. es genügt wenn wir uns damit beschäftigen wie wir Linien beschneiden (wir ignorieren Splines hier).
- Das Fenster kann Punkte ganz oder gar nicht beinhalten
- Das Fenster kann teilweise Linien vollständig beinhalten, gar nicht oder teilweise

$\Rightarrow$ Clipping von Punkten
$\Rightarrow$ Linienclipping nach Cohen und Sutherland

Rendering

Rendering Pipeline - Clipping

Clipping von Punkten




```
inside =
   (x >= wx1) &&
   (x <= wx2) &&
   (y >= wy1) &&
   (y <= wy2);
```

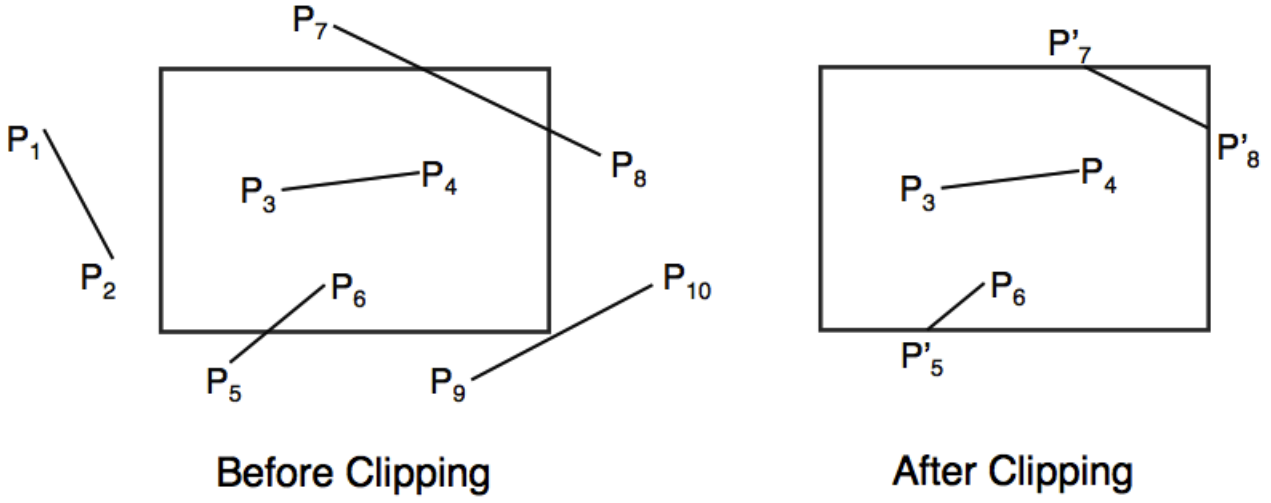
Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm

Rendering

Rendering Pipeline - Clipping

Linien-Clipping nach Cohen und Shuterland

Ausgangspunkt und Zielsetzung:



Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm

Rendering

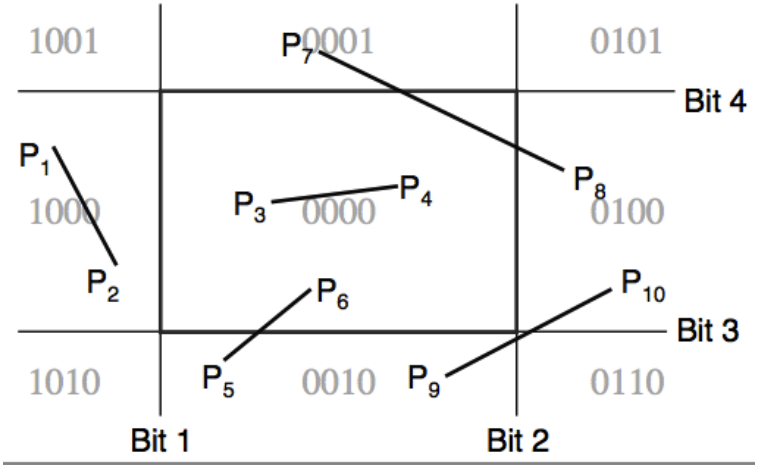
Rendering Pipeline - Clipping

9 Segmente beschrieben durch 4-Bit Code

- ersten 2-Bit definieren horizontal-position (rechts=10,mitte=00,links=01)
- zweiten 2-Bit definieren vertikal-position (unten=10, mitte=00, oben=01)

Logische Bitoperationen auf Linie (P_1,P_2)

1. $P_1\ OR\ P_2 = 0000 \Rightarrow$ Linie ist im Clipping Fenster
2. $P_1\ AND\ P_2 \neq 0000 \Rightarrow$ Linie ist in den Randbereichen, d.h. nicht zu zeichnen
3. Falls $P_1 \neq 0000$ prüfe Schnitt mit Rand (e.g. 0010=rechter Rand)
4. Falls $P_2 \neq 0000$ prüfe Schnitt mit Rand (e.g. 0010=rechter Rand)
5. Bei Schnitt wähle Schnittpunkt P_1' als neuen Linienpunkt.

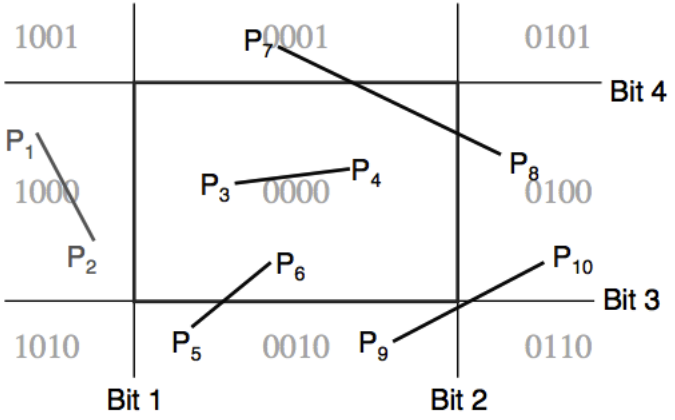


Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm

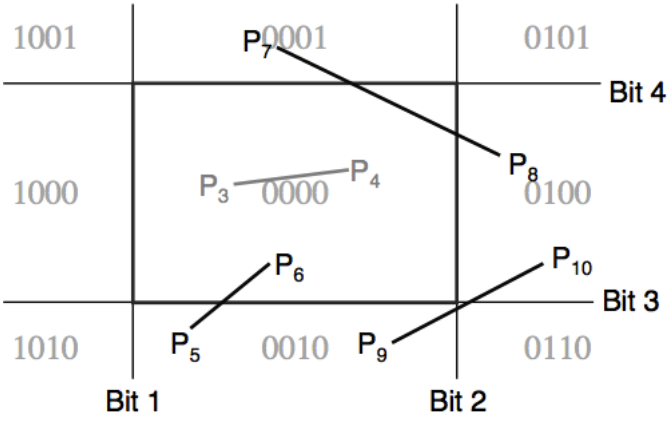
Rendering

Rendering Pipeline - Clipping

Schritte 1. und 2.: schnelle Klassifikation von Linien ohne Schnittpunkte mit Fenstergrenzen



Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm

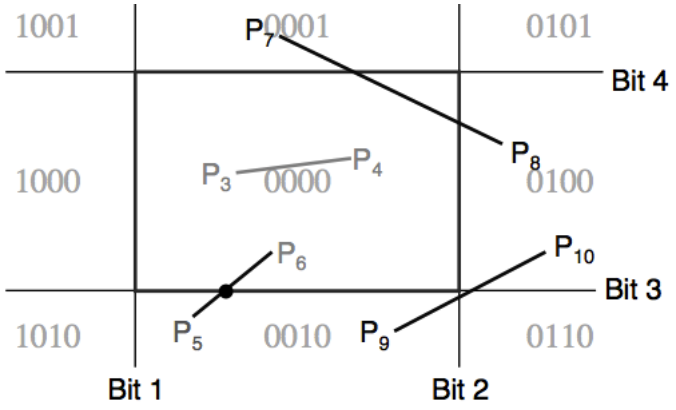


Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm

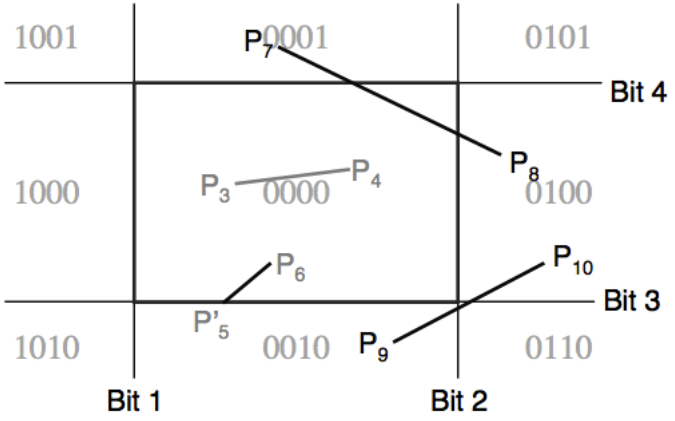
Rendering

Rendering Pipeline - Clipping

Schritte 3.-5.: ermitteln von Schnittpunkten für Linien mit Schnittpunkten



Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm



Bildquelle http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/pipeline/index.htm

Rendering

Von Welt- nach Bildkoordinaten

Clipping erfolgte noch im Weltkoordinatensystem, d.h wir benötigen ein Transformation in das Bildkoordinatensystem des Rasterbilds

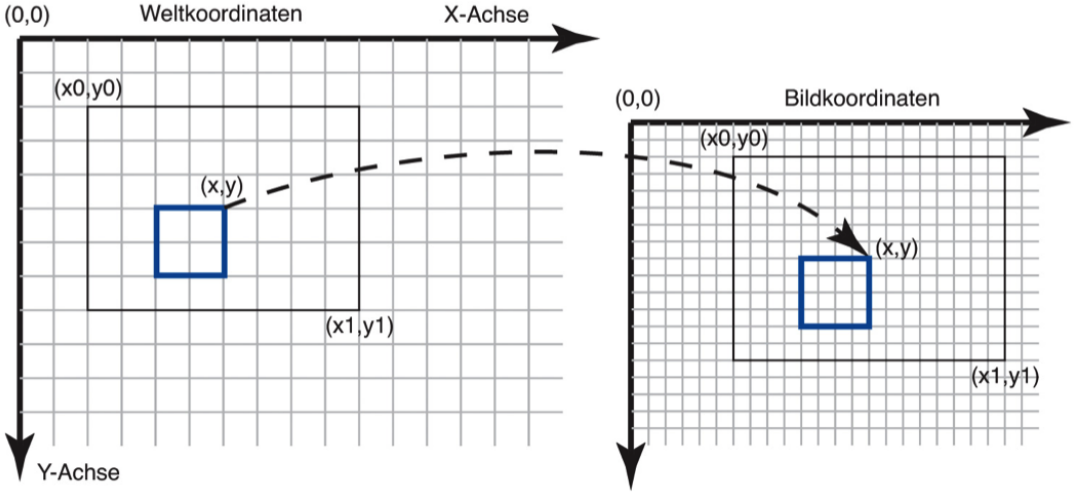


Abbildung 7.8: Transformation von Weltkoordinaten in Bildkoordinaten

[1]

$$x_{bild} = x0_{bild} + (x_{welt} - x0_{welt}) * (x1_{bild} - x0_{bild})/(x1_{welt} - x0_{welt})$$

$$y_{bild} = y0_{bild} + (y_{welt} - y0_{welt}) * (y1_{bild} - y0_{bild})/(y1_{welt} - y0_{welt})$$

Rendering

Rendering

Rendering Prozess: Etablierte Abfolge von Arbeitsschritten zur Darstellung der Vektorgrafik an Raster-basierten Präsentationsmedien

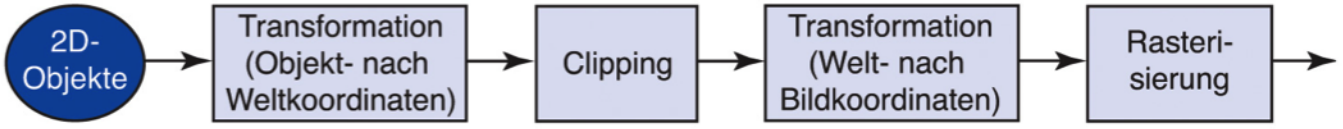


Abbildung 7.5: Die 2D Rendering Pipeline

Bildquelle [1]

- Ausgangspunkte: 2D Objekte als Gruppe primitiver 2D Objekte
- Transformation der Objekte in Weltkoordinaten
- Beschneiden des Anzeigebereichs
- Transformation von Weltkoordinaten in Bildschirmkoordinanten
- Rasterisierung: Zeichen von Linien und Punkten

Rendering

Rasterisierung

Nach Bestimmung der darzustellenden Punkte und Linien müssen deren darzustellende Pixel ermittelt und gezeichnet werden (Ausnahme: Vektorgrafikgeräten wie Plotter, Laserprojektor, Fräsmaschinen)

Naiver Ansatz zur Rasterisierung von Linien

- Annahme: Punkte bereits in Bildkoordinaten und X-Richtung ist länger als Y-Richtung
- Ermittle Steigung der Linie $k = (y_1 - y_2)/(x_1 - x_2)$
- Laufe in einer Schleife über all x Werte zwischen x_1 und x_2
- Setze Pixel $y = round(k * (x - x_1)) + y_1$ auf 1 (d.h. wir nehmen immer das nächstgelegene Pixel)

