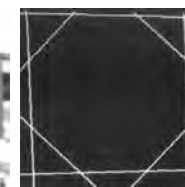
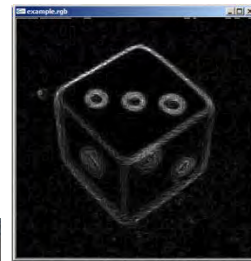
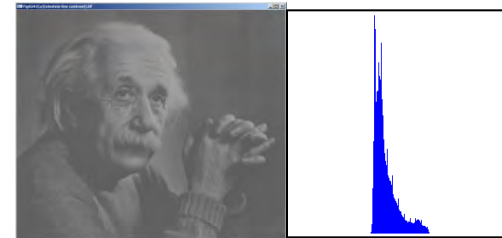
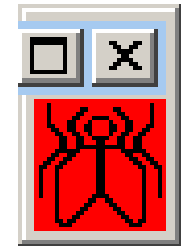
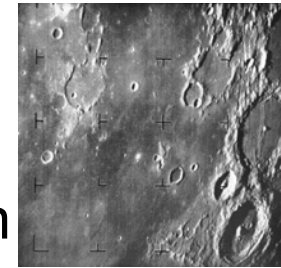


Stationen dieser Vorlesung:

- Geschichtlicher Abriß, Begriffsklärungen
- Fenstersysteme, Bildspeicherung und -ausgabe
- Bildauswertung, -optimierung
- Pixeloperationen, lokale, globale Operationen
- Fourier-Transformation, -Spektrum, -Filter
- Erkennung von geraden Linien (...)



1921: Erste Übertragung eines gerasterten Bildes in ca.3 h
N.York ⇒ London (Fernschreiber mit Typen-Aufsätzen)



Hierzu später (Rechner-Kontext): **das** (seltener: der) **Pixel**
– aus „picture element“, eher: „pic cell“, dt.: Bildpunkt

Pixel-Grafiken als (Hilfs-)Mittel, zunehmend als Ergebnis
techn.-wiss. Tätigkeit: Bild-Markierung ⇒ ... ⇒ Kartographie

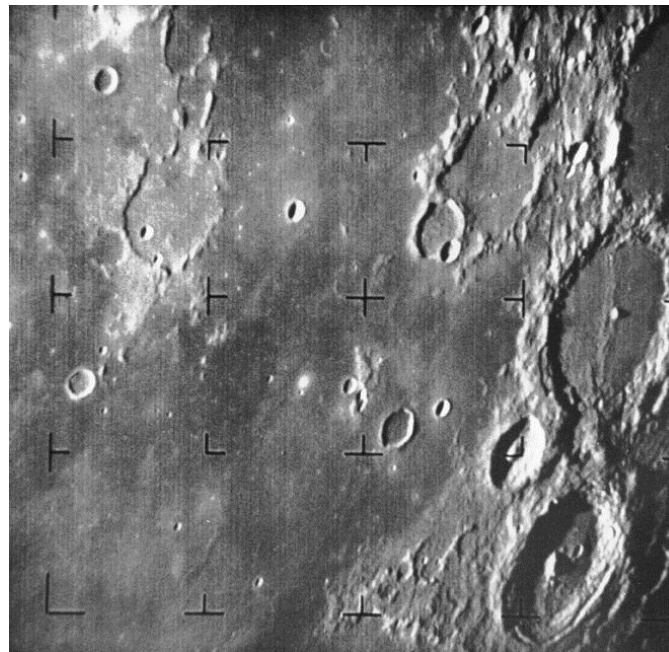
Bild-Retusche ⇒ ... ⇒ Zeichentrick ⇒ ... ⇒ Simulation / VR

Allen Anwendungen gemeinsam: Abb. in Rastern (Matrizen)

Begriff aus TV-Technik: Scan Conversion („Abtastumwandlg“)

Einleitung

- Ende 1950er / Anfang 1960er: Bildcodierung/-darstellung mit Computer; Auftrieb durch NASA
- Ab 1964: Empfang (gestörter TV-) Bilder von Raumsonden



