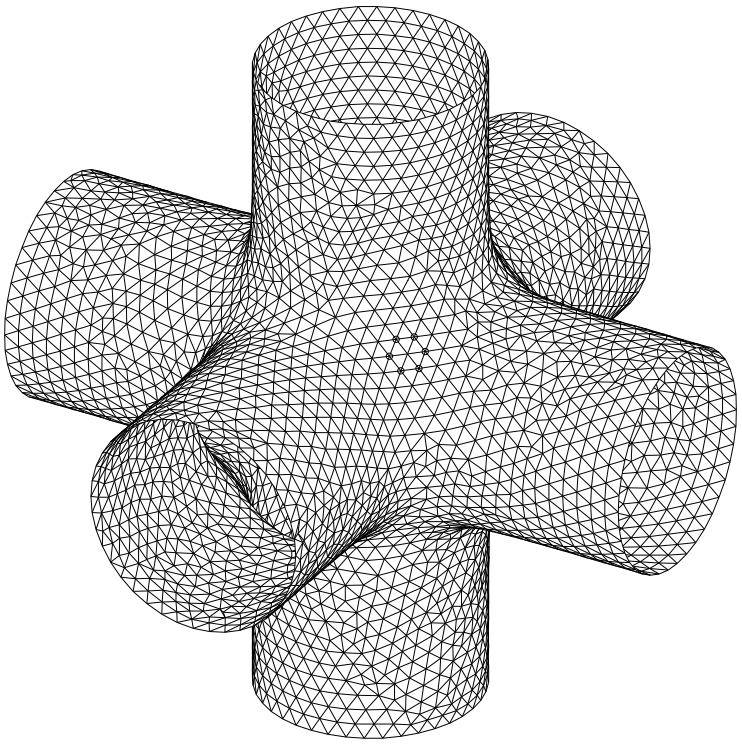
Computerunterstützte Darstellende und Konstruktive Geometrie



Erich Hartmann

Technische Universität Darmstadt
WS 97/98

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Aufgabe der DARSTELLENDEN Geometrie

Das Ziel der Darstellenden Geometrie ist, Bilder von räumlichen Gegenständen wie Häuser, Maschinenteile ... in einer Zeichenebene herzustellen. Dabei verwendet man hauptsächlich zwei Methoden:

I) Parallelprojektion.
Hierbei projiziert man die Objekte (Punkte, Kanten, Kurven,...) mit Hilfe paralleler Strahlen auf eine Ebene (Bildtafel). Steht die Bildtafel senkrecht zu den Projektionsstrahlen, so spricht man von *senkrechter Parallelprojektion* im anderen Fall von *schiefer Parallelprojektion*. Projiziert man schief auf eine horizontale Ebene (z.B. x-y-Ebene), so nennt man diese Art *Vogelperspektive*. Bei einer *Kavaliersperspektive* projiziert man schief auf eine senkrecht stehende Ebene.

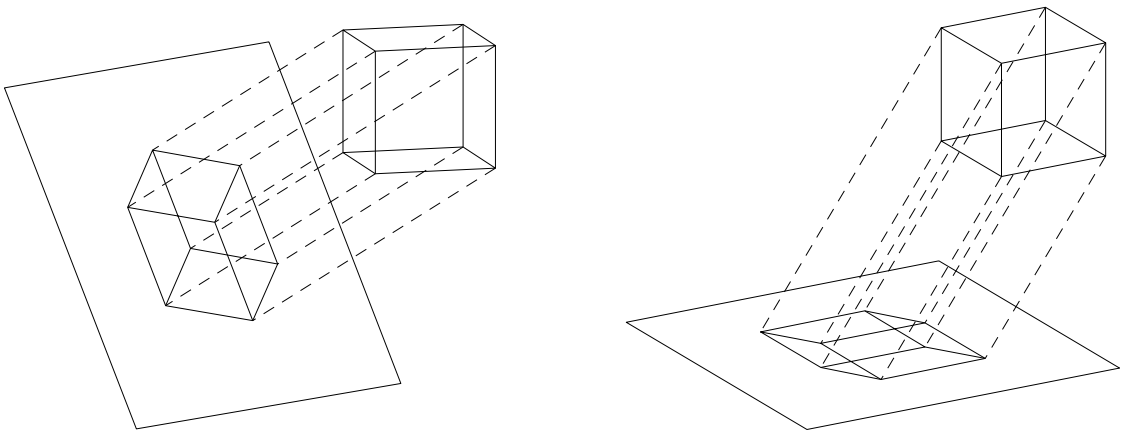


Abbildung 1.1: Senkrechte PARALLELprojektion bzw. Vogelperspektive eines Würfels

II) Zentralprojektion.
Dem Sehen ähnlicher ist die Zentralprojektion. Hier werden die Objekte mit Hilfe von durch einen Punkt Z (das Zentrum oder der Augpunkt) gehende Strahlen zur Abbildung auf einer Bildtafel benutzt.
Die Gestaltungsmöglichkeiten bei Zentralprojektion ist durch die Verwendung der zusätzlichen Pa-

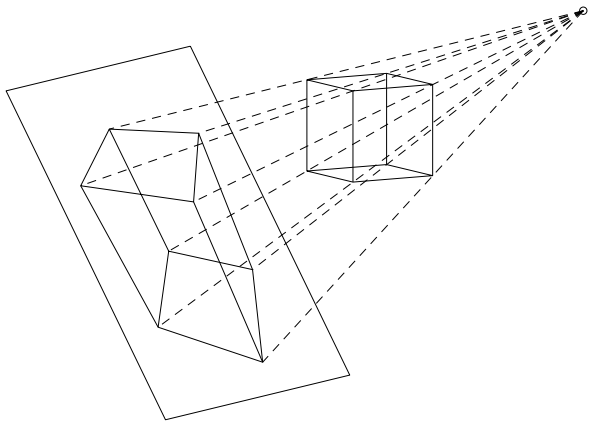


Abbildung 1.2: ZENTRALprojektion eines Würfels

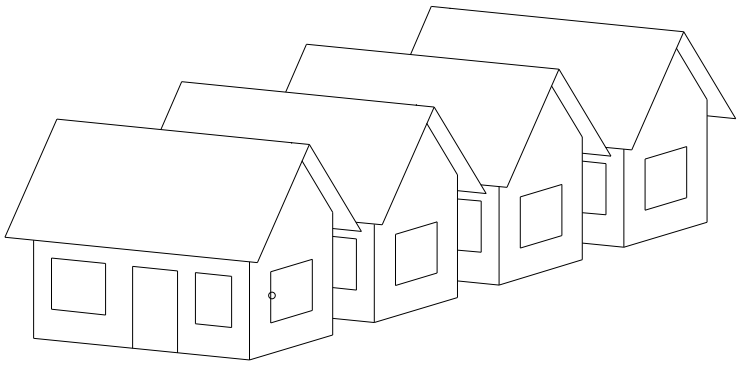


Abbildung 1.3: PARALLELprojektion einer Häuserreihe

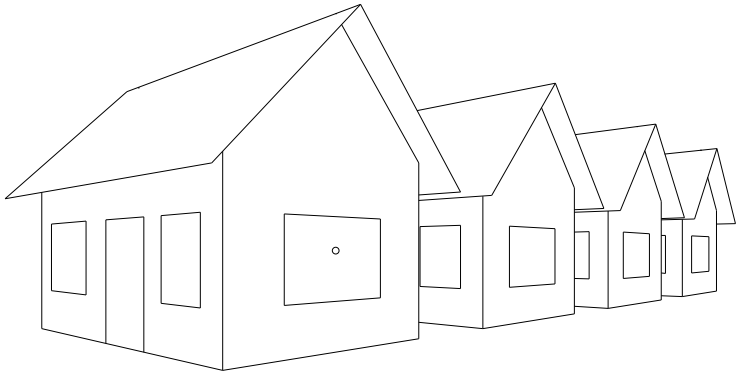


Abbildung 1.4: ZENTRALprojektion einer Häuserreihe

rameter Augpunkt und Distanz (des Augpunktes zur Bildtafel) vielfältiger. Arbeitet man mit Zirkel und Lineal, so ist allerdings der Aufwand zur Erstellung einer Zeichnung auch wesentlich größer. Auch bei Verwendung eines Rechners muß man den Vorteil von "schönen" Bildern durch eine etwas

längere Rechenzeit erkaufen, da eine Zentralprojektion, im Gegensatz zu einer Parallelprojektion, nicht durch eine lineare Abbildung beschrieben werden kann. In der Technik gibt man i.a. der senkrechten Parallelprojektion den Vorzug, da bei Parallelprojektionen Proportionen (Teilverhältnisse) erhalten bleiben.

1.2 Aufgabe der KONSTRUKTIVEN Geometrie

Im Rahmen der konstruktiven Geometrie versucht man Probleme wie das Schneiden von Flächen, Erzeugung von Übergangsflächen, Abwicklung von Flächen, Ausmittelung von Dächern,... zu lösen und die Lösung mit Hilfe der Darstellenden Geometrie informativ darzustellen.

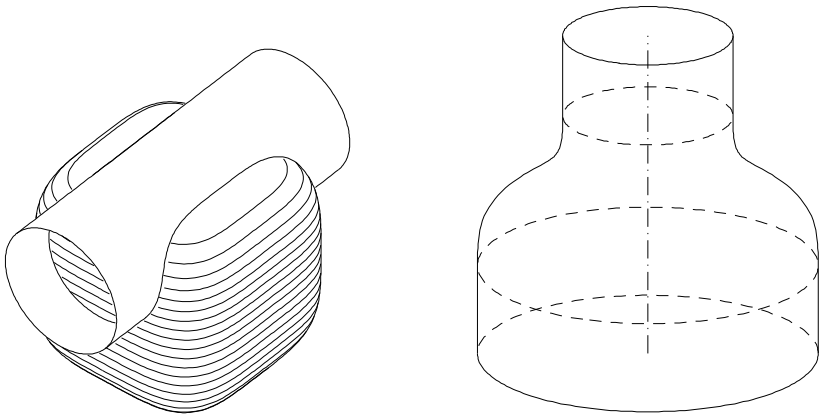


Abbildung 1.5: a) Schnitt zweier Flächen b)Übergangsfläche zwischen zwei Zylindern

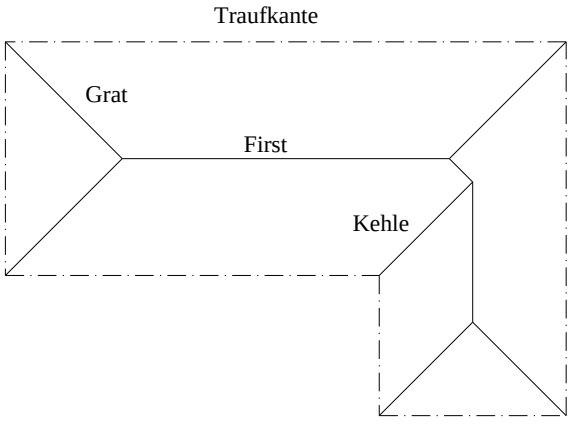


Abbildung 1.6: First, Grat- und Kehllinien eines Daches (Dachausmittelung)

1.3 Über den Inhalt

Im Kapitel **Hilfsmittel** werden zunächst die Anforderungen formuliert, die wir an die zu verwendende Graphik-Software stellen. Es wird die hier benutzte Datenstruktur eingeführt und an einem einfachen Beispiel (N-Eck) ihre Verwendung demonstriert. Die zugrunde liegende Programmiersprache ist PASCAL. Doch lassen sich alle Prozeduren ohne Mühe in andere Sprachen übersetzen. Die übersetzung in C kann sogar "automatisch" mit einer geeigneten Software vorgenommen werden. Ferner enthält das Kapitel viele Grundroutinen aus der analytischen Geometrie. Die zugehörigen PASCALprogramme (auch viele zu den restlichen Kapiteln) sind über das Internet unter

`http://fb04159.mathematik.tu-darmstadt.de/download.html`

zu beziehen.

Kapitel 2

Hilfsmittel

Wichtige Hilfsmittel der Computerunterstützten Darstellenden Geometrie stammen aus der Analytischen Geometrie. Es müssen einfache Operationen, wie Summe von Vektoren, oder Schnitte, wie Schnitt einer Gerade mit einem Kreis, berechnet werden. Hierfür stehen keine Standardbefehle in PASCAL zur Verfügung, sodaß wir zunächst entsprechende Unterprogramme bereitstellen müssen.

2.1 Aufbau eines Zeichenprogramms

Bevor wir auf konkrete Unterprogramme eingehen, werden häufig verwendete globale Konstanten, Typen und Variablen definiert. Sehr wesentlich sind die Typen `vt2d,vt3d,vts2d,vts3d`, die 2- bzw. 3-komponentige Vektoren bzw. Felder von solchen deklarieren.

2.1.1 Globale Konstanten: Datei "geoconst.pas"

Die Datei "geoconst.pas" enthält die Konstante `array_size`, die für die in 2.1.2 erklärten Typen benutzt wird, die Zahlen $\pi, 2\pi, \frac{\pi}{2}$ und eps1, ... , eps8, die bei Abschätzungen nützlich sind. Die Konstanten `black,`... werden zum setzen von Farben (s.u.) verwendet.

```
array_size= 1000;  {...20000 fuer Hiddenline-Alg.}
pi= 3.14159265358;       pi2= 6.2831853;       pih= 1.5707963;
eps1=0.1;      eps2=0.01;       eps3=0.001;     eps4=0.0001;
eps5=0.00001;  eps6=0.000001;   eps7=0.0000001; eps8=0.00000001;
default=-1; black=0; blue=1; green=2; cyan=3; red=4; magenta=5; brown=6;
lightgray=7; darkgray=8; lightblue=9; lightgreen=10; lightcyan=11;
lightred=12; lightmagenta=13; yellow=14; white=15;
```

2.1.2 Globale Typen: Datei "geotype.pas"

```
r_array = array[0..array_size] of real;
i_array = array[0..array_size] of integer;
b_array = array[0..array_size] of boolean;
vt2d    = record x,y: real; end;
vt3d    = record x,y,z: real; end;
vts2d   = array[0..array_size] of vt2d;
vts3d   = array[0..array_size] of vt3d;
matrix3d= array[1..3,1..3] of real;
```

2.1.3 Globale Variablen: Datei "geovar.pas"

```
   null2d:vt2d;  null3d:vt3d;    {Nullvektoren}
{**fuer area_2d and curve2d:}
   origin2d:vt2d;
{**fuer Parallel- und Zentral-Projektion:}
   u_angle,v_angle,              {Projektionswinkel}
   rad_u,rad_v,                  {rad(u), rad(v)}
   sin_u,cos_u,sin_v,cos_v:real; {sin-,cos- Werte von u, v}
   e1vt,e2vt,n0vt:vt3d;          {Basis-Vektoren und}
                                 {Normalen-Vektor der Bildebene}
{**fuer Zentral-Projektion:}
   mainpt,                       {Hauptpunkt}
   centre:vt3d;                  {Zentrum}
   distance:real;                {Distanz Hauptpunkt-Zentrum}
```

2.1.4 Anforderung an die Graphik-Software

Graphik-Software bietet heute sehr viel Komfort. Doch hängt dieser Komfort stark von der verwendeten Software ab. Um die hier angegebenen Programme leicht auf den verschiedensten Systemen zum Laufen zu bringen , wollen wir uns nur auf die folgenden 9 rechnerabhängige Befehle stützen.

1. **graph_on(ipl),** `ipl:integer`, ruft die Grafik-Software und belegt die Vektoren `null2d, null3d` mit Nullen;
   `ipl` $= 0$: Ausgabe nur auf dem Bildschirm,
   `ipl` $\neq 0$: es wird nach jedem Aufruf von `draw_area(...)` (s.u.) eine Datei `name.pld` angelegt, in die alle Zeichenbefehle der aktuellen Zeichnung incl. Farben und Linienstärke geschrieben werden. Mit dem Programm `pldview` der Diskette läßt sich dann die Zeichnung noch einmal auf den Bildschirm schicken oder eine *POSTSCRIPT*–Datei herstellen, die man anschließend zu einem Drucker schicken oder in TEX–Dokumente einbinden kann.

2. **draw_area(width,height,x0,y0,scalefactor)** löscht den Bildschirm und legt eine Zeichenfläche mit einem rechtwinkligen Koordinatensystem fest.
   `origin2d = (x0,y0)` ist eine globale Variable.
   Alle Längen werden in mm angegeben (real-Zahlen) !
   1 mm soll auch auf dem Bildschirm als 1 mm erscheinen, falls scalefactor=1 gesetzt wird.
   Falls eine **Skalierung** (Streckung am Koordinaten-Nullpunkt der Bildtafel) gewünscht wird, so muß scalefactor entsprechend gewählt werden.

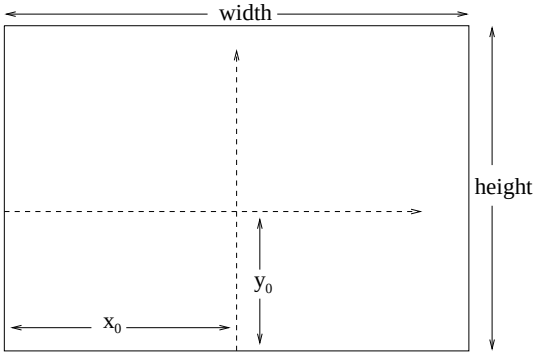


Abbildung 2.1: Koordinatenursprung in der Zeichenfläche

3. **draw_end** schließt die Zeichnung ab. Falls eine neue begonnen werden soll, muß zuerst wieder `draw_area` aufgerufen werden.

4. **graph_off** verabschiedet die Zeichensoftware endgültig.

ZEICHENBEFEHLE:

5. **pointc2d(x,y,style),** `x,y:real; style:integer`,
markiert den Punkt (x,y) durch o falls `style` = 0, + falls `style` = 1 , ...
Für `style` = 10 oder 50 oder 100 erhält man kleinere ausgefüllte Kreise zur Markierung von Punkten.
point2d(p,style), `p:vt2d; style:integer`,
wie `pointc2d`, nur mit Hilfe des Typs `vt2d` des Punktes.

6. **linec2d(x1,y1,x2,y2,style),** `x1,y1,x2,y2:real; style:integer`,
zeichnet die Strecke (x1,y1)(x2,y2) und zwar so:
———— , falls style = 0, − − − − −, falls style = 1, − · − · −, falls style = 2, ...
Die folgenden Befehle können mit Hilfe von `linec2d` definiert werden.

   (a) **line2d(p1,p2,style),** `p1,p2:vt2d; style:integer`,
   wie `linec2d`, nur unter Verwendung des Typs `vt2d` für Anfangs- und Endpunkt.

   (b) **arrowc2d(x1,y1,x2,y2,style),** `x1,y1,x2,y2:real; style:integer`
   zeichnet einen Pfeil von (x1,y1) nach (x2,y2).

   (c) **arrow2d(p1,p2,style),** `p1,p2:vt2d; style:integer`
   zeichnet einen Pfeil von $\mathbf{p}_1$ nach $\mathbf{p}_2$.

   (d) **curve2d(p,n1,n2,style),** `p:vts2d; style:integer`,
   zeichnet den Polygonzug durch die Punkte $\mathbf{p}_{n1}, ..., \mathbf{p}_{n2}$.
   Für die Darstellung von Kurven im Rahmen von Hiddenline-Algorithmen ist das folgende um ein Feld vom Typ `b_array` erweiterte `curve2d`-Programm von Nutzen:

   (e) **curve2d_vis(p,n1,n2,style,visible),**
   `p:vts2d; n1,n2,style:integer; visible:b_array;`
   verbindet benachbarte "visible" Punkte gemäß `style` . Für style=10 werden benachbarte "visible" Punkte durch ————- und benachbarte "not visible" Punkte durch − − −− verbunden.

   Alle Längen und Koordinaten von Vektoren sind in **mm** (Millimeter) anzugeben!

7. **new_color(color)**, `color: integer`,
setzt eine neue Farbe. Dabei werden die in TURBO-Pascal üblichen Integercodes benutzt. Z.B.: `color = red`. `color = default` setzt die Standardfarbe weiß (TURBO-P.) oder schwarz (LINUX-P.).

8. **new_linewidth(factor)**, `factor: real`,
setzt eine neue Linienstärke. `factor`=1 bedeutet normale Linienstärke.