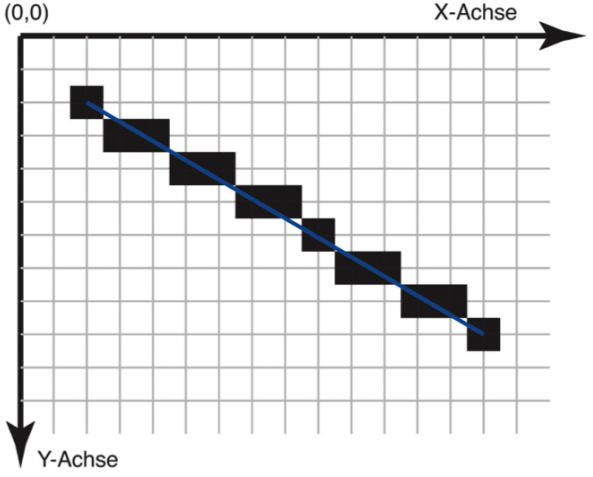


Abbildung 7.9: Rasterisierung einer Linie mit einem naiven Verfahren

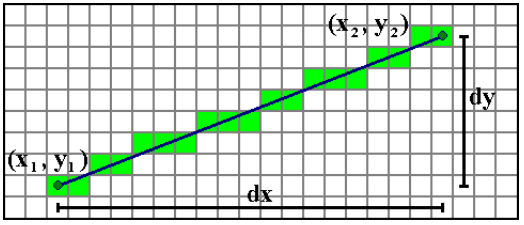
[?]

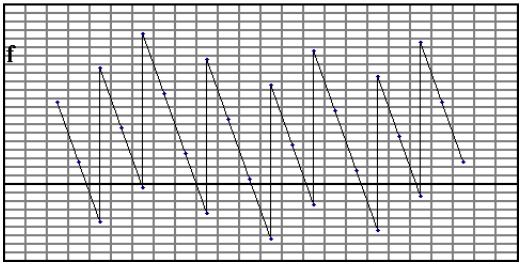
Rendering

Rasterisierung - Bresenham-Algorithmus

Naiver Ansatz ist zeitintensive, da für jeden Punkt eine Multiplikation notwendig ist. Der Bresenham-Algorithmus kommt nur mit vergleichen, addieren und Bit-verschieben aus.

Grundidee:





Bildquelle Wikipedia

- ermittle eine schnelle Richtung (Richtung in der die Koordinate schneller wächst) und eine langsame Richtung

  Bei einer Steigung $k < 1$ wächst x-Achse schneller als die y-Achse
- Fehlerterm: Abweichung im Bereich $[-0.5 : 0.5]$ zwischen gezeichneten (gerundeten) Pixel zum wirklichen Linienwert
- Erhöhen den Fehlerterm mit jedem Schritt in die schnelle Richtung um die Steigung der Kurve $k = \frac{y_2 - y_1}{x_2 - x_1}$
- Liegt der Fehler über 0.5, erhöhe die langsame Richtung um 1 und reduziere den Fehler um 1.0

Rendering

Rasterisierung - Antialiasing mit Algorithmus von Wu

Einbringen von Anti-Aliasing Techniken durch den Algorithmus von Wu

http://en.wikipedia.org/wiki/Xiaolin_Wu

- Für jeden ermittelten y-Wert einer Linie, setze mehrere Pixel
- Färbe die Pixel entsprechend ihres Abstandes von der wahren Linie

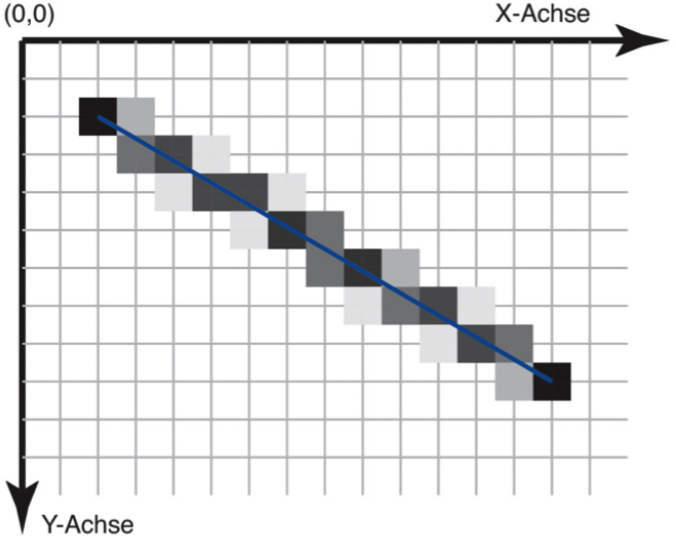


Abbildung 7.10: Rasterisierung einer Linie nach dem Algorithmus von Wu

Bildquelle Wikipedia

Rendering

Rasterisierung von gefüllten Polygonen

Neben effizienten Zeichnen von Linien sollte auch Polygone effizient befüllt werden können.

Painters Algorithmus: Scanline basiertes Verfahren

Beobachtung: Die Anzahl der Schnittpunkte eines Polygons definiert, ob ein Punkt innerhalb oder außerhalb des Polygons liegt

- Gerade Anzahl - Punkt liegt außerhalb
- Ungerade Anzahl - Punkt liegt innerhalb
- gilt für beliebige Polygone

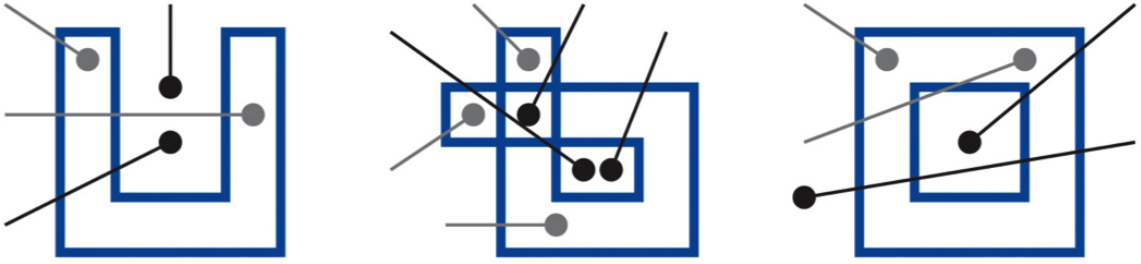


Abbildung 7.11: Parität verschiedener Punkte innerhalb und außerhalb von Polygonen: Schwarz bedeutet gerade und grau bedeutet ungerade Parität.

