



2 Medientypen

2.4 Grafik

Grafik







- Geometrische Objekte werden als Grafiktypen verwendet
- Darstellung als Bild durch Rendering
 - Rendering: Grafiktyp → Bildtyp
 - Übertragung der grafischen Darstellung in ein Punkteraster des Ausgabemediums
 - Bildschirm, Drucker



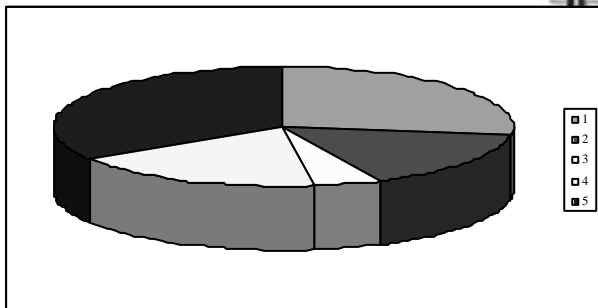
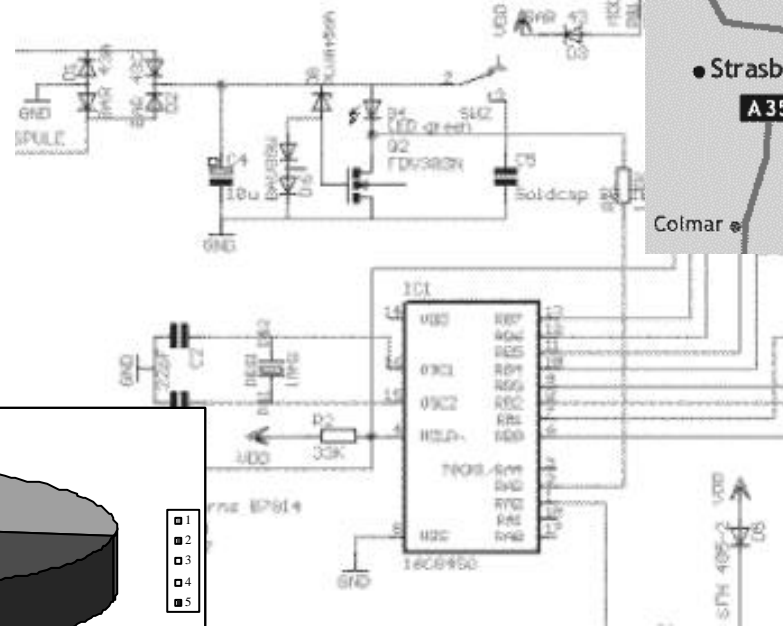
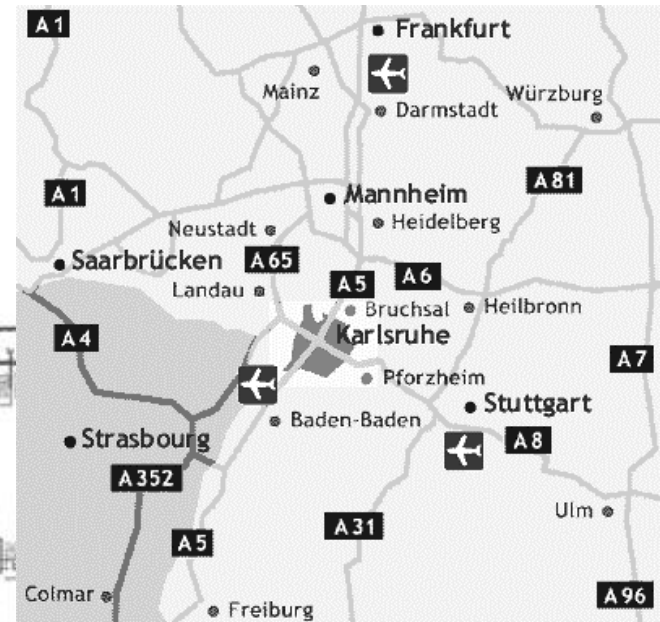
Grafik



- Typische geometrische Objekte eines Grafiksystems
 - Linien, Strecken 
 - Rechtecke 
 - Polygone und geglättete Polygone 
 - Kreise und Ellipsen 
- Begrenzte Anzahl Primitive vs. Semantikerhaltung beliebiger Objekte
 - Ziel: effiziente Implementierung



