



Mediale Informatik

Prof. Dr. Michael Weber



Kapitel A

User Interface Software



Kapitel A1

Grundlagen grafischer Systeme

A1.1 Bildmodelle



- Visuelle Ausgabe üblicherweise auf 2D-Bildschirmen
- Bildmodell dient der
 - Darstellung
 - Modellierung
 - Interaktion
- 3 Basismodelle
 - Linienmodell, Vektormodell
 - Pixelmodell
 - Regionenmodell



Linienmodell bzw. Vektormodell



- Früheste Form der Repräsentation geometrischer Objekte im Computer
- Alle Bilder bestehen aus Strichen mit Attributen
 - Linientyp
 - Koordinaten
 - Farbe
 - Dicke

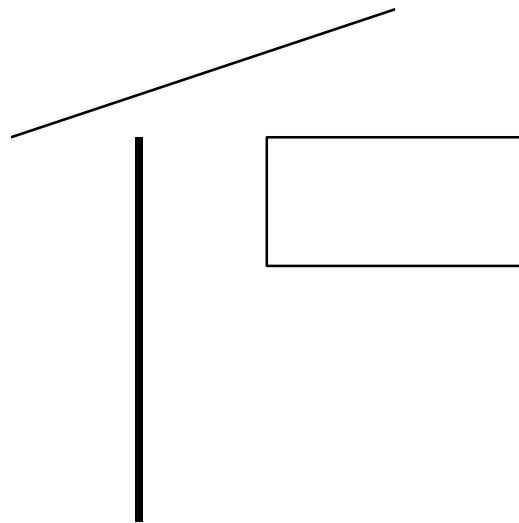


Linienmodell bzw. Vektormodell



■ Beispiele

- Line ((1,3) (4,4) thick 1)
- Line ((2,0) (2,3) thick 3)
- Rectangle ((3,2) (5,3) thick 1)



Linienmodell bzw. Vektormodell



- 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)



Linienmodell bzw. Vektormodell



- 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)



Pixelmodell



- Bilder bestehen aus Pixeln mit Attribut
 - Pixelwert
- Pixelwert ist je nach Modell
 - Ein/Aus → Schwarz/weiß
 - Grauwert
 - Farbwert
 - Farb-/Grauwert und Wert des Alphakanals (Transparenz)



Pixelmodell



- Schlüsselmaße

- Räumliche Auflösung in X und Y

- Beispiele

- PC-Bildschirme: 1024x768, 1280x1024, 1600x1200
 - Andere Bildschirme: PDA ~300x200, GSM-Telefon ~64x64
 - Laserdrucker: 3000x2400, 6000x4800

- Bildtiefe

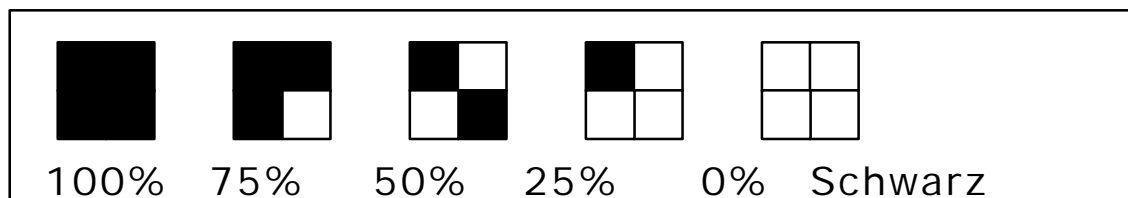
- Bit pro Pixel zur Grauwert-/Farbkodierung



Pixelmodell



- 4 grundlegende Typen von Pixelbildern
- 1.: Bitmap
 - 1 Bit pro Pixel → Schwarz/Weiß
 - Speichervolumen = räumliche Auflösung [bit]
 - Unter Umständen Graudarstellungen durch SW-Muster
 - Halbtonverfahren – Dithering
 - Gute Graudarstellung bei hoher Auflösung
 - Einsatz z.B. bei Druckern



Pixelmodell



- 2.: mehr Bit/Pixel
 - 2 Bit → 4 Graustufen (NeXT 1989)
 - Nicht viel besser für Bilder, aber deutlich besser für Interfaceelemente
 - 8 Bit → 256 Graustufen oder 256 Farben
 - Bei 1024x1024 Bildgröße → 1 Megabyte Speicher



Pixelmodell



- 3.: Truecolor
 - 24 Bit → je 1 Byte pro Farbkanal (rot, grün, blau)
 - 32 Bit → zusätzlich 1 Byte für Alphakanal (Transparenz)
 - Realistische Wiedergabe aller vom Menschen wahrnehmbaren Farben
 - Foto mit 3000x2400 Auflösung braucht 22 MB Speicher
 - Kompressionsverfahren werden oft notwendig, um Speicherplatz zu sparen



Pixelmodell



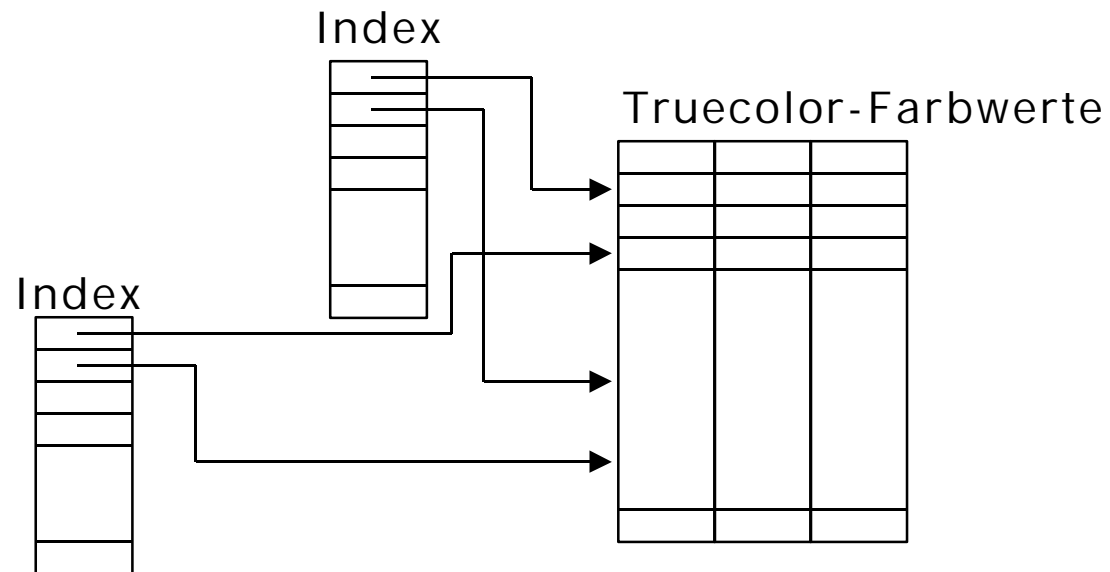
- 4.: Farbtabelle
 - Typisch 1 Byte/Pixel
 - Gespeichert wird Index in Farbtabelle und nicht der Farbwert selbst
 - Oft Farbtabelle pro Anwendung
 - D.h. anwendungsspezifische Tabelle
 - „Flackern“ beim Wechsel der Anwendungsfenster



