2D-Vektorgraphik

Jörg Cassens

Medieninformatik
WS 2019/2020

Universität Hildesheim
medieninformatik
IMAI – Institut für Mathematik und Angewandte Informatik

Einleitung

- Zeichnen und Malerei uralte Kulturtechniken
- Bisher Fokus auf Rasterbildern
- Angemessen für Fotographien, also abgetastete Abbilder der Welt
- Zeichnungen lassen sich auch grundsätzlich anders codieren
  - Menge von Linien, Kurven, Flächen wie der Künstler sie angeordnet hat
  - Darstellung quasi eine Wiederholung der Pinselstriche
- Mathematisch als Vektoren und Polygone in einer zweidimensionale Ebene
- Während ein abgetastetes Bild eine feste Auflösung besitzt können Vektorgraphiken ohne Qualitätsverlust skaliert werden (modulo Rechengenauigkeit)
- Speicherbedarf von Rastergraphik steigt quadratisch zu seiner Kantenlänge; Vektorgraphik hat unabhängig von Auflösung den gleichen Speicherbedarf

Lernziele

- Grundlegende Konzepte der 2D-Vektorgraphik
- Darstellung einer Verarbeitungskette
- Schritte zur Darstellung von Vektorgraphiken am Bildschirm
- Grundlegende Animationsverfahren
- Zwei offene und verbreitete Codierungen für Vektorgraphiken

1 Grundlagen

1.1 Vektorraum

Koordinatensystem, Punkte, Geraden

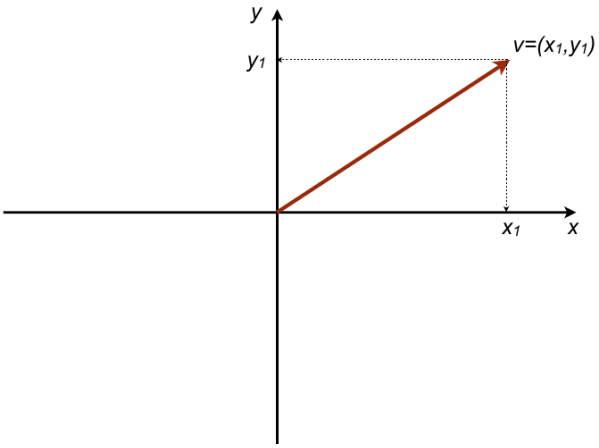
- Mathematische Grundlage: zweidimensionaler Vektorraum
- In der Regel kartesisches, orthogonales Koordinatensystem mit linear aufgeteilten Achsen
- Bezeichnung horizontal meist X, vertikal Y
- Punkt durch Koordinaten (x,y) eindeutig beschreibbar

- Gerade beschrieben mit Start- und Endpunkt
- Polygon aus mehreren Geraden, Endpunkt der einen Startpunkt der nächsten
  - Wenn der letzte Endpunkt gleich dem Startpunkt ist, ist das Polygon geschlossen
  - Sonst offen
- Bei SVG und den meisten anderen Systemen liegt der Ursprung links oben
- PostScript hat den Ursprung links unten

Vektorraum

- Vektorraum oder linearer Raum ist eine algebraische Struktur
- Elemente eines Vektorraumes heißen Vektoren
- Diese können addiert, subtrahiert werden oder mit Skalaren multipliziert bzw. durch diese dividiert werden
- Resultat einer solchen Operation ist immer Vektor desselben Vektorraumes
- Vektoren dienen zur Darstellung geometrischer Objekte (Koordinaten)
- In der Medieninformatik typischerweise zwei-, drei- oder vierdimensional

2D-Vektoren



Quelle: Frank Steinicke

Punkte, Gerade, Polygone

- Punkt wird durch eine Koordinate $P = (x_1, y_1)$ beschrieben
- Gerade wird durch zwei Koordinaten $L = \{(x_1, y_1), (x_2, y_2)\}$ beschrieben, dem Start- und Endpunkt
- Polygone bestehen aus mehreren zusammenhängenden Geraden, also $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
  - Geschlossenen Polygone: Endpunkt ist gleich dem Startpunkt, d.h. $(x_1, y_1) = (x_n, y_n)$

Konzepte

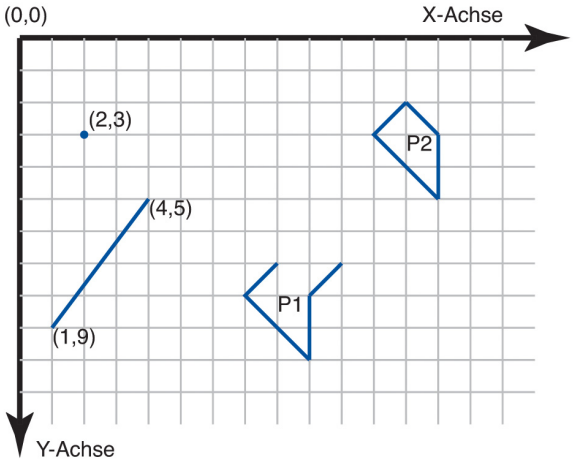


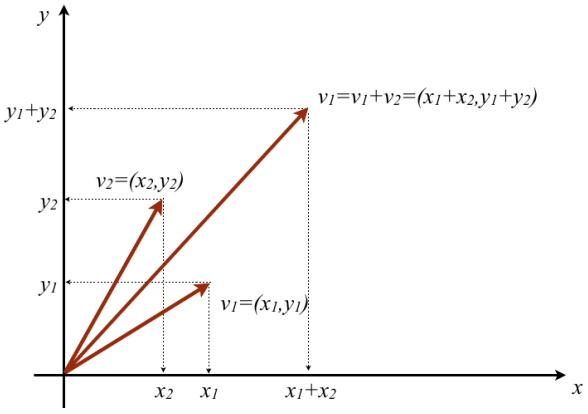
Abbildung 7.1: Zweidimensionaler Vektorraum mit Punkt (2,3), Gerade von (1,9) nach (4,5), offenem (P1) und geschlossenem Polygon (P2)

Addition von Vektoren

- Addition erfolgt komponentenweise
- Für n-dimensionale Vektoren v, w gilt:

$$v + w = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} + \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} v_1 + w_1 \\ \vdots \\ v_n + w_n \end{pmatrix}$$

Addition von Vektoren



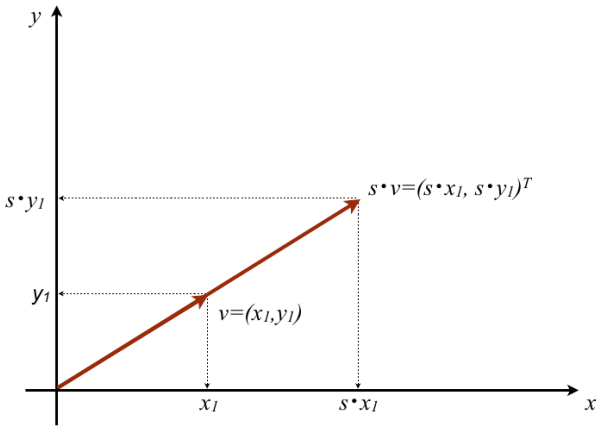
Quelle: Frank Steinicke

Skalieren von Vektoren

- Jeder Vektor v kann mit einer reellwertigen Zahl s skaliert werden:

$$s * v = s * \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} s * v_1 \\ \vdots \\ s * v_n \end{pmatrix}$$

Skalieren von Vektoren



Quelle: Frank Steinicke

Skalarprodukt

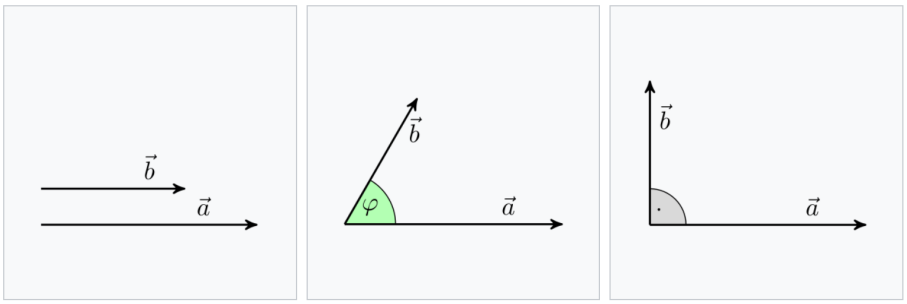
- Skalarprodukt (dot product) ist eine reellwertige Funktion
- Gegeben: Zwei Vektoren u und v
- Dann ist das Skalarprodukt $u \cdot v$ definiert wie folgt:

$$u \cdot v = u_1 * v_1 + u_2 * v_2 + \ldots + u_n * v_n$$

- Im zweidimensionalen Fall also:

$$u \cdot v = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 * v_1 + u_2 * v_2$$

Skalarprodukt



$\vec{a}$ und $\vec{b}$ gleichgerichtet
$\vec{a} \cdot \vec{b} = 5 \cdot 3 \cdot \cos 0^\circ = 15$