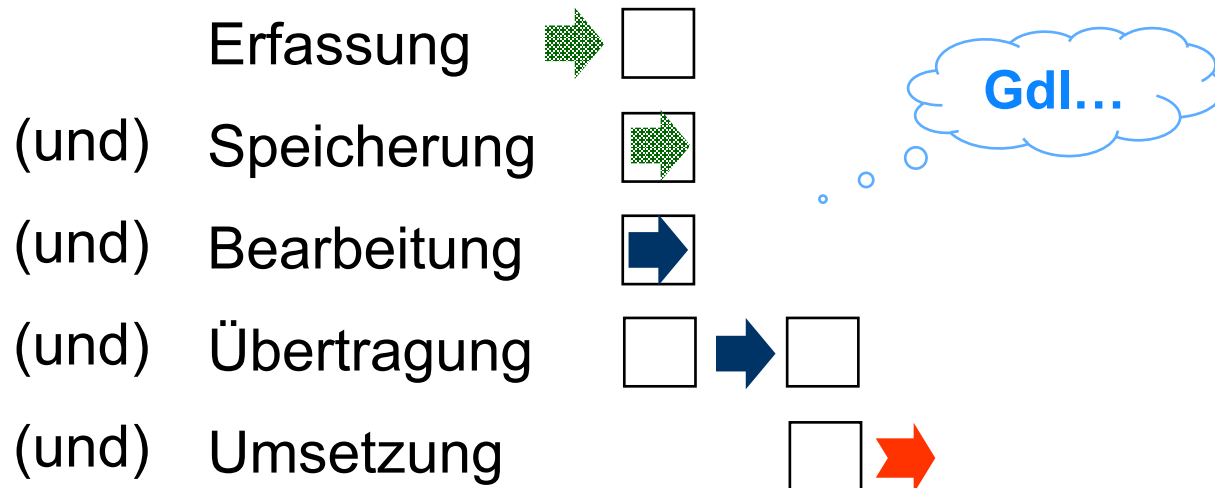
Erstes jemals empfangenes Mond-(TV-)Bild (1964, Ranger 7)

Bild: „Digital Image Processing“ Prentice Hall 2008

- Bedarf an Methoden zum genauen, reproduzierbaren, schnellen, massenhaften „Umgang“ mit Bildern (geometr./radiometrische Korrekturen, Merkmalsextraktion etc.)

Bildverarbeitung (BV): Gebiet der **Datenverarbeitung** (DV), das sich mit visuellen (sichtbaren, bildhaften) Daten befaßt

Datenverarbeitung ist die Tätigkeit u. das Wissensgebiet der



von **Daten** zum Zwecke der **Informationsgewinnung**.

Daten sind Angaben, die etwas kennzeichnen.

Information sind Daten, die in Entscheidungen einfließen.

BV wird oft auch als Oberbegriff zu Bildbearbeitung verwendet.

Bearbeiten: etwas behandeln (Möbel mit Politur, Gegner mit Fäusten), sich damit beschäftigen (Antrag), gestalten (Theaterstück für TV), überarbeiten (Manuskript), verändern (elterliche Meinung)

➔ Bearbeitung bedeutet Änderung, Anpassung d. Qualität

Verarbeiten: (etw.) als Material, Ausgangsstoff verwenden und in einem Prozeß zu etw. neuem anderem machen

(Gold zu Schmuck, Fleisch zu Wurst)

➔ Verarbeitung bedeutet Erzeugung einer neuen Qualität

Bearbeitung u. Verarbeitung sind zweckgerichtete Prozesse
(vgl. „Bug vs. Feature“)

Terminologische Unterscheidung:

Bild-analyse

- **Bildbearbeitung:** Gezielte Merkmalsveränderung, Bild-Manipulation /-Retusche (engl. *Image Editing*) oder Bild-Verbesserung (engl. *Image Enhancement*)
- **Bildverarbeitung:** Informationsgewinnung, Extraktion von Eigenschaften, Merkmalen etc. (engl. *Image Processing*)

Bild-synthese

- **Bildgenerierung:** Erzeugung visueller Darstellungen unter Verwendung logisch-mathematischer oder künstlerischer Methoden (engl. *Image Generation, Computer Graphics*)

- Fragestellungen u. Verfahren der Bildanalyse und -synthese größtenteils gleich, Grenzen unscharf (vgl. Filmtricks)
- Eingangs-/ Ausgangsgröße als Unterscheidungskriterium:

Bottom-up
Prozeß

Bildverarbeitung: Bild ⇒ Beschreibg., Folgerg., ..