Bildquelle [**?**]

Rendering

Rasterisierung von gefüllten Polygonen

Painters Algorithmus im Überblick

- Scanline: Bestimme für jede Zeile von Pixeln all Schnittpunkte mit den Kanten des Polygons und sortiere sie aufsteigend nach X-Koordinate
- Ermittle für jedes Pixel innerhalb der Zeile seine Parität. Vor dem ersten Schnittpunkt haben alle Pixel die Parität null und bei jedem weiteren wird die Parität um eins erhöht
- Färbe alle Pixel mit ungerader Parität mit der Füllfarbe ein

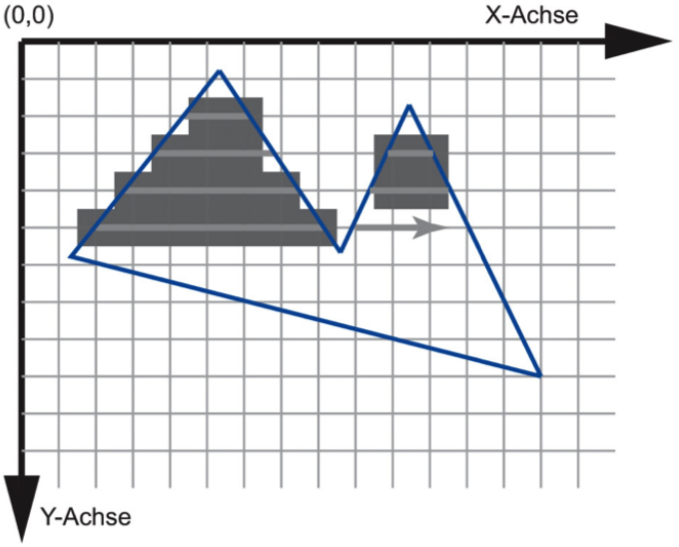


Abbildung 7.12: Ausfüllen eines Polygons mit dem Scanline-Algorithmus

Bildquelle [**?**]

Rendering

Animationen

Computeranimationen können in Vektorgrafiken einfach durch zeitlich Veränderung der Punkte und anschließendem Rendering definiert werden (i.e. Animtaions-transformationen auf Szenegraph)

Keyframeanimation

- Kontrollpunkte und primitive geometrische werden zu zwei Zeitpunkten (z.B. Sekunde 0 und Sekunde 10) bestimmt (die sogenannten Schlüsselbilder/Keyframes)
- **Keyframing:** Die restlichen Bilder dazwischen werden interpoliert
- Interpolation kann linear oder nicht-linear erfolgen (z.B. über Splines zwischen Kontrollpunkten)
- Zusätzlich Interpolation der Farbe
- Benötigt entsprechende Designprogramme (z.B. Adobe Flash)

Weitere Formen der Interaktion:

- **Partikelsysteme**: Animation über physikalische Simulation
- **Scripting**: Animation über Programmcode
  - Interaktionen können berücksichtigt werden
  - Hyperlink Definition möglich
  - Beispielformate: Flash und SVG

Zusammenfassung

Zusammenfassung

Vektorgrafiken

- Vektorgrafiken bestehend aus Koordinatensystem und primitiven geometrischen Objekten
- Punkte, Geraden, Kreise und Interpolationskurven
  - Bezierkurven
- Rendering: Szenengraph/Weltkoordinaten, Clipping, Bildschirmkoordinaten, Rasterisierung
  - Bresenham Algorithmus zur schnellen Rasterisierung
  - Antialisasing durch Färbung
  - Befüllung von Polygonen - Painters Algorithmus

Literatur

[1] Malaka, Butz, Hussmann (2009) - Medieninformatik: Eine Einführung (Pearson Studium - IT), Kapitel 7.1 und 7.2

Kapitel Medientechnik: V

I. Medientechnik - Vektorgrafik

- Vektorgrafik Allgemein
- Codierung am Beispiel Scalable Vector Graphics (SVG)

Codierung mittels Scalable Vektor Grafik

Lernziel

Unterthemen

- Überblick SVG und ähnliche Formate
- Statische SVG Bilder
- Animationen in SVG
- Beispiele
- Erstellungsprogramme

Kodierung von Vektorgrafiken

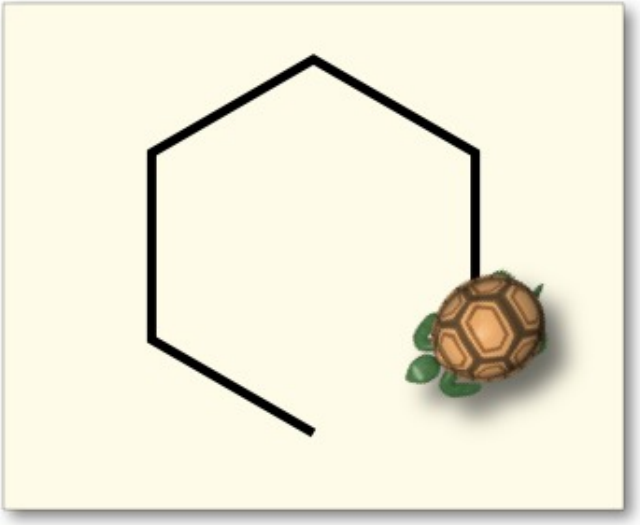
Kodierung von Vektorgrafiken

Grundelemente

Zur **Kodierung** von Vektorgrafiken ist eine entsprechende Sprache zur Definition von Geometrie und Animation notwendig.

Beispiel Turtle Grafik

- Bekannt durch Logo Programmiersprache aus den 1970iger Jahren zum Lernen von Programmierung
- Turtle: Position, Orientierung, Stift (Größe, Farbe etc.)
- Sprache beschreibt Weg: “move forward 10 units”; “lift pen”; “turn left 90°”
- Ähnlich dem Turtle Robot (physisch) aus der frühen Robotik Forschung 1960



Bildquelle http://www.alancsmith.co.uk/logo/

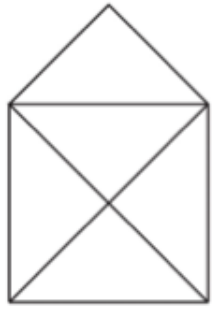
Kodierung von Vektorgrafiken

Grundelemente

Post Script/Encapsulated PostScript (EPS)/Portable Document Format (PDF)

- PostScript entwickelt 1984 von Adobe zur geräteunabhängigen Darstellung formatierter Texte
- Darstellung von Vektorgrafik + Rastergrafik
- Vollständige Programmiersprache
- Wird meist von Druckern (e.g. Laserdrucker) implementiert
- Angabe von Punkten und Pfaden. Pfad wird dann mit Zeichengerät (e.g. Stift, Pinsel) gezeichnet (siehe Beispiel)
- PDF als Nachfolger: Bessere Komprimierung, dafür keine vollständige Programmiersprache mehr

```
%!PS
100 100 newpath moveto
100 200 lineto
200 200 lineto
150 250 lineto
100 200 lineto
200 100 lineto
100 100 lineto
200 200 lineto
200 100 lineto
stroke
showpage
```



Bildquelle [1]

Scalable Vektor Grafik - Grundlagen

Scalable Vektor Grafik - Grundlagen

SVG-Überblick

Sprache für 2D-Graphik in XML, welche kombinierbar mit anderen Web-Standards

Drei Typen grafischer Objekte

- Shapes (Pfade aus Kurven und geraden Linien)
- Bilder (Raster-Graphik)
- Text

Grafische Objekte können

- gruppiert
- gestyled (CSS)
- transformiert
- zusammengesetzt werden

Vorteile

- Zusammengesetzte Transformationen
- Clipping paths (Bilder flexibel zuschneiden)
- Alpha-Masken (Durchsichtigkeit von Objekten)
- Filter-Effekte
- Objektvorlagen