Beispiele



Motivation Vektorformat



- Basis-Grafikobjekte bestehen aus Vektoren, Polygonen etc.
- Probleme bei Bitmaps
 - Größe, Auflösung, Zoom
 - Editierbarkeit, keine Objekte
 - Keine Information über den Inhalt



Vektorformat



- Anforderungen (Auszug)
 - beliebige Zoomstufen
 - Desktop vs. PDA vs. Drucker
 - Objekte anpassen, z.B.
 - Strichstärke 3 pt.
 - Hintergrund blau
 - kleine Dateien
 - Pixel vs. Beschreibung



Eigenschaften Vektorformat



- Direkt darstellbar auf
 - Vector-refresh Displays
 - Direct-view Storage Tube
 - Plotter
- Modell üblich in interaktiven Anwendungen
 - Zeichenprogrammen
 - CAD-Programmen
 - Grafikpakete von Programmiersprachen (z.B. `java.awt.graphics`)
- Auch in Modellierungssprachen, z.B.
 - OpenGL (Grafiksprache)
 - Postscript (Druckersprache)



Bewertung Vektorformat



- Vorteile
 - Nah am mathematischen, geometrischen Modell
 - Nah am programmiertechnischen Modell
 - Damit leicht transformierbar
- Nachteile
 - Die meisten Bildschirme sind pixelorientiert
 - Schlecht für natürliche Bilder (z.B. Fotos)



Rasterkonvertierung



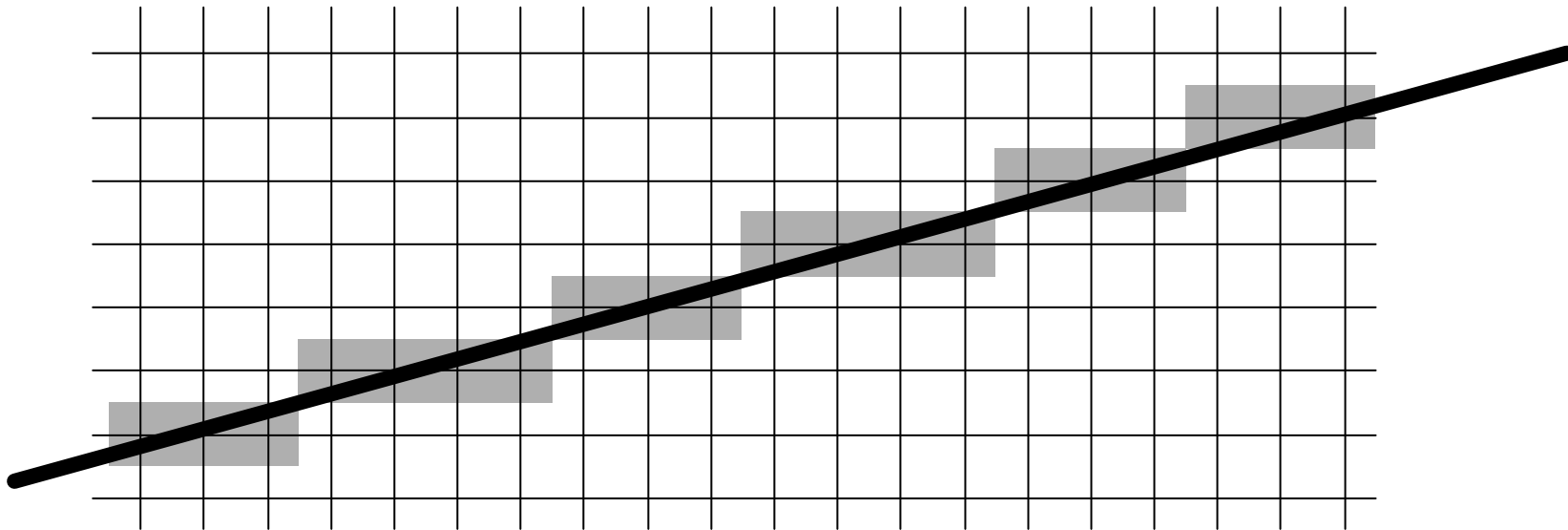
- Rasterkonvertierung:
 - Übertragung der bildlichen Darstellung in Punktraster des Gerätes
- Aliasierung:
 - Objekte werden nicht an ihrem mathematisch tatsächlichen Ort, sondern an genäherten Aliaspositionen gezeichnet
- Punktkonvertierung:
 - $P(x,y) \rightarrow \text{Pixel}(\text{round}(x), \text{round}(y))$
 - Darstellung als kleine Pixelfläche