Bildbearbeitung: Bild ⇒ „besser geeignetes“ Bild

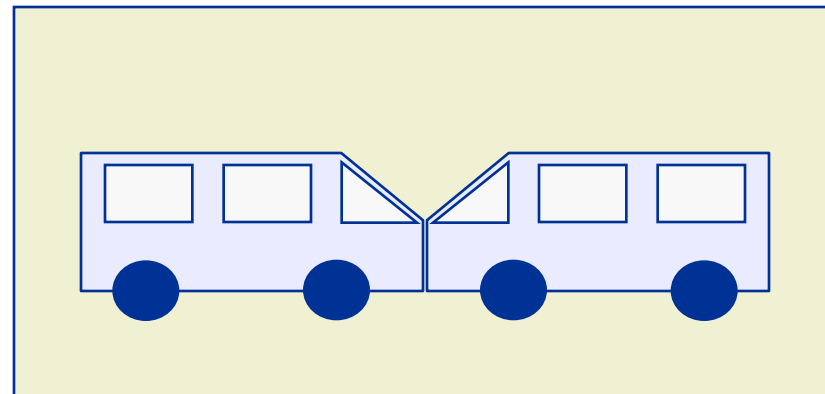
Top-down-
Prozeß

Grafik: Beschreibung ⇒ Bild

Wichtige Teilgebiete der Bildverarbeitung:

- Bildsignalauswertung, -verarbeitung
z.B. Histogramm, Ermittlung globaler Helligkeits-/Farbparameter
- Bildverbesserung (engl. *enhancement*, oft: Vorverarbeitung)
z.B. Anpassung von Helligkeit, Farbe, Konturschärfe
- Bildfilterung, -restaurierung (radiometrische Transform.)
z.B. Entfernung von Rauschen/Unschärfe, Gamma-Korrektur
- Bildverzerrung (geometrische Transformation)
z.B. Satellitenbild-Auswertung, Photogrammetrie, Kartographie
- Wissensbasierte Bildverarbeitung, Bildfolgenverarbeitung
z.B. Segmentierung, Barcode-/QR-Code-Auswertung, OCR
- Bildanalyse
z.B. Gewebeprobe-Untersuchung, thermische Zielerkennung

Inzwischen eigenständiges Teilgebiet der Künstl. Intelligenz:
Wissensbasiertes Bildverstehen, Szenenanalyse:



... dennoch für diesen Kurs weiterführend

„Niedere“ Bildverarbeitung /
Mustererkennung:

1 Dreieck
2 Rechtecke
2 Kreise
1 Fünfeck

} 2x

Höhere Bildverarbeitung /
Bildverstehen:

Kollision zweier Kleinbusse

Zwei Probleme seit den Anfängen der digit. Bildverarbeitung:

- Verfügbare Technik (TV) analog
- Benötigter Datenfluß enorm – z.B.:

Digitalisierung der 625 Zeilen eines monochromen TV-Bildes (PAL) mit Seitenverhältnis 4:3 bedeutet:

$$625 \cdot 4/3 \text{ Pixel/Zeile} = 833,33... \Rightarrow 834 \text{ Pixel/Zeile}$$

$$834 \text{ Pixel/Zeile} \cdot 625 \text{ Zeilen/Bild} = 521.250 \text{ Pixel/Bild}$$

Übertragung von 50 Halbbildern/sec (interlaced) mit 256 Graustufen heißt:

$$1 \text{ Byte/Pixel} \cdot 521.250 \text{ Pixel/Bild} \cdot 25 \text{ (Voll-)Bilder/sec}$$

$$= 13.031.250 \text{ Byte/sec} \approx 12,4 \text{ MB/sec (für schwarz-weiß)}$$

[Zum Vergleich:

Datenübertragungsrate Video-DVD: ca. 0,75 MB/sec]

Lösung:
Kompression

Teurer Lösungsansatz (z.T. noch aktuell): „Bildspeicher“:

- Analog/Digital-Wandler zur Digitalisierung des Video-Signals
- (Digitaler) Speicher für das Bild
- D/A-Wandler zur Anpassung an den Monitor

Einführung des PC durch IBM (1980er) führte zu Standardisierung und Modularisierung (Wettbewerb)

⇒ Preisverfall, Massenmarkt ⇒ Steckkarte „Frame Grabber“ (nicht standardisiert)

⇒ Herstellerspezifische Hw-nahe Programme (Treiber) nötig

Verwendung von CCD-Chips in Kameras (Charge-Coupled Device, späte 1980er) lieferte digitales Bild – darauf :