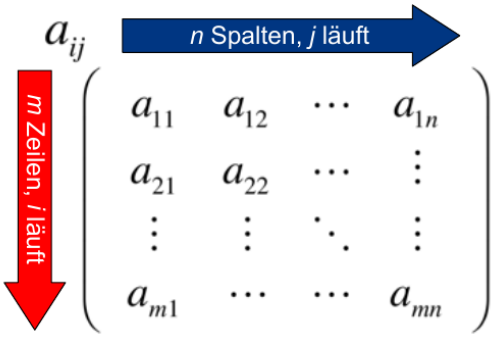
$\vec{a}$ und $\vec{b}$ im 60°-Winkel
$\vec{a} \cdot \vec{b} = 5 \cdot 3 \cdot \cos 60^\circ = 7{,}5$

$\vec{a}$ und $\vec{b}$ orthogonal
$\vec{a} \cdot \vec{b} = 5 \cdot 3 \cdot \cos 90^\circ = 0$

☞ de.wikipedia.org/wiki/Skalarprodukt

Matrizen

- Matrix (Plural: Matrizen) ist eine rechteckige Anordnung von Elementen



Matrizen

- Einsatz: Beschreibung von Transformationen
- Hauptsächlich quadratische Matrizen, $2 \times 2, 3 \times 3, 4 \times 4$

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}, n \in \{2,3,4\}$$

Matrizen: Multiplikation mit Skalaren

- Elementweise Multiplikation

$$s * \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} s * a_{11} & s * a_{12} & s * a_{13} \\ s * a_{21} & s * a_{22} & s * a_{23} \\ s * a_{31} & s * a_{32} & s * a_{33} \end{pmatrix}$$

Matrizen: Addition

- Elementweise Addition

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{pmatrix}$$

Matrizen: Multiplikation

- Multiplikation "Zeile mal Spalte"
- Gegeben: $(m \times n)$-Matrix A und $(n \times p)$-Matrix B
- Ergebnis: $(m \times p)$-Matrix C

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

Matrizen: Multiplikation

- Zeilen-/Spaltenweise Multiplikation

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31} = c_{11}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31} = c_{21}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$a_{31} * b_{11} + a_{32} * b_{21} + a_{33} * b_{31} = c_{31}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32} = c_{12}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$a_{i1} * b_{1j} + a_{i2} * b_{2j} + a_{i3} * b_{3j} = c_{ij}$$

Matrix-Vektor-Multiplikation

- Wichtiger Spezialfall
- Motivation: Matrix beschreibt eine Transformation, der Vektor einen Punkt
- D.h. Matrizen können benutzt werden, um graphische Objekte zu manipulieren

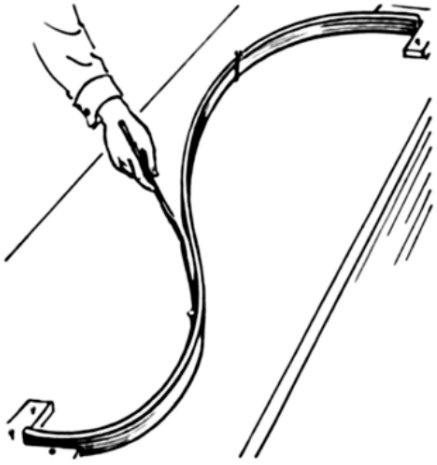
$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} * \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} a_{11} * v_1 + \dots + a_{1n} * v_n \\ \vdots \\ a_{n1} * v_1 + \dots + a_{nn} * v_n \end{pmatrix}$$

Kurven

- Weitere Elemente neben Punkten, Geraden, Polygonen
  - Kreis durch Mittelpunkt und Radius
  - Interpolationskurven durch Kontroll- und Stützpunkte
- Interpolationskurven:
  - Darstellung wirklich beliebiger Kurven eher unüblich, in der Praxis Annäherung
  - In der Praxis oft: Splines
  - Begriff aus dem Schiffbau: mit Hilfe von elastischen Latten, die an bestimmten Punkten fixiert harmonische (und teilweise optimale Formen) zu bilden

1.2 Splines

Splines



Quelle: Frank Steinicke

Splines

- Ein Spline n-ten Grades ist stückweise aus Polynomen maximal n-ten Grades zusammengesetzt
- Falls die Polynome linear sind ist der Spline linear, d.h. ein Polygonzug ist ein linearer Spline
- Für jedes Segment können Randbedingungen angegeben werden:
  - Steigung
  - Krümmung
  - Krümmungsveränderung
  - Erste bis dritte Ableitung

  jeweils an den beiden Endpunkten
- Häufig in Graphikprogrammen durch Kontrollinien angegeben
- Richtung der Kontrollinien Steigung, Länge Steifigkeit

Polynome

- Summe von Vielfachen von Potenzen mit natürlichzahligen Exponenten einer Variable

$$
\begin{aligned}
P(x) &= \sum_{i=0}^{n} a_i * x^i, n \geq 0 && (1)\\
&= a_n * x^n + \ldots + a_1 * x^1 + a_0 * x^0 && (2)\\
&= a_n * x^n + \ldots + a_1 * x + a_0 && (3)
\end{aligned}
$$

Splines

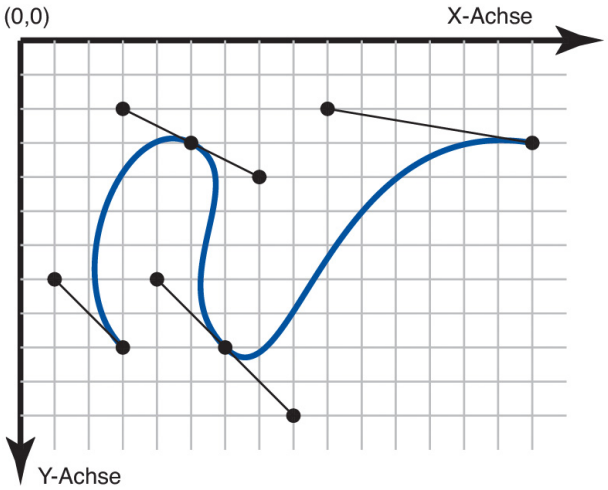


Abbildung 7.2: Eine Spline-Kurve mit Kontrolllinien an den Grenzpunkten zwischen ihren Segmenten

Grad der Polynome

- Meist kubisch oder quadratisch
- Höhergradige polynomiale Kurven erhöhen Rechenaufwand und erschweren Kontrolle
- Mit Polynomen niedrigeren Grades kann die Form der Kurve nicht flexibel genug modelliert werden

Bézier-Splines

- Besondere Familie von Interpolationskurven: Bézier-Kurven
- Konstruiert aus Bernsteinpolynomen
- Aus Beziér-Kurven werden die Bézier-Splines zusammengesetzt
- Unabhängig voneinander von Pierre Bézier bei Renault und Paul de Casteljau bei Citroën entwickelt
- Nach Art der verwendeten Polynome Bézier-Kurven ersten, zweiten, dritten Grades
- Bézier-Kurven n-ten Grades beschrieben durch $n + 1$ Kontrollpunkte
- Diese bilden das Stützpolygon
  - Bézier-Kurve ersten Grades hat ein Stützpolygon aus zwei Punkten, ist also eine Linie
- Kurvenverlauf durch den Algorithmus von Casteljau rekursiv berechenbar (siehe Graphik)

Beispiel Bézier-Kurven

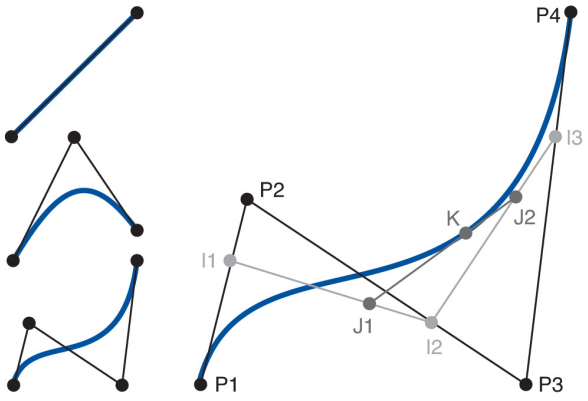


Abbildung 7.3: Links von oben nach unten Bézier-Kurven ersten, zweiten und dritten Grades, rechts eine Darstellung des Algorithmus von Casteljau

Bernsteinpolynome: Grundlagen

Das $n+1$ Bernstein-Basis-Polynom zur Basis n ist definiert als:

$$b_{\nu,n}(x) = \binom{n}{\nu} x^{\nu} (1-x)^{n-\nu}, \quad \nu = 0, \ldots, n.$$

wobei

$$\binom{n}{\nu}$$

ein Binomial-Koeffizient ist.

Die lineare Kombination von Bernstein-Basis-Polynomen

$$B_n(x) = \sum_{\nu=0}^{n} \beta_\nu b_{\nu,n}(x)$$

wird Bernstein-Polynom genannt, die Koeffizienten β_ν Bernstein-Koeffizienten.