Pixelmodell



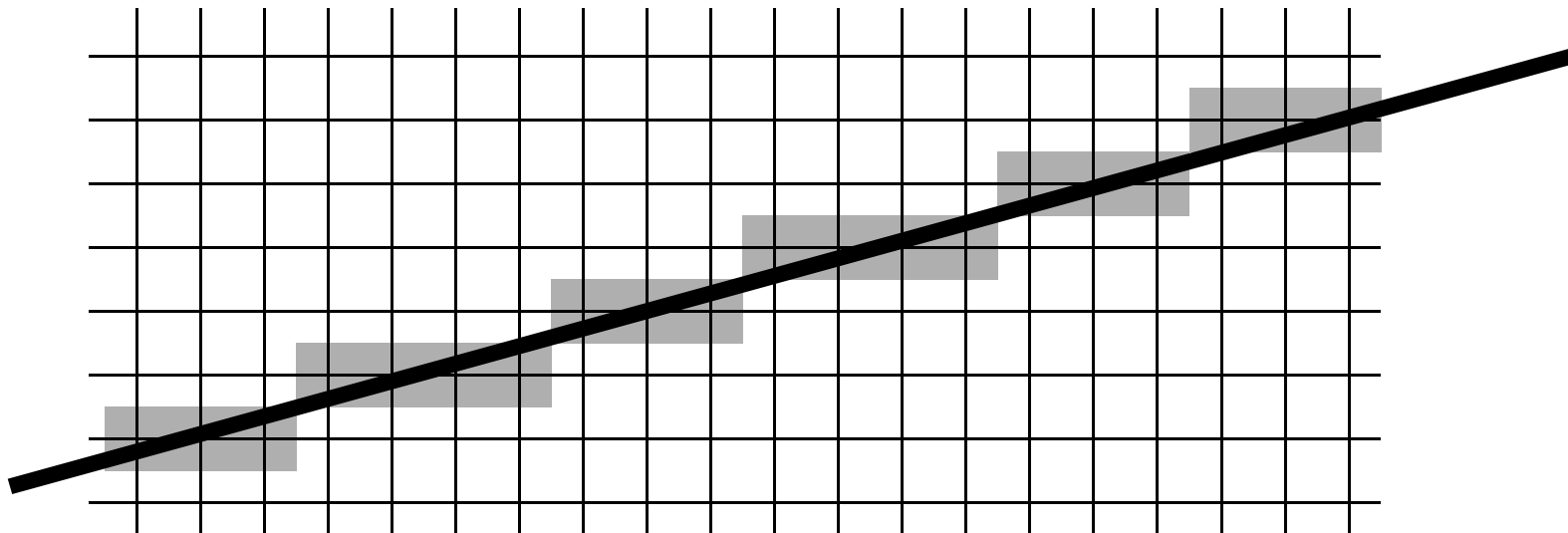
- Farbtabelle



Pixelmodell



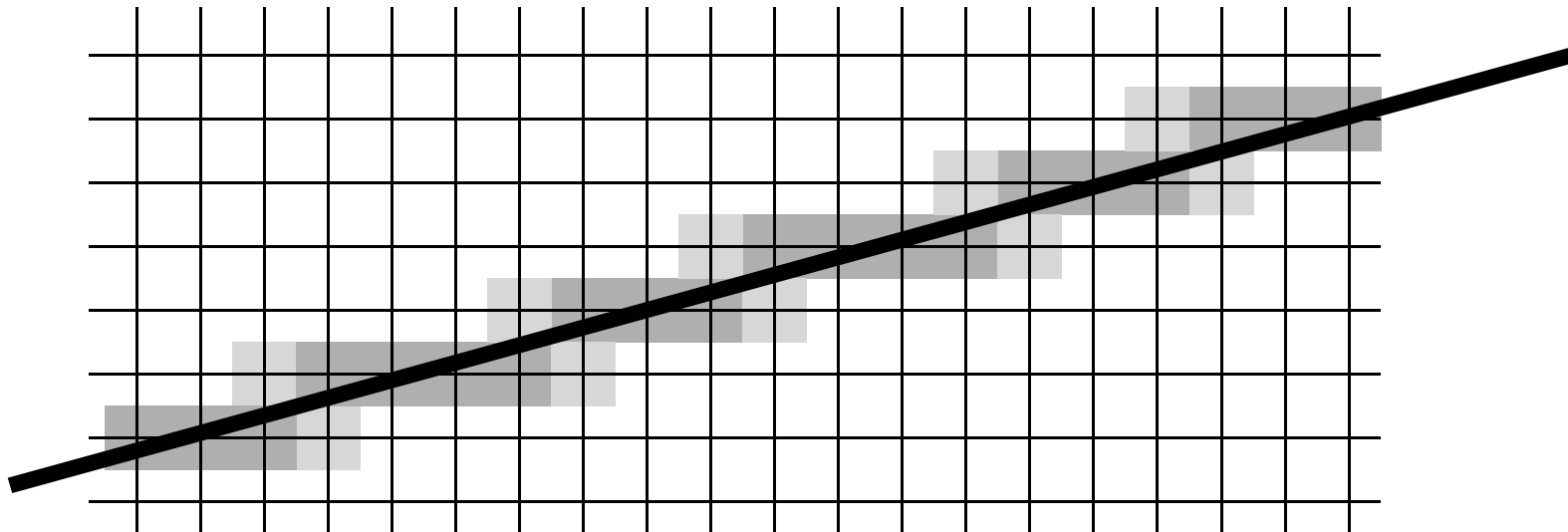
- Darstellung geometrischer Objekte durch Annäherung
→ Aliasing-Effekt



Pixelmodell



- Antialiasing durch Einfügen farblich abgeschwächter Pixel
→ Glättungseffekt



Pixelmodell



- Berechnung der Pixel aufgrund geometrischer Vorgaben
- Beispiel Linie
 - Annahmen:
 - Raster und Linie haben Ganzzahl-Koordinaten
 - Steigung $|m| \leq 1$



Pixelmodell



- 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



Pixelmodell



- 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



Pixelmodell



- DDA in Java

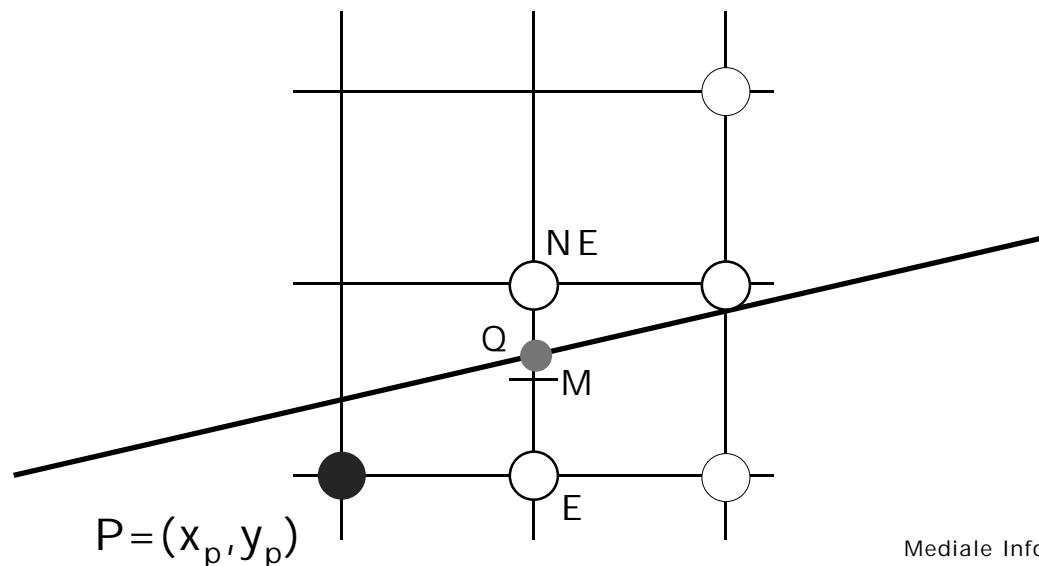
- ```
public void Line(int x0, int y0,
 int x1, int y1,
 Color color)
{
 float dy = y1 - y0;
 float dx = x1 - x0;
 float m = dy/dx;
 float y = y0;
 int x;
 for (x=x0; x<=x1; x++){
 writePixel(x, Math.round(y), color);
 y += m;
 }
}
```