- D/A-Umsetzung in der Kamera (zur Anpassung an TV)
- A/D-Umsetzung des Video-Signals im Frame Grabber
- (Digitaler) Speicher für das Bild
- D/A-Umsetzung zur Anpassung an den Monitor

Situation (auch nach 2000):

- Zeit- / Qualitätsverluste zur Laufzeit
- Abhängigkeit von Hersteller-Treibern in der Sw-Entwicklung

Maßnahmen (Ende 1990er):

1. Entwicklung von Kameras mit digitalem Ausgang
 - ⇒ Auslassung der D/A- und der A/D-Umsetzung – aber:
Notwendigkeit eines sog. „Digital Frame Grabber“:
Digitales Ausgangssignal nicht standardisiert
2. Entwicklung des Busses „IEEE 1394“ („FireWire“)
 - ⇒ Kameras und PCs mit IEEE 1394-Schnittstelle – aber:
Protokoll zum Datenaustausch nicht spezifiziert
3. Entwicklung d.DCAM-Protokolls (Sony, Hamamatsu, IIDC)
 - ⇒ „DCAM-Treiber“: herstellerunabhängig, BS-spezifisch
(DECAM für Linux, für Windows etc.)

Einleitung

Verbleibender Bedarf: API zur Kommunikation der Applikation mit der bildgebenden Hw über d. Betriebssystem

Häufige API-Wahl: **DirectX**



Microsoft®
DirectX®

Vorteile: Verbreitung • ständige Aktualisierung • Nähe zum „Consumer Market“

Nachteil: Abhängigkeit v. Microsoft (Verfügbarkeit, Notation)

API-Wahl hier: **OpenGL**



Vorteile: Verbreitung • ständige Aktualisierung • BS-Unabhängigkeit • Nutzung mit allen verbreiteten BSen (inkl. embedded), wo OpenGL installiert ist • Kombination mit Grafik • Nähe zu Open Source (s.u.)

Nachteile: ?

OpenCV (Open Computer Vision):

- Open-Source-Programmbibliothek mit Hunderten von Algorithmen (je nach Zählung: $> 500 \dots \leq 2.500$ Funktionen) für Windows, Linux, Mac OS X, Android u.a..
- 1999: Forschungsprojekt von Intel-Rußland z.B. für
 - Verarbeitung von Satellitenfotos,
 - Filterung medizinisch-diagnostischer Aufnahmen,
 - Aufbau von Überwachungssystemen,
 - Qualitätskontrolle industrieller Produktion,
 - Steuerung unbemannter Fahr- / Flugzeuge und U-Boote,
 - Bild- und Stimmen-Wiedererkennung.
- 2006: Version 1.0 in C, danach auch C++
- 2008: Beteiligung v. Willow Garage Inc. (CA, USA)
- Bezug: <http://opencv.org> (Version 08/2018: 3.4.3)



- Pixel- / Farbcodierung ab einem Bit (Strichzeichnungen):
„Bitmap“ (OpenGL- / SGI-Terminus)
Bildtechnik heute (TV, Monitore, Handys) Rot, Grün, Blau
- Befriedigende Grau-/Farbtondarstellung benötigt i.d.R. ca. 6 Bit (64 Abstufungen) – nach erfolgter Adaptation (⇔ Tunnelfahrt, NASA-Simulation Erdumkreisung) meist (je Farbkomponente): 8 Bit; selten: 16 Bit
- Grauwert- / Farbcodierung in Bild u. Grafik gemeinsam, oft mit Transparenz-Komponente „Alpha“: RGBA
Gebräuchliche Datentypen zur Grauwert- / Farbcodierung:
 - ⇒ `unsigned char` bzw. `char` (8 Bit: monochrom, LUT)
 - ⇒ `unsigned int` (16 o. 32 Bit: RGB, RGBA)
 - `unsigned long int` (32 Bit: RGBA)
 - `unsigned long long int` (64 Bit: RGBA)

Anpassung
an Licht-
verhältnisse

Bilder werden i.a. als indizierte Variablen (Arrays) dargestellt.
Sie werden typischerweise eindimensional codiert.

