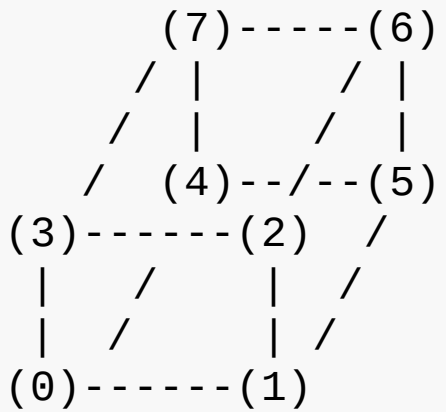
Codierung von Grafik-Modellen

Speicherung eines Grafik-Objekts (z.B. eines Würfels):

```
cube.cgf ; Objekt-Name - CGF Version 0.0
8 ; Anzahl Punkte
-1., -1.,  1. ; Koord. Pkt[0] etc.
 1., -1.,  1.
 1.,  1.,  1.
-1.,  1.,  1.
-1., -1., -1.
 1., -1., -1.
 1.,  1., -1.
-1.,  1., -1.
6 ; Anzahl Flaechen
4, 4, 4, 4, 4, 4 ; Pkte je Flaeche
0, 1, 2, 3 ; Pkt-Idx ab Flaeche[0]
1, 5, 6, 2
2, 6, 7, 3
3, 7, 4, 0
4, 7, 6, 5
5, 1, 0, 4
```

Geometrie der Objektpunkte (ihre räumliche Lage)

Topologie der Objektpunkte (Beziehungen zwischen ihnen)



(Xface?)

Codierung von Grafik-Modellen

Anmerkung zur Codierung von Grafik-Objekten (z.B. Würfel):

Grundsätzlich zwei technisch realisierbare Codierungs-philosophien für die (4D-)Flächenecken (z.B.: **`facePnt`**):

1. Individuelle Speicherung aller Flächenecken-Koord., d.h. (6 Flächen x 4 Punkte/Fläche x 4Koord./Punkt=) 96 Werte:

   `float facePnt[6][4][DIM];` **`#define DIM 4`**

   ⇨ Unnötig erhöhter Speicherbedarf

   Bei Formänderungen (z.B.: Erhöhung einer Kuppelspitze) ist zu berücksichtigen, daß jeder Objektpunkt in mehreren (beim Würfel: in 3) Flächen vorkommt:

   ⇨ Unnötig erhöhter Rechenzeitbedarf

   ⇨ Kaum wartbare Fehler-Quelle

   ➥ nicht gebräuchlich, nicht zu empfehlen (quick & dirty)

2. Alternative: Trennung von Objekt-Geometrie u. -Topologie

Codierung von Grafik-Modellen

2. Geometrie-Codierung von 3D-Objekten durch (einmalige) Speicherung der Koordinaten – z.B. Würfel:
   `float cubVrtx[8][DIM];  /*32 Koordinaten*/`

   Zwei gebräuchliche Verfahren zur Topologie-Erfassung:

   2a) Erfassung der Punkt-Indizes für jede Fläche – z.B.:
   `int facePnt[6][4];  /*24 Indizes*/`

   Bsp. Zuordnung Flächen- / Objektpunkt:
   `facePnt[5][3]=7;  /*4.Pkt d.6.Flaeche≡8.ObPkt*/`

   ⇨ Unhandliche, fehleranfällige Ausdrücke, z.B.:
   `float x;  /*...*/  x=cubVrtx[facePnt[5][3]][0];`

   ➥ Gebräuchlich eher in„Rapid-Prototyping“-Programmen

   2b) Erfassung der Punkt-Adressen für jede Fläche – z.B.:
   ```
   float *facePnt[6][4], x; /*...*/
   facePnt[5][3]=cubVrtx[7];   x=facePnt[5][3][0];
   ```