Aliasing



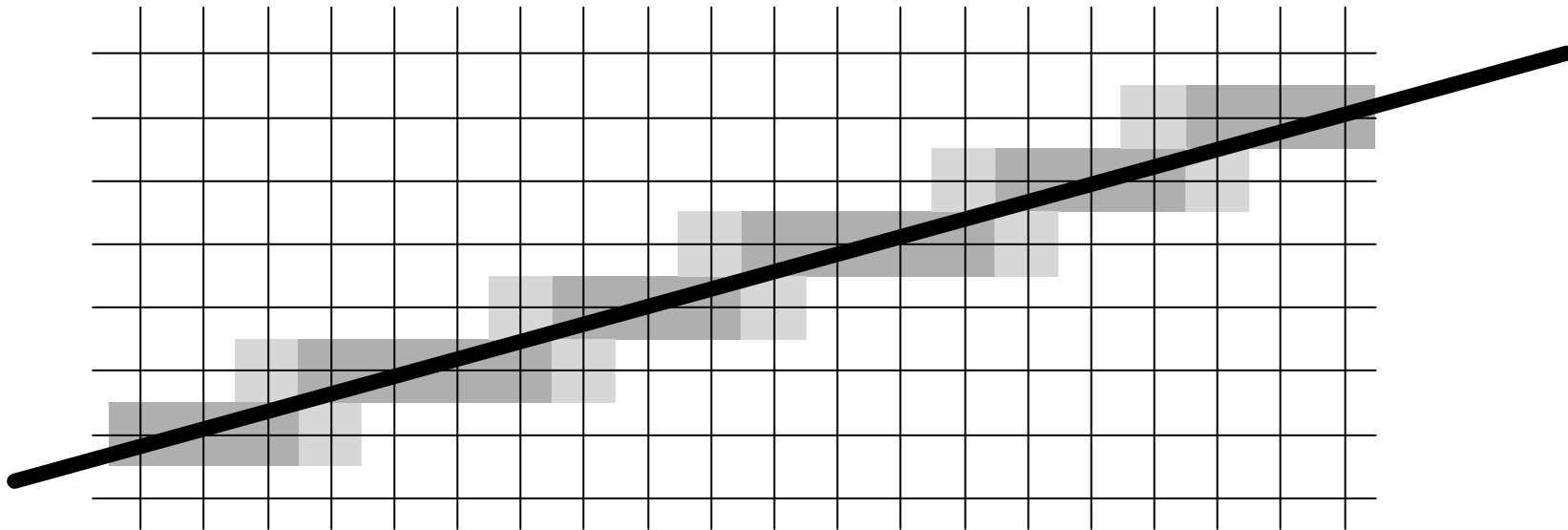
- Darstellung geometrischer Objekte durch Annäherung
→ Aliasing-Effekt



Antialiasing



- Antialiasing durch Einfügen farblich abgeschwächter Pixel
→ Glättungseffekt



Rasterkonvertierung



- Berechnung der Pixel aufgrund geometrischer Vorgaben
- Bewertung:
 - Sehr aufwendig
 - Reeller Zahlenraum wäre zu verwenden
 - Viele Pixel fallen aufeinander
- Beispiel Linie
 - Für jeden Punkt der Linie ist eine Punktkonvertierung durchzuführen
 - Annahmen:
 - Raster und Linie haben Ganzzahl-Koordinaten
 - Steigung $|m| \leq 1$



Punktkonvertierung einer Linie



- Einfacher Algorithmus
 - Berechne Steigung $m = \Delta y / \Delta x$
 - Beginne am linken Endpunkt
 - Inkrementiere x jeweils um 1 und berechne $y_i = mx_i + B$ für jedes x_i
 - Setze das Pixel an $(x_i, \text{round}(y_i))$
- Ineffizient, da jede Iteration 1 Multiplikation, 1 Addition und 1 Rundung mit Gleitkommazahlen durchführt



Punktkonvertierung einer Linie



- Eliminierung der Multiplikation
 - $y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$
- Falls $\Delta x = 1$
 - $y_{i+1} = y_i + m$ und
 - $x_{i+1} = x_i + 1$
- Inkrementeller Algorithmus:
Digital Differential Analyzer (DDA)
 - Nachteil: Fehlerkumulation durch sukzessive Rundung