   Für das lesen einer Integer-Datei verwenden wir

9. **read_integer_file(file_name,n_dat, int_var)**,
`file_name:string; n_dat:integer, int_var:i_array`,
liest `n_dat` Integer aus der Datei `file_name` in die Variable `int_var`.

Die rechnerabhängigen Befehle sind für den Fall von TURBO-Pascal in der Datei `geoproc.pas` der Diskette enthalten.

AUFBAU eines ZEICHENPROGRAMMS:

Packt man alle globalen Konstanten, Typen, Variablen und Prozeduren in ein *unit* `geograph`, so hat ein Zeichenprogramm die einfache Gestalt:

```
program name;
uses geograph;
const  ...
type   ...
var    ...:vts2d;
       ...:integer;
       ...:real;
       ...
      {$i procs.pas}       {weitere Prozeduren}
{******************}
 begin  {Hauptprogramm}
 graph_on(...);
 ...
{Zeichnen:}
 draw_area(...);
 ...
 ...
 draw_end;
 ....
 graph_off;
end.
```

2.2 Funktionen auf $\mathbb{R}$, Operationen mit Vektoren

Im Folgenden werden PASCAL-Funktionen bzw. Prozeduren für einige reelle Funktionen und Operationen mit Vektoren zusammengestellt. Da ihre Realisierungen einfach sind, geben wir hier nur ihre Prozedurköpfe an. Die Prozedur-Texte sind (auf der Diskette) in der Datei `proc_ag.pas` enthalten.

2.2.1 Funktionen auf $\mathbb{R}$

1. $r \to \text{sign}(r)$ (Vorzeichen von r)
   `function sign(a:real):integer;`

2. $a, b \to \max\{a, b\}$ (Maximum von a,b)
   $a, b \to \min\{a, b\}$ (Minimum von a,b)
   `function max(a,b:real):real;  function min(a,b:real):real;`

3. $x \to \cosh x$ (Kosinushyperbolikus)
   $x \to \sinh x$ (Sinushyperbolikus)
   `function cosh(x:real):real;  function sinh(x:real):real;`

2.2.2 Operationen mit Vektoren

1. $x, y \to \mathbf{v} = (x, y)$,
   $x, y, z \to \mathbf{v} = (x, y, z)$
   ```
   procedure put2d(x,y:real; var v:vt2d);
   procedure put3d(x,y,z:real; var v:vt3d);
   ```
   $\mathbf{v} = (x, y, z) \to x, y, z$
   ```
   procedure get3d(v:vt3d; var x,y,z:real);
   ```

2. $r, \mathbf{v} \to r\mathbf{v}$ (Skalierung)
   ```
   procedure scale2d(r:real; v:vt2d; var vs:vt2d);
   procedure scale3d(r:real; v:vt3d; var vs:vt3d);
   ```
   $r_1, r_2, (x, y) \to (r_1 x, r_2 y)$ bzw.
   $r_1, r_2, r_3, (x, y, z) \to (r_1 x, r_2 y, r_3 z)$ (Skalierung der Koordinaten)
   ```
   procedure scaleco2d(r1,r2:real; v:vt2d; var vs:vt2d);
   procedure scaleco3d(r1,r2,r3:real; v:vt3d; var vs:vt3d);
   ```

3. $\mathbf{v}_1, \mathbf{v}_2 \to \mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$ (Summe zweier Vektoren)
   ```
   procedure sum2d(v1,v2:vt2d; var vs:vt2d);
   procedure sum3d(v1,v2:vt3d; var vs:vt3d);
   ```
   $\mathbf{v}_1, \mathbf{v}_2 \to \mathbf{v} = \mathbf{v}_2 - \mathbf{v}_2$ (Differenz zweier Vektoren)
   ```
   procedure diff2d(v1,v2:vt2d; var vd:vt2d);
   procedure diff3d(v1,v2:vt3d; var vd:vt3d);
   ```

4. $r_1, \mathbf{v}_1, r_2, \mathbf{v}_2 \to \mathbf{v} = r_1\mathbf{v}_1 + r_2\mathbf{v}_2$ (Linearkombination von Vektoren)
   ```
   procedure lcomb2vt2d(r1:real; v1:vt2d; r2:real; v2:vt2d; var vlc:vt2d);
   procedure lcomb2vt3d(r1:real; v1:vt3d; r2:real; v2:vt3d; var vlc:vt3d);
   ```
   und analog Linearkombinationen von 3 bzw. 4 Vektoren:
   ```
   lcomb3vt2d(r1,v1, r2,v2, r3,v3, vlc);
   lcomb3vt3d(r1,v1, r2,v2, r3,v3, vlc);
   lcomb4vt2d(r1,v1, r2,v2, r3,v3, r4,v4, vlc);
   lcomb4vt3d(r1,v1, r2,v2, r3,v3, r4,v4, vlc);
   ```

5. $\mathbf{v} = (x, y) \to |x| + |y|$ bzw. $\mathbf{v} = (x, y, z) \to |x| + |y| + |z|$
   ```
   function abs2d(v:vt2d):real;     function abs3d(v:vt3d):real;
   ```

6. $\mathbf{v} = (x, y) \to \|\mathbf{v}\| = \sqrt{x^2 + y^2}$ bzw.
   $\mathbf{v} = (x, y, z) \to \|\mathbf{v}\| = \sqrt{x^2 + y^2 + z^2}$
   ```
   function length2d(v:vt2d):real;     function length3d(v:vt3d):real;
   ```

7. $\mathbf{v} \to \mathbf{v}/\|\mathbf{v}\|$
   ```
   procedure normalize2d(var v:vt2d);  procedure normalize3d(var v:vt3d);
   ```

8. $\mathbf{p}, \mathbf{q} \to \|\mathbf{p} - \mathbf{q}\|$ $\mathbf{p}, \mathbf{q} \to \|\mathbf{p} - \mathbf{q}\|^2$
   ```
   function distance2d(p,q:vt2d):real;
   function distance3d(p,q:vt3d):real;
   function distance2d_square(p,q:vt2d):real;
   function distance3d_square(p,q:vt3d):real;
   ```

9. $\mathbf{v}_1, \mathbf{v}_2 \to \mathbf{v}_1 \cdot \mathbf{v}_2$ (Skalarprodukt)
   ```
   function scalarp2d(v1,v2:vt2d):real;
   function scalarp3d(v1,v2:vt3d):real;
   ```

10. $\mathbf{v}_1, \mathbf{v}_2 \to \mathbf{v}_1 \times \mathbf{v}_2$ (Vektorprodukt)
    `procedure vectorp(v1,v2:vt3d; var vp:vt3d);`

11. $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \to |\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3|$
    (Spatprodukt $\mathbf{v}_1 \cdot (\mathbf{v}_2 \times \mathbf{v}_3)$, 3x3-Determinante)
    `function determ3d(v1,v2,v3:vt3d):real;`

12. $\cos\varphi, \sin\varphi, \mathbf{p} = (x, y) \to \mathbf{p}_r = (x\cos\varphi - y\sin\varphi, x\sin\varphi + y\cos\varphi)$
    (Rotation um den Nullpunkt, Drehwinkel:φ)
    `procedure rotor2d(cos_rota,sin_rota:real; p:vt2d; var pr:vt2d);`

13. $\cos\varphi, \sin\varphi, \mathbf{p}_0, \mathbf{p} \to \mathbf{p}_r$
    (Rotation um den Punkt $\mathbf{p}_0$, Drehwinkel:φ)
    `rotp02d(cos_rota,sin_rota,p0,p, pr);`

14. $\cos\varphi, \sin\varphi, \mathbf{p} \to \mathbf{p}_r$
    (Rotation um x-Achse bzw. y-Achse, z-Achse)
    ```
    procedure rotorx(cos_rota,sin_rota,p, pr);
    procedure rotory(cos_rota,sin_rota,p, pr);
    procedure rotorz(cos_rota,sin_rota,p, pr);
    ```

15. $\cos\varphi, \sin\varphi, \mathbf{p}_0, \mathbf{p} \to \mathbf{p}_r$
    (Rotation um eine zu einer Koordinatenachse parallele Achse durch $\mathbf{p}_0$ im $\mathbb{R}^3$)
    ```
    procedure rotp0x(cos_rota,sin_rota,p0,p, pr);
    procedure rotp0y(cos_rota,sin_rota,p0,p, pr);
    procedure rotp0z(cos_rota,sin_rota,p0,p, pr);
    ```

16. Vertauschen von Zahlen bzw. Vektoren:
    $a \leftrightarrow b$ bzw. $\mathbf{v}_1 \leftrightarrow \mathbf{v}_2$
    ```
    procedure change1d(var a,b:real);
    procedure change2d(var v1,v2:vt2d);
    procedure change3d(var v1,v2:vt3d);
    ```

Beispiel 2.1 *Das folgende Programm zeichnet ein regelmäßiges* **n-Eck** *und, auf Wunsch, mit allen möglichen Kanten. Die Punkte des n-Ecks liegen auf einem Kreis. Der Mittelpunkt des Kreises sei der Punkt (0,0), der Radius sei r. Ist (r,0) der "0-te" Punkt des n-Ecks, so hat der i-te Punkt die Koordinaten*

$$x_i = r\cos(i\Delta\varphi), \quad y_i = r\sin(i\Delta\varphi) \quad \text{mit} \quad \Delta\varphi = 2\pi/n, \quad i = 0, ..n-1.$$

Im Programm wird der Punkt P_{i+1} durch Rotation des Punktes P_i mit `rotor2d` *berechnet.*

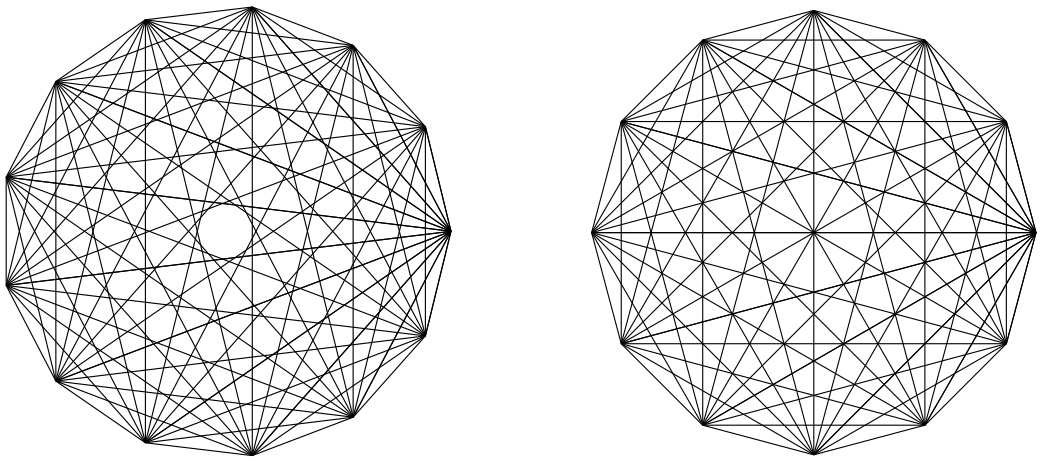


Abbildung 2.2: N-Eck mit allen Diagonalen (Beispiel 2.1)

```
{*****************************}
{***  Regelmaessiges n-Eck  ***}
{*****************************}
program n_eck;
uses geograph;
var  p : vts2d;
     n,iverb,i,j,inz: integer;
     r,dw,cdw,sdw: real;
{******************}
 begin  {Hauptprogramm}
   graph_on(0);
   repeat
     writeln('***  n-Eck  ***');
     writeln('n ?  Radius r des zugeh"origen Kreises ?');         readln(n,r);
     writeln('Jeden Punkt mit jedem Punkt verbinden ? (Ja=1)');   readln(iverb);
{Berechnung der Eckpunkte:}
     put2d(r,0, p[0]);  dw:= pi2/n;  cdw:= cos(dw);  sdw:= sin(dw);
     for i:= 0 to n-1 do  rotor2d(cdw,sdw,p[i], p[i+1]);
     draw_area(2*r+20,2*r+20,r+10,r+10,1);
{Zeichnen:}   new_color(yellow);
     if iverb=1 then
        for i:= 0 to n-1 do
            for j:= i+1 to n do
                line2d(p[i],p[j],0)
       else
        curve2d(p,0,n,0);
     draw_end;
     writeln('Noch eine Zeichnung? (ja:1, nein:0)');  readln(inz);
   until inz=0;
   graph_off;
 end.
```

Aufgabe 2.1 *Schreibe ein Programm, das die folgenden Bilder erzeugt.*

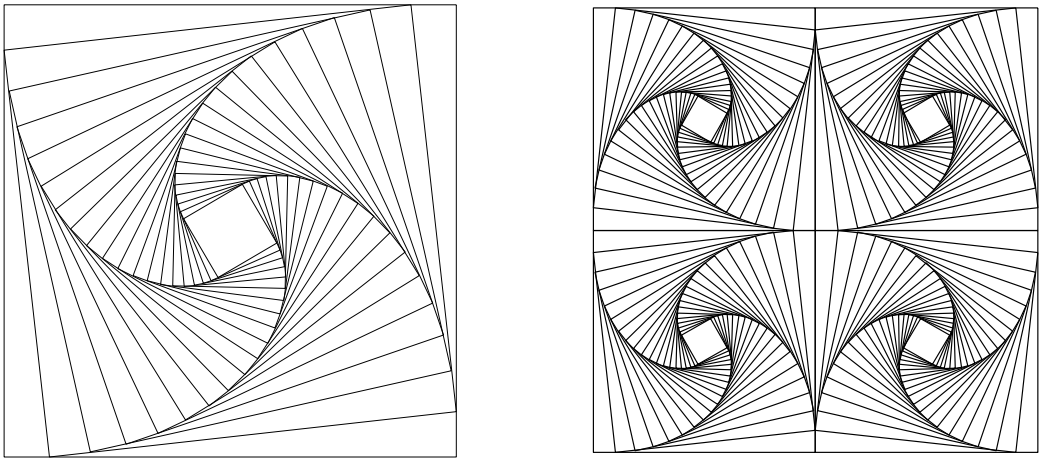


Abbildung 2.3: zur Aufgabe 2.1

2.3 Programme zur analytischen Geometrie

2.3.1 Polarwinkel und quadratische Gleichung

a) Bei der Umrechnung von rechtwinkligen Koordinaten im $\mathbb{R}^2$ in Polarkoordinaten verwenden wir die folgende Funktion `polar_angle`, die dem Punkt `(x,y)` den zugehörigen Polarwinkel zuordnet:
`function polar_angle(x,y:real):real;`

b) Reelle Lösungen einer quadratischen Gleichung $ax^2 + bx + c = 0$:
(Die Lösungen sind der Größe nach geordnet. `ns` ist die Anzahl der reellen Lösungen)
`procedure equation_degree2(a,b,c:real; var x1,x2:real; var ns:integer);`

Die Texte dieser und der folgenden Prozeduren befinden sich auf der Diskette in der Datei `proc_ag.pas`.

2.3.2 Schnitt Gerade-Gerade, Kreis-Gerade, Kreis-Kreis

a) Schnitt Gerade-Gerade :
Das Unterprogramm `is_line_line` verwendet die CRAMERsche Regel um den Schnittpunkt zweier Geraden zu bestimmen.

```
procedure is_line_line(a1,b1,c1, a2,b2,c2:real; var xs,ys:real; var nis:integer);
{Schnittpunkt (xs,ys) (nis=1) der Geraden a1*x+b1*y=c1, a2*x+b2*y=c2.
 Falls die Geraden parallel sind ist nis<>1.}
```

b) Schnitt Kreis-Gerade:
Kreis : $(x - x_m)^2 + (y - y_m)^2 = r^2, \quad r > 0.$
Gerade : $ax + by = c, \quad (a, b) \neq (0, 0)$
Die Substitution $\xi = x - x_m,\ \eta = y - y_m$ führt auf
$a\xi + b\eta = c' \quad$ mit $\quad c' = c - ax_m - by_m \quad$ und
$\xi^2 + \eta^2 = r^2$.
Falls $r^2(a^2 + b^2) - c'^2 > 0$ ist, erhält man die Lösungen
$\xi_{1/2} = (ac' \pm b\sqrt{r^2(a^2 + b^2) - c'^2}/(a^2 + b^2)\ , \qquad \eta_{1/2} = (bc' \mp a\sqrt{r^2(a^2 + b^2) - c'^2}/(a^2 + b^2)$
und damit
$x_{1/2} = x_m + \xi_{1/2}, \qquad y_{1/2} = y_m + \eta_{1/2}.$

```
procedure is_circle_line(xm,ym,r, a,b,c:real; var x1,y1,x2,y2:real; var nis:integer);
{Schnitt Kreis-Gerade: sqr(x-xm)+sqr(y-ym)=r*r, a*x+b*y=c,
 Schnittpkte: (x1,y1),(x2,y2). Es ist x1<=x2, nis Anzahl der Schnittpunkte.}
```

Da wir sehr oft den Schnitt des Einheitskreises ($x^2 + y^2 = 1$) mit einer Gerade berechnen müssen, spezialisieren wir die vorstehende Prozedur für diesen Fall:

```
procedure is_unitcircle_line(a,b,c:real; var x1,y1,x2,y2:real; var nis:integer);
```

Man beachte, dass in beiden Prozeduren $x_1 \leq x_2$ gilt.

c) Schnitt Kreis-Kreis :
1. Kreis : $(x - x_1)^2 + (y - y_1)^2 = r_1^2, \quad r_1 > 0,$
2. Kreis : $(x - x_2)^2 + (y - y_2)^2 = r_2^2, \quad r_2 > 0, \quad (x_1, y_1) \neq (x_2, y_2).$
Dieses Gleichungssystem ist zu dem folgenden äquivalent:
$(x - x_1)^2 + (y - y_1)^2 = r_1^2, \quad ax + by = c \quad$ mit
$a = 2(x_2 - x_1), \quad b = 2(y_2 - y_1) \quad$ und $\quad c = r_2^2 - x_1^2 - y_1^2 - r_2^2 + x_2^2 + y_2^2.$
D.h. die Schnittpunkte der beiden Kreise sind identisch mit den Schnittpunkten des 1. Kreises und der Geraden $ax + by = c$.

```
procedure is_circle_circle(xm1,ym1,r1,xm2,ym2,r2:real; var x1,y1,x2,y2:real; var nis:integer);
{Schnitt Kreis-Kreis. Es ist x1<=x2. nis = Anzahl der Schnittpunkte.}
```

2.3.3 Gleichung einer Ebene

Gegeben: 3 Punkte $P_i : \mathbf{p}_i, \quad i = 1, 2, 3.$
Gesucht: Gleichung $\mathbf{n} \cdot x = d, \quad$ d.h. Normalenvektor $\mathbf{n}$ und d.
Lösung : $\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1) \quad$ und $\quad d = \mathbf{n} \cdot \mathbf{p}_1.$
Das folgende Unterprogramm `plane_equ` berechnet $\mathbf{n}$ und d. Es setzt die boolsche Variable `error` auf `true`, falls $\mathbf{n} \approx \mathbf{0}$.

```
procedure plane_equ(p1,p2,p3:vt3d; var nv:vt3d; var d:real; var error:boolean);
{Berechnet die Gleichung nv*x=d der Ebene durch die Punkte p1,p2,p3.
 error=true: die Punkte spannen keine Ebene auf. }
```

2.3.4 Schnitt Gerade-Ebene

Gegeben: Gerade $\mathbf{x}(t) = \mathbf{p} + t\mathbf{r}, \quad$ Ebene $\mathbf{n} \cdot \mathbf{x} = d.$
Gesucht: Schnittpunkt $\mathbf{p}_{is}$ der Gerade mit der Ebene.
Lösung: $\mathbf{p}_{is} = \mathbf{p} - ((\mathbf{n} \cdot \mathbf{p} - d)/\mathbf{n} \cdot \mathbf{r})\mathbf{r}.$
Das Unterprogramm `is_line_plane` berechnet den Schnittpunkt, falls er existiert.

```
procedure is_line_plane(p,rv,nv:vt3d; d:real; var pis:vt3d; var nis:integer);
{Schnitt Gerade-Ebene. Gerade: Punkt p, Richtung r. Ebene:  nv*x = d .
 nis=0: kein Schnitt ,nis=1: Schnittpunkt, nis=2: Gerade liegt in der Ebene.}
```

2.3.5 Schnitt dreier Ebenen

Gegeben: Drei Ebenen $\varepsilon_i : \quad \mathbf{n}_i \cdot \mathbf{x} = d_i, \quad i = 1, 2, 3, \quad \mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3$ linear unabhängig.
Gesucht: Schnittpunkt $\mathbf{p}_{is} : \varepsilon_1 \cap \varepsilon_2 \cap \varepsilon_3.$
Der Ansatz $\mathbf{p}_{is} = \xi(\mathbf{n}_2 \times \mathbf{n}_3) + \eta(\mathbf{n}_3 \times \mathbf{n}_1) + \zeta(\mathbf{n}_1 \times \mathbf{n}_2)$
führt auf die Lösung
$\mathbf{p}_{is} = (d_1(\mathbf{n}_2 \times \mathbf{n}_3) + d_2(\mathbf{n}_3 \times \mathbf{n}_1) + d_3(\mathbf{n}_1 \times \mathbf{n}_2))/\mathbf{n}_1 \cdot (\mathbf{n}_2 \times \mathbf{n}_3).$
(Falls die Normalen nicht linear unabhängig sind, existiert eine Schnittgerade oder zwei Ebenen sind parallel.)