# Pixelmodell



- Bresenham (1965)
  - Eliminierung des Rundens  $\rightarrow$  Zeitersparnis
  - Inkrementeller Integralalgorithmus  $\rightarrow$  Zeit und Genauigkeit
  - Midpoint Line Algorithm



# Pixelmodell



- 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:  $F(x, y) > 0$



# Pixelmodell



- 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



# Pixelmodell



- 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:
  - $d_{\text{neu}} = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c$
  - $d_{\text{neu}} = d_{\text{alt}} + a + b$



# Pixelmodell



- 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



# Pixelmodell



## ■ 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);
 }
 }
```



# Pixelmodell



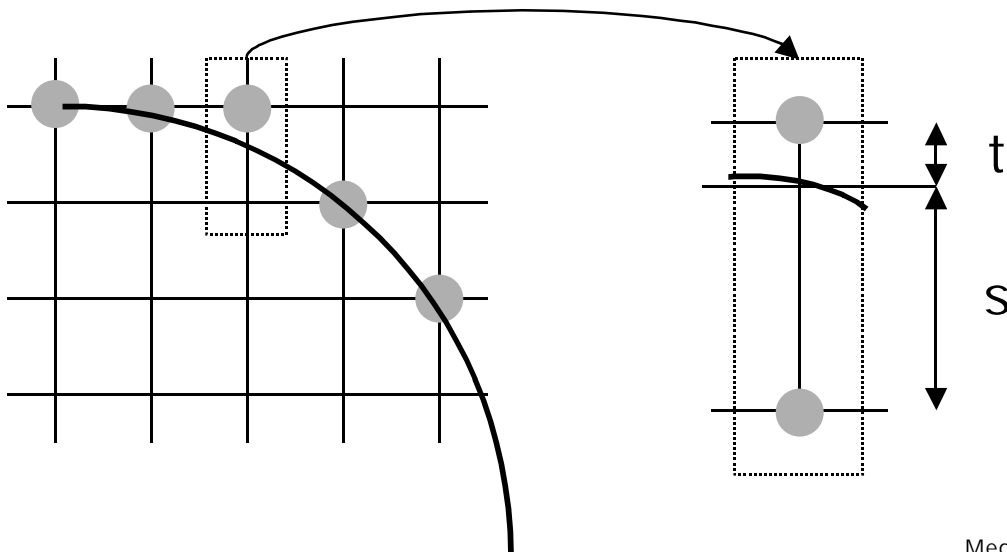
- Für andere Steigungen
  - Einfache Umschreibung des Algorithmus bzw. Spiegelung/Drehung
- Für andere geometrische Objekte gibt es analoge Algorithmen
  - Kreis
  - Ellipse



# Pixelmodell



- 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



# Regionenmodell



- Rand eines Objekts wird als Vektorgrafik gezeichnet
- Inneres wird gefüllt mit
  - Farbe
  - Füllmuster
  - Verlauf
  - ...
- Vorteil:
  - Für Füllungen muss die Auflösung des Displays nicht bekannt sein
  - Bei hochauflösenden Displays spart dies viel Speicher und evtl. Kommunikationsvolumen
  - z.B. Drucker



# A1.2 Koordinatensysteme



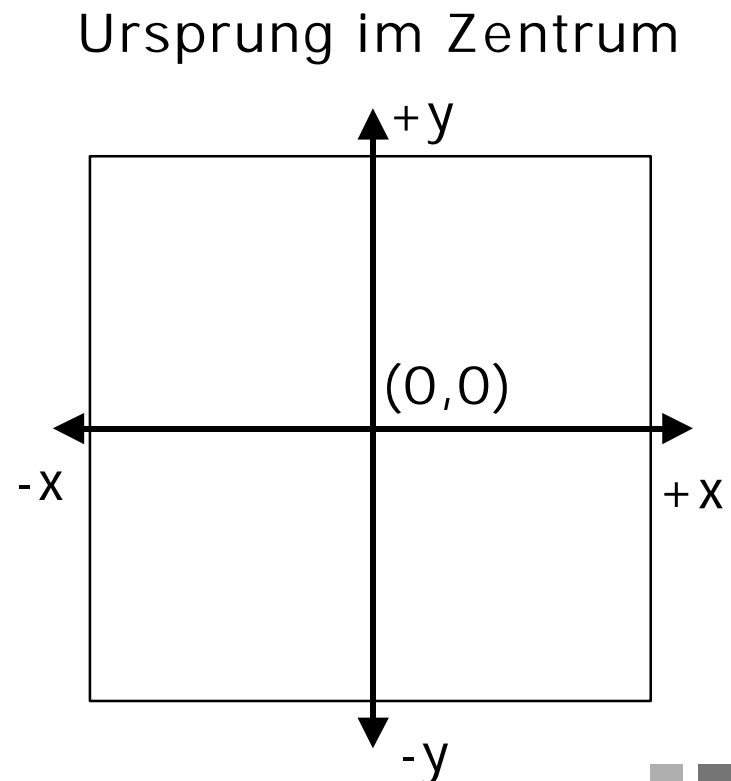
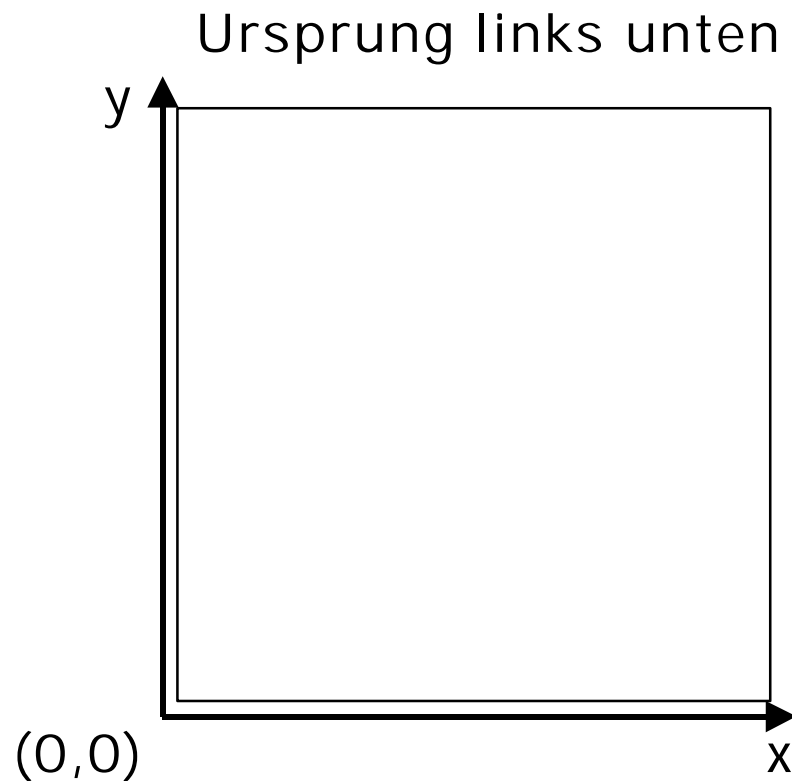
- Koordinatensysteme bestimmen die Lage von Objekten
- Verschiedene Koordinatensysteme müssen beachtet werden
  - Kartesische Koordinaten
  - Gerätekoordinaten
  - Physikalische Koordinaten
  - Modellkoordinaten
  - Interaktive Koordinaten



# Kartesisches Koordinatensysteme



- spannt 2-dimensionalen Vektorraum auf



# Gerätekoordinaten



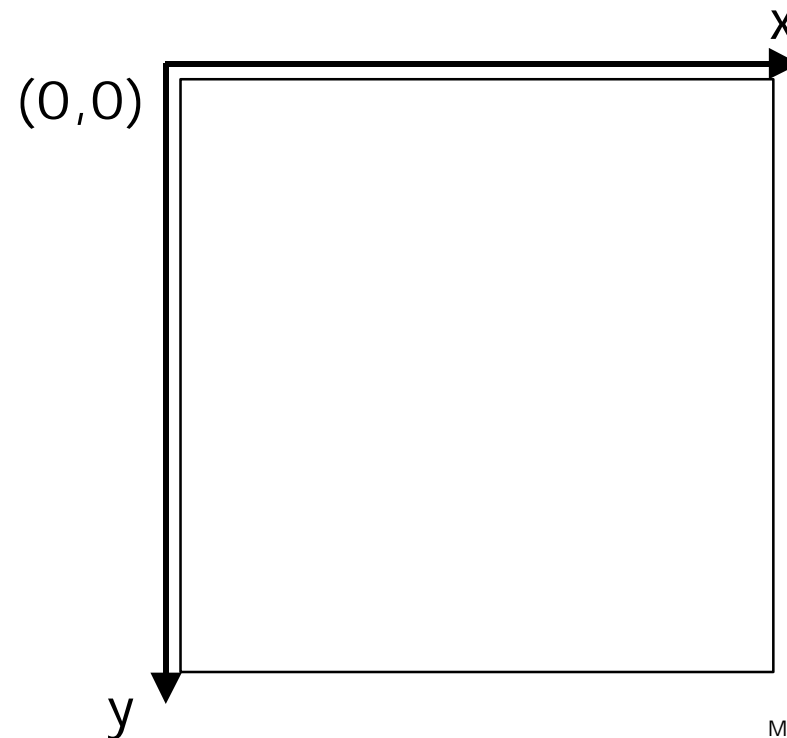
- Koordinaten des tatsächlichen Ausgabegeräts
  - gespiegeltes kartesisches Koordinatensystem
  - Ursprung links oben
    - Man schreibt von links oben nach rechts unten (in der westlichen Welt)
    - der Elektronenstrahl des Bildschirms beschreibt von links oben nach rechts unten



# Gerätekoordinaten



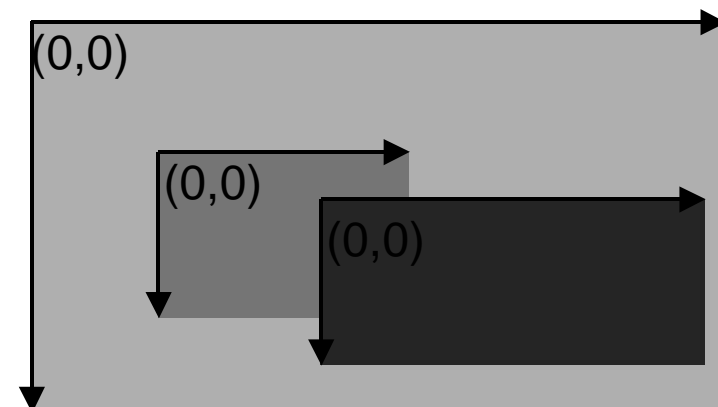
- Es gibt (fast immer) nur positive Koordinaten
- Meist gemessen in Pixel



# Gerätekoordinaten



- Fenster als logisches, virtuelles Ausgabegerät
  - In fast allen modernen Interface Toolkits
- globale und lokale Koordinatensysteme