Bernsteinpolynome

Darstellung der ersten Basis-Polynome

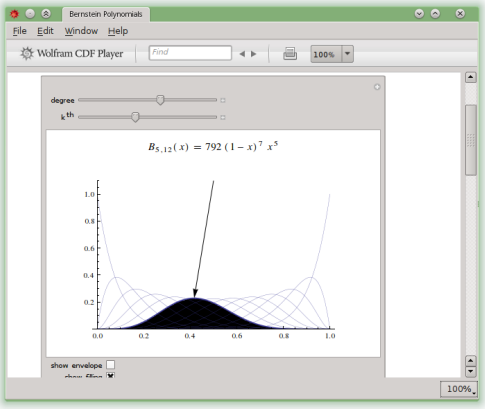
$$n = 0 : b_{0,0}(x) = 1$$

$$n = 1 : b_{0,1}(x) = 1 - x, b_{1,1}(x) = x$$

$$n = 2 : b_{0,2}(x) = (1-x)^2, b_{1,2}(x) = 2x(1-x), b_{2,2}(x) = x^2$$

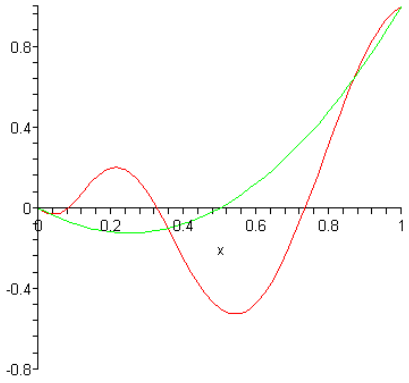
$$n = 3 : b_{0,3}(x) = (1-x)^3, b_{1,3}(x) = 3x(1-x)^2,$$
$$b_{2,3}(x) = 3x^2(1-x), b_{3,3}(x) = x^3$$

Bernsteinpolynome: Beispiel



☞ demonstrations.wolfram.com/BernsteinPolynomials/

Visualisierung Bernsteinpolynome

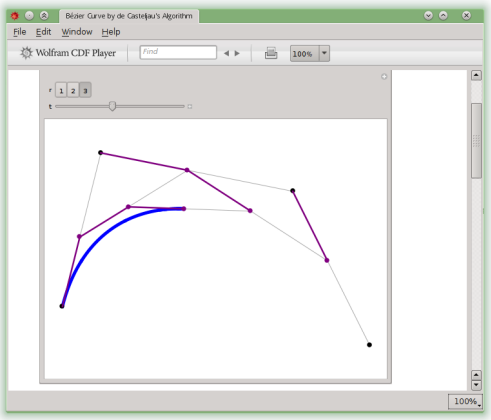


☞ Wikipedia-User James Pic

Casteljau

- Um die gezeigte Kurve mit den Kontrollpunkten $P1$ bis $P4$ zu interpolieren werden zunächst Punkte auf den Linien des Stützpolygons interpoliert
- Die lineare Interpolation B zwischen zwei Punkten A und B ist dabei für einen Interpolationswert i: $0 \leq i \leq 1 : B = (1 - i) * A + i * C$
- Im Beispiel: I1 zwischen P1 und P2, I2 zwischen P2 und P3, I3 zwischen P3 und P4
- Zwischen den Interpolierten Punkten wird weiter interpoliert
- J1 wandert von I1 nach I2, J2 von I2 nach I3
- Zwischen J1 und J2 wird dann der Kurvenpunkt K interpoliert
- So kann für jeden Wert von i zwischen 0 und 1 der Kurvenwert K berechnet werden
- Beispiel zeigt Interpolation für $i = 0.66$

Casteljau: Beispiel



☞ demonstrations.wolfram.com/BezierCurveByDeCasteljausAlgorithm/

Harmonische Spline-Kurven

- Aus den beschriebenen Segmenten lassen sich beliebig lange harmonisch gebogenen Spline-Kurven zusammenfügen
- An den Grenzpunkten durch Manipulation der Kontrollpunkte sicherstellen, daß Steigung und Krümmung benachbarter Element gleich sind
- Parallele Kontrollinien = Steigung der Kurvenstücke ist identisch

- Gleich lange Kontrollinien = Krümmung identisch
- Solcherart gebildete Bézier-Splines ahmen das Verhalten der Holzleisten nach

1.3 Transformation

Geometrische Transformationen

- Innerhalb der Koordinatensysteme können Punkte geometrisch transformiert, z.B. verschoben werden
- Durch Verschiebung der Koordinaten einzelner Punkte können alle Objekte, die durch Punkte beschrieben werden, geometrisch transformiert werden
- Bei der Klasse linearer Transformationen ergeben sich die neuen Koordinaten durch lineare Funktionen

Geometrische Transformationen

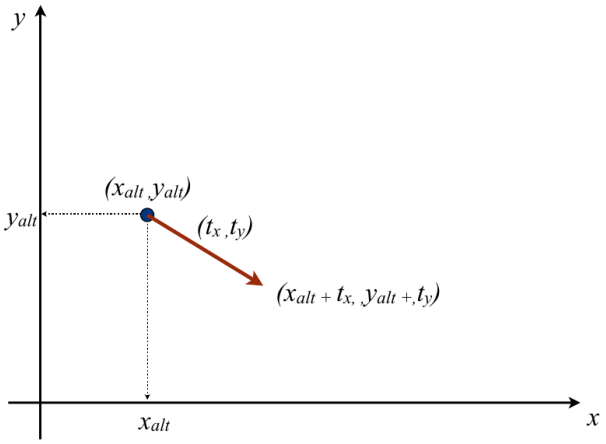
- Grundlegende lineare Transformationen sind:
  - Translation
  - Rotation
  - Skalierung
  - Scherung
- Nebenbemerkung: Die Repräsentation von Beziér-Splines macht diese invariant zu Rotation, Skalierung, Verschiebung

Translation

Bei der Translation wird jeder betroffene Punkt um den gleichen Vektor (t_x, t_y) verschoben

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x_{alt} & + & t_x \\ y_{alt} & + & t_y \end{pmatrix}$$

Beispiel Translation



Quelle: Frank Steinicke

(Uniforme) Skalierung

- Skalierung von $(x_1, y1)$ um s_x und s_y
- Skalierung heißt uniform, wenn $s_x = s_y$:

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = s \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} sx_{alt} \\ sy_{alt} \end{pmatrix}$$

- Nicht-uniforme Skalierung:

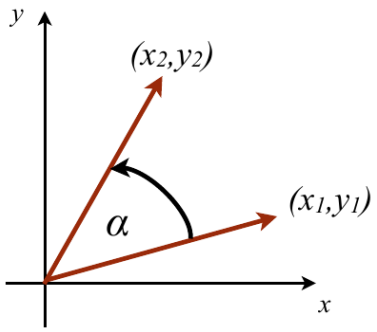
$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} s_x x_{alt} \\ s_y y_{alt} \end{pmatrix}$$

Rotation

Mathematisch positive Rotation (gegen den Uhrzeigersinn) um den Winkel α

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} \cos\alpha x_{alt} - \sin\alpha y_{alt} \\ \sin\alpha x_{alt} + \cos\alpha y_{alt} \end{pmatrix}$$

Beispiel Rotation



Quelle: Frank Steinicke

Scherung

Eine Scherung entlang der x-Achse verändert x-Koordinate in Abhängigkeit von der y-Koordinate

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} x_{alt} + my_{alt} \\ y_{alt} \end{pmatrix}$$

Zusammenfassendes Beispiel

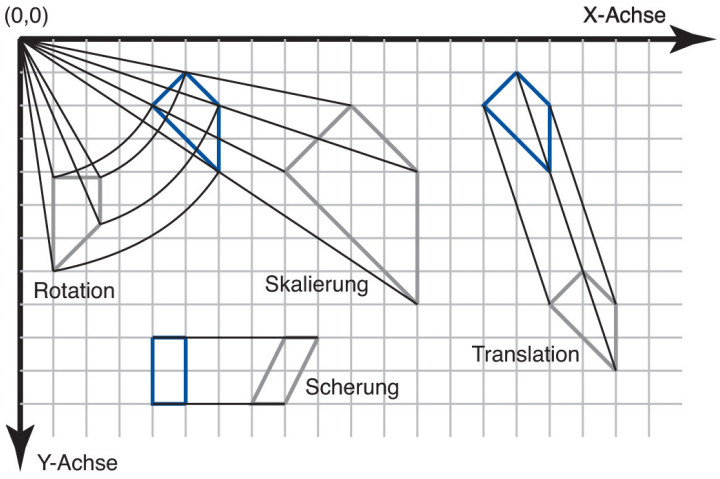


Abbildung 7.4: Die elementaren linearen Transformationen

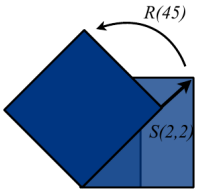
Weitere Transformationen

- Mit Hilfe der definierten Transformationen sind weitere konstruierbar
- Spiegelung: Skalierung um den Faktor -1
- Rotation um andere Punkte als den Mittelpunkt durch Translation, danach Drehung, danach Translation
- Alle Operationen mit Ausnahme der Translation als Matrix-Multiplikation ausdrückbar
- Matrix-Multiplikation ist assoziativ