Das Unterprogramm `is_3_planes` berechnet den Schnittpunkt. Es liefert `error= true`, falls der Schnitt nicht aus einem Punkt besteht.

```
procedure is_3_planes(nv1:vt3d; d1:real; nv2:vt3d; d2:real; nv3:vt3d; d3:real;
                                                  var pis:vt3d; var error:boolean);
{Schnitt der Ebenen nv1*x=d1, nv2*x=d2, nv3*x=d3.
 error= true: Schnitt besteht nicht aus einem Punkt.}
```

2.3.6 Schnitt zweier Ebenen

Gegeben: Zwei Ebenen $\varepsilon_i: \quad \mathbf{n}_i \cdot \mathbf{x} = d_i, \quad i = 1, 2, \quad \mathbf{n}_1, \mathbf{n}_2$ linear unabhängig.
Gresucht: $\varepsilon_1 \cap \varepsilon_2 : \mathbf{x} = \mathbf{p} + t\mathbf{r}$.
Die Richtung der Schnittgerade ist $\mathbf{r} = \mathbf{n}_1 \times \mathbf{n}_2$. Einen Punkt $P : \mathbf{p}$ der Schnittgerade erhält man, indem man die Ebenen $\varepsilon_1, \varepsilon_2$ mit der Ebene $\varepsilon_3 : \mathbf{x} = s_1\mathbf{n}_1 + s_2\mathbf{n}_2$ schneidet. s_1 und s_2 ergeben sich durch Einsetzen in die Gleichungen der Ebenen ε_1 und ε_2.

$$P: \ \mathbf{p} = \frac{d_1{\mathbf{n}_2}^2 - d_2(\mathbf{n}_1 \cdot \mathbf{n}_2)}{{\mathbf{n}_1}^2{\mathbf{n}_2}^2 - (\mathbf{n}_1 \cdot \mathbf{n}_2)^2}\mathbf{n}_1 + \frac{d_2{\mathbf{n}_1}^2 - d_1(\mathbf{n}_1 \cdot \mathbf{n}_2)}{{\mathbf{n}_1}^2{\mathbf{n}_2}^2 - (\mathbf{n}_1 \cdot \mathbf{n}_2)^2}\mathbf{n}_2$$

Das Unterprogramm `is_plane_plane` berechnet den Richtungsvektor $\mathbf{r}$ und einen Punkt P der Schnittgerade. Es liefert error= true, falls die Ebenen parallel sind.

```
procedure is_plane_plane(nv1:vt3d; d1:real; nv2:vt3d; d2:real;
                              var p,rv:vt3d; var error:boolean);
{Schnitt der Ebenen nv1*x=d1, nv2*x=d2. Schnittgerade: x = p + t*rv .
 error= true: Schnitt besteht nicht aus einer Gerade.}
```

2.3.7 Punkt "vor" einer Ebene

Im Zusammenhang mit Hiddenline-Algorithmen muß oft beurteilt werden, auf welcher Seite einer Ebene ein Punkt liegt.
Gegeben: Punkt $P : \mathbf{p}$, Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$.
P liegt "vor" der Ebene ε, wenn $\mathbf{n} \cdot \mathbf{p} - d > 0$ ist. Dies entscheidet die folgende boolsche Funktion `pt_before_plane`.

```
function pt_before_plane(p,nv: vt3d; d: real) : boolean;
{...stellt fest, ob der Punkt p "vor" der Ebene  nv*x-d=0 liegt. }
```

2.3.8 ξ-η-Koordinaten eines Punktes in einer Ebene

Gegeben: Ebene $\varepsilon : \mathbf{x} = \mathbf{p}_0 + \xi\mathbf{v}_1 + \eta\mathbf{v}_2$ und Punkt P: $\mathbf{p}$ in ε .
Gesucht: ξ, η so, daß $\mathbf{p} = \mathbf{p}_0 + \xi\mathbf{v}_1 + \eta\mathbf{v}_2$ ist.
Durch skalare Multiplikation des Ansatzes für $\mathbf{p}$ mit den Vektoren $\mathbf{v}_1, \mathbf{v}_2$ erhält man das lineare Gleichungssystem
$(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{v}_1 = \xi{\mathbf{v}_1}^2 + \eta\mathbf{v}_1 \cdot \mathbf{v}_2, \qquad (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{v}_2 = \xi\mathbf{v}_1 \cdot \mathbf{v}_2 + \eta{\mathbf{v}_2}^2,$
Die Prozedur `ptco_plane3d` berechnet ξ, η mit Hilfe der CRAMERschen Regel. Es setzt `error=true`, falls die Determinante des Gleichungssystems ≈ 0 ist. (Liegt der Punkt P nicht in ε und ist ξ, η die Lösung des obigen Gleichungssystems, so ist P': $\mathbf{p}' = \mathbf{p}_0 + \xi\mathbf{v}_1 + \eta\mathbf{v}_2$ der Fußpunkt des Lotes von P auf die Ebene ε.)

```
procedure ptco_plane3d(p0,v1,v2,p:vt3d; var xi,eta:real; var error:boolean);
{v1,v2 sind linear unabhaengig, p-p0 linear abhaengig von v1,v2.
 Es werden Zahlen  xi,eta berechnet mit  p = p0 + xi*v1 + eta*v2.}
```

2.3.9 Punkte auf einer Gerade

Für Hiddenline-Algorithmen sind die folgenden Unterprogramme von Nutzen: `line_pts2d` und `line_pts3d` bestimmen zur Strecke $P1, P2$ Punkte $P_0, ..., P_{n2}$ auf dieser Strecke mit $P_0 = P1, P_{n2} = P2$.

```
procedure line_pts2d(p1,p2:vt2d; var p:vts2d; var n2:integer);
{Berechnet n2 Punkte der Strecke p1,p2 im Abstand 2/scalefactor.}

procedure line_pts3d(p1,p2:vt3d; var p:vts3d; var n2:integer);
```

Kapitel 3

PARALLELPROJEKTION

In der klassischen Darstellenden Geometrie unterscheidet man zwei Arten von Parallelprojektionen:
a) senkrechte Parallelprojektion b) schiefe Parallelprojektion,
je nachdem, ob die Projektionsstrahlen senkrecht oder schief (nicht senkrecht) zur Bildtafel stehen. Zwar liefern schiefe Parallelprojektionen nicht so gute Bilder wie senkrechte Parallelprojektionen, aber in Form der Kavalier- und Vogelperspektiven lassen sich in vielen Situationen schnell anschauliche Bilder erstellen. Einem Rechner ist es allerdings gleichgültig, ob er eine senkrechte oder schiefe Projektion berechnet. Deshalb werden wir hier fast nur senkrechte Parallelprojektionen behandeln. Im Kapitel 3.4 werden wir aber kurz auf die Vogelperspektive eingehen und an Beispielen demonstrieren.

3.1 Senkrechte Parallelprojektion

3.1.1 Die Projektionsformeln

Um die Parallelprojektion rechnerisch erfassen zu können, führen wir im Raum ein zur Beschreibung des abzubildenden Gegenstandes geeignetes rechtwinkliges Koordinatensystem $(O; x, y, z)$ ein. Die Ebene (Bildtafel), auf die senkrecht projiziert werden soll, nennen wir ε_0. Da eine Verschiebung der Bildtafel an dem Bild des Gegenstandes (außer seine Lage) nichts ändert, können wir annehmen, daß ε_0 den Nullpunkt O des Koordinatensystems enthält. Die Lage der Ebene ε_0 und damit die senkrechte Parallelprojektion ist durch die Angabe eines Normalenvektors $\mathbf{n}_0$ von ε_0 eindeutig bestimmt. Wir wählen den Vektor $\mathbf{n}_0$ so, daß er die Länge 1 ($|\mathbf{n}_0| = 1$) hat und der Projektionsrichtung entgegengesetzt ist ($\mathbf{n}_0$ zeigt zur "Sonne"). Beschreibt man $\mathbf{n}_0$ durch seine Kugelkoordinatenwinkel u, v (u ist die "geographische Länge", v die "geographische Breite"), so gilt:

$$\mathbf{n}_0 = (\cos u \cos v, \sin u \cos v, \sin v), \quad 0 \le u \le 2\pi,\ -\pi/2 \le v \le \pi/2.$$

Zur Beschreibung der Bildpunkte verwenden wir ein rechtwinkliges Koordinatensystem $(O; x_e, y_e)$ in der Bildtafel ε_0, dessen Nullpunkt O mit O übereinstimmt und dessen y_e-Achse im Falle $|v| < \pi/2$ das Bild der z-Achse ist. Die x_e-Achse liegt dann in der Schnittgerade von ε_0 mit der x-y-Ebene. Die Vektoren

$$\mathbf{e}_1 = (-\sin u, \cos u, 0), \qquad \mathbf{e}_2 = (-\cos u \sin v, -\sin u \sin v, \cos v)$$

bilden eine Orthonormalbasis in ε_0 und $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0\}$ ist eine Orthonormalbasis des $\mathbb{R}^3$.
Um die Bildkoordinaten (x_e, y_e) eines Punktes $Q : \mathbf{q} = (x, y, z)$ zu erhalten, muß man also nur die

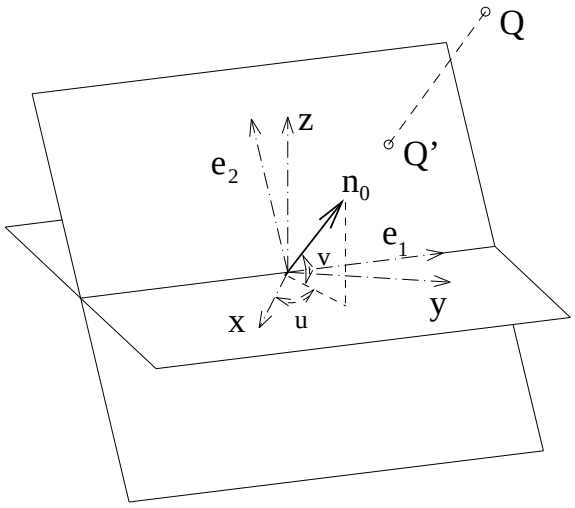


Abbildung 3.1: Parallelprojektion eines Punktes Q

ersten beiden Koordinaten von Q bezüglich der Basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0\}$ bestimmen:

$$\begin{aligned} x_e &= \mathbf{e}_1 \cdot \mathbf{q} = -x \sin u + y \cos u \\ y_e &= \mathbf{e}_2 \cdot \mathbf{q} = -(x \cos u + y \sin u) \sin v + z \cos v. \end{aligned}$$

Eine senkrechte Parallelprojektion ist also eine lineare Abbildung. Die Koeffizienten der zugehörigen Abbildungsmatrix ergeben sich aus den Projektionsformeln.

3.1.2 Prozeduren zur senkrechten Parallelprojektion

Da die Zahlen $\sin u, \cos u, \sin v, \cos v$ und der Normalenvektor $\mathbf{n}_0$ der Bildtafel für eine bestimmte Parallelprojektion oft gebraucht werden, werden wir sie in dem Unterprogramm `init_parallel_projection` nach dem Einlesen der Winkel u, v berechnen und über globale Variablen allen anderen Unterprogrammen zur Verfügung stellen. Die weiteren, unten aufgeführten, Unterprogramme werden durch Kommentare erläutert. All diese Programme sind in der Datei `proc_pp.pas` enthalten.

```
procedure init_parallel_projection;
 begin
   writeln('*** PARALLEL-PROJEKTION ***');
   writeln;
   writeln('Projektionswinkel u, v ? (in Grad)');
   readln(u_angle,v_angle);
   rad_u:= u_angle*pi/180;      rad_v:= v_angle*pi/180;
   sin_u:= sin(rad_u)    ;      cos_u:= cos(rad_u)    ;
   sin_v:= sin(rad_v)    ;      cos_v:= cos(rad_v)    ;
{Normalen-Vektor der Bildebene:}
   n0vt.x:= cos_u*cos_v;  n0vt.y:= sin_u*cos_v;  n0vt.z:= sin_v;
 end; { init_parallel_projection }
{**************}
procedure pp_vt3d_vt2d(p:vt3d; var pp:vt2d);
 {Berechnet das Bild eines Punktes}
{*************}
```

```
procedure pp_point(p:vt3d; style:integer);
 {Projiziert einen Punkt und markiert ihn gemaess style}
{*************}
procedure pp_line(p1,p2:vt3d ; style:integer);
 {Projiziert die Strecke p1,p2 gemaess style}
{*************}
procedure pp_arrow(p1,p2:vt3d; style:integer);
 {Projiziert einen Pfeil}
{*************}
procedure pp_axes(al:real);
 {Projiziert die Koordinatenachsen, al:Achsenlaenge}
{*************}
procedure pp_vts3d_vts2d(var p:vts3d; n1,n2:integer; var pp:vts2d);
 {Berechnet die Bilder pp einer Punktreihe p.}
{*************}
procedure pp_curve(var p:vts3d; n1,n2,style:integer);
 {Projiziert das 3d-Polygon p[n1]...p[n2]}
{*************}
procedure pp_curve_vis(var p:vts3d; n1,n2,style:integer; visible:b_array);
 {Projiziert ein 3d-Polygon. Es werden je zwei benachnarte "visible"
 Punkte verbunden. style=10: Rest wird gestrichelt.}
{*************}
```

3.2 Hiddenline-Algorithmus für konvexe Polyeder

Will man eine Szene, bestehend aus Kanten und ebenen n-Ecken, unter Berücksichtigung der Sichtbarkeit darstellen, so kann dies ein zeitaufwendiges Problem sein. Wir werden in Kapitel 13 einen relativ allgemeinen Algorithmus hierfür besprechen und angeben. Um Standardkörper, wie Würfel, Pyramide, Dodekaeder, usw. schon jetzt darstellen zu können, wollen wir uns einen einfachen Hiddenline -Algorithmus für konvexe Polyeder überlegen.

Zunächst erklären wir, was man unter einem konvexen Polyeder versteht. Die Oberfläche eines Körpers K mit den Eigenschaften
a) K wird von ebenen n-Ecken ($n \geq 3$) begrenzt.
b) Für jede Begrenzungsebene ε_i gilt: K liegt vollständig auf einer Seite von ε_i.
heißt *konvexes Polyeder*.
BEISPIELE sind: Quader, Pyramide, Oktaeder, ...

Idee des Algorithmus': Jede ebene Begrenzungsfläche wird mit Hilfe des Skalarproduktes der nach außen weisenden Normalen und der Projektionsrichtung auf Sichtbarkeit geprüft. Ist der Wert des Skalarproduktes positiv, so ist die entsprechende Fläche unsichtbar.

Wir demonstrieren den Algorithmus am Beispiel eines QUADERS:

Ein Quader besitzt 8 Ecken und 6 Seitenflächen (Rechtecke). Wir numerieren die Ecken von 1 bis 8 und übergeben dem Rechner, die Informationen, welche Ecken in einer Fläche liegen, in dem Integer-array `ifp`:

```
4,3,2,1,0,1,2,6,5,0,2,3,7,6,0,3,4,8,7,0,4,1,5,8,0,5,6,7,8,0
1.Fl"ache, 2.Fl"ache, 3.Fl"ache, 4.Fl"ache, 5.Fl"ache, 6.Fl"ache
```

Die Daten der verschiedenen Flächen trennen wir durch eine 0, um die Übersicht zu behalten. Da für

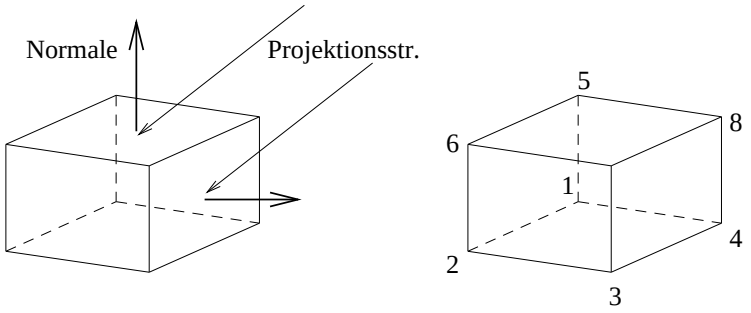


Abbildung 3.2: a) Normalentest b) zu `ifp` eines Quaders

den Sichtbarkeitstest nach außen weisende Normalen der Flächen berechnet werden müssen, ist die Anordnung der Punkte in `ifp` sehr wesentlich. Sie müssen (von außen gesehen) im Gegenuhrzeigersinn angeordnet sein. Das Feld `ifp` wird aus einer Integer-Datei $\cdots$ `.dat` eingelesen (s.u.). Außerdem übergeben wir
np: die Anzahl der Punkte (hier: 8) ,
nf: die Anzahl der Flächen (hier: 6) ,
und die Punkte $P_i,\ i = 1, ..., 8$.
(Man kann natürlich np,nf aus dem Feld `ifp` erkennen. Aber zur Kontrolle ist es nützlich, diese Daten extra anzugeben.)