   ➥ Intuitive, übersichtliche, robuste Ausdrücke

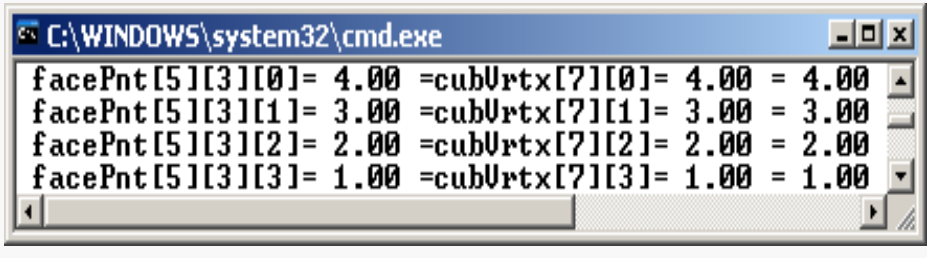
Codierung von Grafik-Modellen

Beispiel: „Harte“ Codierung eines Würfels

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

```
int main()
{ int    j1=0;
  float  cubVrtx[8][DIM]; //Koordinaten der Wuerfelecken
  float *facePnt[6][4];   //Flaechen-Def.: Punkt-Adressen
  /*Koordinaten-Zuweisung fuer Wuerfel-Punkt (z.B.):*/
  for(j1=0;j1<DIM;j1++) cubVrtx[7][j1]=4.f-j1;//=4,3,2,1
  /*Verknuepfung Flaechen-Ecke / Wuerfel-Punkt (z.B.):*/
  facePnt[5][3] = cubVrtx[7]; /* = &cubVrtx[7][0]; */
  for (j1=0; j1<DIM; j1++)      /*Koordinaten-Ausgabe:*/
  { printf ("\n\r facePnt[5][3][%d]=%5.2f"
    " = cubVrtx[7][%d]=%5.2f",
    j1, facePnt[5][3][j1],
    j1, cubVrtx[7][j1]);

    printf (" =%5.2f \n\r",*(*(*(facePnt+5)+3)+j1));
  } _getch();  return 0;  }
```



s.Projekt: 010ObjCod

Codierung von Grafik-Modellen

- Besonders interessanter Spezialfall: Zuweisung von phys. Eigenschaften (z.B. Farbe) einem bestimmten Objektpunkt

  ⇨ Ermittlung des Objektpunkt-Index aus der Objpkt.-Adresse

Beispiel:

```
#define DIM 4
float  cubVrtx[8][DIM];
float *facePnt[6][4]; /*...*/
facePnt[5][3]= facePnt[4][1]= cubVrtx[7];
```

6 Startadressen (**facePnt[i]**) von Eckpunkt-Listen mit je 4 Startadressen (**facePnt[i][j]**) von Startadressen (**cubVrtx[k]**) von Koordinaten-Listen

Gemeinsamkeit aller Instanzen **facePnt** eines Objektpunktes **cubVrtx** auf unterschiedlichen Flächen:

Adressen-Abstand zu **cubVrtx[0]** – z.B.:

```
printf("Idx=%d\n",(facePnt[5][3]-cubVrtx[0])/DIM);
printf("Idx=%d\n",(facePnt[4][1]-cubVrtx[0])/DIM);
```

(Ausgabe beider Anweisungen: „**Idx=7**“)

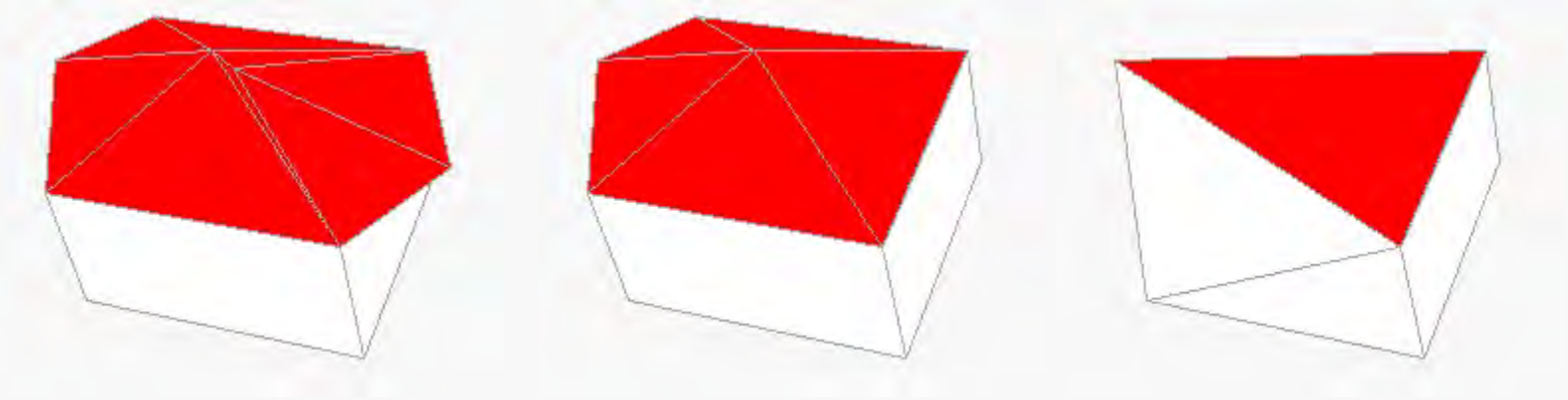
s.Projekt: `ObjCod`

Codierung von Grafik-Modellen

Veränderte Aufgabenstellung in der Praxis:

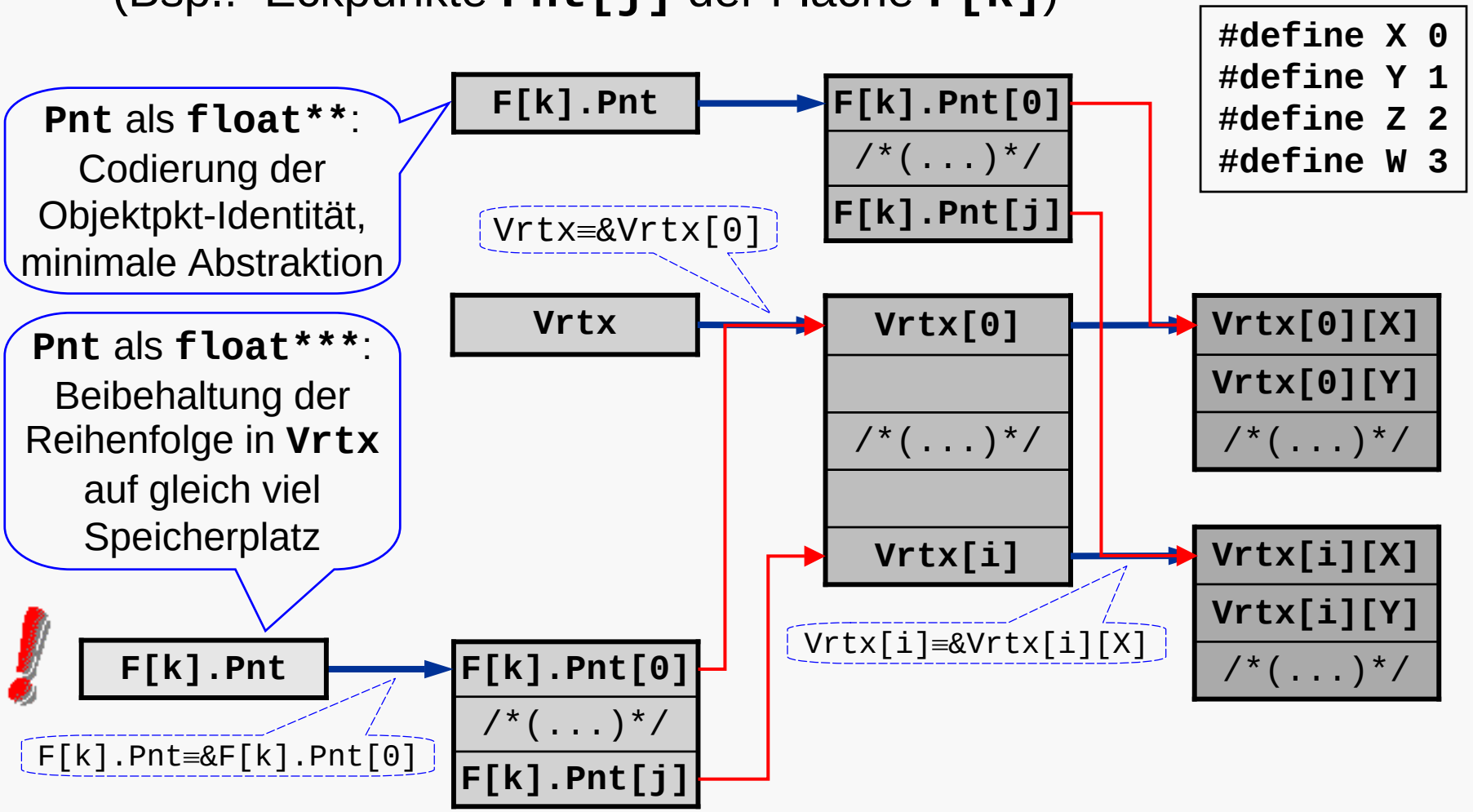
- Anzahl von Punkten u. Flächen erst beim Laden bekannt (⇨ Speicherplatz-Belegung zur Laufzeit)
- Anzahl von Punkten pro Fläche zudem i.d.R. ungleich (⇨ z.T. verkettete Listen statt Feldern)
- Je nach Anwendung:

  Anzahl von Objektpunkten und Flächen, aber auch Identität von Flächen-Eckpunkten variabel



Codierung von Grafik-Modellen

Codierung d. Objekt-/Eckpkt-Zuordnung prinzipiell wählbar:
(Bsp.: Eckpunkte **`Pnt[j]`** der Fläche **`F[k]`**)



Codierung von Grafik-Modellen

Codierung von Geometrie (**`Vrtx`**) u. Topologie (**`Pnt`**) in der Praxis:

- Speicherreservierung für Objektpunkte je nach Objektgröße:

```
Vrtx = (float**) calloc (nVrtx,sizeof(float*));
for (j=0; j<nVrtx; j++)
  Vrtx[j] = (float*)calloc(NDIM,sizeof(float));
```