SVG Zeichnungen sind potenziell

- interaktiv und
- dynamisch

W3C Standard `http://www.w3.org/Graphics/SVG/`

Scalable Vektor Grafik - Grundlagen

Grundelemente SVG

Koordinatensystem:

- Koordinatensystem: $(0, 0)$ links oben
- User Koordinaten System korrespondiert mit Bildschirmkoordinatensystem per Default (100 Pixel in SVG Datei entsprechen 100 Pixel am Bildschirm)
- Änderungen am User Koordinatensystem möglich

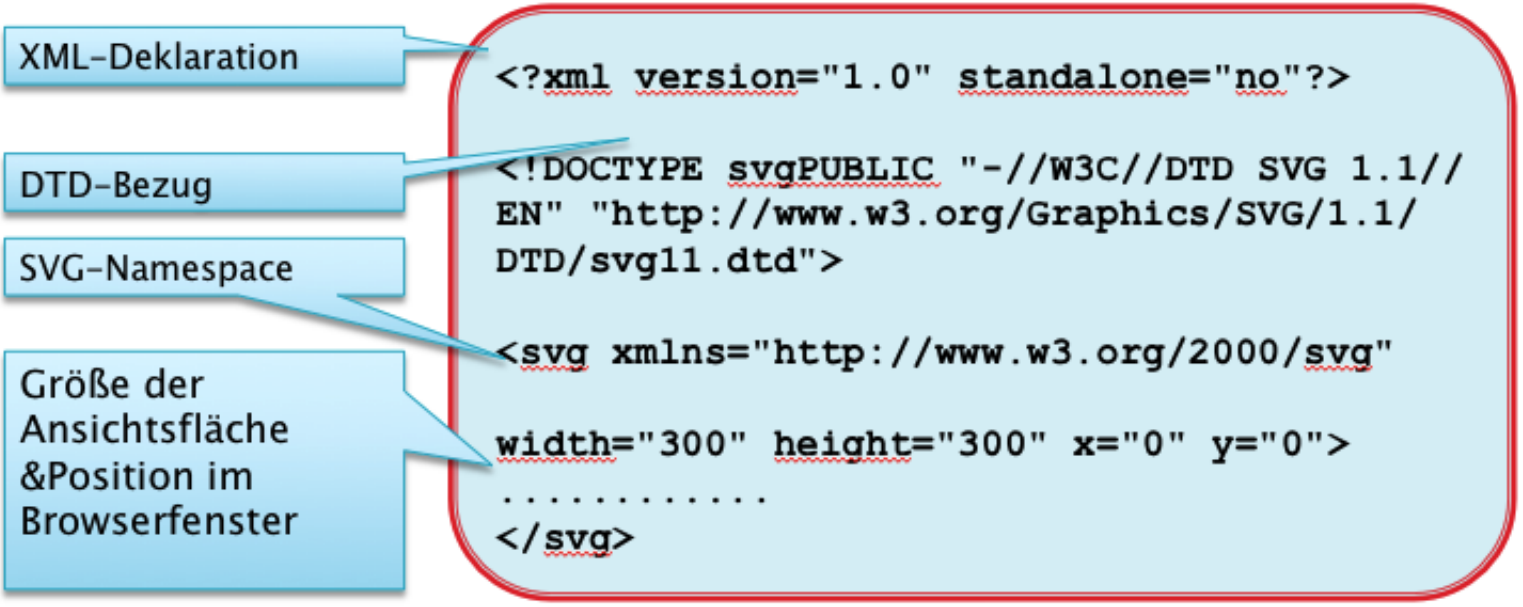
- Pfade (ähnlich der Turtle) als grundlegende Zeichenobjekt
- Erweiterung um geometrische Objekte (e.g. Kreis, Rechteck etc.)
- Definition von Attributen (e.g. Strichbreite) pro Pfad/Objekt

$\Rightarrow$ Scalable Vector Graphics (SVG) 1.1 (Second Edition)

`http://www.w3.org/TR/SVG/Overview.html`

Scalable Vektor Grafik - Grundlagen

XML Grundstruktur SVG



Scalable Vektor Grafik - Grundlagen

XML Grundstruktur SVG

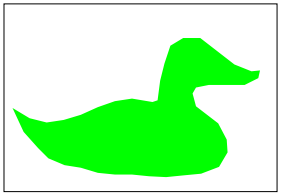
- `<svg>`
  1. Definitionen wieder verwendbarer Bestandteile
     - Pfade
     - Gradienten
     - Filter
  2. Zeichnen unter Verwendung der Definitionen und Grundoperationen
- `</svg>`

Scalable Vektor Grafik - Grundlagen

Einfaches SVG Beispiel

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    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 251 149 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"/>
         </g> </g>
</svg>
```

Quelle: Prof. Butz, LMU



Zum

Ausprobieren: http://www.w3schools.com/svg/tryit.asp?filename=trysvg_myfirst

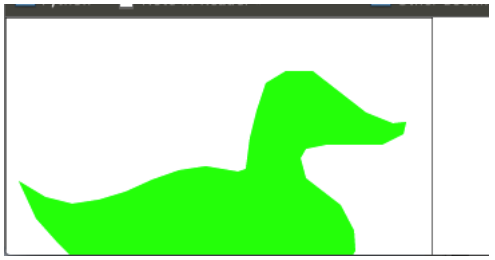
Scalable Vektor Grafik - Grundlagen

SVG - Canvas Größe

Bestimmung der Größe der Zeichenfläche auf 2 Arten möglich

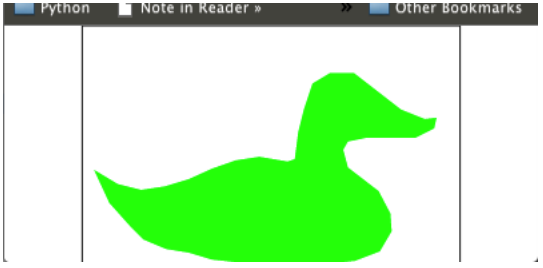
Absolute Größenangabe, d.h. Grafik wird bei Verkleinerung abgeschnitten

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



Angabe eines Sichtfensters, d.h. Größe wird bei Änderung des Fensters skaliert

```
<svg viewBox="0 0 320 200">
```



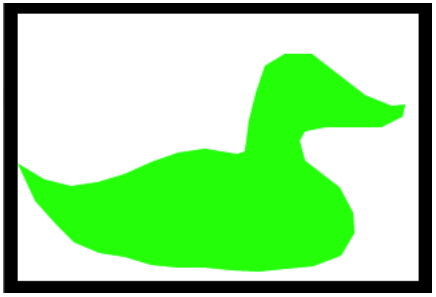
Scalable Vektor Grafik - Grundlagen

SVG - Rendering Attribute

Beeinflussung eines grafischen Objektes mit Attributen

Angabe der Attribute direkt in XML Tag, über `style` Definition in CSS2-Syntax oder über CSS2-Stylesheet

- Füllfarbe `fill`
- Transparenz `opacity`
- Linienfarbe und -stärke `stroke` und `stroke-width`
- Linienenden `stroke-linecap`
- Schriftfamilie und -größe `font-family` und `font-size`