Die folgenden Variablen müssen im Hauptprogramm bereit gestellt werden(s. Beispiel 3.1):

```
Variablen fuer aux_convex_polyh, pp_convex_polyh im Hauptprogramm:
      ep1,ep2 : array[1..150]  of integer;
      ifp,npf,ife : i_array;
      p : vts3d;     { Ecken }
      edge_drawn,visible_face : array[0..150]  of boolean;
      np,nf,ne,style : integer;
```

Das Unterprogramm `aux_convex_polyh` ist ein Hilfsprogramm. Es

a) numeriert die Kanten beginnend mit der 1.Fläche, dann die 2.Fläche,..., und berechnet die Anzahl `ne` der Kanten.

b) berechnet die Anzahl `npf(i)` der Punkte (Kanten) in der i-ten Fläche.

c) berechnet das Informationsfeld `ife` für die Kanten, das zeigt, welche Kanten in der i-ten Fläche, i = 1,...,nf, liegen. `ife` ist ähnlich aufgebaut wie `ifp`:
`ife : ....,0, ....,0, ....,0, .... ,`

d) berechnet Anfangs- und Endpunkte `ep1(i), ep2(i)` der i-ten Kante.

```
procedure aux_convex_polyh;
{Berechnet aus np,nf und ifp die Variablen: ne, ife, npf, ep1, ep2 .
 np,ne,nf : Anzahl der Punkte,Kanten,Fl"achen
 npf[i] : Anzahl der Punkte (Kanten) der i-ten Fl"ache
 ep1[k], ep2[k] : Anfangs- bzw. Endpunkt der k-ten Kante
 ifp bzw. ife : enthaelt bis zur naechsten "0" die Punkte bzw. Kanten
                einer Flaeche (positiv orientiert!!!)}
```

Der eigentliche Hiddenline-Algorithmus ist in `pp_convex_polyh` enthalten. Es

e) testet die Flächen auf Sichtbarkeit, indem es aus jeder Fläche zwei Vektoren $\mathbf{x}_1, \mathbf{x}_2$ auswählt, das Kreuzprodukt $\mathbf{x}_1 \times \mathbf{x}_2$ (nach außen weisende Normale der Fläche) und schließlich das Skalarprodukt $\mathbf{n}_0 \cdot (\mathbf{x}_1 \times \mathbf{x}_2)$ mit der Projektionsrichtung $\mathbf{n}_0$ berechnet. (Man beachte: $\mathbf{n}_0 \cdot (\mathbf{x}_1 \times \mathbf{x}_2)$ ist ein Spatprodukt und kann mit Hilfe einer 3x3_Determinante berechnet werden.)

f) zeichnet die sichtbaren Kanten und beachtet (mit Hilfe des Feldes `edge_drawn`), daß jede Kante nur einmal gezeichnet wird .

g) setzt `visible_face(i)=true`, falls die i-te Fläche sichtbar ist. (Dies kann man zur weiteren Behandlung von sichtbaren Flächen verwenden, s. u.)

h) zeichnet alle sichtbaren Kanten (`style=0`) und strichelt die unsichtbaren (`style=10`).

```
procedure pp_convex_polyh(style : integer);
{ style = 0 : unsichtbare Kanten werden weggelassen,
         10 : unsichtbare werden gestrichelt.
 edge_drawn[k]=true : k-te Kante wurde schon gezeichnet
 visible_face[i]=true: i-te Flaeche sichtbar }
 var  p32,p12 : vt3d;            pp : vts2d;
      i,j,k,ia,ib,i1,i2,j1,j2,j3,ik : integer;
      test : real;
 begin
   pp_vts3d_vts2d(p,1,np,pp);
   for i:= 1 to ne do edge_drawn[i]:= false;
   for i:= 1 to nf do visible_face[i]:= false;
{ Sichtbare Kanten berechnen und zeichnen: }
   ia:= 1;
   for i:= 1 to nf do
       begin
       j1:= ifp[ia];  j2:= ifp[ia+1];  j3:= ifp[ia+2];
       diff3d(p[j3],p[j2], p32);       diff3d(p[j1],p[j2], p12);
       test:= determ3d(n0vt,p32,p12);
       if test>=0 then    { Flaeche sichtbar }
          begin
          visible_face[i]:= true;
          ib:= ia + npf[i] - 1;
          for k:= ia to ib do
              begin
              ik:= ife[k];
              if not edge_drawn[ik] then
                 begin
                 i1:= ep1[ik] ;      i2:= ep2[ik] ;
                 line2d(pp[i1],pp[i2],0);
                 edge_drawn[ik]:= true;
                 end;
              end;  { for k }
          end;  {test<0}
       ia:= ia + npf[i]+1;
       end;  { for i }
{ Unsichtbare Kanten stricheln : }
   if style=10 then
      for i:= 1 to ne do
          begin
          if not edge_drawn[i] then
             begin
```

```
            i1:= ep1[i] ;     i2:= ep2[i] ;
            line2d(pp[i1],pp[i2],1);
            end;
          end;  {for}
 end;   { pp_convex_polyh }
{*************}
```

Die Unterprogramme `aux_convex_polyh` und `pp_convex_polyh` sind in der Datei `proc_pkp.pas` enthalten.

Beispiel 3.1 *QUADER*
Das folgende Programm zeigt die Anwendung des Hiddenline-Algorithmus auf einen Quader:

```
{************************************************************}
{***  Projektion eines Quaders mit achsenparallelen Kanten ***}
{************************************************************}
program quader;
uses geograph;
var    i,iachs,inz : integer;
       a,b,c,achsl : real;
{ Variablen fuer aux_convex_polyh, pp_convex_polyh: }
       ep1,ep2 : array[1..150] of integer;
       ifp,npf,ife : i_array;
       p : vts3d;   { Ecken }
       edge_drawn,visible_face : array[1..150] of boolean;
       np,nf,ne,style : integer;
    {$i proc_pkp.pas}  {enthaelt aux_convex_polyh und pp_convex_polyh}
{*********************}
  begin  {Hauptprogramm}
    graph_on(0);
    np:= 8;    nf:= 6;
    for i:= 1 to 30 do ifp[i]:= 0;
{ Eckpunkte der Fl"achen : }
    read_integer_file('quader.dat',30, ifp);
    writeln('*** Quader ***');  writeln;
    writeln(' x-L"ange, y-L"ange, z-L"ange ?');           readln(a,b,c);
    put3d(0,0,0,p[1]);  put3d(a,0,0,p[2]);
    put3d(a,b,0,p[3]);  put3d(0,b,0,p[4]);
{ Deckel: }
    for i:= 1 to 4 do put3d(p[i].x, p[i].y, c, p[4+i]);
    aux_convex_polyh;
    repeat
      init_parallel_projection;
      writeln(' Koordinaten-Achsen ? (Ja = 1)');       readln(iachs);
      if iachs=1 then  begin writeln('Achslaenge ?');  readln(achsl);  end;
      writeln(' Unsichtbare Kanten stricheln ? (Ja:1, nein:0)');
      readln(style);  if style=1 then style:= 10;
{** Zeichnen: }
      draw_area(180,140,90,70,1);
      if iachs=1 then  pp_axes(achsl);
      pp_convex_polyh(style);
      draw_end;
      writeln('Noch eine Zeichnung ? (ja: 1)');        readln(inz);
    until inz=0;
```

```
  graph_off;
end.
```

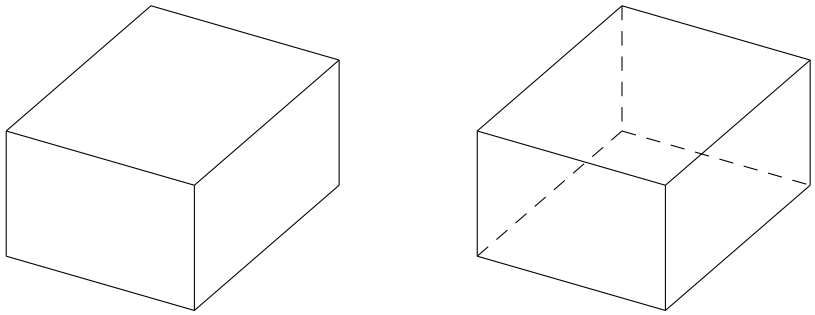


Abbildung 3.3: Quader a) `style=0`, b) `style=10`

Aufgabe 3.1 *Schreibe jeweils ein Programm, das*
a) eine PYRAMIDE bzw. b) ein OKTAEDER bzw. c) ein HAUS bzw.
d) einen PYRAMIDENSTUMPF
projiziert.

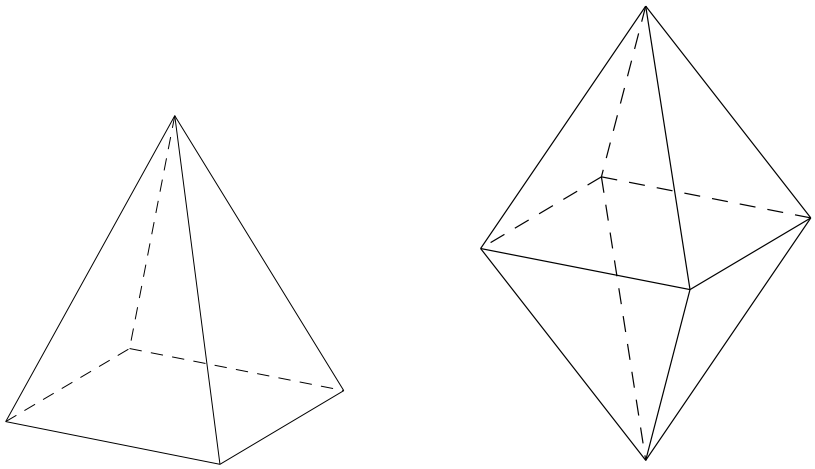


Abbildung 3.4: a) Pyramide, b) Oktader

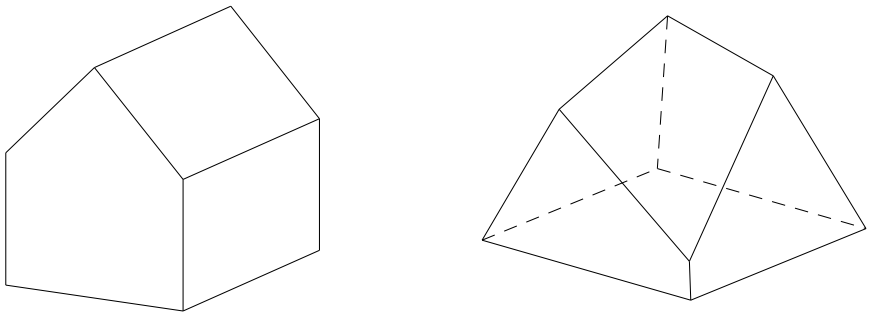


Abbildung 3.5: a) Haus, b) Pyramidenstumpf

Beispiel 3.2 *: DODEKAEDER*
Wir geben hier nur das Informationsfeld ifp und die Koordinaten von vier Punkten an. Die Koordinaten der restlichen Punkte lassen sich leicht aus der Zeichnung ermitteln.

```
ifp : 1,2,3,4,5,0,       4,6,7,8,5,0,       5,8,9,10,1,0,
      1,10,11,12,2,0,    13,17,16,15,14,0, 17,18,7,6,16,0,
      17,13,20,19,18,0, 20,13,14,12,11,0, 2,12,14,15,3,0,
      3,15,16,6,4,0,     7,18,19,9,8,0,     9,19,20,11,10,0
```

Punkte: 1: (a,0,c), 2: (b,b,b), 3: (0,c,a), 12: (c,a,0).
Dabei ist $b = a(\sqrt{5}+1)/2, \quad c = a(\sqrt{5}+3)/2$.
a ist die halbe Kantenlänge.

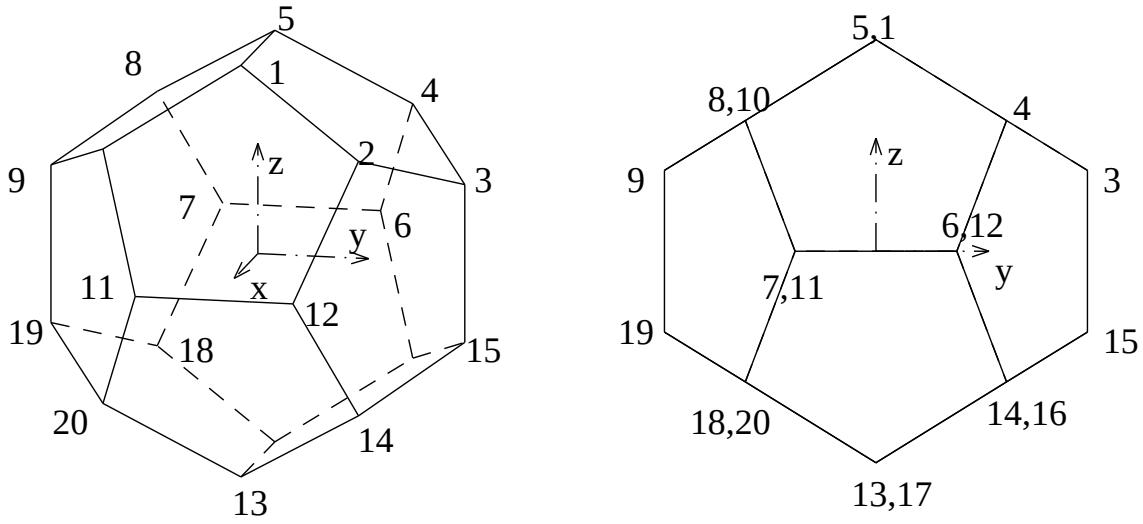


Abbildung 3.6: Dodekaeder (Beispiel 3.2)

Beispiel 3.3 *: IKOSAEDER*
Wir geben nur die Koordinaten von drei wesentlichen Punkten an. Die Koordinaten der restlichen

Punkte und das Informationsfeld ifp lassen sich aus der Zeichnung ablesen.
Punkte: 1: (0,a,b), 2: (b,0,a), 3: (a,b,0).
Dabei ist $b = a(\sqrt{5}+1)/2$. *a ist die halbe Kantenlänge.*

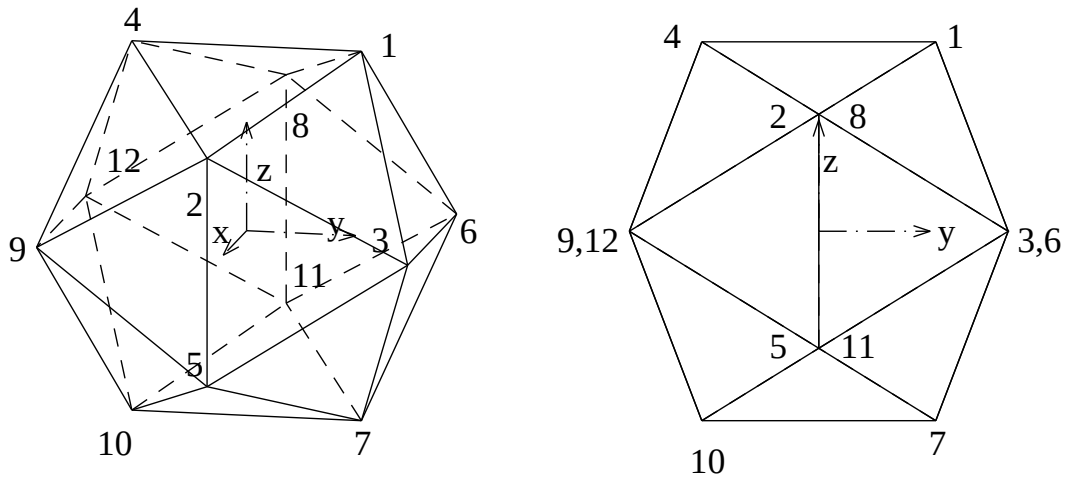


Abbildung 3.7: Ikosaeder (Beispiel 3.3)

Aufgabe 3.2 *a) Durch geeignetes Abschneiden der Ecken eines Würfels entstehen die die HALBREGULÄREN POLYEDER in Abb. 3.8 mit regelmäßigen 3- und 4-Ecken bzw. 4- und 6-Ecken.*
b) Das RHOMBEN-DODEKAEDER entsteht, indem man auf die Seitenflächen eines Würfels geeignete Pyramiden aufsetzt. (Die Seitenflächen benachbarter Pyramiden liegen in einer Ebene !)
c) Durch geeignetes Abschneiden der Ecken eines Ikosaeders entsteht das halbreguläre FUSSBALLPOLYEDER mit regelmäßigen 5- und 6-Ecken.
d) Durch geeignete Unterteilungen der Dreiecke eines Ikosaeders in regelmäßige Dreiecke und anschließende Projektion der Ecken auf die Umkugel des Ikosaeders entstehen TRIANGULIERUNGen DER KUGEL.
e) Mit Hilfe des in `pp_convex_polyh` *berechneten Feldes* `visible_face` *lassen sich z.B. KURVEN AUF sichtbaren FLÄCHEN von konvexen Polyedern zeichnen.*

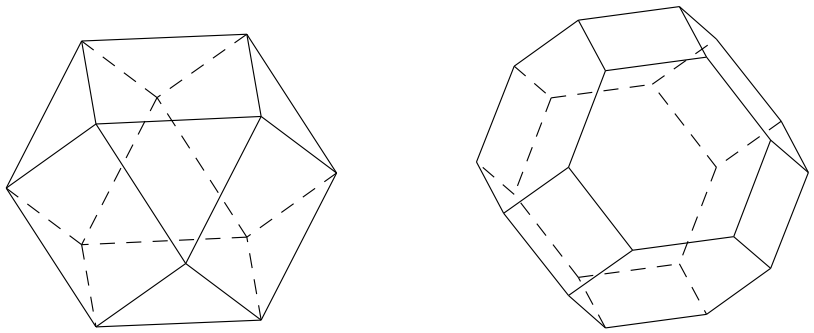


Abbildung 3.8: halbreguläre Polyeder

3.3 Kurve vor einer Ebene

In diesem Paragraphen wird ein Unterprogramm angegeben, das den Teil einer Kurve projiziert und zeichnet, der "vor" einer Ebene liegt. Es ist nützlich, um verdeckte Linien auf einer Quadrik

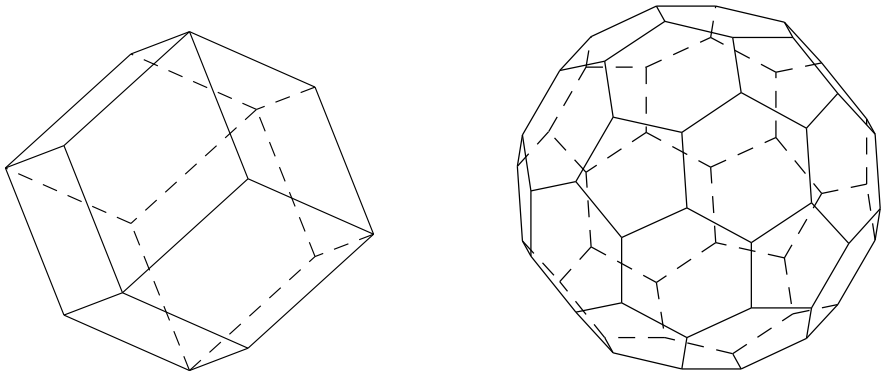


Abbildung 3.9: a) Rhomben-Dodekaeder b) Fußball-Polyeder

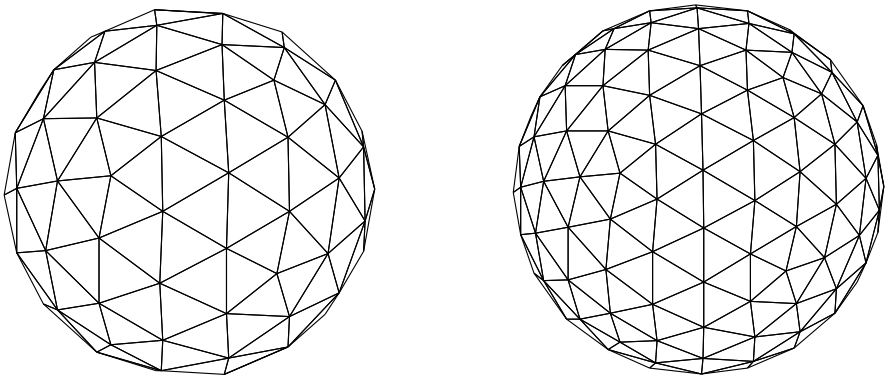


Abbildung 3.10: Triangulierung einer Kugel

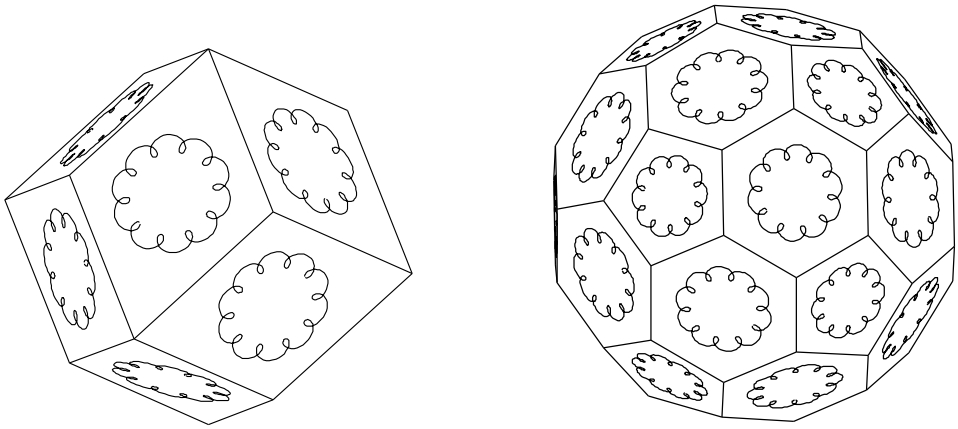


Abbildung 3.11: Kurven auf einem Polyeder

wegzulassen. Denn der Umriß einer Quadrik liegt sowohl bei Parallelprojektion als auch bei einer Zentralprojektion in einer Ebene. Da allerdings häufig Ellipsen abzubilden sind, wird für diesen Fall später ein besonderes Programm angegeben, das schneller als das allgemeine "Kurve vor einer Ebene" Programm ist.