Beispiel: Komposition

- Komposition von Skalierung und Rotation

$$\underbrace{\begin{pmatrix} cos(45) & -sin(45) \\ sin(45) & cos(45) \end{pmatrix}}_{\text{(iii)}} \cdot \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}}_{\text{(ii)}} = \underbrace{\begin{pmatrix} 2 \cdot cos(45) & -2 \cdot sin(45) \\ 2 \cdot sin(45) & 2 \cdot cos(45) \end{pmatrix}}_{\text{(i)}}$$

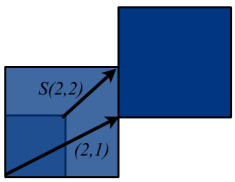


Quelle: Frank Steinicke

Beispiel: Komposition

- Komposition von Skalierung und Translation

$$\underbrace{\begin{pmatrix} 2 \\ 1 \end{pmatrix}}_{\text{(ii)}} + \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}}_{\text{(i)}}$$



Quelle: Frank Steinicke

Homogene Koordinaten

- Matrix-Multiplikation ist assoziativ
- Daher wäre es hilfreich, falls Translation auch so ausgedrückt werden könnte
- Lösung: Hinzunahme einer 3. Dimension bei allen Berechnungen
  - Sogenannte homogene Koordinaten, da alle Transformationen als Matrix darstellbar
  - In homogenen Koordinaten wird jeder 2D-Punkt (x, y) repräsentiert durch $(x, y, 1)$
  - $(x, y, 1)$ und (x, y, W) repräsentieren den gleichen Punkt genau dann, wenn $W \neq 0$
  - Matrizen erhalten in der dritten Zeile/Spalte jeweils zwei Nullen und eine 1 auf der Diagonale

Homogene Koordinaten

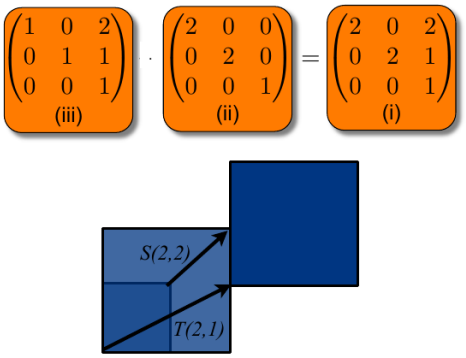
$$\begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \text{ sowie } \begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix} \Rightarrow \begin{pmatrix} m_{1,1} & m_{1,2} & 0 \\ m_{2,1} & m_{2,2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Und damit gilt für die Translation

$$\begin{pmatrix} x_{neu} \\ y_{neu} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{alt} + t_x \\ y_{alt} + t_y \\ 1 \end{pmatrix}$$

Beispiel: Komposition

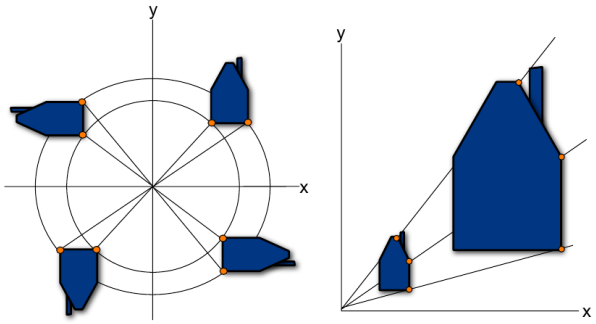
- Komposition von Skalierung und Translation



Quelle: Frank Steinicke

Komposition

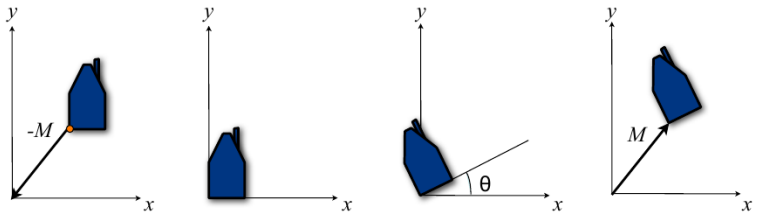
- Rotation und Skalierung finden relativ zum Ursprung statt



Quelle: Frank Steinicke

Komposition

- Um Objekte an einem beliebigen Bezugspunkt zu rotieren oder zu skalieren, ist eine Folge von Transformationen notwendig

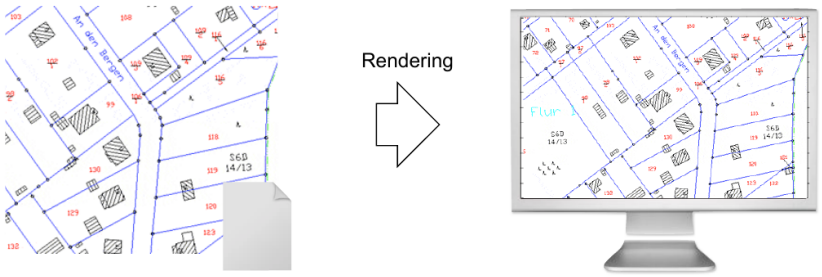


Quelle: Frank Steinicke

2 2D Pipeline

2D-Rendering

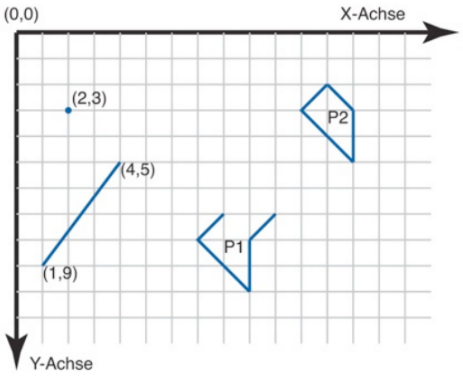
- Ausgang: 2D-Vektorraum
- Ziel: 2D-Rasterbildschirm



Quelle: Frank Steinicke

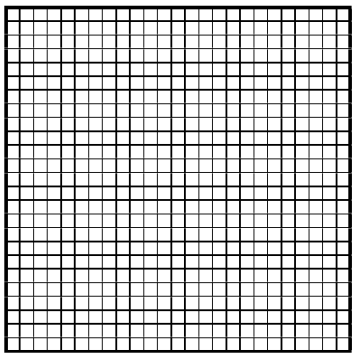
2D-Vektorraum

- Mathematisch exakte Beschreibung einer Menge von Punkten



2D-Rasterbildschirm

- Diskretes Raster von Pixeln, auf das Elemente des 2D-Vektorraumes abgebildet werden müssen (Abtastung)



Quelle: Frank Steinicke

2D Rendering Pipeline

- Bei Berechnung und Darstellung einer 2D-Vektorgraphik gibt es eine etablierte Abfolge von Arbeitsschritten
- Diese wird im allgemeinen als Rendering-Pipeline beschrieben

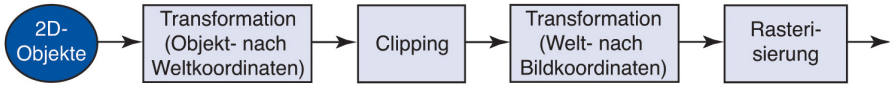
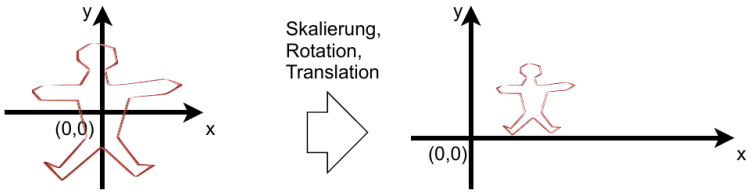


Abbildung 7.5: Die 2D Rendering Pipeline

2.1 Weltkoordinaten

Objekt- und Weltkoordinaten

- 2D-Primitive werden in Objektkoordinaten beschrieben und dann in Weltkoordinaten transformiert



Quelle: Frank Steinicke

Szenegraph

- Alle eingeführten geometrischen Primitive sind durch Punkte beschrieben
- Theoretisch könnten die Objekte alle an dem Ort, an dem sie später zu sehen sein sollen, beschrieben werden
- Praktisch meist hierarchisch organisiert
  - Gemeinsam verschieben
  - Gemeinsam aninmiert
  - Objekteigenschaften wie Strichbreite, Füllfarbe für Gruppen von Objekten festgelegt werden können
- Um diese effizient anwenden zu können werden die Objekte in einem Szenegraphen organisiert

Szenegraph

- An den Blättern geometrische Objekte
- An den inneren Knoten Transformationen oder Gruppierungen
- Im einfachsten Fall ein Baum
- Bei mehrfacher Verwendung eines Objektes/einer Gruppe ein DAG (Gerichteter Azyklischer Graph)
- Mehrfachnutzung eines Objektes spart Modellierungsaufwand
- Ein Objekt kann dann durch mehrere Transformation an die jeweils richtige Stelle verschoben werden
- Geometrische Eigenschaften werden aufmultipliziert
- Nichtgeometrische Eigenschaften werden vererbt
- Die Definition, was ein Szenegraph ist, wird unterschiedlich gesehen