Punktkonvertierung einer Linie



- DDA in Java

- ```
public void Line(int x0, int y0,
 int x1, int y1,
 Color color)
{
 float dy = y1 - y0;
 float dx = x1 - x0;
 float m = dy/dx;
 float y = y0;
 int x;
 for (x=x0; x<=x1; x++){
 writePixel(x, Math.round(y), color);
 y += m;
 }
}
```



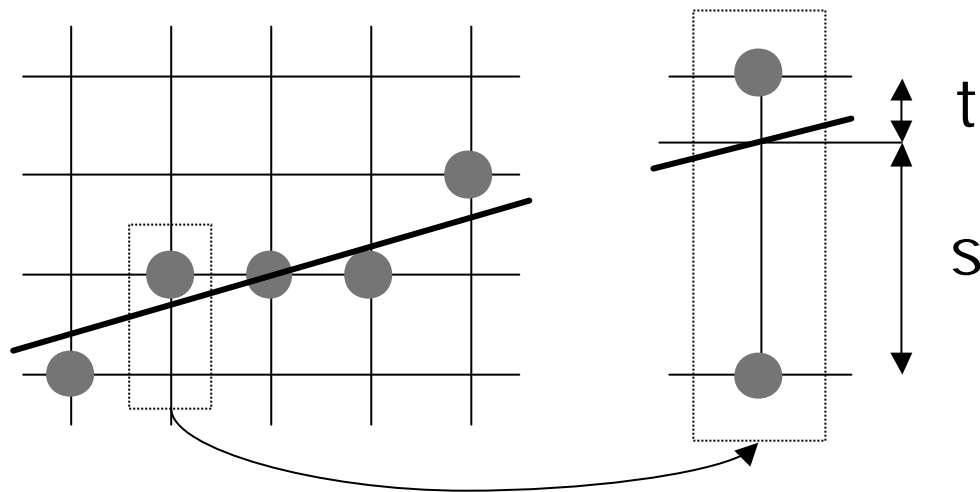
# Algorithmus von Bresenham



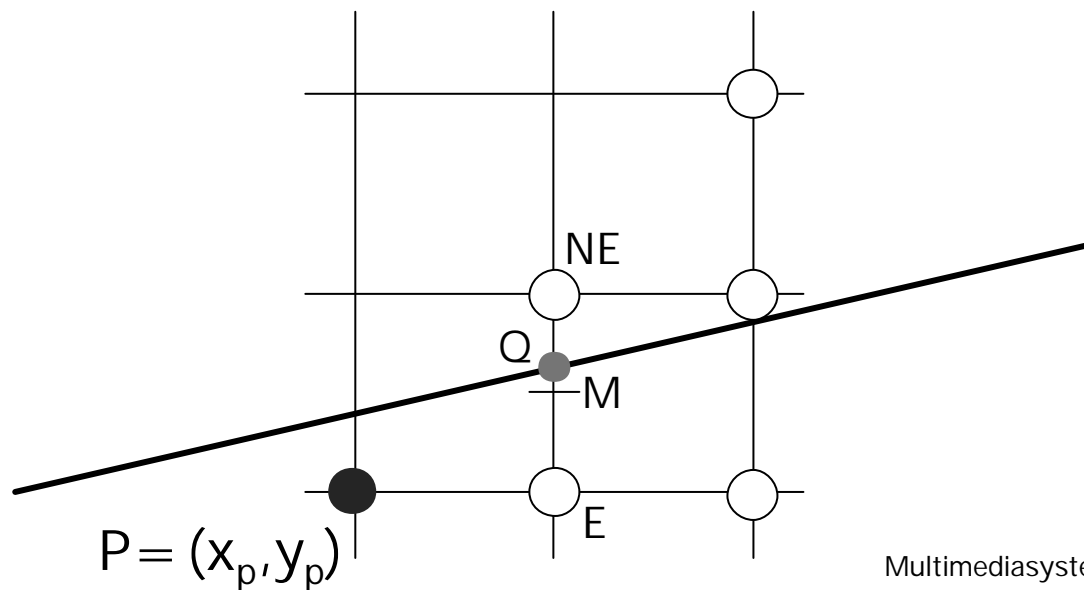
- Bresenham (1965)
  - Eliminierung des Rundens,  
nur einfache Operationen: Addition,  
Subtraktion, Multiplikation mit 2  
→ Zeitersparnis
  - Inkrementeller Integralgorithmus  
→ Zeit und Genauigkeit
  - Midpoint Line Algorithm