3.3.1 Strecke vor einer Ebene, Kurve vor einer Ebene

Das folgende Unterprogramm `pp_line_before_plane(p1,p2,nv,d,side,style)` projiziert den Teil der Strecke $\overline{P_1P_2}$, $P_i : \mathbf{p}_i$, der "vor" der Ebene $\mathbf{n}\cdot\mathbf{x} = d$ liegt, d. h. der der Bedingung $side\cdot(\mathbf{n}\cdot\mathbf{x}-d) \geq 0$ genügt. Die Integer-Variable "side" hat den Wert 1 oder -1, jenachdem , welche Seite der Ebene man als "Vorderseite" haben möchte. Mit Hilfe der beiden Zahlen $dis1 = \mathbf{n}\cdot\mathbf{p}_1 - fd$ und $dis2 = \mathbf{n}\cdot\mathbf{p}_2 - d$ wird zunächst festgestellt, ob die Strecke $\overline{P_1P_2}$ die Ebene durchstößt. Dies ist der Fall, wenn $dis1$ und $dis2$ verschiedene Vorzeichen besitzen. Gegebenenfalls wird dann der Parameter t (bzgl. der Parameterdarstellung $\mathbf{p}_1 + t(\mathbf{p}_2 - \mathbf{p}_1)$) des Schnittpunktes P_3 berechnet und $\overline{P_1P_2}$ oder $\overline{P_2P_3}$ projiziert, je nachdem, ob $dis1$ oder $dis2$ positiv ist. Der "unsichtbare" Teil der Strecke wird gestrichelt, falls **style**= 10 ist.

```
procedure pp_line_before_plane(p1,p2,nv: vt3d; d : real; side,style: integer);
 {Zeichnet, falls style=0, den Teil der Strecke p1, p2, f"ur den
 side*(nv*x - d) >= 0 ist und strichelt den Rest, falls style = 10 ist.}
```

Die Prozedur `pp_curve_before_plane` ist das Analogon zu `pp_line_before_plane` für einen Polygonzug.

```
procedure pp_curve_before_plane(var p: vts3d ; n1,n2: integer; nv: vt3d;
                                          d: real; side,style : integer);
 {Projiziert eine Kurve (Polygonzug) "vor" der Ebene nv*x = d und strichelt
 den Rest, falls style=10 .}
```

3.4 Vogelperspektive

In der Darstellenden Geometrie mit Zirkel und Lineal verwendet man häufig schräge Parallelprojektionen, z. B. "Vogelperspektive" (schräge Parallelprojektion auf die $x-y-$Ebene) oder "Kavalierperspektive" (schräge Parallelprojektion auf die $x-z-$ oder $y-z-$Ebene). Sie ermöglichen in gewissen Situationen, schnell anschauliche Bilder von Objekten herzustellen (vgl. GR,BA '73 oder WU'66/'67 oder ...), weil z.B. bei der Vogelperspektive jede zur $x-y-$Ebene parallele Figur unverzerrt abgebildet wird. Sobald allerdings Kreise in allgemeiner Lage oder Kugeln abgebildet werden müssen, ergeben sich (mit Zirkel und Lineal) größere Schwierigkeiten als bei der senkrechten Parallelprojektion. Weil die Vogelperspektive durch kleine Abänderungen der Unterprogramme `pp_vt3d_vt3d`, `pp_vts3d_vts2d` besonders einfach zu realisieren ist, wollen wir hier kurz darauf eingehen.
Es sei Φ_{uv} die zu den Winkeln u, v mit $v \neq 0$ gehörige senkrechte Parallelprojektion auf die Ebene ε_{uv} mit der Normalen $\mathbf{n}_0$ (negative Projektionsrichtung, s. Kap. 3.1). Das Bild Q'' eines Punktes $Q : \mathbf{q} = (x, y, z)$ unter der schrägen Parallelprojektion Σ_{uv} auf die $x-y-$Ebene mit derselben Projektionsrichtung erhält man, indem man das Bild $Q' : \mathbf{q}' = (x_e, y_e)$ unter Φ_{uv} auf die $x-y-$Ebene weiterprojiziert. Zur Beschreibung des Bildes Q'' in der $x-y-$Ebene benutzen wir $\mathbf{e}_1$ (aus ε_{uv}) und das normierte Bild von $\mathbf{e}_2$ (s. Abschn 3.1). Ein Punkt $Q : \mathbf{q} = (x, y, z)$ wird also durch Σ_{uv} auf den Punkt $Q'' : \mathbf{q} = (x_e, y_e/\sin v)$ abgebildet. Um die Vogelperspektive eines Objektes zu erhalten, braucht man demnach nur die Projektionsformeln in den Unterprogrammen `pp_vt3d_vt2d` und `pp_vts3d_vts2d` in der hier beschriebenen Weise abzuändern.

Beispiele:

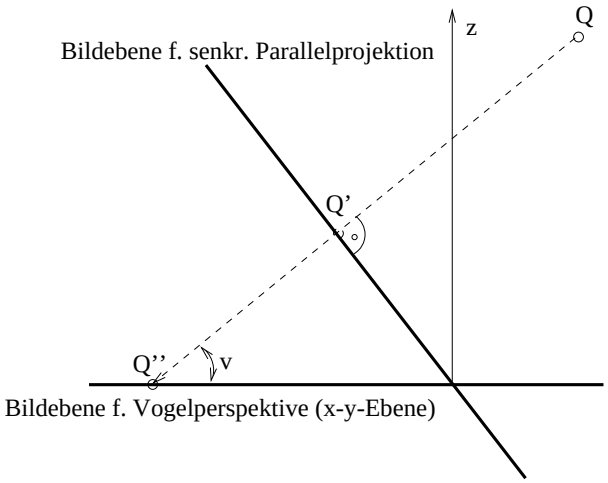


Abbildung 3.12: Bild des Punktes Q bei Vogelperspektive

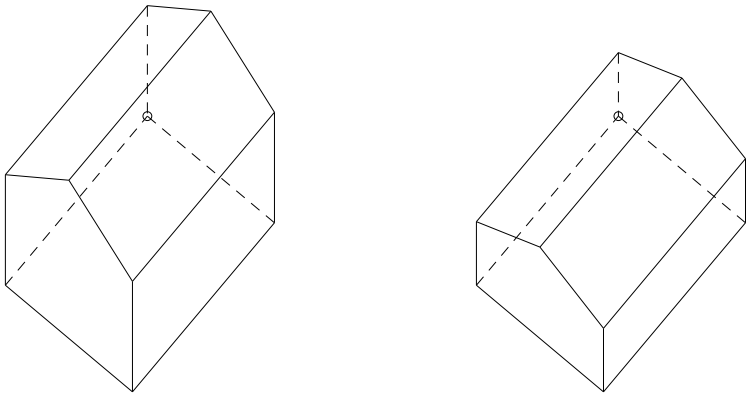


Abbildung 3.13: Haus in Vogelperspektive (links: $v = 45°$, rechts: $v = 60°$)

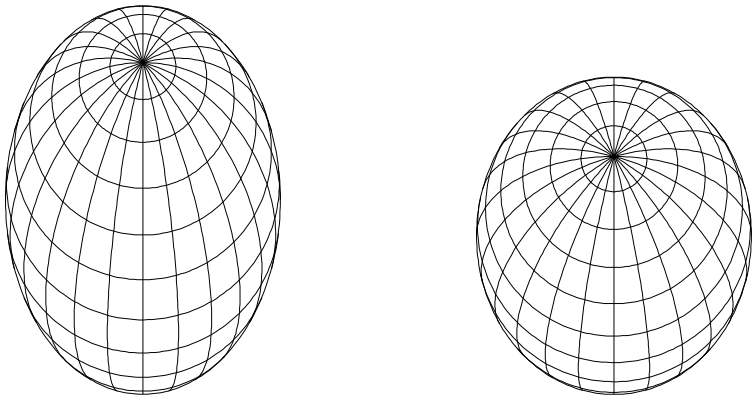


Abbildung 3.14: Kugel in Vogelperspektive (links: $v = 45°$, rechts: $v = 60°$)

Kapitel 4

ZENTRALPROJEKTION

Ein höheres Maß an Anschaulichkeit erreicht man durch Darstellung eines Gegenstandes in Zentralprojektion. Man verwendet dabei Strahlen, die von einem festen Punkt Z, dem Zentrum oder Augpunkt, ausgehen. Den Gewinn an Anschaulichkeit muß man allerdings i.a. durch einen Verlust an Maßgenauigkeit erkaufen. Ein typischer Unterschied zur Parallelprojektion besteht darin, daß parallele Geraden i.a. in einer Zentralprojektion nicht mehr parallel sind, sondern durch einen Punkt, dem Fluchtpunkt des Parallelbüschels gehen (s. Abb. 4.1).

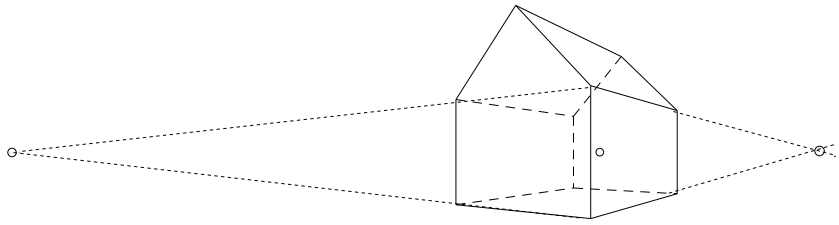


Abbildung 4.1: Haus in Zentralprojektion

Wir werden sehen, daß sich die meisten Programme , die wir für Parallelprojktion geschrieben haben, leicht für Zentralprojektion abändern lassen.

4.1 Die Projektionsformeln

Sei ε eine Ebene und $Z : \mathbf{z}$ ein nicht in ε gelegener Punkt des $\mathbb{R}^3$. Die Abbildung Φ, die einem beliebigen Punkt $P : \mathbf{p}$ den Schnittpunkt P' der Geraden $\overline{ZP}$ mit der Ebene ε zuordnet, falls dieser existiert, heißt **Zentralprojektion von Z auf ε**.

Z heißt das **Zentrum** oder der **Augpunkt** der Zentralprojektion Φ (s. Abb. 4.2). Der Lotfußpunkt $H : \mathbf{h}$ des Lotes von Z auf die Bildtafel ε_0 heißt **Hauptpunkt** von Φ. Ist $\mathbf{n}_0 := (\cos u \cos v, \sin u \cos v, \sin v), u \in [0, 2\pi], v \in [-\pi/2, \pi/2]$, die Normale von ε_0, so hat ε_0 die Gleichung: $(\mathbf{x} - \mathbf{h}) \cdot \mathbf{n}_0 = 0$. Für das Zentrum $Z : \mathbf{z}$ und den Hauptpunkt H gilt $\mathbf{z} = \mathbf{z} + \delta\mathbf{n}_0$ mit $\delta > 0$. Der Abstand δ des Zentrums Z zur Ebene ε_0 heißt die **Distanz** von Φ. Alle Punkte des $\mathbb{R}^3$, die durch Φ nicht abgebildet werden können, liegen in der zu ε_0 parallelen Ebene ε_v durch den Augpunkt Z.
ε_v heißt die **Verschwindungsebene** von Φ.

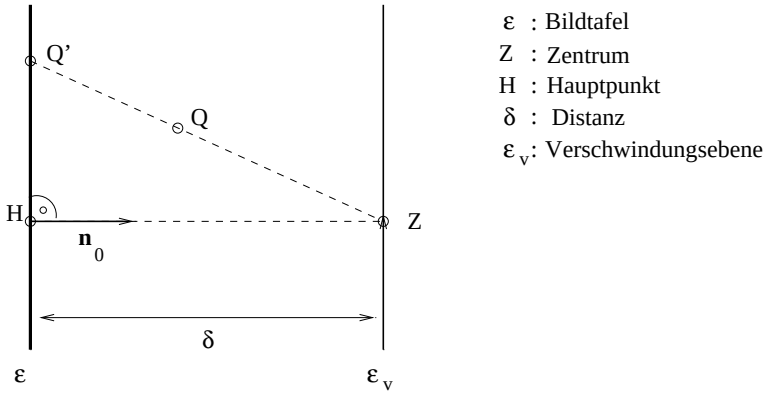


Abbildung 4.2: Hauptpunkt, Distanz und Verschwindungsebene

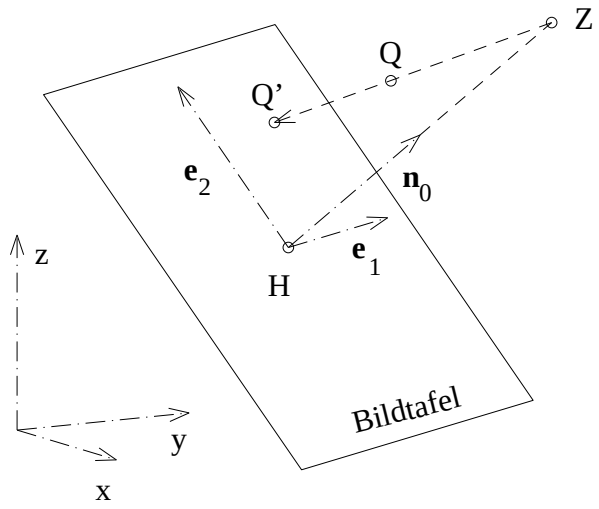


Abbildung 4.3: Zentralprojektion eines Punktes

Die Zentralprojektion Φ ist durch die Vorgabe der Parameter u, v, des Hauptpunktes H und der Distanz δ eindeutig bestimmt. Ein Punkt $P : \mathbf{p}$, der nicht in der Verschwindungsebene ε_v liegt, wird auf den Punkt $P' := \overline{ZP} \cap \varepsilon_0$ abgebildet. Führt man diesen Schnitt des Projektionsstrahls mit der Bildtafel aus, so ergibt sich

$$P' : \mathbf{p}' = \mathbf{z} + \frac{(\mathbf{h}-\mathbf{z})\cdot\mathbf{n}_0}{(\mathbf{p}-\mathbf{z})\cdot\mathbf{n}_0}(\mathbf{p}-\mathbf{z}) = \mathbf{h} + \delta\mathbf{n}_0 + \frac{\delta}{\delta-(\mathbf{p}-\mathbf{h})\cdot\mathbf{n}_0}(\mathbf{p}-\mathbf{h}-\delta\mathbf{n}_0)$$

Da P' in ε_0 liegt, gilt ferner $(\mathbf{p}'-\mathbf{h})\cdot\mathbf{n}_0 = 0$.

In der Ebene ε_0 führen wir jetzt so Koordinaten ein, daß H der Nullpunkt ist und (analog zur senkrechten Parallelprojektion)

$$\mathbf{e}_1 := (-\sin u, \cos u, 0), \quad \mathbf{e}_2 := (-\cos u \sin v, -\sin u \sin v, \cos v)$$

die Basisvektoren sind. $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0\}$ ist eine ON-Basis des $\mathbb{R}^3$.(vgl. Abb. 4.3)

Es gibt Zahlen x_{ez}, y_{ez} (die Koordinaten von P' bzgl. der Basis $\{\mathbf{e}_1, \mathbf{e}_2\}$), sodaß $\mathbf{p}' = \mathbf{h} + x_{ez}\mathbf{e}_1 + y_{ez}\mathbf{e}_2$ ist. Mit Hilfe der obigen Darstellung von $\mathbf{p}'$ ergibt sich (unter Beachtung der Orthogonalität von $\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0$)

$$x_{ez} = \frac{\mathbf{e}_1 \cdot (\mathbf{p} - \mathbf{h})}{1 - (\mathbf{p} - \mathbf{h}) \cdot \mathbf{n}_0/\delta} \qquad y_{ez} = \frac{\mathbf{e}_2 \cdot (\mathbf{p} - \mathbf{h})}{1 - (\mathbf{p} - \mathbf{h}) \cdot \mathbf{n}_0/\delta}$$

(Für $\mathbf{h} = 0$ und $\delta \to \infty$ ergeben sich die Formeln für die senkrechte Parallelprojektion !) Bei der Ausführung der Projektion ist es üblich, nur solche Punkte zu projizieren , die "vor" der Verschwindungsebene ε_v liegen, die also der Bedingung $(\mathbf{p} - \mathbf{h}) \cdot \mathbf{n}_0 < \delta$ genügen. Der Einfachheit halber wollen wir hier zunächst auch nur Strecken und Kurven projizieren, die **vollständig** "vor" der Verschwindungsebene liegen. Für den allgemeinen Fall werden wir in 4.2 einen geeigneten "Clippingalgorithmus" einbauen.
Wir geben jetzt die für die Zentralprojektion wesentlichen Unterprogramme für die Projektion von Punkten, Strecken, Kurven,... an. Dabei beachte man, daß der Hauptpunkt `mainpt`, das Zentrum `centre`, die Normale `n0vt` und die Distanz `distance` globale Variablen sind und in der Datei `geovar`.pas enthalten sind. Die Zentralprojektion eines Punktes führen wir formal in zwei Schritten durch:

(1) Koordinatentransformation in das System $(H; \mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0)$ mit dem Nullpunkt H und der Basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0\}$.

$$\mathbf{p} = (x, y, z) \to \bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{z}) \text{ mit } \bar{x} = (\mathbf{p} - \mathbf{h}) \cdot \mathbf{e}_1,\ \bar{y} = (\mathbf{p} - \mathbf{h}) \cdot \mathbf{e}_2,\ \bar{z} = (\mathbf{p} - \mathbf{h}) \cdot \mathbf{n}_0,$$

(2) Zentralprojektion in dem System $(H; \mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0)$ auf die $\bar{x} - \bar{y}$- Ebene:

$$\bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{z}) \to \left(\frac{\bar{x}}{1 - \bar{z}/\delta},\ \frac{\bar{y}}{1 - \bar{z}/\delta}\right)$$

Da wir später die Koordinatentransformation (1) benötigen, wird sie hier schon angegeben.

```
procedure init_central_projection;
 begin
   writeln('*** ZENTRAL-PROJEKTION ***');
   writeln;
   writeln('Hauptpunkt ?');  readln(mainpt.x,mainpt.y,mainpt.z);
   writeln('Distanz ?');     readln(distance);
   writeln('Projektionswinkel u, v ? (in Grad)');  readln(u_angle,v_angle);
   rad_u:= u_angle*pi/180;        rad_v:= v_angle*pi/180;
   sin_u:= sin(rad_u);            cos_u:= cos(rad_u);
   sin_v:= sin(rad_v);            cos_v:= cos(rad_v);
{Basis e1,e2 und Normale n0 der Bildebene:}
   e1vt.x:= -sin_u;              e1vt.y:= cos_u;             e1vt.z:= 0;
   e2vt.x:= -cos_u*sin_v;        e2vt.y:=-sin_u*sin_v;       e2vt.z:= cos_v;
   n0vt.x:= cos_u*cos_v;         n0vt.y:= sin_u*cos_v;       n0vt.z:= sin_v;
 {Zentrum:}
   lcomb2vt3d(1,mainpt, distance,n0vt, centre);
 end;   {init_central_projection}
{**************}
procedure transf_to_e1e2n0_base(p : vt3d; var pm : vt3d);
{Berechnet Koordinaten bzgl. System mit Hauptpkt. als Nullpkt. und der
 Basis e1,e2,n0.}
 var pd : vt3d;
 begin
```

```
   diff3d(p,mainpt, pd);
   put3d(scalarp3d(pd,e1vt),scalarp3d(pd,e2vt),scalarp3d(pd,n0vt), pm);
 end;  { transf_to_e1e2n0_base }
{***************}
procedure cp_vt3d_vt2d(p: vt3d; var pp : vt2d);
 {Zentralprojektion (Koordinaten) eines Punktes}
 var xe,ye,ze,cc : real;   pm : vt3d;
 begin
   diff3d(p,mainpt, pm);
   xe:= scalarp3d(pm,e1vt);     {Koordinaten von p bzgl. dem Koord.-System:}
   ye:= scalarp3d(pm,e2vt);     {Nullpunkt = Hauptpunkt}
   ze = scalarp3d(pm,n0vt);     {und Basis e1,e2,n0}
   cc:= 1-ze/distance;
   if cc>eps6 then begin  pp.x:= xe/cc; pp.y:= ye/cc; end     {Projektion}
              else
              writeln('Punkt liegt in oder hinter der Verschwindungsebene !!');
 end;    {cp_vt3d_vt2d}
```