Hintergrund:

Matrizen-Deklaration darf max. 1. Index (v.li.) offen lassen:

```
unsigned char matrix[][COL]; //feste Breite
```

Größere Flexibilität mit eindimensionalen Feldern:

```
unsigned char *image; /*Speicher zuweisen!*/
```

```
int jx, jy, jz; /* (...) */
```

```
/*Graukeil mit 256 Stufen:*/
```

```
#define HEIGHT 1080  
#define WIDTH 1920  
#define CMPNTS 4
```

```
for(jy=0; jy<HEIGHT; jy++)//image[jz][jy][jx]
```

```
    for(jx=0; jx<WIDTH; jx++)
```

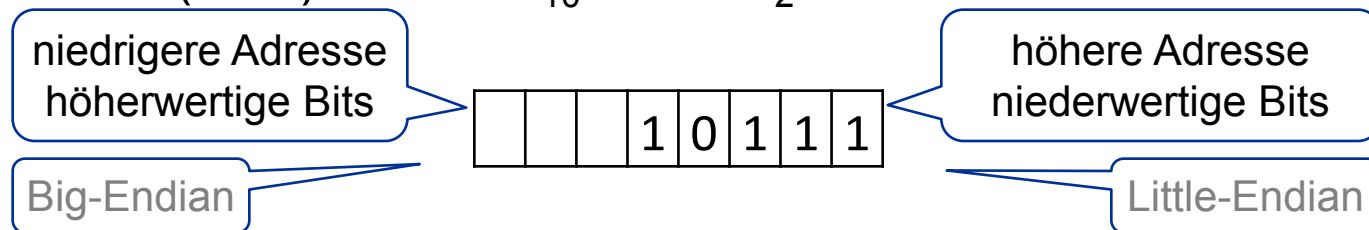
```
        for(jz=0; jz<CMPNTS; jz++)
```

```
            image[jy*WIDTH*CMPNTS + jx*CMPNTS + jz]
```

```
                =(255*jx)/(WIDTH-1);
```

Systemübergreifende Nutzbarkeit von Bilddateien erfordert Auseinandersetzung mit Zahlencodierung an Rechnern:

- Ablage der Ziffern in (gräko-romanischer) Schreibweise von links nach rechts belegt jedes Byte mit den höherwertigen (signifikanteren) dualen Ziffern (Bits) an den niedrigeren Adressen (links) – z.B. $23_{10} = 1\ 0111_2$:

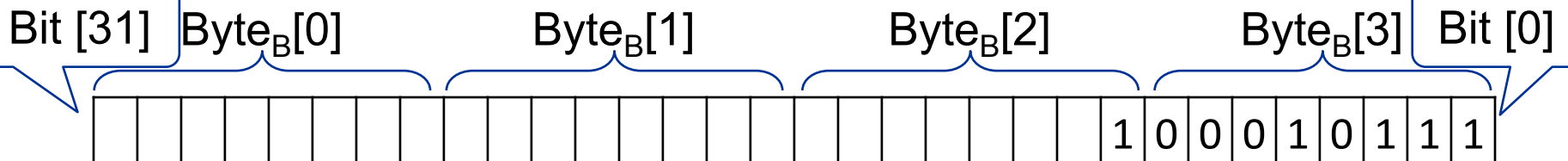


Da ein Byte i.d.R. die kleinste adressierbare Speicher-Einheit ist, gibt es dafür herstellerübergreifende Kompatibilität.

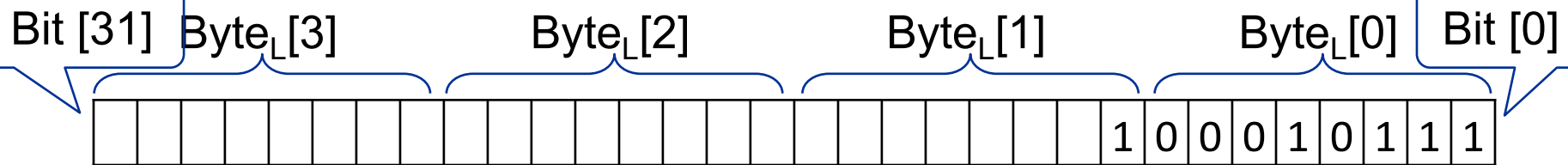
- Keine einheitlichen Standards gibt es dafür, was bei Mehr-Byte-Darstellungen (z.B.: 4-Byte-`int`) am **Anfang**, d.h. an niedrigen Speicher-Adressen mit höherwertigen Bits, stehen soll:
 - **höherwertige Bytes an höherwertige Bits: Big-Endian**
 - **niederwertige Bytes an höherwertige Bits: Little-Endian**
- } „Endianness“