```
<rect ..... stroke-width="20"/>
```

Scalable Vektor Grafik - Grundlagen

SVG mit Stylsheet

```
<?xml-stylesheet type="text/css" href="renderstyle.css" ?>
<svg viewBox="0 0 300 300">
  <rect class="heavy" 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

Quelle Butz LMU

Scalable Vektor Grafik - Grundlagen

SVG - Pfad Syntax

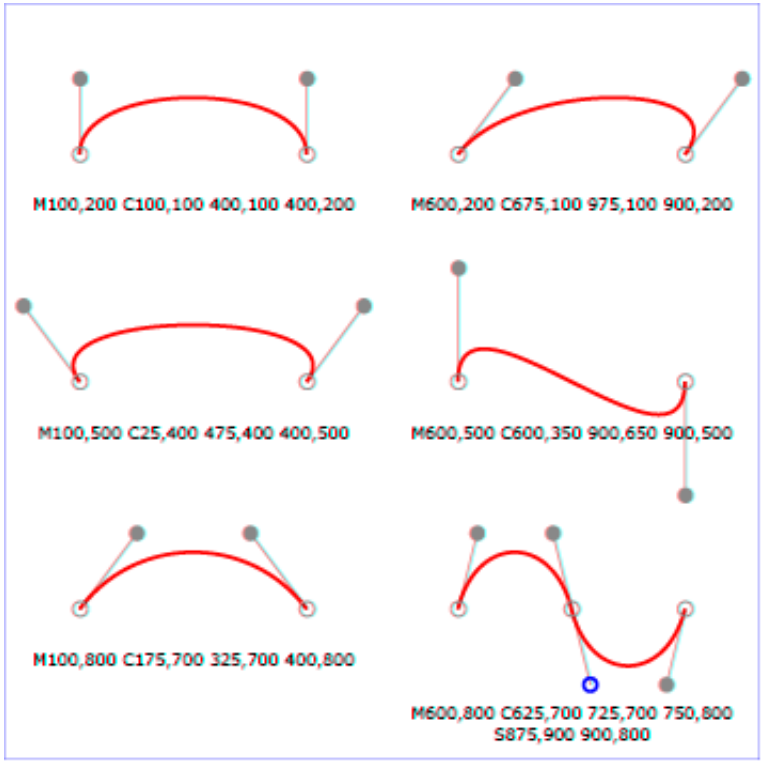
Pfade definieren eine Folge von Zeichenkommandos für einen virtuellen Zeichenstifft

- Syntax ist knapp gehalten, um Speicherplatz zu sparen
  - Kommandos mit Zeichenlänge 1, relative Koordinaten, keine Token Separatoren wenn möglich, Berzier-Kurven Formulierung
  - Zusätzlich Möglichkeit der verlustfreien Kompression (z.B. Huffmann)
- Kommandos
  - `M X Y` Startpunkt auf Koordinate X,Y
  - `L X Y` Linie nach X Y
  - `Z` Gerade Linie zurück zum Startpunkt
  - `H X` Horizontale Linie bis Koordiante X
  - `V X` Vertikale Linie bis Koordiante X
  - `Z` Gerade Linie zurück zum Startpunkt
  - `Q cx cy x y` Quadratische Berzier-Kurve nach X,Y mit Kontrollpunkt cx,cy
  - `C c1x c1y c2x c2y x y` Kubische Berzier-Kurve nach X,Y mit den beiden Kontrollpunkten (c1x,c1y) und (c2x,c2y)
  - `A rx ry x-rot la-flag sweep-flag x y` Elliptische Kurve
  - Kleinbuchstaben Versionen des Kommandos stehen für relative Koordinaten (e.g.`l X Y`)