Die folgenden Unterprogramme können wörtlich aus der Parallelprojektion übernommen werden. Man muß nur überall die drei Zeichen `pp_` durch `cp_` ersetzen.

```
procedure cp_point(p: vt3d; style: integer);
 {markiert einen projizierten Punkt}
{***************}
procedure cp_line(p1,p2 : vt3d ; style : integer);
 {projiziert die Strecke p1 p2}
{***************}
procedure cp_arrow(p1,p2 : vt3d; style : integer);
 {projiziert einen Pfeil}
{***************}
procedure cp_axes(al : real);
 {projiziert die Koordinatenachsen}
{***************}
procedure cp_vts3d_vts2d(p: vts3d; n1,n2 : integer; var pp : vts2d);
 {Koordinaten von projizierten Punkten p[i] , i= n1...n2.}
{*************}
procedure cp_curve(p: vts3d; n1,n2,style : integer);
 {projiziert ein 3D-Polygon}
{*************}
procedure cp_curve_vis(p : vts3d; n1,n2,style : integer; visible: b_array );
 {Es werden je zwei benachnarte "visible" Punkte verbunden.}
{*************}
procedure cp_line_before_plane(p1,p2,nv: vt3d; d : real; side,style: integer);
 {Zeichnet ,falls style=0, den Teil der Strecke p1, p2, f"ur den
 side*(nv*x - d) >= 0 ist und strichelt den Rest, falls style = 10 ist.}
{*************}
procedure cp_curve_before_plane(p: vts3d ; n1,n2: integer; nv: vt3d; d: real; side,style: integer);
 {Projiziert eine Kurve (Polygonzug) "vor" der Ebene nv*x = d und strichelt
 den Rest, falls style=10.}
```

Aufgabe 4.1 *Ändere das Unterprogramm* `pp_convex_polyh` *aus 3.2 in* `cp_convex_polyh` *ab, das Zentralprojektionen von konvexen Polyedern erstellt. (Es muß die (negative) Projektionsrichtung* $\mathbf{n}_0$ *durch die Projektionsrichtung* $\mathbf{z} - \mathbf{p}_i$ *des Punktes* P_i *ersetzt werden.)*

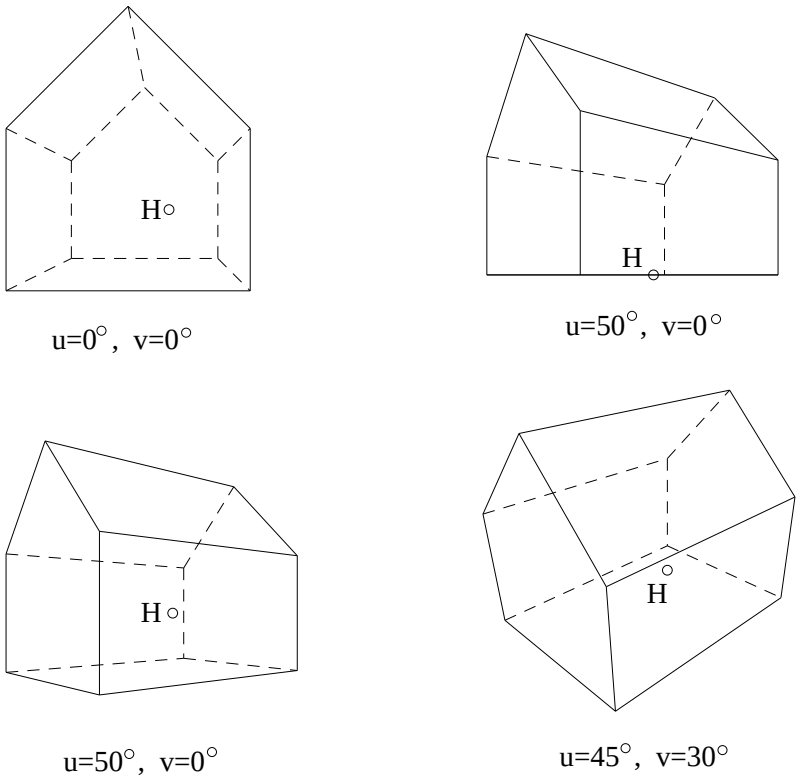


Abbildung 4.4: Zentralprojektionen eines Hauses

4.2 3-D-Clipping

Will man von Objekten auch "Innen"-Ansichten herstellen, wie zum Beispiel von einem Haus, so ist die Voraussetzung "Objekt liegt vollständig vor der Verschwindungsebene", die wir in Kap. 1 gemacht haben, nicht mehr erfüllt. Man muß dann Punkte , Strecken und Kurven vor der Projektion testen und, falls nötig, beschränken, **clippen**. Wir wollen nur solche Punkte, Strecken- und Kurventeile projizieren, die innerhalb eines nach unten offenen **Pyramidenstumpfes** (vgl. 4.5) liegen. Die Spitze der zugehörigen Pyramide ist das Projektionszentrum Z und die Gerade durch Z und den Hauptpunkt H die Symmetrieachse.
Zur Beschreibung der Seitenflächen der Pyramide und zum Clippen verwenden wir $\bar{x}-\bar{y}-\bar{z}-$Koordinaten bzgl. der Basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0\}$ mit dem Hauptpunkt H als Nullpunkt (vgl. Kap. 4.1). Die $\bar{z}-$Achse ist dann die Achse der Pyramide. "Deckel" und Seitenflächen des Pyramidenstumpfes wählen wir in den folgenden Ebenen:

$$
\begin{array}{lcrcrcl}
\varepsilon_0 & : & & & \overline{z} & = & \overline{z}_0 < \delta \quad (\delta \text{ ist die Distanz.}) \\
\varepsilon_1 & : & \gamma\overline{x} & + & \overline{z} & = & \delta \\
\varepsilon_2 & : & -\gamma\overline{x} & + & \overline{z} & = & \delta \\
\varepsilon_3 & : & \gamma\overline{y} & + & \overline{z} & = & \delta \\
\varepsilon_4 & : & -\gamma\overline{y} & + & \overline{z} & = & \delta.
\end{array}
$$

Die **Clipkonstanten** γ(= `gamma_clip`), z_0(= `zmax_over_screen`) und c_0(= `xy_max_on_screen`) werden global bereitgestellt. Letztere wird als Parameter von `init_central_projection_clip` im

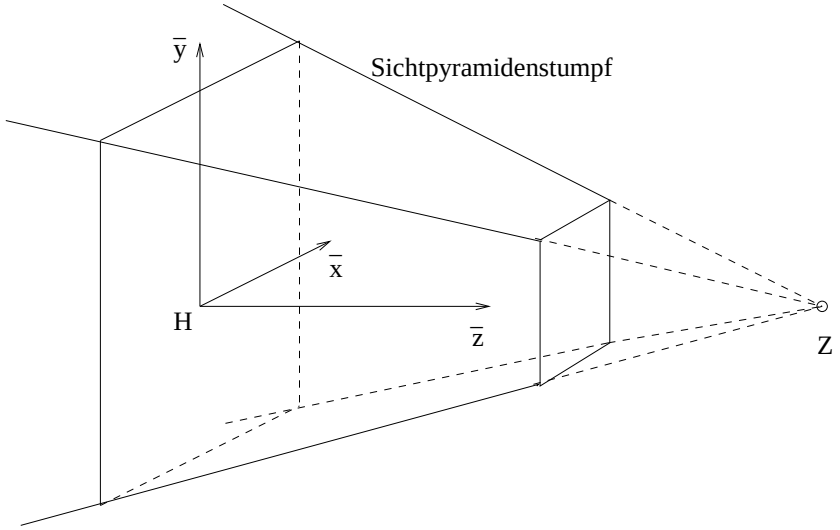


Abbildung 4.5: Sichtpyramidenstumpf

Hauptprogramm festgelegt.

Um ein Objekt auch von innen projizieren zu können, müssen lediglich die Unterprogramme `cp_vt3d_vt2d, cp_point, cp_line, cp_axes, cp_curve, cp_ellipse_arc` gegen die folgenden Versionen ausgetauscht werden. Man beachte, daß die Parameterliste von `cp_vt3d_vt2d` um die Kontrollvariable `vis` erweitert ist (vis=true: Punkt ist "sichtbar", d.h. innerhalb des Sichtpyramidenstumpfes.).

```
{*****************}
var   xy_max_on_screen,      zmax_over_screen,      gamma_clip: real;
{*****************}
procedure init_central_projection_clip(xym: real);
 begin
   writeln('*** ZENTRAL-PROJEKTION mit 3D-CLIPPING ***');
   writeln;
   writeln('Hauptpunkt ?');  readln(mainpt.x,mainpt.y,mainpt.z);
   writeln('Distanz ?');     readln(distance);
   writeln('Projektionswinkel u, v ? (in Grad)');  readln(u_angle,v_angle);
   rad_u:= u_angle*pi/180;        rad_v:= v_angle*pi/180;
   sin_u:= sin(rad_u);            cos_u:= cos(rad_u);
   sin_v:= sin(rad_v);            cos_v:= cos(rad_v);
{Basis e1,e2 und Normale n0 der Bildebene:}
   e1vt.x:= -sin_u;         e1vt.y:= cos_u;          e1vt.z:= 0;
   e2vt.x:= -cos_u*sin_v;   e2vt.y:=-sin_u*sin_v;    e2vt.z:= cos_v;
   n0vt.x:= cos_u*cos_v;    n0vt.y:= sin_u*cos_v;    n0vt.z:= sin_v;
{Zentrum:}
   lcomb2vt3d(1,mainpt, distance,n0vt,  centre);
{Clip-Konstanten:}
  xy_max_on_screen:= xym;
  zmax_over_screen:= distance-1;
  gamma_clip:= distance/xy_max_on_screen;
 end; { init_central_projection_clip }
{**************}
```

```
procedure cp_vt3d_vt2d(p: vt3d; var pp : vt2d; var vis: boolean);
{Zentralprojektion eines Punktes (Koordinaten) innerhalb der Sichtpyramide. }
 var x,y,z,denom,d,g : real;   q : vt3d;
 begin
   transf_to_e1e2n0_base(p, q);
   x:= q.x;  y:= q.y;  z:= q.z;
   vis:= false;  d:= distance;  g:= gamma_clip;
   if (z<zmax_over_screen) then
      if (g*x+z-d<0) then  if (-g*x+z-d<0) then
         if (g*y+z-d<0) then  if (-g*y+z-d<0) then   vis:= true;
   if vis then     { Punkt innerhalb der Sichtpyramide }
      begin
      denom:= 1-z/d;   pp.x:= x/denom;   pp.y:= y/denom;
      end;
 end; { cp_vt3d_vt2d }
{**************}
procedure cp_point(p: vt3d; style: integer);
{Markiert einen projizierten Punkt}
 var pp : vt2d;  vis : boolean;
 begin
   cp_vt3d_vt2d(p,pp,vis);  if vis then  point2d(pp,style);
 end;  { cp_point }
{**************}
procedure cp_line(p1,p2: vt3d; style: integer);
{Projiziert den Teil einer Strecke, der innerhalb des Sichtpy... liegt.}
 var g,d,z0,gx1,gy1,z1,z2,gx2,gy2,t1,t2,denom : real;
     q1,q2,q21,q1c,q2c : vt3d;  pp1,pp2 : vt2d;
{**}
function still_vis(ix,iy: integer; z0 : real) : boolean;
 var dis1,dis2,dist1,dist2,s : real;
 begin
   dis1:= ix*gx1 +iy*gy1 + z1 - z0;    dis2:= ix*gx2 +iy*gy2 + z2 - z0;
   dist1:= dis1 + t1*(dis2-dis1);     dist2:= dis1 + t2*(dis2-dis1);
   if ((dist1<0) and (dist2>0)) or ((dist1>0) and (dist2<0)) then
      begin
      s:= dis1/(dis1-dis2);     still_vis:= true;
      if dist1>0 then t1:= s  else  t2:= s;
      end
     else   if dist1<0   then still_vis:= true else still_vis:= false;
 end; { still_vis }
{**}
 begin
   transf_to_e1e2n0_base(p1, q1);   transf_to_e1e2n0_base(p2, q2);
   diff3d(q2,q1, q21);
   g:= gamma_clip;   d:= distance;    z0:= zmax_over_screen;
   gx1:= g*q1.x;  gy1:= g*q1.y;  z1:= q1.z;
   gx2:= g*q2.x;  gy2:= g*q2.y;  z2:= q2.z;
   t1:= 0;   t2:= 1;
   if still_vis(0,0,z0) then
      if still_vis(1,0,d) then
         if still_vis(-1,0,d) then
            if still_vis(0,1,d)then
               if still_vis(0,-1,d) then
      begin   { Projektion der Rest-Strecke }
```

```
      lcomb2vt3d(1,q1, t1,q21, q1c);  lcomb2vt3d(1,q1, t2,q21, q2c);
      denom:= 1-q1c.z/d;  pp1.x:= q1c.x/denom;  pp1.y:= q1c.y/denom;
      denom:= 1-q2c.z/d;  pp2.x:= q2c.x/denom;  pp2.y:= q2c.y/denom;
      line2d(pp1,pp2,style);
      end;
  end;  { cp_line }
{**************}
procedure cp_vts3d_vts2d(p: vts3d; n1,n2 : integer; var pp : vts2d;
                                                    var vis : b_array);
{Berechnet die Bildkoordinaten einerPunktreihe und stellt die Sichtbarkeit fest.}
 var i : integer;
 begin
   for i:= n1 to n2 do cp_vt3d_vt2d(p[i],pp[i],vis[i]);
 end;  { cp_vts3d_vts2d }
{**************}
procedure cp_curve(p: vts3d; n1,n2,style : integer);
{Projiziert ein Polygon.}
 var pp : vts2d;   vis : b_array;
 begin
   cp_vts3d_vts2d(p,n1,n2,pp,vis);   curve2d_vis(pp,n1,n2,style,vis);
 end;   { cp_curve }
{**************}
procedure cp_curve_vis(p : vts3d; n1,n2,style : integer; vis: b_array );
{Es werden je zwei benachnarte "visible" Punkte verbunden.}
 var pp : vts2d;  i : integer;
 begin
   for i:= n1 to n2 do  if vis[i] then cp_vt3d_vt2d(p[i],pp[i],vis[i]);
   curve2d_vis(pp,n1,n2,style,vis);
 end;  { cp_curve_vis }
{**************}
procedure cp_ellipse_arc(pc0,pc1,pc2,pc3,pc4 : vt3d; style: integer);
 {Projektion eines "geclippten" Ellipsenbogens }
 var p : vts3d;      i,n2 : integer;
 begin
   ellipse_pts3d(pc0,pc1,pc2,pc3,pc4, p,n2);
   cp_curve(p,0,n2,style);
   if style=10 then
      begin
      ellipse_pts3d(pc0,pc1,pc2,pc4,pc3, p,n2);     cp_curve(p,0,n2,1);
      end;
 end; { cp_ellipse_arc}
{**************}
procedure cp_ellipse(pc0,pc1,pc2 : vt3d; style: integer);
 begin   cp_ellipse_arc(pc0,pc1,pc2,pc1,pc1,style);   end;
{**************}
```

Aufgabe 4.2 *Schreibe ein Programm, das einen FESTSAAL (s. Abb. 4.6) von verschiedenen inneren Standpunkten aus projiziert.*

Abbildung 4.6: Projektion des Innern eines Festsaales (1)

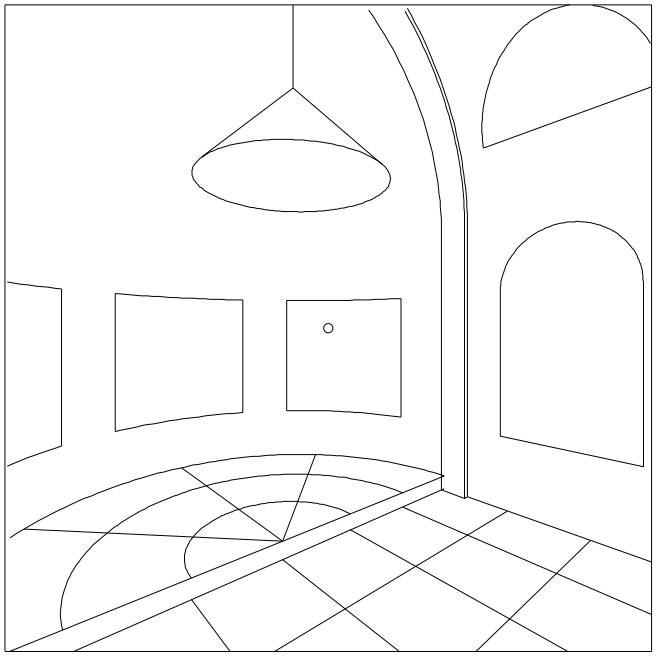


Abbildung 4.7: Projektion des Innern eines Festsaales aus anderer Sicht(2)

Kapitel 5

ELLIPSEN, HYPERBELN UND PARABELN

Es gibt viele Möglichkeiten eine Ellipse, Hyperbel oder Parabel zu definieren. Wir verwenden hier, daß eine Ellipse bzw. Hyperbel bzw. Parabel das affine Bild des "Einheitskreises" bzw. der "Einheitshyperbel" bzw. "Einheitsparabel"ist. Aber zunächst erklären wir, was man unter einer "affinen Abbildung" versteht.

Im Folgenden sei $B := \{\mathbf{e}_1, \mathbf{e}_2\}$ eine Orthonormalbasis (d.h.: $\mathbf{e}_1 \perp \mathbf{e}_2$ und $\|\mathbf{e}_1\| = \|\mathbf{e}_2\| = 1$) der Anschauungsebene. Ein Punkt P läßt sich dann durch einen Vektor $\mathbf{p}$ repräsentieren und es gibt $x, y \in \mathbb{R}$ mit

$$\mathbf{p} = x\mathbf{e}_1 + y\mathbf{e}_2 = \begin{pmatrix} x \\ y \end{pmatrix}_e$$

x, y heißen die *Koordinaten* von P bzgl. der Basis B . Falls keine Mißverständnisse möglich sind, schreiben wir kurz

$$P : \mathbf{p} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ oder } \mathbf{p} = (x, y).$$

Ist A eine reguläre 2×2-Matrix (d. h. ihre Determinante ist nicht 0) und $(x_0, y_0) \in \mathbb{R}^2$, so heißt die Abbildung

$$\Phi : \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + A \begin{pmatrix} x \\ y \end{pmatrix} \text{ von } \mathbb{R}^2 \text{ auf sich } \textbf{affine Abbildung.}$$

In Vektorform $\Phi : \mathbf{p} \to \mathbf{p}_0 + A\mathbf{p}$. Analog zur Definition einer affinen Abbildung im $\mathbb{R}^2$ ist eine affine Abbildung im $\mathbb{R}^3$ erklärt.

5.1 Ellipsen

Bezüglich der obigen Basis B ist $K_1 := \{(x, y) \in \mathbb{R}^2 | x^2 + y^2 = 1\}$ ein Kreis mit Radius 1 (Einheitskreis).
Ein affines Bild des Einheitskreises K_1 heißt **Ellipse**. (vgl. Abb. 5.1)

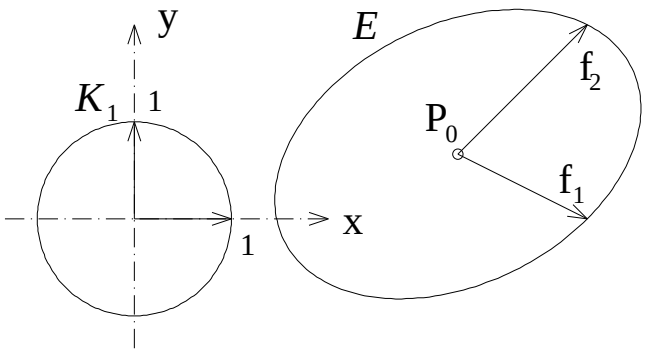


Abbildung 5.1: Ellipse als affines Bild des Einheitskreises

Beschreibt man K_1 folgendermaßen

$$K_1 := \{(\cos t, \sin t) | 0 \leq t \leq 2\pi\} = \{\begin{pmatrix}1\\0\end{pmatrix} \cos t + \begin{pmatrix}0\\1\end{pmatrix} \sin t | \ldots\}$$

und bedenkt, daß es zu je zwei linear unabhängigen Vektoren $\mathbf{f}_1, \mathbf{f}_2$ eine reguläre Matrix A gibt mit

$$\mathbf{f}_1 = A\begin{pmatrix}1\\0\end{pmatrix}, \quad \mathbf{f}_2 = A\begin{pmatrix}0\\1\end{pmatrix},$$

so erkennt man die Gültigkeit der folgenden Aussage:

Zu jeder Ellipse E gibt es einen Vektor $\mathbf{p}_0$ und zwei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2$, sodaß gilt:

$$E = \{\mathbf{p}_0 + \mathbf{f}_1 \cos t + \mathbf{f} \sin t | 0 \leq t \leq 2\pi\}.$$

Umgekehrt ist für einen Vektor $\mathbf{p}_0$ und zwei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2$ die obige Punktmenge E eine Ellipse.