  - Bildschirm besitzt ein globales Koordinatensystem
  - Fenster haben eigene lokale Koordinatensysteme
  - Maus operiert evtl. auf globalem Koordinatensystem
  - Konversionen notwendig



# Physikalische Koordinaten



- Pixel-basierte Koordinaten sind nicht auflösungsunabhängig
  - Ausgabe auf Bildschirm bzw. Drucker ist unterschiedlich groß
- Koordinaten in physikalischen Einheiten
  - Inch, Zentimeter, Druckerpunkte (dots per inch)
- z.B.
  - $5 \text{ cm} \cdot 600 \text{ dpi} = 3000 \text{ Pixel}$  (Drucker)
  - $5 \text{ cm} \cdot 60 \text{ dpi} = 360 \text{ Pixel}$  (17" Bildschirm)



# Physikalische Koordinaten



- Fontgrößen werden in Punkt angegeben
- Ursprung im Bleisatz
- Größe festgelegt durch Kegelhöhe des Bleiletters
  - d.h. etwas größer als das Druckbild
  - beinhaltet das Fleisch um das Druckbild
  - Einheit Didot-Punkt (0,376 mm) [Didot, 1784]
- Computerschriften verwenden Pica-Punkte (0,351mm)
  - d.h. Pica Punkte entsprechen 72 dpi
  - d.h. 12 pt Pica entspricht ca. 11 pt Didot
  - Höhe von 12 pt Text auf 400 dpi Drucker = 67 Pixel



# Physikalische Koordinaten



- Vorteile
  - Exakter Bezug zur realen Welt
  - Unterschiedliche Geräte werden normiert
- Nachteil
  - Jede Koordinate muss per Multiplikation berechnet werden



# Modellkoordinaten



- Anwendungsspezifisches Koordinatensystem
  - z.B. reale Millimeter für Wordprocessor zur exakten Positionierung auf A4-Papier
  - 1:100 Maßstab für Statikprogramm
- Umrechnung von Modellkoordinaten in Gerätekoordinaten brauchen im Allgemeinen Dreisatzrechnung



# Interaktive Koordinaten



- Umgekehrter Vorgang wie bisher
  - Eingabeposition (z.B. der Maus) muss auf Koordinate abgebildet werden
- Allgemeine Formel einen Modellpunkt in einem Fenster darzustellen
  - $\text{Modellkoordinate} \cdot \text{Zeichenmaßstab} \cdot \text{physikalische-zu-Pixel-Konversion} \cdot \text{Fensterursprung} = \text{Ausgabekoordinate}$
- Allgemeine Formel eine Mauskoordinate (relativ zum Bildschirm) einem Modellpunkt zuzuordnen
  - $(\text{Eingabekoordinate} - \text{Fensterursprung}) / (\text{physikalische-zu-Pixel-Konversion} \cdot \text{Zeichenmaßstab}) = \text{Modellkoordinate}$
- In Realität
  - Anwendung homogener Koordinaten und Matrix Algebra



## A1.3 Geschwindigkeit menschlichen Sehvermögens



- Ab ca. 20 Bilder pro Sekunde werden diese als kontinuierlicher Strom empfunden → Film
  - PAL Fernsehen → 25 Bilder/Sek.
  - NTSC Fernsehen → 29,997 Bilder/Sek.
- Für die Darstellung am Rechner bedeutet das