Codierungsaspekte

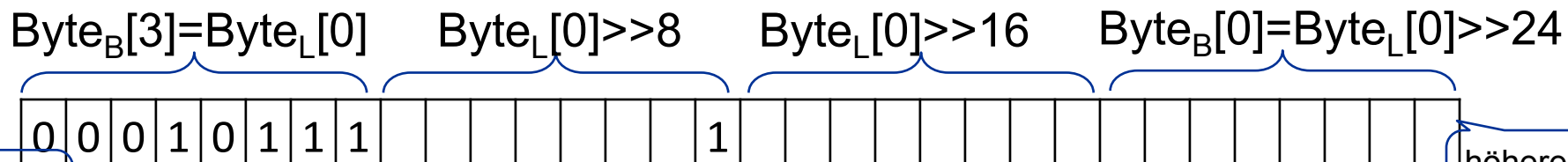
Vier-Byte-Darstellung von $279_{10} = 1\ 00010111_2$ bei Motorola, SGI (Big-Endian) – zugleich physikalische Byte-Anordnung:



Vier-Byte-Darstellung von 279_{10} bei Intel (Little-Endian):



Physikalische Byte-Anordnung bei Little-Endian (z.B. Intel / PC):



niedrigere Adresse

höhere Adresse

→ Umspeicherungsbedarf bei Erzeugung von SGI-Dateien am PC!

Codierungsaspekte

Auswirkungen der internen Darstellung (Byte-Folge) am Beispiel

$50.462.976_{10} = 11\ 00000010\ 00000001\ 00000000_2$:

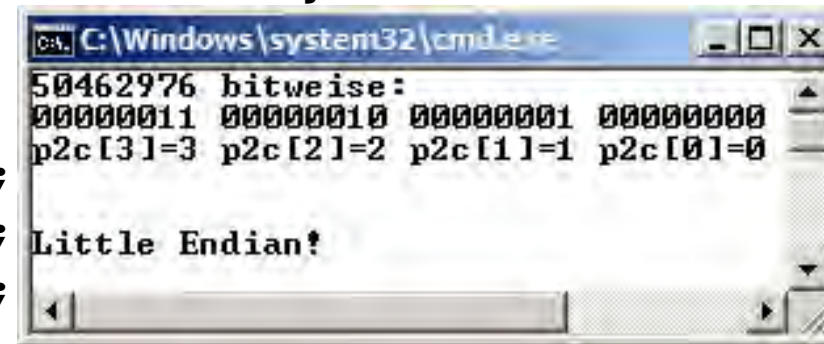
```
int main(void)
{
    int bit=0;
    unsigned int *p2i, i=50462976;
    unsigned char *p2c, c[]={3,2,1,0, 0,1,2,3};

    printf ("%u bitwise:\n", i);
    for (bit=0; bit<32; bit++)
    { if(i&1<<(31-bit))printf("1");
      else printf("0");
      if(!((bit+1)%8)) printf(" ");
    } printf ("\n");

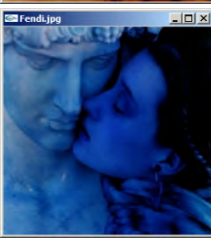
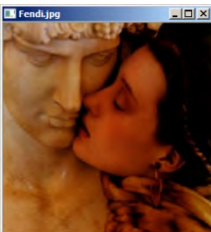
    p2c = (unsigned char *) &i;
    printf("p2c[3]=%d p2c[2]=%d p2c[1]=%d p2c[0]=%d\n\n",
           p2c[3], p2c[2], p2c[1], p2c[0]);

    p2i = (unsigned int *) c;
    if(i==*p2i) printf ("Big Endian!"); //(i==p2i[0])
    if(i==*(p2i+1)) printf ("Little Endian!"); _getch();}
```

```
#include <conio.h>
#include <stdio.h>
```