# Algorithmus von Bresenham



(s-t) dient als  
Entscheidungsvariable



# Algorithmus von Bresenham



- Anfangspunkt:  $(x_0, y_0)$   
Endpunkt:  $(x_1, y_1)$
- Linie:  $F(x, y) = ax + by + c = 0$
- $dy = y_1 - y_0$ ;  $dx = x_1 - x_0$
- $\rightarrow y = dy/dx \cdot x + B$
- $\rightarrow F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$
- Für Punkte auf der Linie:  $F(x, y) = 0$
- Über der Linie:  $F(x, y) < 0$
- Unter der Linie:  $F(x, y) > 0$



# Algorithmus von Bresenham



- Mittelpunktskriterium:  
Vorzeichentest von  $F(M) = F(x_p + 1, y_p + 1/2)$
- Entscheidungsvariable:  
 $d = a(x_p + 1) + b(y_p + 1/2) + c$
- Falls  $d > 0$  wähle Pixel NE
- Falls  $d < 0$  wähle Pixel E
- Falls  $d = 0$  wähle Pixel E
  - könnte auch NE sein



# Algorithmus von Bresenham



- Nächste Iteration
- E war gewählt:
  - $d_{\text{neu}} =$   
 $F(x_p + 2, y_p + 1/2) = a(x_p + 2) + b(y_p + 1/2) + c$
  - $d_{\text{neu}} = d_{\text{alt}} + a$
- NE war gewählt:
  - $d_{\text{neu}} =$   
 $F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c$
  - $d_{\text{neu}} = d_{\text{alt}} + a + b$



# Algorithmus von Bresenham



- Erster Wert für d
  - $F(x_0 + 1, y_0 + 1/2) = a(x_0 + 1) + b(y_0 + 1/2) + c$   
 $= ax_0 + by_0 + c + a + b/2$   
 $= F(x_0, y_0) + a + b/2$
  - $F(x_0, y_0) = 0$
  - $\rightarrow d_{\text{start}} = a + b/2 = dy - dx/2$
- Definiere F als 2F neu
  - $F(x, y) = 2(ax + by + c)$
  - Damit fällt der Bruch weg, die Vorzeichen bleiben aber



# Algorithmus von Bresenham



## ■ Bresenham in Java

```
 public void MidPointLine(int x0, int y0, int x1, int y1,
 Color color)
 {
 int dx = x1 - x0; int dy = y1 - y0;
 int d = 2*dy -dx;
 int incrE = 2*dy; int incrNE = 2*(dy-dx);
 int x = x0; int y = y0;
 writePixel(x,y,color);

 while(x<x1){
 if(d<=0){
 d+=incrE;
 x++;
 }
 else{
 d+=incrNE;
 x++;
 y++;
 }
 writePixel(x,y,color);
 }
 }
```



# Algorithmus von Bresenham



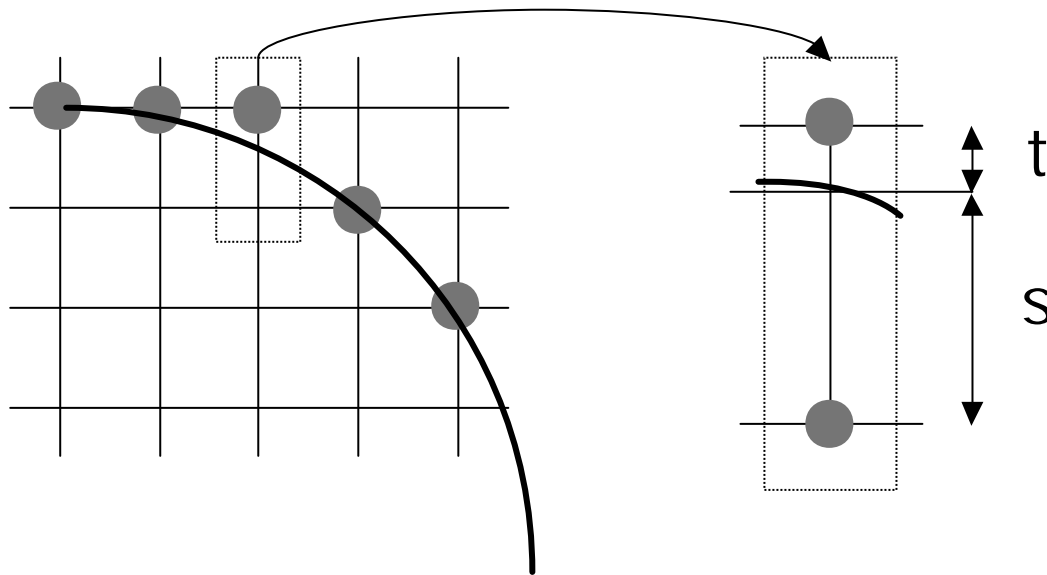
- Für andere Steigungen
  - Einfache Umschreibung des Algorithmus bzw. Spiegelung/Drehung
- Für andere geometrische Objekte gibt es analoge Algorithmen
  - Kreis
  - Ellipse