  - 300 MIPS Rechner bei 1 Mio. Pixels und 30 Hz Bildrate → maximal 10 Instruktionen pro Pixel, falls alle manipuliert werden
- Interaktives Feeling
  - 5 Hz bei Objekt-Dragging ausreichend
  - 0,5 – 1 Hz bei Kontextwechseln, z.B. neues Fenster öffnet



## A1.4 Grafikhardware



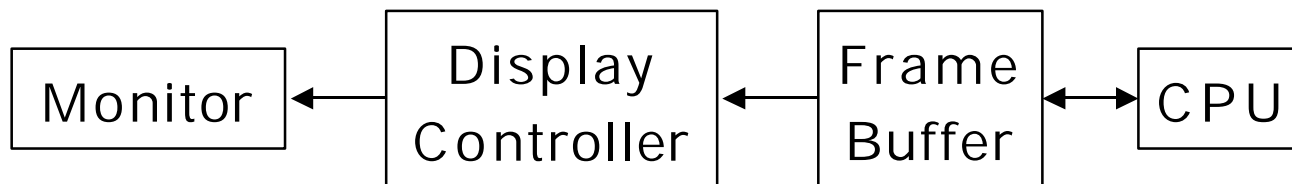
- Frame Buffer Architektur
- CRT Bildschirm
- LCD Bildschirm
- Druckausgabegeräte



# Frame Buffer Architektur



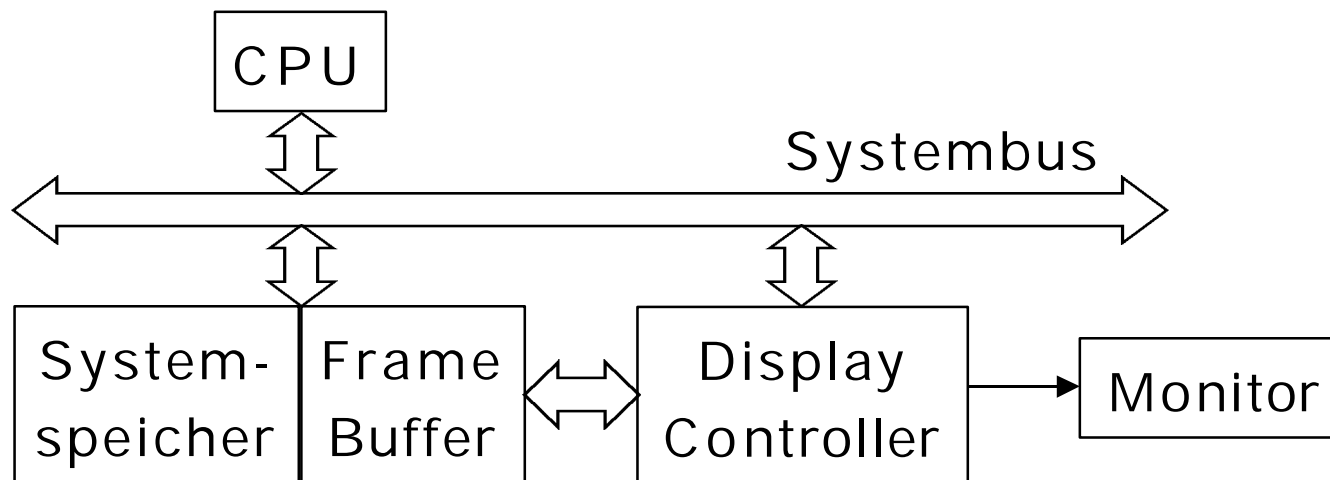
- Architektur aller gängiger Bildschirmperipherie
- Frame Buffer = Bildspeicher
- CPU kann meist auch aus Frame Buffer lesen



# Frame Buffer Architektur



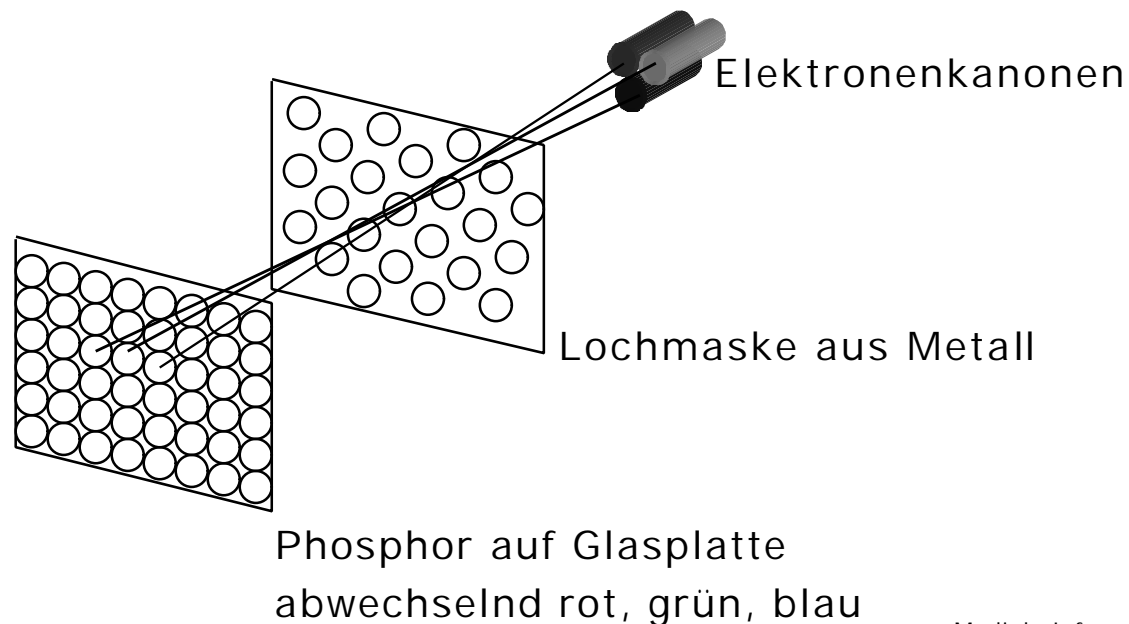
- Frame Buffer ist meist Dual-Port Memory, um vom Display Controller zugreifbar zu sein, ohne den Systembus zu belasten



# CRT Bildschirm



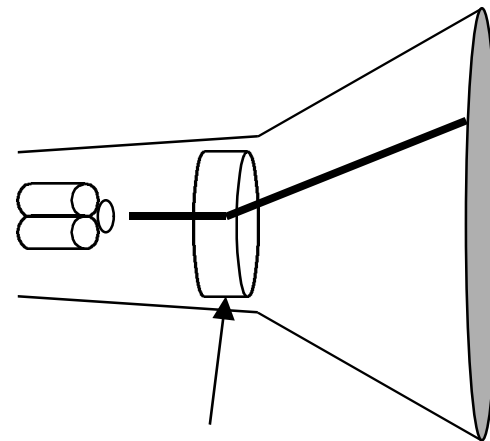
- CRT = cathode ray tube, Kathodenstrahlröhre
- Herkömmliche Fernsehetechnik
- Elektronenstrahlen bringen Phosphor auf dem Bildschirm zum Leuchten



# CRT Bildschirm



- Strahl tastet Bildschirm von links oben nach rechts unten ab
- Magnetfeld steuert diesen Vorgang



Magnet



# CRT Bildschirm



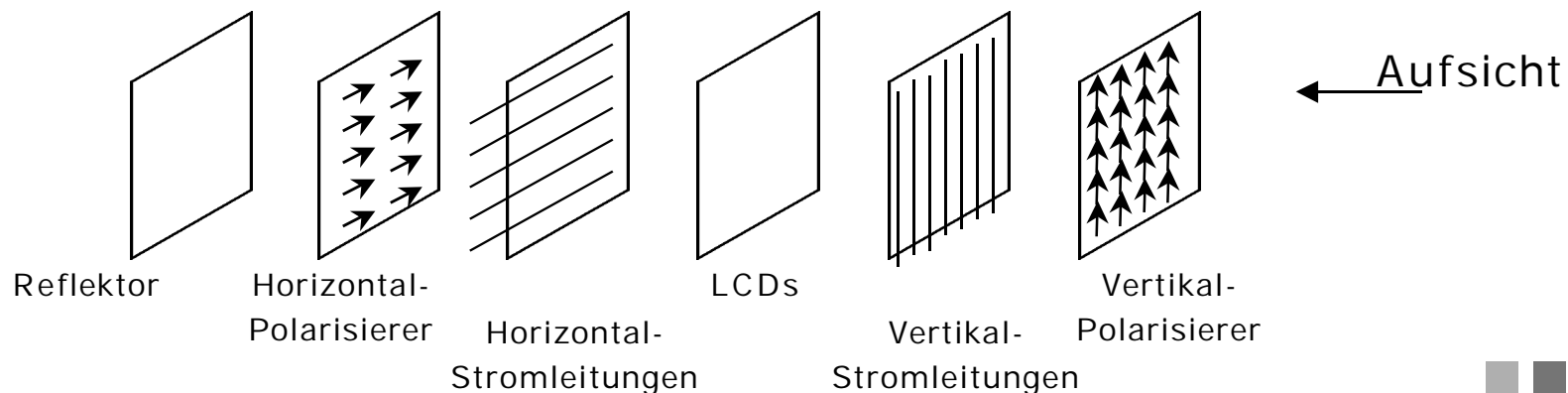
- Nachleuchten des Phosphor im Konflikt mit der menschlichen Wahrnehmung
  - Flackern
  - Spuren bewegter Objekte
- Deshalb Bildwiederholraten von 60 Hz und höher bei Computerbildschirmen
- Bei Fernsehen → Interlacing
  - Abwechselnd Zeilen beschreiben



# LCD Bildschirm



- Flüssigkristalle in Matrixanordnung werden durch elektrisches Feld polarisierend
- Falls Feld anliegt, ändern Kristalle die Polarisierung des Lichts nicht. Licht wird dann vom hinteren Polarisierer verschluckt



# LCD Bildschirm



- Passivmatrix
  - Nur an disjunkten Kreuzungspunkten kann Spannung erzeugt werden
  - Langes Nachleuchten notwendig
- Aktivmatrix
  - Transistor an jedem Kreuzungspunkt (Pixel)
  - Zustand kann individuell geändert werden



# LCD Bildschirm



- Vorteile
  - Flach, leicht
  - Niedriger Stromverbrauch
- Nachteile
  - Teurerer als CRT
  - Kleinere Bildschirmfläche
  - Blickwinkel, eigene Leuchtstärke eingeschränkter als bei CRT



# Druckgeräte



- Indirekt mit User Interface verbunden
- Fast immer soll aber auch gedruckt werden können
- Druckgeräte haben wesentlich höhere Auflösung als Bildschirme
  - Frame Buffer nicht mehr möglich
    - 600 dpi Seite bräuchte 3,6 MByte
  - Seitenbeschreibungssprachen
    - Regionenmodell
    - Postscript