Beispiel Szenegraph

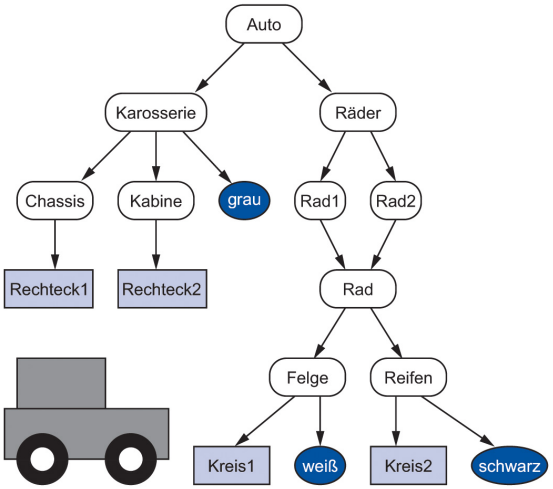


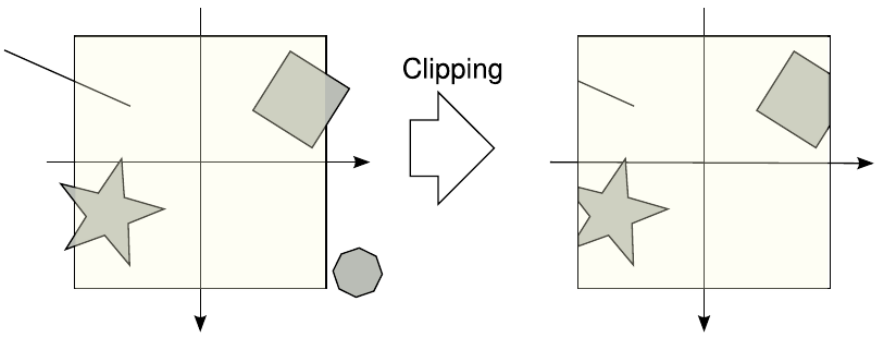
Abbildung 7.6: Szenegraph eines Autos mit zwei identischen Rädern

2.2 Clipping

Clipping

- Zweidimensionaler Vektorraum, in dem unsere Objekte liegen, erst einmal unendlich groß
- Praktisch vorkommende Ausgabegeräte endlich
- Nur ein Unterraum darstellbar
- Prinzipiell: Sichtfenster festlegen
- Liegen alle Objekte darin, können sie ausgegeben werden
- Liegen sie komplett außerhalb, können sie weggelassen werden (durch Bounding Box schnell feststellbar)
- Liegen sie teilweise im, teilweise außerhalb des Sichtfensters muß es beschnitten werden (clipping)

Beispiel: Clipping

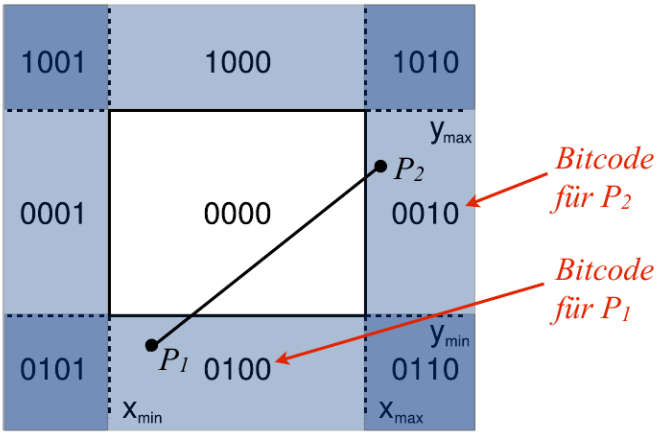


Quelle: Frank Steinicke

Clipping

- Clipping von Punkten: Punkt liegt innerhalb des Sichtfensters
- Clipping beliebiger Polygone auf Clipping von Linien zurückführen
- Klassisches Verfahren für das Clipping gerader Linien am einem rechteckigen Ausschnitt ist das Line Clipping von Cohen und Sutherland
  - Gegeben Linie mit Start- und Endpunkten P_1 und P_2 sowie ein Sichtfenster mit Koordinaten x_{min}, x_{max}, y_{min} und y_{max}
  - Unterteile den Vektorraum in 9 Teilbereiche aufgeteilt
  - Jeder Teilraum erhält einen 4 Bit langen Code

Beispiel: Bitcode



Quelle: Frank Steinicke

Bitmuster Cohen und Sutherland

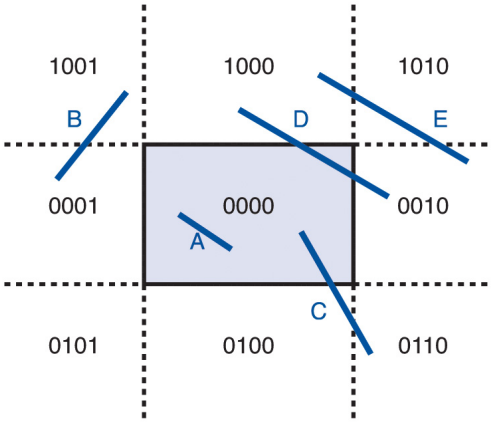
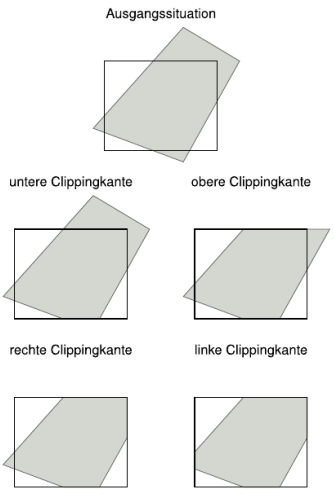


Abbildung 7.7: Line Clipping nach Cohen und Sutherland

Cohen Sutherland (contd.)

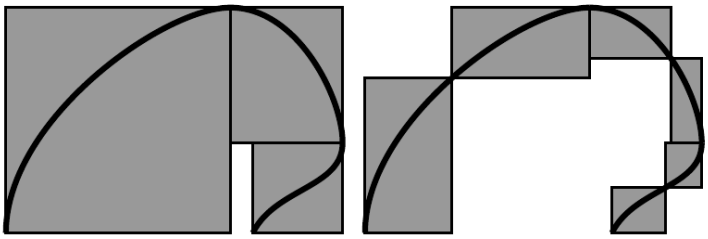
- Start- und Endpunkt der zu schneidenden Linie wird der Code des Teilraums zugewiesen, in dem er liegt
  1. Codes P_1 und P_2 mit **"or"** verknüpft **ergeben** 0000: Objekt liegt im Viewport (Beispiel A: $0000 \vee 0000 = 0000$)
  2. Codes P_1 und P_2 mit **"and"** verknüpft **ergeben nicht** 0000: P_1 und P_2 liegen auf der gleichen "Seite" außerhalb des Viewports, das Objekt kann weggelassen werden (B: $0001 \vee 1000 = 1001; 0001 \wedge 1000 = 0000$)
  3. Ansonsten muss die Linie – abhängig vom Code – geschnitten werden
- Falls der Code von P_1 nicht 0000 ist, muß die Gerade mit den von dort erreichbaren Rändern des Viewports geschnitten werden
  - Vom Feld 0010 ist z.B. nur der rechte Rand zu erreichen (Beispiele D und E)
- Schneidet die Linie das Randsegment wird der Schnittpunkt S als neuer Wert von P_1 bestimmt
- Falls auch der Code von P_2 nicht 0000 ist muß analog geschnitten werden

Polygon-Clipping



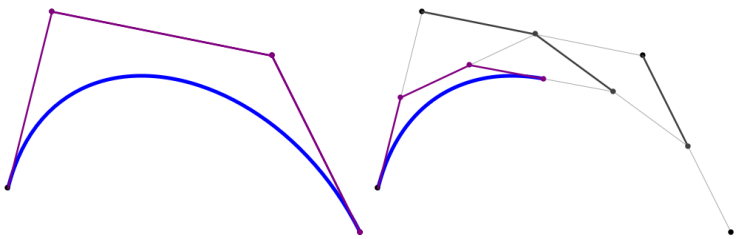
Nach Sutherland and Hodgman (Bildquelle: Frank Steinicke)

Kurven-Clipping



☞ T. W. Sederberg, BYU, Computer Aided Geometric Design, Course Notes

Kurven-Clipping



Reminder: Casteljau

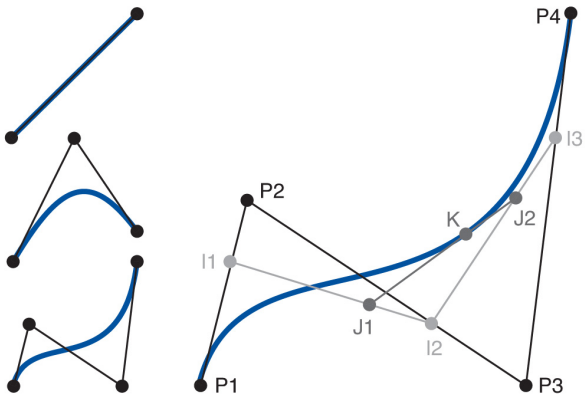
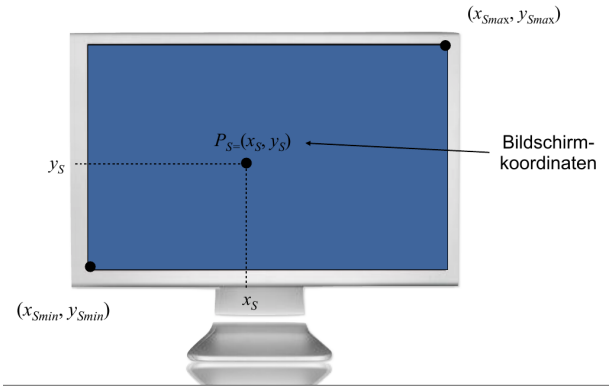