# Rastern von Kreisen nach Bresenham



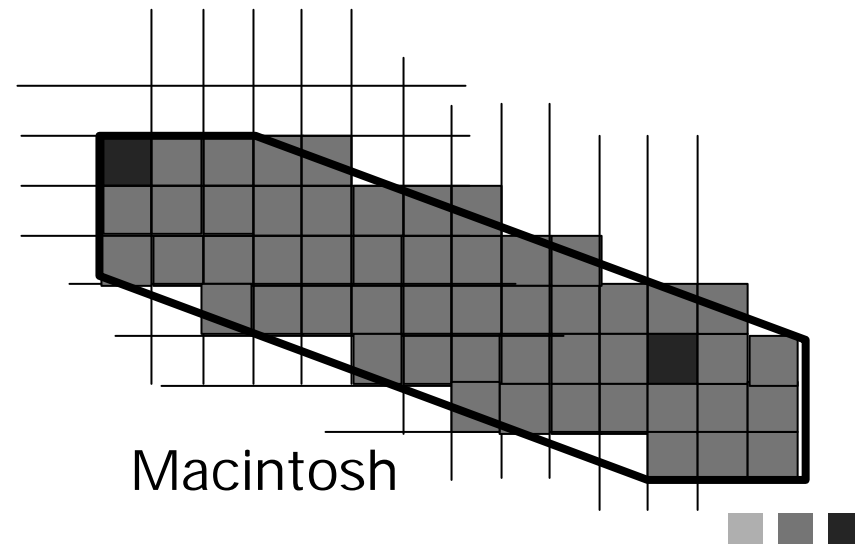
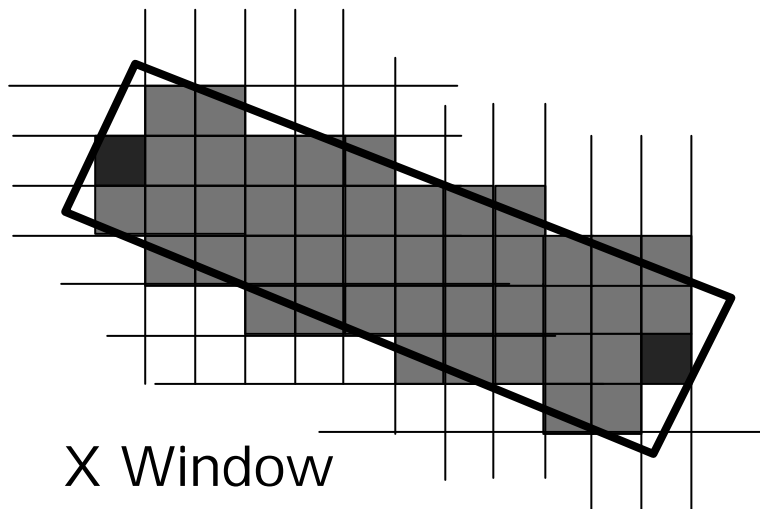
- Wegen der Symmetrieeigenschaften genügt es die Punkte des Achtelkreises zwischen  $90^\circ$  und  $45^\circ$  zu bestimmen
- Rest durch Spiegelung



# Grafikoperationen



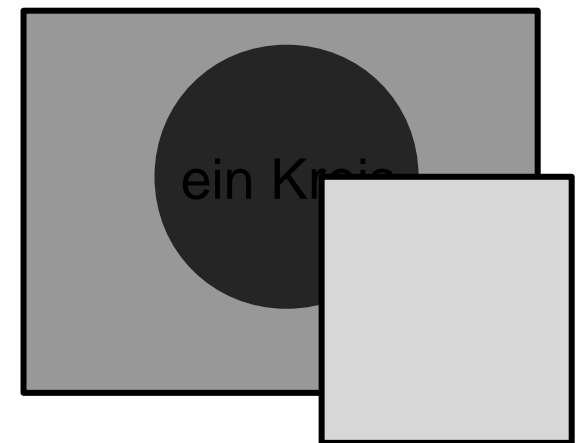
- Verschiedene Systeme haben unterschiedliches Verhalten, z.B.
  - gehört der Rand zum Kreis?
  - wo beginnt die Linie, wo endet sie?
  - Liniendicke absolut oder veränderlich mit der Zeichenrichtung?
  - sind Polygone automatisch geschlossen?