- Druckertypen
  - Laser, Tintenstrahl, Thermotransfer, Plotter, ...



# A1.5 Grafiksoftware



- Einleitung
- Grafiksoftware mit Java
- Clipping
- Farbe



# Grafiksoftware



- Um die Präsentation von Grafikausgabe zu unterstützen, macht eine abstrakte Klasse Sinn
- Allgemeines Modell
  - Höhe, Breite
  - Physikalische Einheiten
  - Allgemeine Zeichenmethoden
- Mögliche Subklassen
  - Fenster
  - Bild im Speicher
  - Hardcopy zum Drucken
  - ...



# Grafiksoftware



- Die gängigen User Interface / Grafikpakete haben ein solches Konzept (oft jedoch nicht wirklich objektorientiert)
  - In Java: `java.awt.Graphics`
  - X Window: `display`, `window`
  - Windows: Graphical Device Interface, GDI
  - Macintosh Quickdraw: GrafPort
  - NeXTSTEP: View



# Grafiksoftware mit Java



- Java Foundation Classes (JFC) umfassen
  - AWT – Abstract Window Toolkit
  - Swing Components
  - Java 2D API
  - Accessibility API
  
- AWT ist der Kern und im weiteren Gegenstand



# Grafiksoftware mit Java



- AWT stellt Infrastruktur für JFC Komponenten zur Verfügung
  - Delegation Event Model
  - Lightweight Components
  - Clipboard and Data Transfer
  - Printing and mouseless Operation



# Grafiksoftware mit Java



- AWT enthält viele Subpackages
  - java.awt
  - java.awt.color
  - java.awt.datatransfer
  - java.awt.dnd
  - java.awt.event
  - java.awt.font
  - java.awt.geom
  - java.awt.im
  - java.awt.image
  - java.awt.print



# Grafiksoftware mit Java



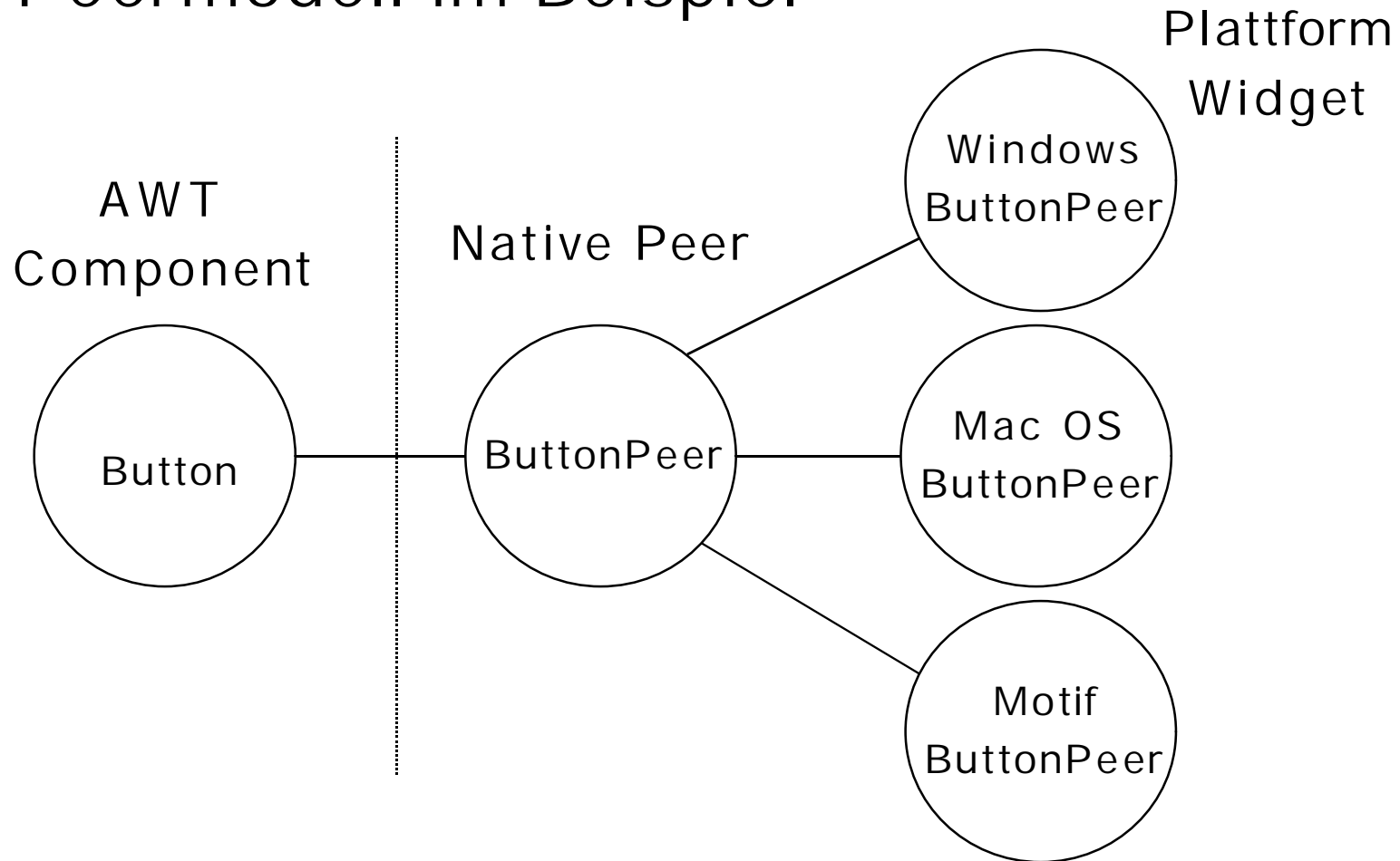
- Java ist plattformunabhängig
- Frage:
  - wie erfolgt die Darstellung unter dem nativen User Interface Management mit nativem Look-&-Feel?
- Lösung:
  - AWT Components delegieren an AWT Peers
  - Für fast jedes AWT-Oberflächenelement gibt es ein Peer-Objekt, das auf der jeweiligen Betriebssystemplattform vorhanden ist
  - Abbildung auf Oberflächenelemente der jeweils unterliegenden Plattform



# Grafiksoftware mit Java



## ■ Peermodell im Beispiel



# Grafiksoftware mit Java



- Das klassische PeermodeLL ist schwergewichtig
  - Jede Komponente hat ihren Peer mit eigenem Fenster
  - Skaliert schlecht
- Lightweight Components
  - Erweitern `java.awt.Component` oder `java.awt.Container`
  - „A lightweight component is a component that is not associated with a native opaque window.“
  - Darstellung in ihrem Container-Fenster ohne „Rahmen“
  - Alle Swing Komponenten sind lightweight



# Grafiksoftware mit Java



## ■ Klassenhierarchie (Auswahl)