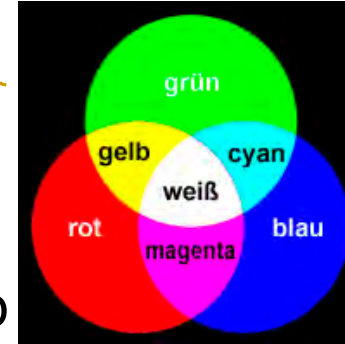
PointerCast.c



RGBA-Farbspeicherung am PC (Intel): 4 Byte ($2^{31} \dots 2^0$), d.h.:
4.294.967.295 ($= 2^{32} - 1$): weiß (opak, d.h.: deckend)
16.777.215 ($= 2^{24} - 1$): weiß (evtl. transparent); darin:
Byte[3]: Alpha; Byte[2]: Blau; Byte[1]: Grün; Byte[0]: Rot

additive Farbmischung

⇒ **ABGR !**



Beispiele: Farben bei voller Helligkeit und Sättigung

$$255 = 2^8 - 1 \quad : \quad \text{rot}$$

$$65.535 = 2^{16} - 1 \quad : \quad \text{rot} + \text{grün} = \text{gelb}$$

$$65.280 = 2^{16} - 1 - (2^8 - 1) \quad : \quad \text{gelb} - \text{rot} = \text{grün}$$

$$16.776.960 = 2^{24} - 1 - (2^8 - 1) \quad : \quad \text{weiß} - \text{rot} = \text{cyan}$$

$$16.711.680 = 2^{24} - 1 - (2^{16} - 1) \quad : \quad \text{weiß} - \text{gelb} = \text{blau}$$

$$16.711.935 = 2^{24} - 1 - [(2^{16} - 1) - (2^8 - 1)] : \quad \text{weiß} - \text{grün} = \text{magenta}$$

Pixel-Codierung als ganze Zahl: Farbwert-Operationen mit bitweiser Verschiebung (Shift) / VerUNDung / VerODERung

Beispiel:

RGB-Codierung eines Pixels mit [R, G, B] = [196, 139, 72] deckend (opak) durch Veroderung der Farbkomponenten, Abfrage der Pixel-Farbanteile durch VerUNDung, wobei:

RGB ([196, 139, 72])

$$= 196 \cdot 2^0 + 139 \cdot 2^8 + 72 \cdot 2^{16} = 4.282.944.452$$

```
unsigned int  rgb;
```

```
unsigned char r, g, b;
```

```
rgb = 196 | 139<<8 | 72<<16 | (((1<<8)-1)<<24);
```

```
r = rgb & ((1<<8)-1);
```

```
g = (rgb & (((1<<16)-1) - ((1<<8)-1))) >> 8;
```

```
b = (rgb & (((1<<24)-1) - ((1<<16)-1))) >> 16;
```

Übung

Anschaulichere Vorgehensweise:

Gleichsetzung des Speichers einer `int`-Variablen und eines vier-Elemente-großen `char`-Feldes, das elementweise mit `[R, G, B, A]` gesetzt wird.

Beispiel:

RGBA-Codierung eines Pixels mit `[R, G, B] = [196, 139, 72]` deckend (opak):

```
enum {R, G, B, A};  
unsigned int    rgb=0;  
unsigned char *pixel=(unsigned char *) &rgb;  
pixel[R]=196;   pixel[G]=139;  
pixel[B]=72;    pixel[A]=255;
```



Übung

Bildcodierung und -speicherung

Allgemeine Datenstruktur zur universellen Bildcodierung:

Grundsätzlich als Zeiger: Größen, die zur Compilierungszeit nicht bekannt sein können

```
#define LENGTH 80
```

```
typedef char String[LENGTH];
```

```
typedef struct
{
    String      Name;      //Bildname

    int         Width;     //Pixel je Bildzeile
    int         Height;    //Anzahl Bildzeilen

    int         Cmpnts;    //Byte je Pixel;s/w:1;RGB:3
    int         Format;     //Kennzahl:GL_RGB etc.

    unsigned int *Data;    //Bilddaten
} DIPimg;
```

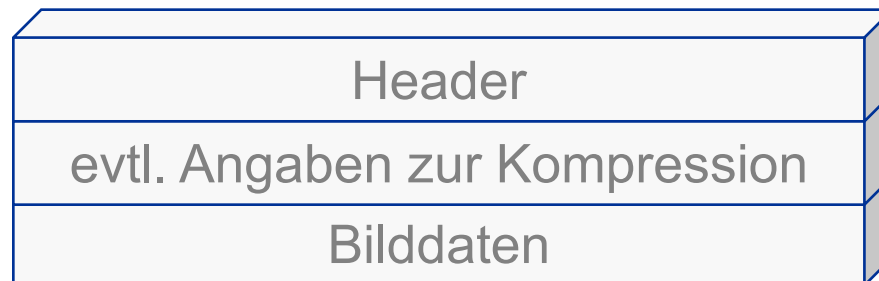
Zusammenfassung von vier Ein-Byte-Objekten (RGBA)