Abbildung 7.3: Links von oben nach unten Bézier-Kurven ersten, zweiten und dritten Grades, rechts eine Darstellung des Algorithmus von Casteljau

2.3 Bildkoordinaten

Bildkoordinaten

- Auflösung und Größe des Bildschirms bzw. Fensters bestimmt Koordinaten



Quelle: Frank Steinicke

Welt- nach Bildkoordinaten

- Die darzustellenden Primitive wurden mit Hilfe des Szenegraphen an die richtige Stelle transformiert
- Die Darstellung wurde mittels Clipping auf ein definiertes Sichtfenster begrenzt
- Jetzt muß in Weltkoordinaten umgerechnet werden

$$x_{\text{bild}} = x0_{\text{bild}} + (x_{\text{welt}} - x0_{\text{welt}}) * (x1_{\text{bild}} - x0_{\text{bild}}) / (x1_{\text{welt}} - x0_{\text{welt}})$$
$$y_{\text{bild}} = y0_{\text{bild}} + (y_{\text{welt}} - y0_{\text{welt}}) * (y1_{\text{bild}} - y0_{\text{bild}}) / (y1_{\text{welt}} - y0_{\text{welt}})$$

Welt- nach Bildkoordinaten

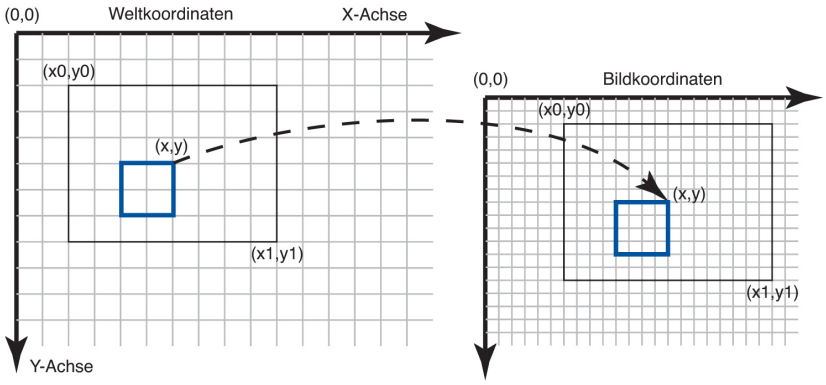


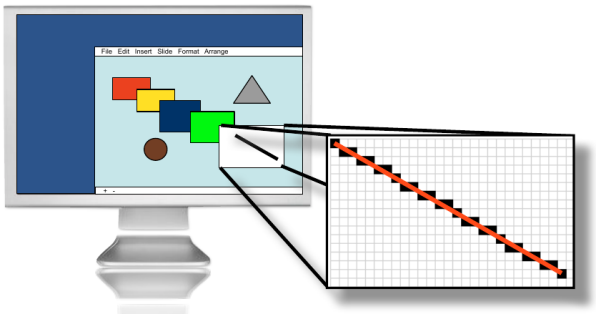
Abbildung 7.8: Transformation von Weltkoordinaten in Bildkoordinaten

$$x_{\text{bild}} = x0_{\text{bild}} + (x_{\text{welt}} - x0_{\text{welt}}) * (x1_{\text{bild}} - x0_{\text{bild}}) / (x1_{\text{welt}} - x0_{\text{welt}})$$
$$y_{\text{bild}} = y0_{\text{bild}} + (y_{\text{welt}} - y0_{\text{welt}}) * (y1_{\text{bild}} - y0_{\text{bild}}) / (y1_{\text{welt}} - y0_{\text{welt}})$$

2.4 Rasterisierung

Rasterisierung

- Objekte in Bildschirmkoordinaten müssen in Pixel-Farbinformationen transformiert werden



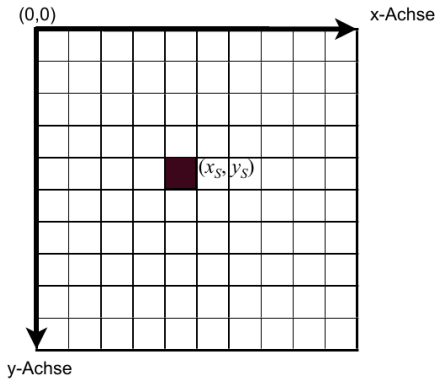
Quelle: Frank Steinicke

Rasterisierung

- Einige Ausgabegeräte können Vektordaten direkt darstellen
  - Plotter, CNC, Laserprojektoren
- Zumeist aber Umrechnung in Pixel
- Begriff: **Rasterisierung**
- Nichts anderes als eine Abtastung, also gelten im besonderen das zum Nyquist-Shannon-Theorem und das zum Alias gesagte
- Abtastrate entspricht der Ortsauflösung des Pixelrasters
- Abtastgenauigkeit entspricht der Farbtiefe

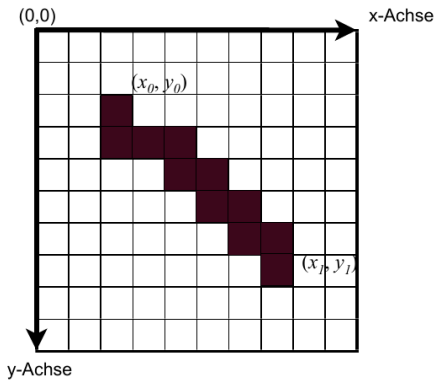
Beispiel: Rasterisierung eines Punktes

- Farbe des Pixels, das den Punkt enthält, wird angepaßt



Quelle: Frank Steinicke

Rasterisierung einer Geraden



Quelle: Frank Steinicke

Naive Rasterisierung SW

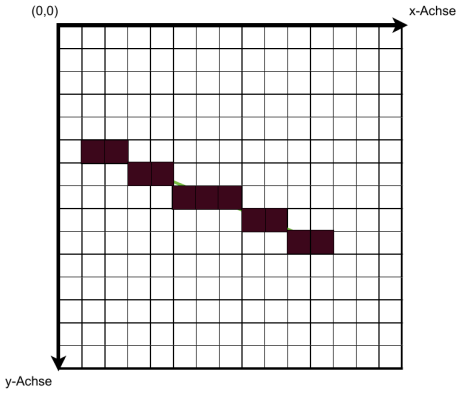
- Zeichnen einer Linie mit einem Bit
- Naiv: Wir nehmen die Steigung der Linie $m = (y_1 - y_2)/(x_1 - x_2))$
- Schleife über alle x Werte zwischen x_1 und x_2. Der zugehörige y-Wert wird als $round(m * (x - x_1))$ genommen und das Pixel eingefärbt

Rasterisierung SW

- Nachteile des naiven Vorgehens:
  - Sehr rechenaufwendig
    - Multiplikation und Rundung für jedes Pixel
- Mehrere bessere Algorithmen wurden vorgeschlagen.
- Hier **Bresenham**: Ich gehe in Richtung der schnelleren Änderung, merke mir die Fehler, und mache ab und zu einen Schritt zum Ausgleich der Fehler
  - Nur noch Addition und Vergleich auf Integer
  - Details der Implementierung variieren

Beispiel: Bresenham

- Wechsel der Änderungen



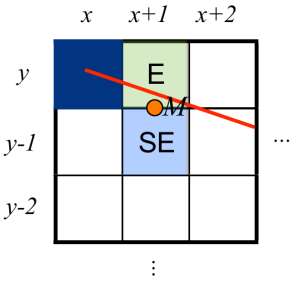
Quelle: Frank Steinicke

Bresenham

- Explizite Funktion einer Geraden: $y = f(x) = mx + c$
- Implizite Funktion einer Geraden: $F(x, y) = y - mx + c$
- Annahme: $-1 < m < 0$
- Unterscheidung von drei Fällen
  - $F(x, y) = 0 \implies (x, y)$ auf der Geraden
  - $F(x, y) > 0 \implies (x, y)$ oberhalb der Geraden
  - $F(x, y) < 0 \implies (x, y)$ unterhalb der Geraden

Bresenham

- Sei Pixel (x, y) bereits als Linienfragment markiert
- Wegen $-1 < m < 0$ ist das nächste Pixel entweder E(east) oder S(outh)E(east)