```
java.lang.Object
|
+-- java.awt.Component -- abstrakte Klasse
| |
| +-- java.awt.Container -- abstr. Klasse
| |
| +-- java.awt.Panel
| |
| +-- java.awt.Window -- abstr. Klasse
| |
| +-- java.awt.Frame
|
+-- java.awt.Graphics -- abstrakte Klasse
|
```



# Grafiksoftware mit Java



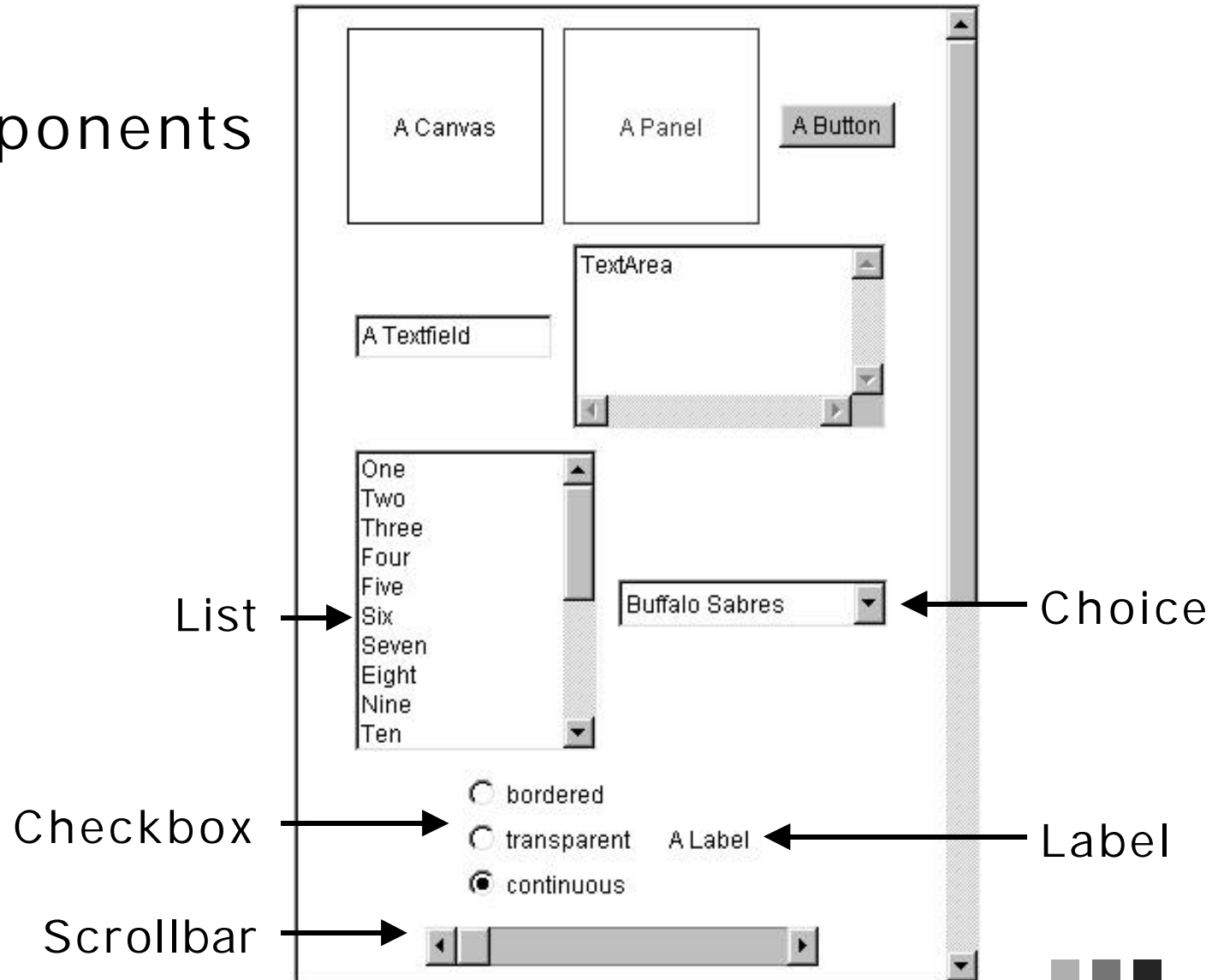
- Components sind die Basis von AWT
  - User Interface Controls
  - „A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.“
  - „Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.“
  - „The Component class is the abstract superclass of the nonmenu-related Abstract Window Toolkit components.“



# Grafiksoftware mit Java



## ■ Components



# Grafiksoftware mit Java



- Grundfunktionalität von Components
  - Graphics Objekt
  - Lokation
  - Größe
  - Native Peer
  - Vater Container
  - Fonts und deren Dimensionen
  - Vorder- und Hintergrundfarbe
  - Locale (internationale Einstellungen)
  - Minimale, maximale und bevorzugte Größe



# Grafiksoftware mit Java



- Container
  - Container enthalten Components
  - Subclasses sind u.a.
    - Panel, Applet, Window, Dialog, Frame, ...
  - Container implementieren das Interface `LayoutManager`
  - „A generic Abstract Window Toolkit (AWT) container object is a component that can contain other AWT components.
  - Components added to a container are tracked in a list.
  - The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list (and hence to the bottom of the stacking order).“



# Grafiksoftware mit Java



- Panel
  - „Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.“
  - „The default layout manager for a panel is the FlowLayout layout manager.“
  - Applet ist eine Subclass von Panel



# Grafiksoftware mit Java



- Window
  - „A Window object is a top-level window with no borders and no menubar. The default layout for a window is BorderLayout.“
  - „A window must have either a frame, dialog, or another window defined as its owner when it's constructed.“



# Grafiksoftware mit Java



- Frame
  - „A Frame is a top-level window with a title and a border.
  - The size of the frame includes any area designated for the border. The dimensions of the border area can be obtained using the `getInsets` method ...
  - The default layout for a frame is `BorderLayout`.“



# Grafiksoftware mit Java



- **LayoutManager**

- public Interface, welches Methoden zur Positionierung und Größeneinstellung für einen Container definiert