Grundvoraussetzung für die Bildverarbeitung:

- Bilder laden aus / speichern in Bilddateien
Laden von Bildern durch Reservierung (*allocation*) von Speicherplatz zur Laufzeit
- Individuelle, unbekannte Bildmerkmale (Größe, Format etc.) werden im zuerst geladenen Dateibereich (Header) abgelegt, damit ein Bild korrekt geladen werden kann.

⇒ Allgemeiner Aufbau einer Bilddatei:



Bestandteile eines universellen Bild-Headers (SGI)

2 Byte	short	Kontrollzahl: 474_{10} („IRIS image file magic number“)
1 Byte	char	Speicherformat (0: ohne Kompression)
1 Byte	char	Byte je Farbkomponente („bytes per pixel channel“)
2 Byte	ushort	1-zeilig (Graukeil) / s/w / Farbbild („dimension“)
2 Byte	ushort	Bildbreite in Pixel
2 Byte	ushort	Bildhöhe in Pixel
2 Byte	ushort	Anzahl Farbkomponenten („channels“)
4 Byte	long	Niedrigster Pixelwert (i.d.R.: 0)
4 Byte	long	Höchster Pixelwert (i.d.R.: 255)
4 Byte	char	Platzhalter („dummy“ – wird übergangen)
80 Byte	char	Bildname
4 Byte	long	normal / dithered / LUT / „SGI-Colormap“ (i.d.R.: 0)
404 Byte	char	Platzhalter („dummy“ – wird übergangen) ⇒ $\Sigma = 512$ Byte

- Namenskonventionen von SGI-Bilddateien:
 - *.**bw** (schwarz-weiß); *.**rgb** (Farbe); *.**rgba** (Farbe, ggf. mit Transparenz); *.**sgi** (nicht festgelegt, i.d.R. RGB)
- Bild-Ursprung ist immer die untere linke Bild-Ecke. Die erste (geschriebene / gelesene) Zeile (Index 0) ist die unterste Bildzeile.
- Die Bilddaten einer Farbkomponente (Kanals) werden vollständig abgelegt; wenn das Bild mehrere Kanäle hat, werden diese nacheinander gespeichert. Die Kanäle werden pixelweise (mit je 1 oder 2 Byte) abgelegt.
- Beim Laden eines Bildes empfiehlt es sich, die Daten jedes Pixels zu einem Datentyp zusammenzuführen (RGBA: `unsigned int`) mit Rot bei Index 0.
- Beim Speichern sind die Farbkomponenten zu trennen und die (SGI-eigene) Byte-Reihenfolge zu beachten.

Man beachte d. Unterschied zw. Codierung u. Speicherung:

- Im **Programm-Code** sind die Angaben zu einem Pixel jeweils in einem Datenelement zusammen.

Üblich: Farbcodierungen mit ≤ 1 bis 2 Byte / Komponente

z.B.: `short` mit 5 Rot-Bit + 6 Grün-Bit + 5 Blau-Bit

hier: `unsigned int` mit Rot[0], Grün[1], Blau[2], Alpha[3]

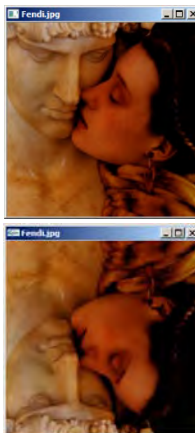
- In der **Bild-Datei** liegen die einzelnen Farbkomponenten jeweils vollständig hintereinander.

Es gibt Farbformate mit 1 – 4 Komponenten

Bsp.: Grautonbild; Farb-Index (LUT); RGB, RGBA etc.

Beachtlicher Unterschied bei der **Zählung** der Pixel:

- **Bildsysteme** (z.B. OpenCV) zählen ab **oben links**.
- **Grafiksysteme** (z.B. OpenGL) zählen ab **unten links**.



Übung

Übung:

(verwendet neben GLUT- und OpenGL-auch OpenCV-Code)

Beginn der Implementierung von `DIPintro.c`
(eines einfachen Bildverarbeitungsprogramms)