$P_0 : \mathbf{p}_0$ heißt der Mittelpunkt der Ellipse,
$\mathbf{f}_1, \mathbf{f}_2$ heißen zueinander **konjugierte Halbmesser** von E.
$\mathbf{f}_1, \mathbf{f}_2$ sind i.a. nicht zueinander orthogonal.

Aber es gilt:
Die Tangente in $P_1 : \mathbf{p}_0 + \mathbf{f}_1$ ist parallel zu $\mathbf{f}_2$ und die Tangente in $P_2 : \mathbf{p}_0 + \mathbf{f}_2$ ist parallel zu $\mathbf{f}_1$. Falls $\mathbf{f}_1 \perp \mathbf{f}_2$ ist, heißen $\mathbf{p}_0 \pm \mathbf{f}_1, \mathbf{p}_0 \pm \mathbf{f}_2$ die (vier) **Scheitel** der Ellipse.

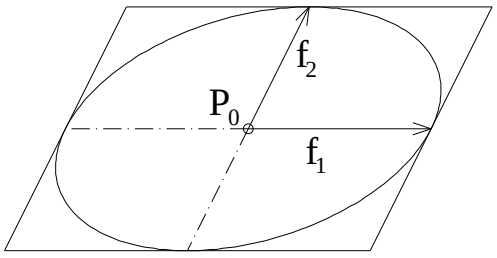


Abbildung 5.2: konjugierte Halbmesser einer Ellipse

Wir verwenden die hier beschriebene Darstellung einer Ellipse, um zunächst Punkte eines Ellipsenbogens zu berechnen:

Gegeben sei eine Ellipse E durch ihren Mittelpunkt $P_0 : \mathbf{p}_0$ und zwei zueinander konjugierte Punkte $P_1 : \mathbf{p}_1, P_2 : \mathbf{p}_2$, d.h. $\mathbf{f}_1 = \mathbf{p}_1 - \mathbf{p}_0$ und $\mathbf{f}_2 = \mathbf{p}_2 - \mathbf{p}_0$ sind zueinander konjugierte Halbmesser. Die $P_1 \to P_2$-Richtung auf der Ellipse sei diejenige Richtung, die man "gehen" muß, um über den kleineren Winkel von P_1 nach P_2 zu kommen. Der Bogen, von dem Punkte berechnet werden sollen, erstrecke sich von dem Punkt P_3 nach dem Punkt P_4 in $P_1 \to P_2$-Richtung.

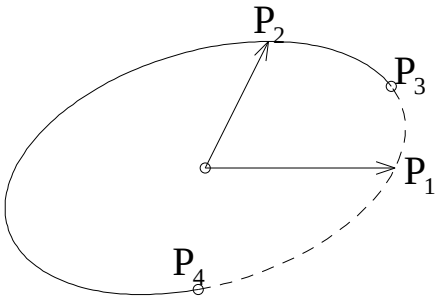


Abbildung 5.3: Ellipsenbogen von P_3 nach P_4 in $P_1 \to P_2$-Richtung

Zum Unterprogamm `ellipse_pts2d(p0,p1,p2,t3,t4, p,n2)`:

a) $P_i = \mathbf{p}_i, i = 0, 1, 2$, sind Mittelpunkt und zwei konjugierte Punkte der Ellipse.

b) Die Parameter von Anfangs- und Endpunkt P_3 und P_4 sind $t3$ und $t4$.

c) dt ist der Parameterabstand der zu berechnenden Punkte.

d) Ist t der Parameter eines Punktes, so ergeben sich die sin- und cos- Werte des darauffolgenden Punktes aus

$$\begin{aligned} \cos(t+dt) &= \cos(t)\cos(dt) - \sin(t)\sin(dt) \\ \sin(t+dt) &= \sin(t)\cos(dt) + \cos(t)\sin(dt) \end{aligned}$$

e) Will man Punkte des komplementären Bogens haben, so muß man $t3$ und $t4$ vertauschen.

```
procedure ellipse_pts2d(p0,p1,p2: vt2d; t3,t4: real; var p: vts2d; var n2: integer);
{Berechnet Punkte p[0],...,p[n2] der Ellipse mit Mittelpunkt p0, den
 konjugierten  Punkten p1,p2 im Parameterbereich  t3,t4  in p1->p2 Richtung
 mit dt=1/100. Beliebiger Punkt: p0 + (p1-p0)*cos(t) + (p2-p0)*sin(t).}
```

Aus dem Unterprogramm `ellipse_pts2d` leiten wir jetzt einige Zeichenprogramme für Ellipsen und Ellipsenbögen ab. Dabei beachte man die erweiterte Bedeutung des Parameters `style`. Falls `style=10` ist, wird der Bogen durchgezogen und der **komplementäre Bogen** zusätzlich **gestrichelt**.

```
procedure ellipse_arc2d(p0,p1,p2: vt2d; t3,t4: real; style: integer);
 {Zeichnet den durch p0,p1,p2,t3,t4 definierten Ellipsenbogen.
  Falls style=10 ist wird der Rest gestrichelt.}
{*************}
procedure circle_arc2d(xm,ym,r,t1,t2: real; style: integer);
 {Zeichnet einen Kreisbogen im Intervall  t1,t2.}
{*************}
procedure ellipse_arc2d_ax(xm,ym,a,b,w,t1,t2: real; style: integer);
 {Zeichnet einen Ellipsenbogen (s.ellipse2d_ax) im Intervall  t1,t2.}
{*************}
```

```
procedure ellipse2d_ax(xm,ym,a,b,w: real; style: integer);
 {Zeichnet eine Ellipse, die aus x*x/(a*a) + y*y/(b*b) = 1 durch Drehung um
  den Winkel w und anschliessender Translation (x,y) -> (x+xm,y+ym) entsteht.}
{*************}
```

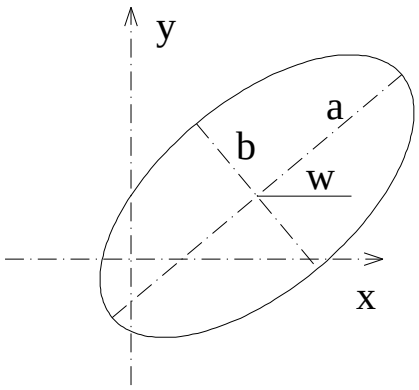


Abbildung 5.4: zu `ellipse_arc2d_ax`

Aufgabe 5.1 *Schreibe ein Programm, das die folgende Zeichnung erzeugt.*

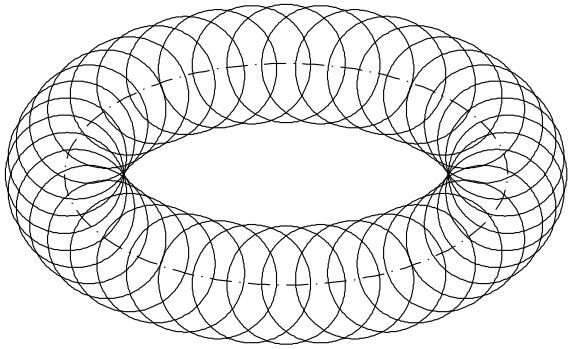


Abbildung 5.5: Kreise mit Mittelpunkte auf einer Ellipse (Aufgabe 5.1)

5.2 Hyperbeln

Bezüglich der Basis $B = \{\mathbf{e}_1, \mathbf{e}_2\}$ sei $H_1 = \{(x, y) \in \mathbb{R}^2 | x^2 - y^2 = 1\}$ (Einheitshyperbel).
Ein affines Bild von H_1 heißt **Hyperbel**.
Ähnlich wie bei der Ellipse können wir folgende Aussage formulieren:

Zu jeder Hyperbel H gibt es einen Vektor $\mathbf{p}_0$ und zwei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2$, sodaß gilt:

$$H = \{\mathbf{p}_0 \pm \mathbf{f}_1 \cosh t + \mathbf{f}_2 \sinh t | -\infty < t < \infty\}.$$

Umgekehrt ist für jeden Vektor $\mathbf{p}_0$ und zwei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2$ die obige Punktmenge H eine Hyperbel. (cosh bzw. sinh ist die hyperbolische Kosinus- bzw. Sinus-Funktion.)

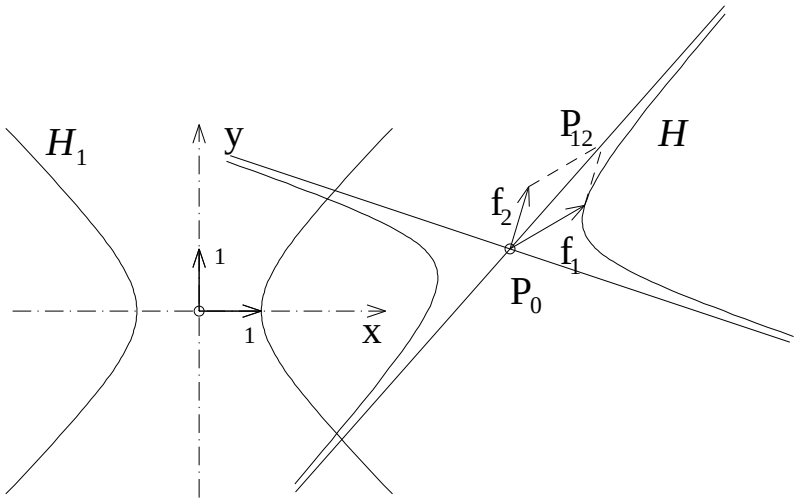


Abbildung 5.6: Einheitshyperbel und ein affines Bild

$P_0 : \mathbf{p}_0$ heißt der Mittelpunkt der Hyperbel.
$\mathbf{f}_1$ heißt ein Halbmesser von H.
$\mathbf{f}_2$ heißt ein zu $\mathbf{f}_1$ **konjugierter Halbmesser**.
Im Gegensatz zur Ellipse liegt der Punkt $P_2 : \mathbf{p}_0+\mathbf{f}_2$ nicht auf H, sondern auf der zu H **konjugierten Hyperbel**

$$H' := \{\mathbf{p}_0 + \mathbf{f}_1 \sinh t \pm \mathbf{f}_2 \cosh t | \ldots\}$$

$\mathbf{f}_1, \mathbf{f}_2$ sind i.a. nicht zueinander orthogonal. Aber es gilt:
Die Tangente in $P_1 : \mathbf{p}_0 + \mathbf{f}_1$ ist parallel zu $\mathbf{f}_2$ und $P_{12} : \mathbf{p}_0 + \mathbf{f}_1 + \mathbf{f}_2$ liegt auf einer Asymptote.

Falls $\mathbf{f}_1 \perp \mathbf{f}_2$ ist, heißen $\mathbf{p}_0 \pm \mathbf{f}_1$ die (zwei) Scheitel der Hyperbel H und $\mathbf{p}_0 \pm \mathbf{f}_2$ ihre **Nebenscheitel**.
Das folgende Unterprogramm `hyperb_pts2d` berechnet n2 Punkte des Hyperbelbogens

$$\begin{aligned} H_+ &:= \{\mathbf{p}_0 + \mathbf{f}_1 \cosh t + \mathbf{f}_2 \sinh t | \ldots\}, \quad \text{falls } ia = 1, \text{ bzw.} \\ H_- &:= \{\mathbf{p}_0 - \mathbf{f}_1 \cosh t + \mathbf{f}_2 \sinh t | \ldots\}, \quad \text{falls } ia = -1 \end{aligned}$$

im Parameterbereich $[t3, t4]$ und Parameterabstand $dt = 0.05$.

`hyberb_arc2d` schließlich verbindet die mit `hyperb_pts2d` berechneten Punkte. Das Unterprogramm `hyperb_arc2d_ax` entspricht dem `ellipse_arc2d_ax` für Ellipsen.

```
procedure hyperb_pts2d(p0,p1,p2: vt2d; t3,t4 : real; ia : integer; var p: vts2d; var n2 : integer);
 {Berechnet Punkte p[0] ,...,p[n2]  des Hyperbelbogens der Hyperbel mit Mittelpunkt p0
 Hyperbelpunkt p1 und dem dazu konjugierten Punkt p2 im Parameterintervall [t3,t4].
 ia=1 bzw. ia=-1 : Ast, auf dem p1  liegt bzw. nicht.
 Beliebiger Punkt : p0 + ia*(p1-p0)*cosh(t) + (p2-p0)*sinh(t). Es ist dt=0.05.}
{*************}
procedure hyperb_arc2d(p0,p1,p2: vt2d; t3,t4: real; ia,style: integer);
{Verbindet die mit hyperb_pts2d berechneten Punkte.}
```

```
{*************}
procedure hyperb_arc2d_ax(xm,ym,a,b,w,t1,t2: real; ia,style: integer);
{Zeichnet einen Hyperbelbogen der aus x=a*cosh(t), y=b*sinh(t), t aus
 [t1,t2], durch Drehung und anschliessende Translation um (xm,ym) entsteht.}
{*************}
```

5.3 Parabeln

Bezüglich der Basis $B = \{\mathbf{e}_1, \mathbf{e}_2\}$ sei $P_1 := \{(x, y) \in \mathbb{R}^2 | y = x^2\}$ (Einheitsparabel).
Ein affines Bild von P_1 heißt **Parabel**.
Für Parabeln gilt :
Zu jeder Parabel P gibt es einen Vektor $\mathbf{p}_0$ und zwei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2$, so daß gilt:

$$P = \{\mathbf{p}_0 + \mathbf{f}_1 t + \mathbf{f}_2 t^2 | t \in \mathbb{R}\}.$$

Umgekehrt ist für jeden Vektor $\mathbf{p}_0$ und zwei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2$ die obige Menge P eine Parabel.

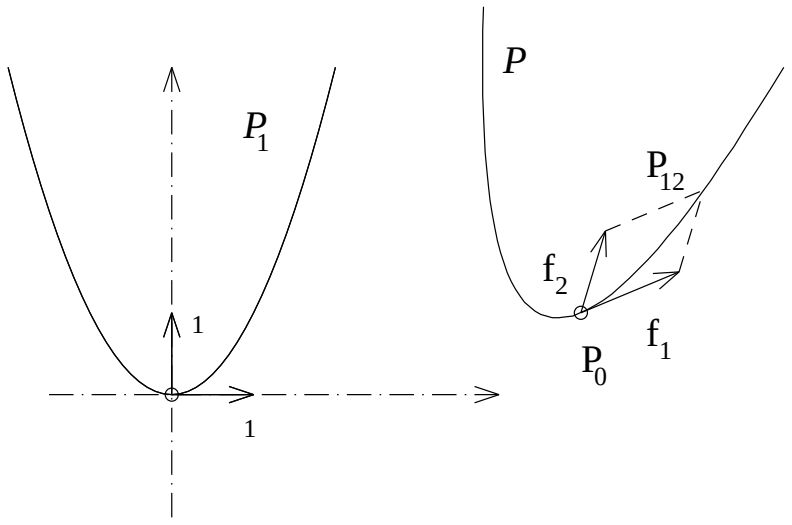


Abbildung 5.7: Parabel und ein affines Bild

$P_0 : \mathbf{p}_0$ ist ein Punkt der Parabel,
$\mathbf{f}_1$ die Richtung der Tangente in P_0, $\mathbf{f}_2$ die Richtung der **Parabelachse** und
$P_{12} : \mathbf{p}_0 + \mathbf{f}_1 + \mathbf{f}_2$ ein weiterer Punkt der Parabel.
$\mathbf{f}_1, \mathbf{f}_2$ sind i. a. nicht zueinander orthogonal.
Falls $\mathbf{f}_1 \perp \mathbf{f}_2$ ist, heißt P_0 der **Scheitel** der Parabel.
Das folgende Unterprogramm `parab_pts2d` berechnet Punkte des Parabelbogens

$$P = \{\mathbf{p}_0 + \mathbf{f}_1 t + \mathbf{f}_2 t^2 | t_3 \leq t \leq t_4\}.$$

`parab_arc2d` verbindet die mit `parab_pts2d` berechneten Punkte. Und `parab_arc2d_ax` entspricht `ellipse_arc2d_ax`.

```
procedure parab_pts2d(p0,p1,p2: vt2d; t3,t4: real; var p: vts2d; var n2: integer);
 {Berechnet Punkte p[0],...,p[n2]  der Parabel mit Punkt p0, und p1 auf der
 Tangente in p0, p2 auf der Achse durch p0 im Parameterintervall [t3,t4] .
 Beliebiger Punkt : p0+t*(p1-p0)+t*t*(p2-p0).}
{*************}
procedure parab_arc2d(p0,p1,p2: vt2d; t3,t4: real; style: integer);
 {Verbindet die mit parab_pts2d berechneten Punkte.}
{*************}
procedure parab_arc2d_ax(a,x0,y0,w,t1,t2: real; style: integer);
 {Zeichnet einen Parabelbogen der aus y=a*x*x, x aus  [t1,t2] , durch Drehung
 um Winkel w und anschliessende Translation um (x0,y0) entsteht.}
{*************}
```

Die Programme zu den Kegelschnitten sind in der Datei `proc_ks.pas` enthalten.

5.4 Ellipsen, Hyperbeln, Parabeln im Raum

Die in 5.1 eingeführte Beschreibung von Ellipsen, Hyperbeln und Parabeln mit Hilfe der Vektoren $\mathbf{p}_0, \mathbf{f}_1, \mathbf{f}_2$ läßt sich analog auf den Raum übertragen. Hierzu muß man nur $\mathbf{p}_0, \mathbf{f}_1, \mathbf{f}_2$ aus dem $\mathbb{R}^3$ wählen. Zur Berechnung von Punkten eines Ellipsen- bzw. Hyperbel- bzw. Parabelbogens im $\mathbb{R}^2$ haben wir Anfangs- und Endparameter benutzt. Im Rahmen der Darstellenden Geometrie wird die Begrenzung eines Kegelschnittbogens im $\mathbb{R}^3$ meistens durch Anfangs- und End**punkte** gegeben sein. Deshalb werden in den folgenden Programmen Anfangs- und Endpunkte als Parameter übergeben. Um allerdings auch schnell die Punkte eines Kreisbogens über ein bestimmtes Winkelintervall berechnen zu können, geben wir für Ellipsenbögen beide Versionen an. Der einzige Unterschied zu den ebenen Unterprogrammen (mit Anfangs- und Endparameter) besteht darin, daß Anfangs- und Endparamter aus Anfangs- und Endpunkte erst berechnet werden müssen.