Scalable Vektor Grafik - Grundlagen

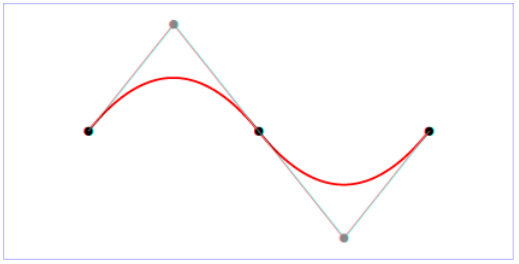
SVG Bezier Kurven

Kubische Bezier Kurven



http://www.w3.org/TR/SVG/paths.html

Quadratische Bezier Kurven

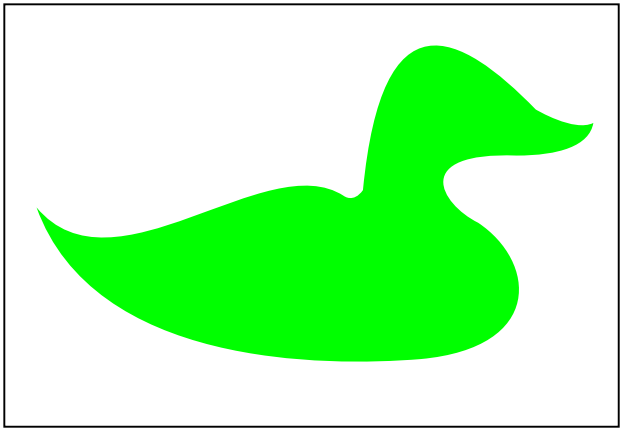


http://www.w3.org/TR/SVG/paths.html

Scalable Vektor Grafik - Grundlagen

SVG Bezier Pfad Beispiele

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    width="320" height="420">
  <rect x="0" y="200" width="320" height="220" fill="white" stroke="black"/>
   <g transform="translate(10 10)">
    <g stroke="none" fill="lime">
      <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"/>
        </g> </g>
</svg>
```



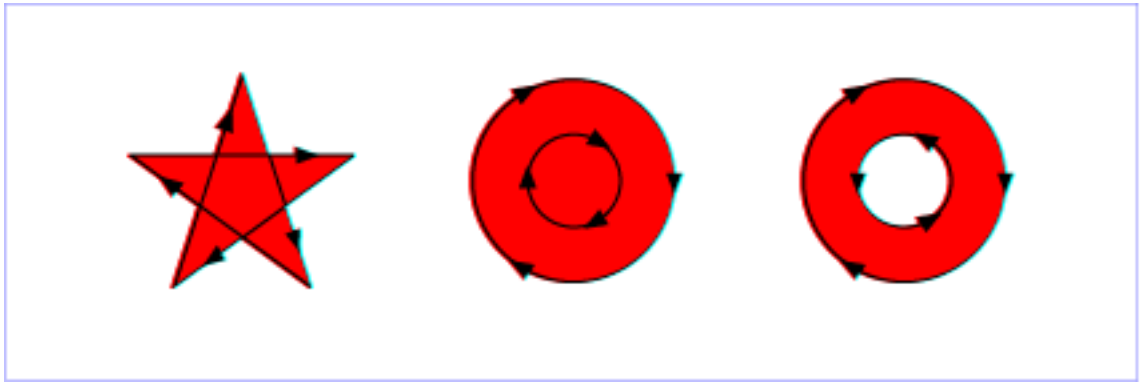
Quelle: [?]

Scalable Vektor Grafik - Grundlagen

SVG - Fülllregeln

Füllen benötigt die Bestimmung, ob ein Punkt innen liegt oder nicht.

nonzero: Für jeden Punkt sende einen Strahl ins unendliche. Für jede links-rechts (rechts-links) gezeichnete Linie erhöhe (erniedrige) den Zähler. Wenn Zähler ungleich 0, dann liegt der Punkt innen



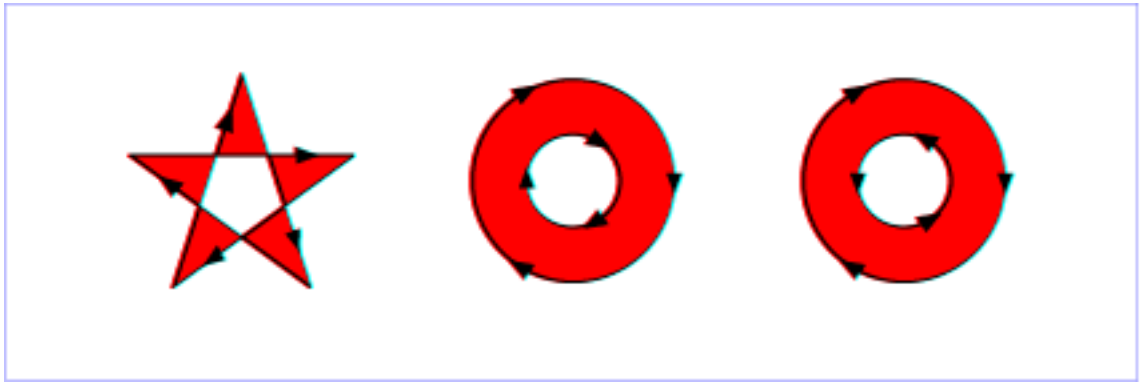
Bildquelle `http://www.w3.org/TR/SVG/painting.html`

Scalable Vektor Grafik - Grundlagen

SVG - Fülllregeln

Bestimme, ob ein Punkt innen liegt oder nicht

evenodd: Für jeden Punkt sende einen Strahl ins unendliche. Für jede links-rechts (rechts-links) gezeichnete Linie erhöhe den Zähler. Wenn Zähler ungerade ist, dann liegt der Punkt innen



Bildquelle `http://www.w3.org/TR/SVG/painting.html`

Scalable Vektor Grafik - Grundlagen

SVG - Text

`<text>`

- Platzierung von Text auf der Leinwand
- Koordinaten-Attribute x und y: Linke untere Ecke des ersten Buchstabens
- Schrift, Größe etc. über Attribute oder Stylesheet

`<tspan>`

- Untergruppe von Text in einem `<text>`-Element
- Einheitliche Formatierung (wie `<span>` in HTML)
- Raltive Position zur aktuellen Textposition: Attribute `dx` und `dy`

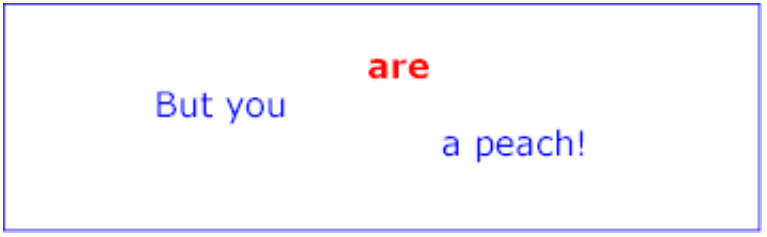
Spezialeffekte

- Drehen einzelner Buchstaben (`rotate`-Attribut)
- Text entlang eines beliebigen Pfades (`<textpath`-Element)

Scalable Vektor Grafik - Grundlagen

SVG Text einfach

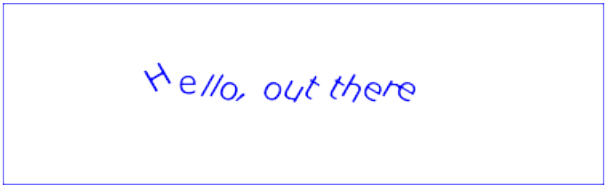
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
    xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example tspan02 - using tspan's dx and dy attributes
      for incremental positioning adjustments</desc>

  <g font-family="Verdana" font-size="45" >
    <text x="200" y="150" fill="blue" >
      But you
      <tspan dx="2em" dy="-50" font-weight="bold" fill="red" >
        are
      </tspan>
      <tspan dy="100">
        a peach!
      </tspan>
    </text>
  </g>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
      fill="none" stroke="blue" stroke-width="2" />
</svg>
```



Scalable Vektor Grafik - Grundlagen

SVG Text rotiert

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
  xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>
    Example tspan04 - The number of rotate values is less than the number of
    characters in the string.
  </desc>
  <text font-family="Verdana" font-size="55" fill="blue" >
    <tspan x="250" y="150" rotate="-30,0,30">
      Hello, out there
    </tspan>
  </text>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
  fill="none" stroke="blue" stroke-width="2" />
</svg>
```



Scalable Vektor Grafik - Grundlagen

SVG - Geometrische Primitive und Transformationen

Pfade definieren i.A. keine geschlossenen Objekte

SVG definiert auch Standard Objekte als geometrische Primitive

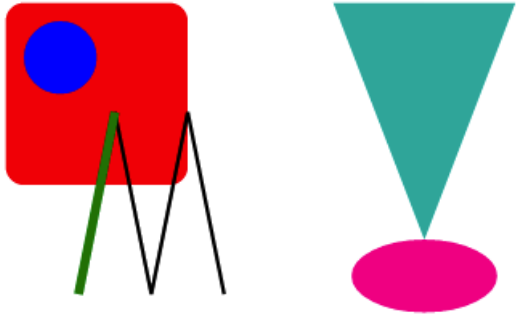
Elementname	Bedeutung	Attribute
<line>	Linie	x1, y1: Erster Punkt x2, y2: Zweiter Punkt
<polyline>	Folge zusammenhängender Linien	points: Folge von x, y
<polygon>	Polygon	points: Folge von x, y
<rect>	Rechteck	x, y: Linke obere Ecke width: Breite, height: Höhe rx, ry: Radien der Ecken
<circle>	Kreis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

Quelle Prof. Butz, LMU

Scalable Vektor Grafik - Grundlagen

SVG Geometrie Beispiel

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
  xmlns="http://www.w3.org/2000/svg" version="1.1">
 <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"/>
</svg>
```