# Regionen



- zur Beschreibung von Bereichen auf der Zeichenebene
  - durch Linienzüge begrenzt
  - mehrere, u.U. nicht zusammenhängende Teile
- Zeichenregion
  - Darstellungsbereich für Grafik
- Clipping
  - Zeichenoperationen dürfen nur im nicht überdeckten Bereich einer Zeichenregion ausgeführt werden



# Datenmodelle für geometrische Objekte



- Modell beschreibt grafische Informationen so genau wie nötig
- Anforderungen:
  - Einfache Beschreibungsmittel für Nutzer
  - Wenig Speicherplatz trotz hoher Genauigkeit
  - Effiziente Algorithmen zur Transformation
  - Geschlossene Operationsräume
    - d.h.  
Operation(gültiges Objekt) → gültiges Objekt



# Beispiele zu Datenmodellen



- Strecken
  - Anfangs- und Endpunkt
- Kreis
  - Mittelpunkt und Radius oder umschließendes Quadrat
- Kurven
  - Folge von Strecken
    - zu aufwendig
    - Schlecht interaktiv zu verändern
  - Interpolation durch Polynome
    - $n$  Punkte erfordern Polynom vom Grad  $n-1$
    - zu aufwendig



# Datenmodelle für Kurven



- Kubische Splines
  - Je 2 Punkte durch Polynom 3. Ordnung verbinden
  - Polynome untereinander stetig verbinden
  - $n$  Punkte  $\rightarrow$   $n-1$  Funktionen
- Beziér-Kurven
  - Siehe Kapitel Text
- B-Splines
  - Kurvenstücke laufen nicht durch die Kontrollpunkte
  - Kontrollpunkte wirken wie Gravitationspunkte



# Datenmodelle für Flächen



- Polygon
  - Liste von n Punkten
- Polygonnetz
  - Polygon wird aus Dreiecken zusammengesetzt
  - Halbkantendarstellung verhindert unbeabsichtigte Öffnungen im Netz



# 2-D Modelle



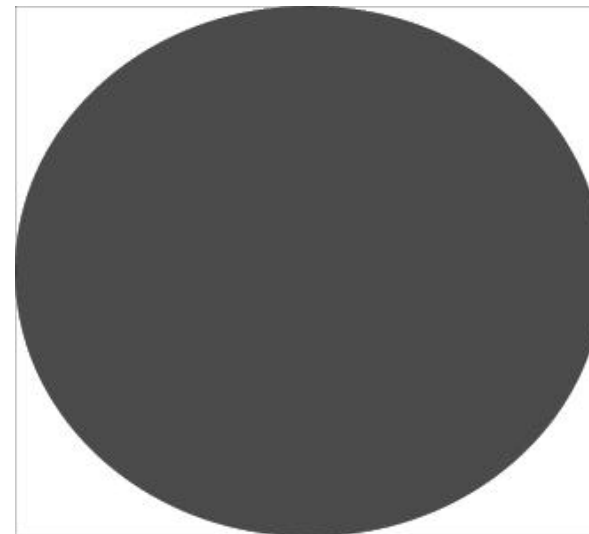
- CAD Formate
  - z.B. Catia
- Allgemeine Vektorgrafik-Formate
  - z.B. SVG (scalable vector graphics)