- Speicherreservierung für Flächenpkte je nach Codiergskonzept:

  Flächenpunkt als **`float***** vs. *`float**`*:

```
for (j1=0; j1<nFace; j1++)
{ Face[j1].Pnt =
 (float***)calloc(Face[j1].nPnt,sizeof(float**));
  (float**)calloc(Face[j1].nPnt,sizeof(float*));
 for (j2=0; j2<Face[j1].nPnt; j2++) { /*...*/
    Face[j1].Pnt[j2] = &Vrtx[j3];
    Face[j1].Pnt[j2] =  Vrtx[j3]; }
}
```

Codierung von Grafik-Modellen

Allgemeine programm-interne Darstellg. eines 3D-Objektes:

```
typedef struct
{ int      nPnt;  //Anzahl Eckpunkte
  float ***Pnt;   //Punkt-Indizes
  char     symbol;//Zeichn f.Konsole
} Polygon;
```

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

stellv. für Flächen-Eigensch.: Farbe, Textur, Rauhigkeit,...

```
#define DIM      4
#define LENGTH  80
```

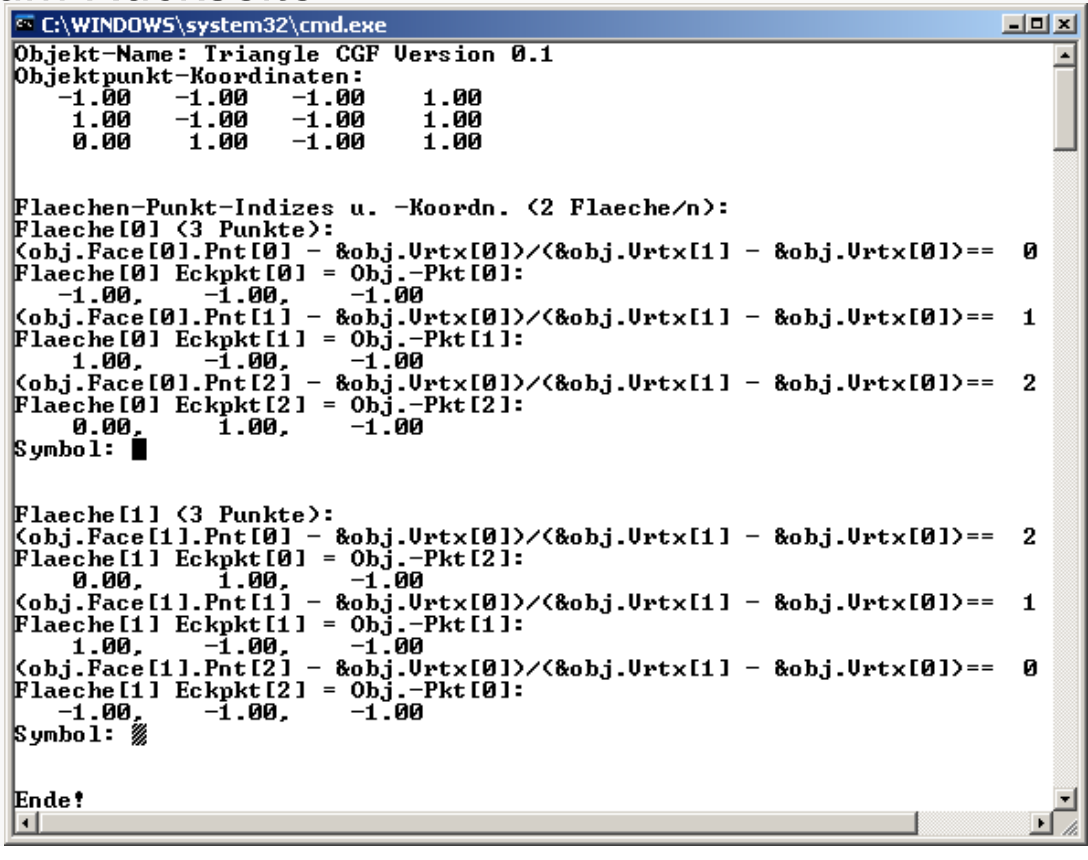
Als Zeiger: Größen, die zur Compi-lierungszeit nicht bekannt sein können

```
typedef struct
{ String   Name;              //Obj.-Name, Header
  int      nVrtx;             //Anzahl Objektpkte
  int      nFace;             //Anzahl Flaechen
  float  **Vrtx;              //Objektpunktkoord.
  Polygon *Face;              //Flaechenliste
  float    posMat[DIM*DIM];//Positionsmatrix
} CGFobject;
```

Übung

Übung: Objektcodierung eines Dreieck-Modells

- Reservierung von Speicherplatz für Strukturelemente
- Belegung der Struktur mit Werten
- Erweiterung des Modells um Rückseite



`LoadSetCGF.exe`