Scalable Vektor Grafik - Grundlagen

SVG -Gruppen und Transformationen

Gruppen - Geometrische Primitive können zu Gruppen kombiniert werden `<g>`

- Einheitliche Attributdefinition für element der Gruppe
- Manipulation der gesamten Gruppe (e.g. Transformation)

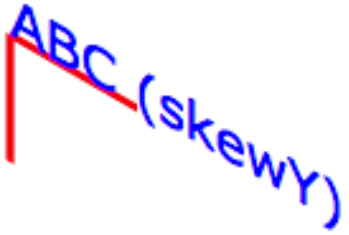
Transformationen

- Verschieben (translate), drehen (rotate), verzerren (skew) oder skalieren (scale)
- SVG Attribut `transform`
- Wert des Attributtes entspricht der Operation plus parameter

Scalable Vektor Grafik - Grundlagen

SVG Geometrie Beispiel

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="120px" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <g transform="translate(200,30)">
    <g transform="skewY(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (skewY)
      </text>
    </g>
  </g>
</svg>
```



Scalable Vektor Grafik - Grundlagen

SVG - Clipping

Clipping - Ausschneiden von Pfaden aus Objekten

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="500px" version="1.1"
    xmlns="http://www.w3.org/2000/svg">
   <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>
</svg>
```

Scalable Vektor Grafik - Grundlagen

SVG- Alpha Composition, Masking

Opacity - Transparenz von Objekten

Masking - Nutzung von beliebigen Objekten zur Maskierung (Überdeckung) eines anderen Objektes

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="500px" version="1.1"
    xmlns="http://www.w3.org/2000/svg">
  <g>
    <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>
  <circle cx="250" cy="150" r="150" opacity="0.5" fill="green"/>
</svg>
```

Scalable Vektor Grafik - Grundlagen

SVG - Links

Hyperlinks können über den Xlink (XML Links) Standard `http://www.w3.org/1999/xlink` eingefügt werden

- Der Namensraum muss im svg Tag spezifiziert werden
- Beispiel:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
 xmlns:xlink="http://www.w3.org/1999/xlink">
  <a xlink:href="http://mics.fim.uni-passau.de">
    <circle cx="50" cy="50" fill="blue" r="20"/>
  </a>
</svg>
```

Scalable Vektor Grafik - Grundlagen

SVG - Symbole

Symbole können zur wiederholten Verwendung definiert werden

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
 xmlns:xlink="http://www.w3.org/1999/xlink">
<symbol id="sym1">
   <circle cx="50" cy="50" fill="blue" r="20"/>
</symbol>
<use xlink:href="#sym1" x="30" y="30"/>
<use xlink:href="#sym1" x="50" y="50" opacity=".3"/>
</svg>
```

Scalable Vektor Grafik - Grundlagen

SVG - Annimationen

SVG-Objekte können zeitabhängig verändert werden.

Elemente

- `animate` - Animation eines einzelnen Attributes über die Zeit. Beispiel für Ausblenden (opacity) über die Zeit:

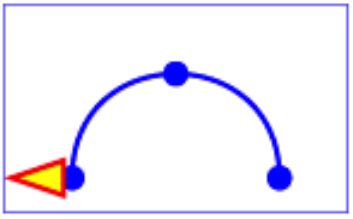
```
<rect> <animate attributeType="CSS"
attributeName="opacity" from="1" to="0" dur="5s"
repeatCount="indefinite" /> </rect>
```

- `set` - setzen eines Attributwertes für ein spezifische Zeit
- `animateMotion` - bewegt ein referenciertes Element entlang eines Pfades
- `animateColor` -Farbtransformation über die Zeit
- `animateTransform` - Transformation (Rotation, Skalierung etc.) über die Zeit
- Grundattribute: `from, by, to` zur Spezifikation der Startwerte, Schrittweite und Endwerte
- Abhängig v. d. Elementart weitere Attribute spezifizierbar

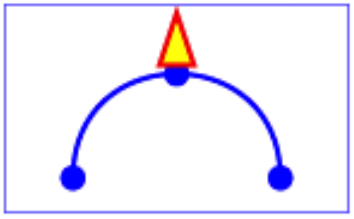
Scalable Vektor Grafik - Grundlagen

SVG - Annimationen

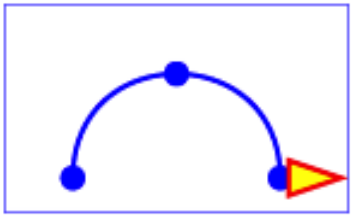
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="3cm" viewBox="0 0 500 300"
    xmlns="http://www.w3.org/2000/svg" version="1.1"
    xmlns:xlink="http://www.w3.org/1999/xlink" >
  <rect x="1" y="1" width="498" height="298"
      fill="none" stroke="blue" stroke-width="2" />
  <path id="path1" d="M100,250 C 100,50 400,50 400,250"
      fill="none" stroke="blue" stroke-width="7.06" />
  <circle cx="100" cy="250" r="17.64" fill="blue" />
  <circle cx="250" cy="100" r="17.64" fill="blue" />
  <circle cx="400" cy="250" r="17.64" fill="blue" />
  <path d="M-25,-12.5 L25,-12.5 L 0,-87.5 z"
      fill="yellow" stroke="red" stroke-width="7.06" >
    <animateMotion dur="6s" repeatCount="indefinite" rotate="auto" >
      <mpath xlink:href="#path1"/>
    </animateMotion>
  </path>
</svg>
```



At zero seconds



At three seconds



At six seconds

Scalable Vektor Grafik - Grundlagen

Software zur Darstellung und Erzeugung von SVG

- Direkte Browserunterstützung:
  - Firefox, Safari, Opera, Chrome
  - (derzeit) nicht in Internet Explorer
  - Diverse Plugins für Internet Explorer, z.B.: Adobe SVG Viewer (nicht weiterentwickelt), Google Chrome Frame
- Früher: Spezialsoftware (Standalone Viewer)
- Vektorgrafik-Editoren mit SVG-Import und Export z.B. Adobe Illustrator, CorelDraw
- SVG-orientierte Grafik-Editoren z.B. Inkscape (Open Source), Sketsa
- XML-Editoren, Keine Grafik-Unterstützung, nur Text-Syntax

Auch für 3D Grafiken gibt es solche Formate (z.B. VRML)

Zusammenfassung

Zusammenfassung

Scalable Vector Graphics

- Standardformat für Vektorgrafiken im WWW
- Kombinierbar mit anderen Web-Standards
- Elemente: Pfade, geometrische Primitive, Bezier, Kreis, Elipse,
- Text, Animationen, Transformationen und Füllungen
- Links, Symboldefinitionen etc.

Literatur

[1] Malaka, Butz, Hussmann (2009) - Medieninformatik: Eine Einführung (Pearson Studium - IT), Kapitel 7.1 und 7.2