```
void addLayoutComponent(String name,Component comp)
```

```
void layoutContainer(Container parent)
```

```
Dimension minimumLayoutSize(Container parent)
```

```
Dimension preferredLayoutSize(Container parent)
```

```
void removeLayoutComponent(Component comp)
```

- **Classes, die LayoutManager implementieren**

- BorderLayout, CardLayout, FlowLayout, GridBagLayout, GridLayout



# Grafiksoftware mit Java



- Graphics
  - Abstrakte Klasse, die einen Grafikkontext und Methoden für Grafikoperationen zur Verfügung stellt.
  - Jede Component besitzt ein assoziiertes Graphics Objekt.
  - „The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.“



# Grafiksoftware mit Java



- Ein Graphics Objekt hält den Zustand des Grafikkontextes
  - Component Objekt auf das gezeichnet wird
  - Ursprung des Koordinatensystems
  - Clippingregion
  - Farbe
  - Font
  - Grafikmodus
    - Paint oder XOR



# Grafiksoftware mit Java



- Koordinatensystem
  - Koordinaten sind unendlich dünn und liegen zwischen den Pixeln des Ausgabegerätes
  - Zeichenoperationen traversieren zwischen den Pixeln
  - Ein pixel-breiter Stift hängt nach rechts unten

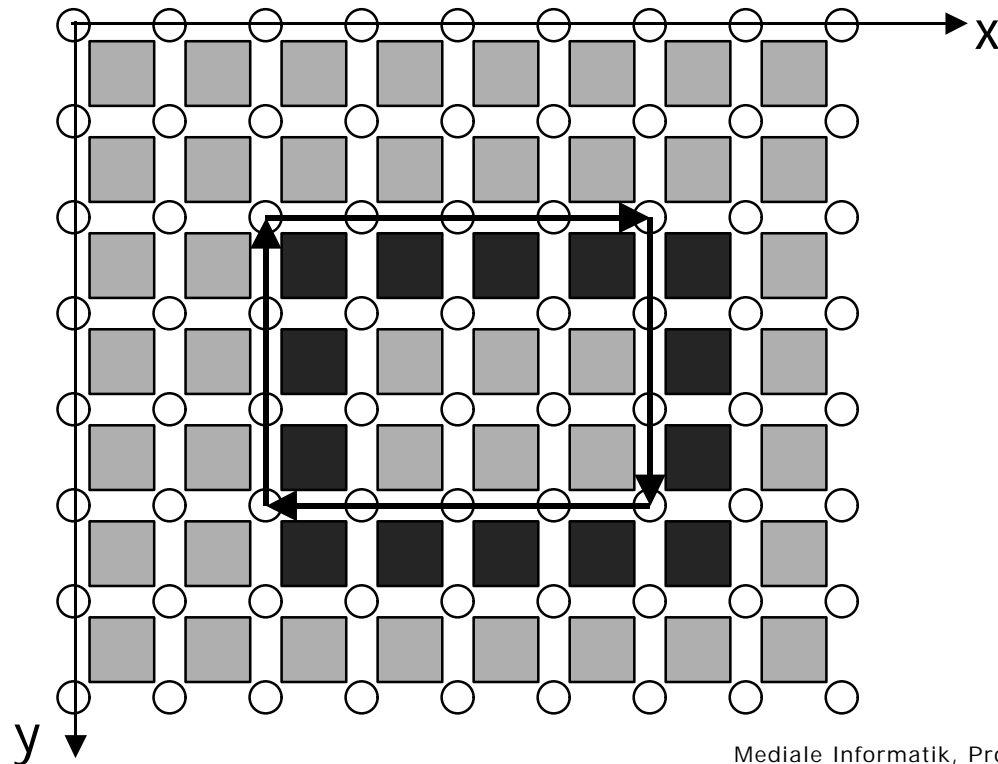


# Grafiksoftware mit Java



- Beispiel zum Koordinatensystem

```
public class RectTest extends Applet {
 public void paint(Graphics g) {
 g.drawRect(2,2,4,3);
 }
}
```



# Grafiksoftware mit Java



- Methoden der Graphics Class (Auszug)
  - Lebenszyklus
    - create
    - dispose
  - Gefülltes Zeichnen
    - fill3DRect
    - fillArc
    - fillOval
    - fillPolygon
    - fillRect
    - fillRoundRect
  - Zeichenmethoden
    - clearRect
    - copyArea
    - draw3DRect
    - drawArc
    - drawBytes
    - drawChars
    - drawImage
    - drawLine
    - drawOval
    - drawPolygon
    - drawPolyline
    - drawRect
    - drawRoundRect
    - drawString



# Grafiksoftware mit Java



- Methoden der Graphics Class (Auszug)
  - Properties abfragen
    - getClip
    - getClipBounds
    - getColor
    - getFont
    - getFontMetrics
    - hitClip
  - Properties setzen
    - clipRect
    - setClip
    - setColor
    - setFont
    - setPaintMode
    - setXORMode
    - translate

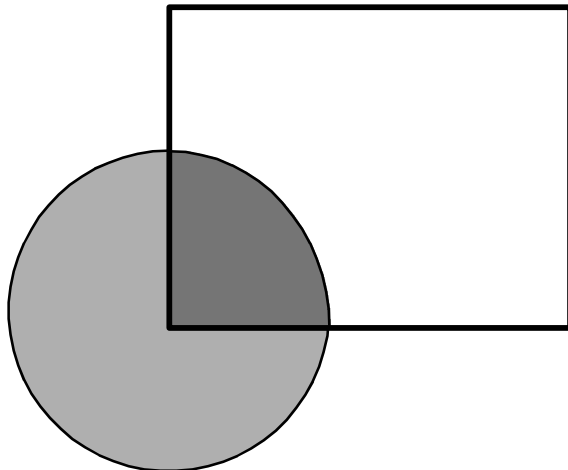


# Clipping

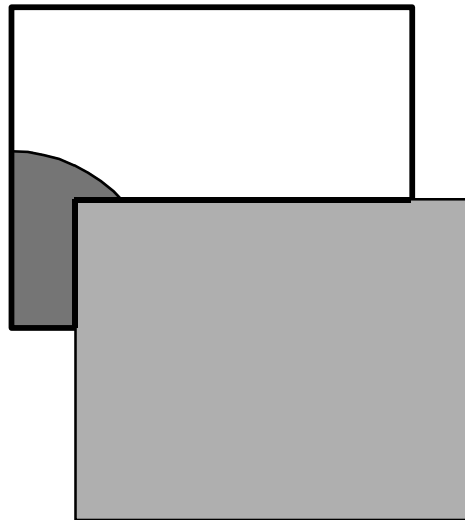


- Clipping ist notwendig, wenn die Zeichnung auf einen bestimmten Bereich beschränkt werden soll

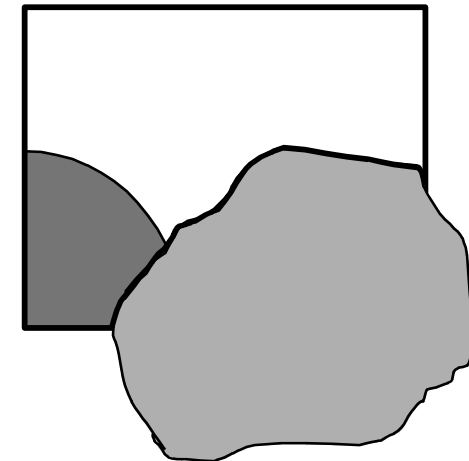
rechteckig



rechtwinklig



Beliebige Form



# Clipping



- Je komplizierter die Clippingform desto aufwendiger die Berechnung
- Algorithmen dazu siehe u.a. Foley, van Dam, Feiner, Hughes: Computer Graphics