# SVG Beispiel (generiert aus CorelDraw)



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
<!-- Creator: CorelDRAW -->
<svg xml:space="preserve" x="-2.56756in" y="-0.82976in" width="3.45454in"
height="3.11119in" style="shape-rendering:geometricPrecision; text-rendering:geometricPrecision;
image-rendering:optimizeQuality"
viewBox="-2568 0 3455 3111">
<defs>
<style type="text/css">
<![CDATA[
.str0 {stroke:#1F1A17;stroke-width:3}
.fil0 {fill:none}
.fil1 {fill:#DA251D}
]]>
</style>
</defs>
<g id="Ebene 1">
<rect id="45591048" class="fil0 str0" x="-2566" y="2" width="3452" height="3108"/>
<ellipse id="45590520" class="fil1 str0" cx="-840" cy="1556" rx="1726" ry="1554"/>
</g>
</svg>
```



# Datenmodelle für 3-D Flächen



- Parameterdarstellung
  - $P(s,t) = (x(s,t), y(s,t), z(s,t))$
- Zusammensetzen aus Teilflächen
- Splines und Beziér-Kurven lassen sich auch näherungsweise auf gekrümmte Flächen anwenden



# Datenmodelle für 3-D Flächen



- Bei Darstellung als Pixelfläche
  - z.B. Quadtree-Verfahren
  - Rekursive Aufteilung der Fläche in Quadrate
- Fraktale Modelle
  - Mandelbrot-Menge
  - Gut für realistisch aussehende Pflanzen, Gelände



# Datenmodelle für Körper



- Kantenmodell
  - Punkte und verbindende Kanten
  - Drahtkörperdarstellung
- Flächenmodelle
  - Zwischen den Kanten werden Flächen aufgespannt
  - Was ist verdeckt, was liegt innen bzw. außen  
→ Ebenengleichung
- Boundary Representation
  - Angabe Begrenzungsfläche plus Innen/Außen



# Datenmodelle für Körper



- Translationskörper
  - Bewege Fläche oder Körper entlang einer Kurve im Raum
  - Daraus entsteht neuer Körper
  - Begriff: Extrusion
- Voxel
  - Räumliche Enumeration
  - Speicherintensiv
  - Octree-Verfahren



# Datenmodelle für Körper



- Zellenzerlegung
  - Körper wird in einfache Teilkörper zerlegt
  - Echter Körper durch Vereinigungsmenge beschrieben
- Constructive Solid Geometry (CSG)
  - Verknüpfung von Grundkörpern
  - Nicht nur Vereinigung, sondern alle booleschen Mengenoperationen



# Geometrische Transformationen



- Operationen durch Multiplikation mit Transformationsmatrizen
  - Verschieben, Translation
  - Skalieren
  - Rotation
- Dazu Repräsentation in einem homogenen Koordinatensystem
  - 3D bei 2D Flächen
  - 4D bei 3D Körpern



# Projektionen



- Darstellung von 3-D Objekten im 2-D Fläche
- vom Projektionszentrum ausgehend verlaufen Projektionsstrahlen durch jeden Objektpunkt
- Schnittpunkte der Strahlen mit der Projektionsebene ergeben 2-D Bild



# Ermittlung sichtbarer Flächen



- Algorithmen,  
die sichtbare Pixel berechnen
  - Z-Puffer-Algorithmus
  - Rasterzeilenalgorithmus
  - Strahlverfolgung, Raytracing
- Algorithmen,  
die unsichtbare Pixel berechnen
  - Entfernen der Rückseiten
  - Teilungsalgorithmus
  - Maleralgorithmus



# Fotorealistische Darstellung



- Lichtquelle wird in die Szene eingefügt
- Parameter
  - Art der Quelle
  - Ausbreitung des Lichts
  - Farbe des Lichts
  - Farben der Objekte
  - Intensität, mit der das Licht auftrifft
  - Brechung
  - Reflexion



# Verfahren



- Schattierung
  - Gourand-Shading
  - Phong-Shading
- Texturen
  - Oberflächenmuster
    - Picture Mapping vs. Procedural Mapping
  - Oberflächenrauhheit
    - Bump Mapping, leichte Verschiebungswerte
- Demobeispiele aus  
R.J. Wolfe: 3D Graphics – A Visual Approach,  
Oxford University Press, 2000.



# Verfahren



- Rekursives Raytracing
  - Erweiterung zur Behandlung von Schatten, Reflexion und Brechung
- Strahlungsverfahren, Radiosity
  - Wechselseitiger Strahlungsaustausch zwischen Teilflächen



# 3-D Modelle



- Beispiele:
  - Graphic Kernel System, GKS, 1985
  - Programmer's Hierarchical Interactive Graphic System, PHIGS, 1989
  - OpenGL, Silicon Graphics, 1993
  - Virtual Reality Markup Language, VRML
  - Java 3D



# Vertiefung Computergrafik



- Vorlesung Computergrafik  
Proffs. Neumann und Toran
- J. Foley, A. van Dam, S. Feiner, J. Hughes:  
Computer Graphics: Principles and Practice,  
Addison-Wesley, 2nd Ed., 1996.
  - Gibt es auch in deutsch
- J. Encarnacao, W. Straßer, R. Klein:  
Grafische Datenverarbeitung,  
Oldenbourg, 1996.