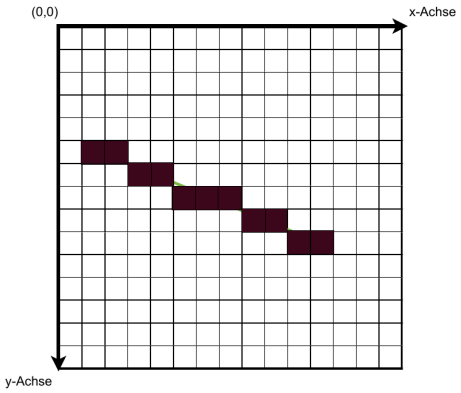
Quelle: Frank Steinicke $M := (x + 1, y - \frac{1}{2})$

$d := F(M) = F(x + 1, y - \frac{1}{2})$

$d < 0 \implies$ E, sonst SE

Beispiel: Bresenham

- Wechsel der Änderungen



Quelle: Frank Steinicke

Antialiasing

- Unter Zuhilfenahme von Graustufen oder Farbabstufungen läßt sich eine visuell bessere Darstellung erreichen
- Dieses Antialiasing mindert den oben zu sehenden Treppencharakter
- Grundidee: zu jedem x-Wert nicht genau einen y-Wert zu setzen, sondern mehrere umliegende Pixel in Abhängigkeit ihres Abstands zur idealen Linie heller oder dunkler einfärben
- Algorithmus von **Wu**: läuft über alle x-Werte und färbt immer die beiden Pixel der nächstgelegenen y-Werte ein
- Je näher das Pixel an der idealen Linie liegt, desto dunkler wird es eingefärbt

Beispiel Algorithmus von Wu

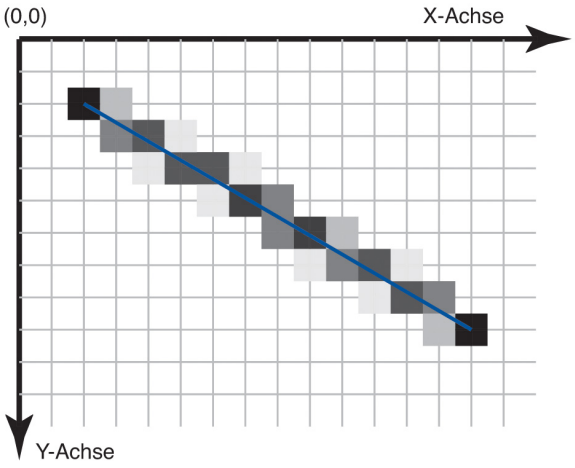
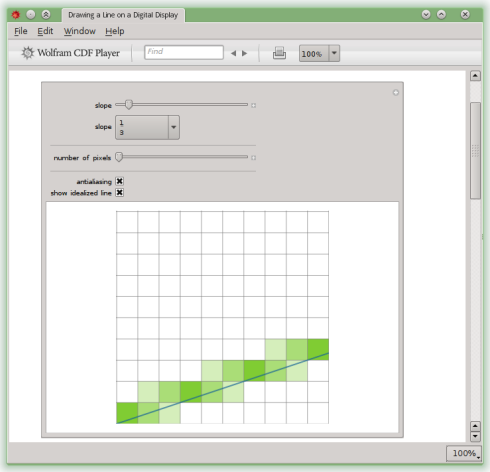


Abbildung 7.10: Rasterisierung einer Linie nach dem Algorithmus von Wu

Line Drawing: Beispiel



☞ demonstrations.wolfram.com/DrawingALineOnADigitalDisplay/

Rasterisierung gefüllter Polygone

- Wie lassen sich geschlossene Polygone als Fläche füllen?
- Verschiedene Algorithmen, mit unterschiedlichen Annahmen
- Verbreitete Grundidee: **Scanline** Algorithmen
- Wenn man von einem beliebigen Punkt einen Strahl zum Rand des Zeichenbereichs schickt und zählt, wie viele Polygonkanten er schneidet, dann definieren wir die Anzahl der Schnitte als Parität
- Punkte mit gerader Parität liegen dabei außerhalb der Polygone

Beispiel Scanline

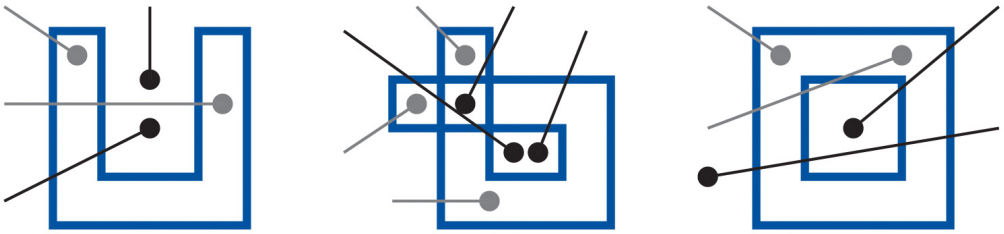


Abbildung 7.11: Parität verschiedener Punkte innerhalb und außerhalb von Polygonen: Schwarz bedeutet gerade und grau bedeutet ungerade Parität.

Scanline

- Bestimme für jede Zeile von Pixeln (Scanline) die Schnittpunkte mit den Kanten des Polygons und sortiere sie aufsteigend nach der x-Koordinate
- Ermittle für jedes Pixel die Parität
- Färbe alle Pixel mit ungerader Parität mit der Füllfarbe

Beispiel Scanline

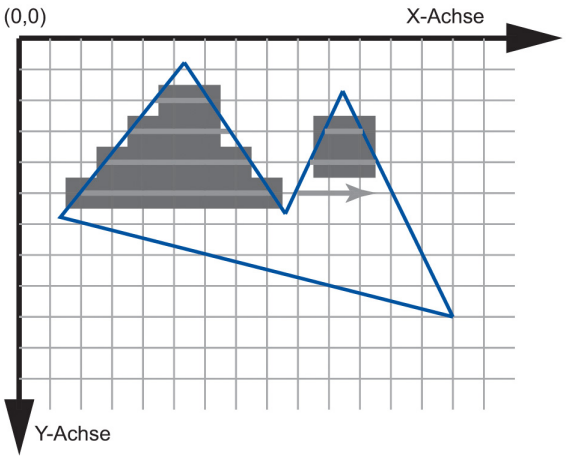
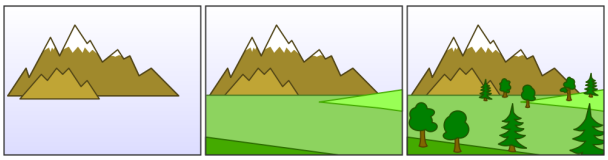


Abbildung 7.12: Ausfüllen eines Polygons mit dem Scanline-Algorithmus

Painter's Algorithm

- Wie geht man mit mehreren, sich gegenseitig teilweise verdeckenden, Polygonen um?
- Genauere Betrachtungen gehören eher in die Generative Graphische Datenverarbeitung (Computergraphik)
- Hier: Grundidee des **Painter's algorithm**:
- Da alle vorkommenden geometrischen Formen in einer Ebene liegen genügt es zur Behandlung von Verdeckungen in 2D-Graphik alle Primitive in der richtigen Reihenfolge zu zeichnen (von hinten nach vorne)



☞ Wikipedia: Painter's algorithm

3 Kodierung

Kompression?

- Keine medienspezifische, auf den Wahrnehmungsapparat zugeschnittene Kompression
- Aber Daten insgesamt noch recht klein
- Evtl. textbasiert und mit allgemeinen Verfahren komprimierbar

PostScript

- Ursprünglich von Adobe 1984 als Seitenbeschreibungssprache für die Ansteuerung von Druckern entwickelt
- Geräteunabhängige Formatierung von Text sowie Raster- und Vektorgraphiken
- Turing-mächtige Programmiersprache
- EPS (Encapsulated PostScript) als Vektorgraphikformat
- Als Darstellungsformat inzwischen von PDF abgelöst (nicht Turing-mächtig)
- PDF hat deutlich bessere Kompression
- PostScript hat den Ursprung 0,0 in der unteren linken Ecke einer Seite
- Längeneinheit ist der (PostScript-) Punkt

Beispiel PostScript

```
%!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
```

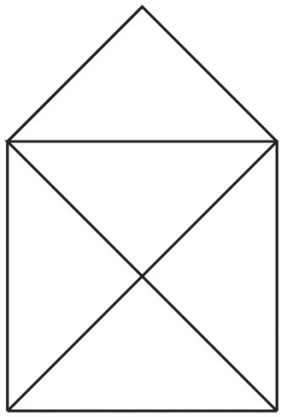


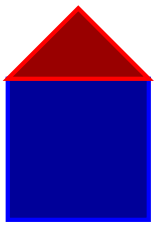
Abbildung 7.13: „Das ist das Haus vom Nikolaus“ in PostScript

Scalable Vector Graphics (SVG)

- Explizit für den Austausch skalierbarer Graphiken im WWW
- Standardisiert durch das W3C
- Basis XML
- Enthält Elemente wie Geraden, Polygone, Kurve, Kreissegmente, Splines
- Ursprung 0,0 in der oberen linken Bildecke
- Hat ein User-Koordinatensystem das per default dem Pixel-Koordinatensystem des Ausgabegerätes entspricht, aber geändert werden kann

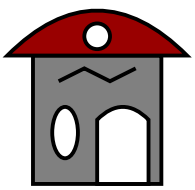
Beispiel SVG I

```
<?xml version="1.1"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <rect stroke="blue" stroke-width="3px"
    fill="#000099" x="100" y="100"
    width="100" height="100"/>
  <polygon stroke="red" stroke-width="3px"
    fill="#990000"
    points="100,100 200,100 150,50" />
</svg>
```



Beispiel SVG II

```
<?xml version="1.1"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <rect stroke="black" stroke-width="3px" fill="grey"
    x="100" y="100" width="100" height="100"/>
  <path stroke="black" stroke-width="3px"fill="#990000"
    d="M 80 100 L 220 100 Q 150 30 80 100 z" />
  <path stroke="black" stroke-width="3px" fill="white"
    d="M 150 200 L 190 200 L 190 150 Q 170 130 150 150
    L 150 200 z" />
  <circle stroke="black" stroke-width="3px" fill="white"
    cx="150" cy="85C r="10"/>
  <ellipse stroke="black" stroke-width="3px" fill="white"
    cx="125" cy="160" rx="10" ry="20"/>
  <polyline stroke="black" stroke-width="3px" fill="none"
    points="120,120 140,110 160,120 180,110"/>
</svg>
```

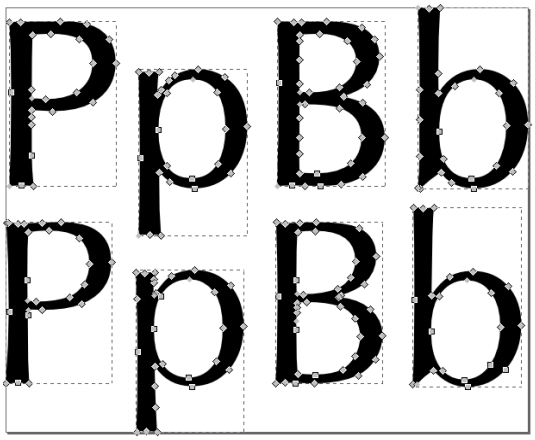


Path-Syntax

- Elemente für Rechtecke, Kreise, Ellipsen und Polylinien fast selbsterklärend
- Die Bedeutung der Syntax für Pfade verdient ein kurze Erläuterung

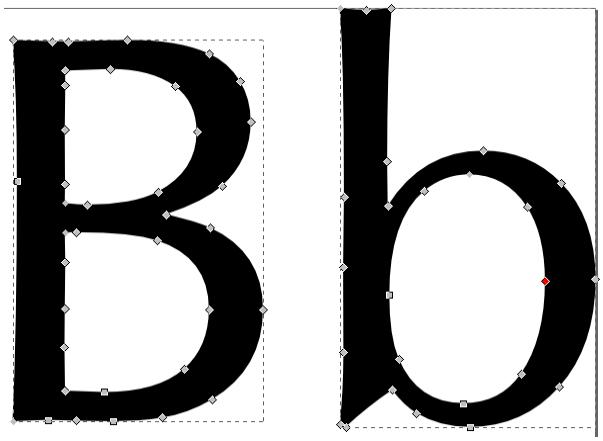
- `<path 'stroke=black' stroke-width='3px' fill='white' d='M 80 100 L 220 100 Q 150 30 80 100 z'/>`
  - Analog PostScript-Pfad, aber Befehl vor Parameter
  - Move to 80, 100
  - Line to 220, 100
  - Quadratic Bézier curve,
    - Kontrollpunkt 150,30
    - Endpunkt 80,100
  - z schließt den Pfad

Beispiel Font



Oben URW Classico (PostScript, kubische Bezier-Splines), unten Softmaker Opus (Truetype, quadratische Bezier-Splines)

Beispiel Font (Detail)



Detail PostScript: URW Classico

4 Animation

Animation

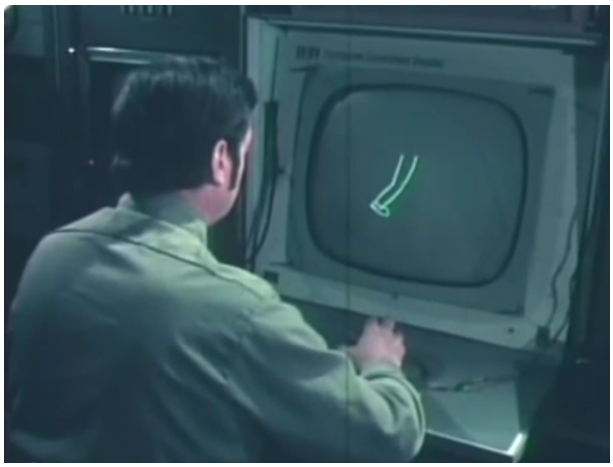
- Bisher besprochen: statische Elemente
- Bei Vektorgraphiken haben wir die Möglichkeit, nicht jedes Bild neu zu übertragen, sondern anzugeben, wie sich der Vektor über die Zeit verändert
- Rasterbilder zur Darstellung können aus dieser Beschreibung und den Vektordaten dann jederzeit generiert werden

- Das nennt sich Computeranimation
- Offensichtliche Animation: da alle geometrischen Formen durch ihre Eck- und Kontrollpunkte gegeben sind, genügt es, diese Kontrollpunkte zu animieren

Keyframe-Animation

- Einfachste Art, die Veränderung eines Parameters anzugeben, ist dessen Wert zu zwei Zeitpunkten anzugeben
- Z.B. Zeitpunkt 0 (Start der Animation, Laden der Datei, Beginn der Darstellung) und dann nach z.B. 10 sek
- Dazwischen wird (linear) interpoliert
- Dieses Vorgehen heißt Schlüsselbildanimation oder **Keyframe-Animation**
- Dabei können wir mehr als 2 Zeitpunkte angeben
- Je nach Software können auch andere als lineare Interpolationen zwischen einzelnen Keyframes definiert werden
  - Beschleunigen und Bremsen (ease-in, ease-out)
  - Interpolation mittels Spline

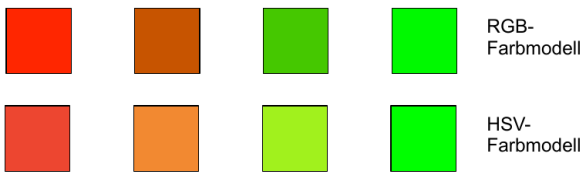
Video 9.1: Keyframe-Animation



☞ Key Frame Animation, Wein and Burtnyk, 1971 (7:21)

Animation von Farben

- Animation von Farbe nicht unmittelbar klar (Farbverläufe in welchem Farbraum?)



Quelle: Frank Steinicke

Animation in SVG

- Neben statischer Darstellung unterstützt SVG einige grundlegende Animationselemente
- Erlauben einerseits die Animation von Position und Orientierung zu animieren
- Andererseits können Transformationen zeitabhängig ausgeführt werden um Parameter wie Farbe oder Transparenz in der Zeit zu ändern

- Animationselemente werden innerhalb der graphischen Elemente spezifiziert
- Elemente können gruppiert werden, um sie gemeinsam zu animieren
- Objekte können mit Hilfe von IDs benannt und wiederverwendet werden

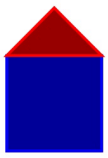
Animationselemente

- **animate**: animiert einen einzelnen Parameter eines Elements, wie x-Position oder Transparenz
- **set**: ein bestimmter Parameter wird für eine bestimmte Dauer auf einen festen Wert gesetzt
- **animateMotion**: bewegt ein Objekt entlang eines Bewegungspfades, wobei das Objekt wahlweise am Pfad ausgerichtet werden kann

  Der Pfad wird durch Reihe von Stützpunkten beschrieben und kann ein Linienzug oder Spline sein
- **animateColor**: animiert Farben von einem Start- zu einem Endwert. ggf. mit Zwischenwert
- **animateTransform**: Erlaubt es, graphische Objekte zu verschieben, zu skalieren, zu rotieren oder zu drehen

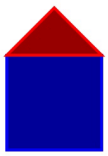
Beispiel SVG-Animation I

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <g id="house">
    <rect .../>
    <polygon .../>
    <animateMotion dur="4s"
      repeatCount="indefinite"
      path="M 0,0 C 0,0 200,0 0,0 Z" />
  </g>
</svg>
```



Beispiel SVG-Animation II

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <g id="house">
    <rect .../>
    <polygon .../>
      <animateTransform
        attributeName="transform"
        type="rotate"   dur="8s"
        from="0 150 75" to="360 150 75"
        repeatCount="indefinite"/>
    </polygon>
    <animateMotion ... />
  </g>
</svg>
```



Beispiele

Abbildung 7.16: Bahnhofsuhr in SVG

☞ Demonstrationen: Animation

5 Bildnachweis

Alle Abbildungen, wenn nicht anders angegeben, aus:
Malaka, Rainer; Butz, Andreas; Hussmann, Heinrich: *Medieninformatik – Eine Einführung*. ISBN 978-3-8273-7353-3, München: Pearson Studium, 2009.