5.4.1 Ellipse und Kreis im Raum

E sei die Ellipse im $\mathbb{R}^3$ mit Mittelpunkt $P_0 : \mathbf{p}_0$, konjugierte Punkte $P_1 : \mathbf{p}_1, P_2 : \mathbf{p}_2$, Anfangspunkt $P_3 : \mathbf{p}_3$, und Endpunkt $P_4 : \mathbf{p}_4$. Dann sind $\mathbf{f}_1 := \mathbf{p}_1 - \mathbf{p}_0, \mathbf{f}_2 := \mathbf{p}_2 - \mathbf{p}_0$ konjugierte Halbmesser und es ist $E = \{\mathbf{p}_0 + \mathbf{f}_1 \cos t + \mathbf{f}_2 \sin t | t_3 \leq t \leq t_4\}$. Die cos- und sin-Werte von Anfangs- und Endparameter t_3, t_4 ergeben sich mit Hilfe des Unterprogramms `ptco_plane3d`, denn es ist $\mathbf{p}_i = \mathbf{p}_0 + \mathbf{f}_1 \cos t_i + \mathbf{f}_2 \sin t_i$, $i = 3, 4$. t_3, t_4 berechnet man dann mit `polar_angle`.

```
procedure ellipse_pts3d(p0,p1,p2,p3,p4: vt3d; var p: vts3d; var n2: integer);
 {Berechnung von Punkten des Ellipsenbogens der Ellipse mit Mittelpunkt
  p0 und den konjugierten Punkten p1, p2 und Anfangspunkt p3 bzw. Endpunkt p4.
  Falls p3=p4 ist, wird eine "ganze"  Ellipse berechnet.}
{************}
procedure ellipse_pts3d_t3t4(p0,p1,p2: vt3d; t3,t4 : real;
                                          var p: vts3d; var n2: integer);
 {Berechnung von Punkten des Ellipsenbogens der Ellipse mit Mittelpunkt p0 und
 den konjugierten Punkten p1, p2 und Anfangsparameter t3 bzw. Endparameter t4.}
{************}
```

Da ein Kreis im Raum schon durch seinen Mittelpunkt $\mathbf{p}_0$, den Radius r und eine Normale $\mathbf{n}$ der Ebene, in der er liegt, bestimmt ist, wollen wir hier für diesen Fall ein besonderes Unterprogramm angeben. Um `ellipse_pts3d` benutzen zu können, müssen wir konjugierte Punkte, d.h. Punkte, die auf zueinander senkrechten Halbmessern des Kreises liegen, bestimmen. Dies geschieht in Abhängigkeit der Komponenten der Normalen.

```
procedure circle_pts3d(p0,nv: vt3d; r: real; var p: vts3d; var n2: integer);
{Berechnet Punkte eines Kreises mit Mittelpunkt p0, Radius r, Normale nv}
var cc: real;  ff1,f1,f2,p1,p2 : vt3d;
begin
  normalize3d(nv);
  if (abs(nv.x) > 0.5) or (abs(nv.y) > 0.5) then put3d(nv.y,-nv.x,0, ff1)
                                           else put3d(0,nv.z,-nv.y, ff1);
  scale3d(r/length3d(ff1),ff1, f1);
  vectorp(nv,f1, f2);  sum3d(p0,f1, p1);  sum3d(p0,f2, p2);
  ellipse_pts3d(p0,p1,p2,p1,p1, p,n2);
end;  {circle_pts3d}
```

5.4.2 Schnitt Ellipse-Ebene

Gegeben: Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d,\ \mathbf{n} = (a,b,c) \neq (0,0,0)$,
Ellipse $E : \{\mathbf{p}_0 + \mathbf{f}_1 \cos t + \mathbf{f}_2 \sin t | \ldots\}$.
Gesucht: Schnittpunkte der Ellipse E mit der Ebene ε.

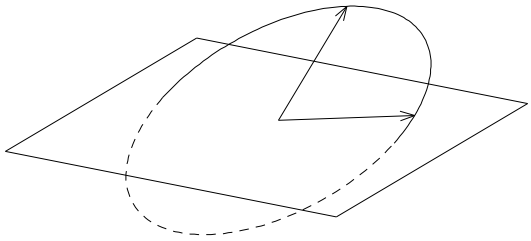


Abbildung 5.8: Ellipse "vor" einer Ebene

Ein Punkt $\mathbf{x}$ der Ebene ε_0, die die Ellipse E enthält, läßt sich durch $\mathbf{x} = \mathbf{p}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta, \quad \xi, \eta \in \mathbb{R}$, (Parameterdarstellung) beschreiben. Die Parameter der Punkte der Schnittgerade $g : \varepsilon \cap \varepsilon_0$ genügen also der Gleichung $\mathbf{n} \cdot \mathbf{f}_1\xi + \mathbf{n} \cdot \mathbf{f}_2\eta = d - \mathbf{p}_0 \cdot \mathbf{n}$. Die Parameter der Schnittpunkte der Gerade g mit der Ellipse E ergeben sich aus dem Gleichungssystem

$$\begin{array}{rcl} \gamma_1\xi + \gamma_2\eta & = & \gamma_3 \quad \text{mit } \gamma_1 = \mathbf{n} \cdot \mathbf{f}_1,\ \gamma_2 = \mathbf{n} \cdot \mathbf{f}_2,\ \gamma_3 = d - \mathbf{n} \cdot \mathbf{p}_0 \\ \xi^2 + \eta^2 & = & 1 \end{array}$$

Die Lösungen dieses Systems berechnet man mit `is_unitcircle_line`. Sind $(\xi_1, \eta_1), (\xi_2, \eta_2)$ solche Lösungen, so erhält man die Schnittpunkte

$$Q_1 = \mathbf{p}_0 + \mathbf{f}_1\xi_1 + \mathbf{f}_2\eta_1, \quad Q_2 = \mathbf{p}_0 + \mathbf{f}_1\xi_2 + \mathbf{f}_2\eta_2$$

Das Unterprogramm `is_ellipse_plane` berechnet die Schnittpunkte Q_1, Q_2 (falls sie existieren) so, daß der Bogen $Q_1 \to Q_2$ "vor" der Ebene ε liegt. Die im Fall $\gamma_3 \neq 0$ (Mittelpunkt liegt nicht in der Ebene) berechnete Zahl "test" ist der orientierte Flächeninhalt des Dreiecks $(0,0)(\xi_1, \eta_1), (\xi_2, \eta_2)$. Im Fall $\gamma_3 = 0$ wird der Testpunkt $(-\eta_1, \xi_1)$ in den Ausdruck $\xi\gamma_1 + \eta\gamma_2 - \gamma_3$ eingesetzt. nis gibt die Anzahl der Schnittpunkte an. $nis = 3$ bedeutet: die Ellipse liegt in der Ebene ε.

```
procedure is_ellipse_plane(pc0,pc1,pc2,nv: vt3d; d: real; var q1,q2: vt3d;
                                                          var nis: integer);
 {Berechnet die Schnittpunkte q1,q2 der durch pc0,pc1,pc2 bestimmten Ellipse
  mit der Ebene nv*x = d. Der Bogen q1->q2 ist "vor" der Ebene. }
 var f1,f2 : vt3d;     i : integer;       change12 : boolean;
     g1,g2,g3,xi1,xi2,eta1,eta2,test : real;
 begin
```

```
   diff3d(pc1,pc0, f1);    diff3d(pc2,pc0, f2);
   g1:= scalarp3d(nv,f1);   g2:= scalarp3d(nv,f2);
   g3:= d - scalarp3d(nv,pc0);
   if (abs(g1)<eps3) and (abs(g2)<eps3)  then  { Ellipse parallel zur Ebene }
        begin  nis:= 0;   if g3=0 then  nis:= 3;   end
      else
        begin
        is_unitcircle_line(g1,g2,g3, xi1,eta1,xi2,eta2,nis);
        if nis > 0  then
              begin
              if nis=2 then     { Bogen q1->q2  liegt vor der Ebene }
                 begin
                 change12:= false;
                 if abs(g3)>eps5 then
                      begin
                      test:= xi1*eta2-eta1*xi2;
                      if (g3*test<0) then  change12:= true;
                      end
                     else
                      if (-eta1*g1 + xi1*g2 - g3 <0) then  change12:= true;
                 if change12 then
                      begin  change1d(xi1,xi2); change1d(eta1,eta2);  end;
                 end; { nis=2 }
              lcomb3vt3d(1,pc0, xi1,f1, eta1,f2, q1);
              lcomb3vt3d(1,pc0, xi2,f1, eta2,f2, q2);
              end; { nis>0 }
        end;
 end;   { is_ellipse_plane }
{*************}
```

5.4.3 Hyperbel und Parabel im Raum

Der Aufbau der folgenden Unterprogramme für Hyperbelbogen bzw. Parabelbogen ist analog dem des Programms `ellipse_pts3d`. Während sich in `parab_pts3d` Anfangs- und Endparameter direkt aus `ptco_plane3d` ergeben, muß man in `hyperb_pts3d` die Beziehung

$$\operatorname{arsinh}(x) = \ln(x + \sqrt{x^2 + 1})$$

verwenden. Sie entspricht der Anwendung von `polar_angle` in `ellipse_pts3d`.

```
procedure hyperb_pts3d(p0,p1,p2,p3,p4: vt3d; var p: vts3d; var n2: integer);
 {Berechnet Punkte des Hyperbelbogens p0+(p1-p0)*cosh(t)+(p2-p0)*sinh(t) mit
  Anfangspunkt p3 und Endpunkt p4.}
{***********}
procedure parab_pts3d(p0,p1,p2,p3,p4: vt3d; var p: vts3d; var n2: integer);
 {Berechnet Punkte des Parabelbogens p0+t*(p1-p0)+t*t*(p2-p0) zwischen p3 und p4.}
{***********}
```

5.5 Parallelprojektion von Ellipse, Hyperbel und Parabel

Eine senkrechte Parallelprojektion Φ ist eine lineare Abbildung und bildet daher einen Ellipsenbogen

$$B = \{\mathbf{p}_0 + \mathbf{f}_1 \cos t + \mathbf{f}_2 \sin t \mid t_3 \le t \le t_4\}$$

auf

$$B' = \{\Phi(\mathbf{p}_0) + \Phi(\mathbf{f}_1)\cos t + \Phi(\mathbf{f}_2)\sin t \mid t_3 \le t \le t_4\}$$

ab. Wir müssen also nicht erst Punkte des Bogens B berechnen und diese Punkt für Punkt projizieren. Es genügt, die Bilder von $\mathbf{p}_0, \mathbf{f}_1$ und $\mathbf{f}_2$ zu bestimmen und dann B' mit `ellipse_arc2d` zu zeichnen. B' ist nur dann wieder ein Ellipsenbogen, wenn $\Phi(\mathbf{f}_1)$ und $\Phi(\mathbf{f}_2)$ linear unabhängig sind. Im anderen Fall ist B' eine Strecke. Diese beiden Fälle müssen aber nicht gesondert betrachtet werden; das Unterprogramm `ellipse_arc2d` wird in jedem Fall das Richtige zeichnen. Die hierzu notwendigen Parametergrenzen t_3, t_4 werden wie bei `ellipse_pts3d` mit Hilfe von `ptco_plane3d` berechnet.
Das Unterprogramm `pp_ellipse_arc` projiziert den Ellipsenbogen der Ellipse mit Mittelpunkt $\mathbf{p}_0$, den konjugierten Punkten $\mathbf{p}_1, \mathbf{p}_2$ und den **Anfangs**- und **Endpunkten** $\mathbf{p}_3, \mathbf{p}_4$. `pp_ellipse` projiziert eine ganze Ellipse. `pp_circle` projiziert einen Kreis mit Mittelpunkt $\mathbf{p}_0$, Radius r und Normale $\mathbf{n}$.

```
procedure pp_ellipse_arc(p0,p1,p2,p3,p4 : vt3d; style : integer);
 {Projiziert den Ellipsenbogen p0+f1*cos(t)+f2*sin(t), fi= pi-p0,
 zwischen p3 und p4 in p1->p2 Richtung. style: s. ellipse_arc2d.}
{*****************}
procedure pp_ellipse(p0,p1,p2 : vt3d; style : integer);
 {Projiziert die durch p0,p1,p2 bestimmte Ellipse.}
{*****************}
procedure pp_circle(p0,nv : vt3d; r : real; style: integer);
{Projiziert den Kreis mit Mittelpunkt p0, Normale nv und Radius r.}
{*****************}
```

Die analogen Unterprogramme für Hyperbeln und Parabeln sind `pp_hyperb_arc` und `pp_parab_arc`.

```
procedure pp_hyperb_arc(p0,p1,p2,p3,p4 : vt3d; style : integer);
 {Projiziert den Hyperbelbogen p0+f1*cosh(t)+f2*sinh(t),  fi= pi-p0,
  Anfangs- bzw. Endpunkt p3 bzw. p4.}
{****************}
procedure pp_parab_arc(p0,p1,p2,p3,p4 : vt3d; style : integer);
{Projiziert den Parabelbogen p0+t*f1+t*t*f2 zwischen p3 und p4, fi= pi-p0.}
{****************}
```

Die Prozedur `pp_ellipse_before_plane` ist eine Erweiterung von `is_ellipse_plane`.

```
procedure pp_ellipse_before_plane(pc0,pc1,pc2,nv: vt3d; d: real;
                                              side,style: integer);
{Projiziert den Teil der durch pc0,pc1,pc2 bestimmten Ellipse, f"ur den
 side*(nv*x-d) > 0 ist und, falls style = 10, strichelt den Rest.}
 var nvv,f1,f2: vt3d;      p0,p1,p2 : vt2d;      nis : integer;
     dd,g1,g2,g3,xi1,xi2,eta1,eta2,s3,s4,test : real;
 begin
   scale3d(side,nv, nvv);    dd:= side*d;
   diff3d(pc1,pc0, f1);      diff3d(pc2,pc0, f2);
   g1:= scalarp3d(nvv,f1);   g2:= scalarp3d(nvv,f2);
   g3:= dd - scalarp3d(nvv,pc0);
   if (abs(g1)<eps4) and (abs(g2)<eps4)
      then                                    {Ellipse parallel zur Ebene !}
        begin
        if g3<=0                    then   pp_ellipse(pc0,pc1,pc2,0);
        if (style=10) and (g3>0)    then   pp_ellipse(pc0,pc1,pc2,1);
        end
```

```
      else
        begin
        is_unitcircle_line(g1,g2,g3, xi1,eta1,xi2,eta2,nis);
        if nis<2  then
           begin
           if g3<=0                    then  pp_ellipse(pc0,pc1,pc2,0);
           if (style=10) and (g3>0)    then  pp_ellipse(pc0,pc1,pc2,1);
           end
          else     {nis=2}
           begin
           s3 := polar_angle(xi1,eta1);  s4:= polar_angle(xi2,eta2);
           if abs(g3)>eps5 then
                begin
                test:= xi1*eta2-eta1*xi2;
                if (g3*test<0) then  change1d(s3,s4);
                end
               else
                if (-eta1*g1 + xi1*g2 - g3 <0) then  change1d(s3,s4);
           pp_vt3d_vt2d(pc0,p0);  pp_vt3d_vt2d(pc1,p1);  pp_vt3d_vt2d(pc2,p2);
           ellipse_arc2d(p0,p1,p2,s3,s4,style)
           end;  { nis=2 }
        end;
 end;   { pp_ellipse_before_plane }
{**************}
```

Die Programme dieses Abschnitts sind in der Datei `proc_pks.pas` enthalten.

5.6 Zentralprojektion von Ellipse, Hyperbel, Parabel

5.6.1 Zentralprojektion einer Ellipse "vor" der Verschwindungsebene

Die Darstellung einer Ellipse in Zentralprojektion ist nicht mehr so einfach wie bei der Parallelprojektion, da eine Zentralprojektion keine lineare Abbildung mehr ist. Das Bild einer Ellipse muß nicht wieder eine Ellipse sein, sondern kann auch eine Hyperbel oder Parabel sein. Welcher Fall vorliegt, richtet sich danach, ob die abzubildende Ellipse die Verschwindungsebene in keinem, einem oder zwei Punkten schneidet. Diese lästige Unterscheidung kann man dadurch umgehen, indem man mit `ellipse_pts3d` Punkte der Ellipse im Raum berechnet und mit Hilfe eines Clipping-"Filters" nur die Punkte projiziert und verbindet, die vor der Verschwindungsebene liegen. Dieses Verfahren wird in 4.2 benutzt. In den meisten Fällen werden wir es mit Ellipsen zu tun haben, die vor der Verschwindungsebene liegen (z. B. Ellipsoid mit Breiten- und Längen-"kreisen"). Für solche Fälle werden wir uns zunächst ein Unterprogramm überlegen, das den Mittelpunkt, zwei konjugierte Punkte und Anfangs- und Endparameter des Bildellipsenbogens bestimmt und den Ellipsenbogen dann mit `ellipse_arc2d` zeichnet. Der Vorteil der letzteren Methode liegt in der größeren Geschwindigkeit, mit der die Zeichnung erstellt wird.

Gegeben: Ellipse $E : \mathbf{x} = \mathbf{p}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi$ vor der Verschwindungsebene ε_v mit Anfangs- und Endpunkte P_3, P_4 eines Bogens. Zentralprojektion Φ.

Gesucht: Mittelpunkt $Q_0 : \mathbf{q}_0$, zwei konjugierte Punkte $Q_1 : \mathbf{q}_1$, $Q_2 : \mathbf{q}_2$ der Bildellipse und Anfangs- und Endparameter t_3, t_4.

Idee: Geraden, die sich in einem Punkt der Verschwindungsebene schneiden, sind nach der Projektion parallel. Zwei Sehnen der Ellipse sind nach der Abbildung 5.9 konjugiert, wenn sich die Tangenten in den Endpunkten der einen Sehne und die Gerade der anderen Sehne auf der Verschwindungsebene in einem gemeinsamen Punkt schneiden.

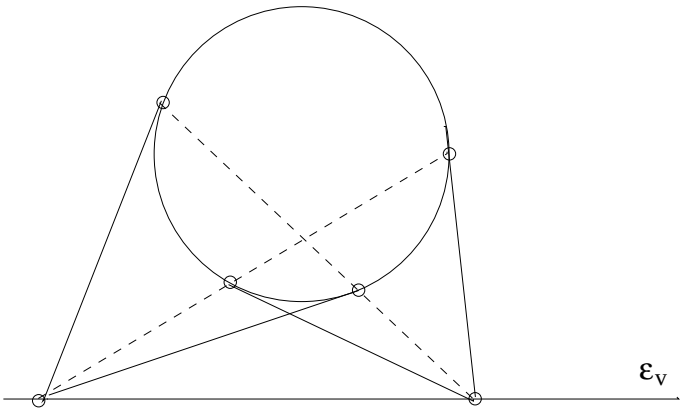


Abbildung 5.9: Zentralprojektion einer Ellipse

(1) Die Gleichung der Verschwindungsebene ε_v ist $(\mathbf{x}-\mathbf{h})\cdot\mathbf{n}_0 = \delta$. ($\mathbf{n}_0$ ist die Normale der Bildtafel, δ ist die Distanz und $H : \mathbf{h}$ ist der Hauptpunkt.)

(2) Die Parameterdarstellung der Ebene ε, die die Ellipse E enthält ist $\mathbf{x} = \mathbf{p}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta$.

(3) Aus (1) und (2) ergibt sich die Gleichung der Verschwindungsgerade $g_v := \varepsilon_v \cap \varepsilon$ in $\xi - \eta-$Koordinaten:

$$\gamma_1\xi + \gamma_2\eta = \gamma_3 \quad \text{mit} \;\; \gamma_1 := \mathbf{f}_1 \cdot \mathbf{n}_0,\; \gamma_2 := \mathbf{f}_2 \cdot \mathbf{n}_0,\; \gamma_3 := \delta - (\mathbf{p}_0 - \mathbf{h}) \cdot \mathbf{n}_0$$

Die Gleichung der Ellipse ist $\xi^2 + \eta^2 = 1$.
Falls $\gamma_1 = \gamma_2 = 0$ ist, ist die Ellipse **parallel** zur Bildtafel. In diesem Fall werden Mittelpunkt und konjugierte Punkte auf ebensolche abgebildet.
Falls $(\gamma_1, \gamma_2) \neq (0, 0)$ ist:

(4) Lotfußpunkt des Lotes l_v von P_0 auf g_v (in $\xi - \eta-$Koordinaten):

$$V : (\gamma_1, \gamma_2)\gamma_3/(\gamma_1^2 + \gamma_2^2)$$

(5) Die Berührpunkte $Q_1 : (\xi_1, \eta_1),\; Q_1' : (\xi_1', \eta_1')$ der Tangenten durch V ergeben sich aus dem Gleichungssystem

$$\begin{array}{rcll} \xi_v\xi + \eta_v\eta & = & 1 & (\text{“Polare” des Punktes } V = (\xi_v, \eta_v)) \\ \xi^2 + \eta^2 & = & 1 & \end{array}$$

Mit $\omega := \sqrt{\gamma_3^2/(\gamma_1^2 + \gamma_2^2) - 1}$ erhält man

$$Q_1 : ((\gamma_1 + \gamma_2\omega)/\gamma_3,\; (\gamma_2 - \gamma_1\omega)/\gamma_3) \quad \text{und} \quad Q_1' : ((\gamma_1 - \gamma_2\omega)/\gamma_3,\; (\gamma_2 + \gamma_1\omega)/\gamma_3)$$

(6) Schnittpunkte des Lotes l_v mit dem “Kreis” $\xi^2 + \eta^2 = 1$:

$$Q_2 : (\gamma_1, \gamma_2)/\sqrt{\gamma_1^2 + \gamma_2^2} \qquad Q_2' : -(\cdots, \cdots)/\sqrt{\cdots}$$

(in $\xi - \eta-$Koordinaten).

Die Tangenten in Q_2, Q_2' sind parallel zur Sehne $\overline{Q_1Q_1'}$ und zu g_v.
Die Tangenten und die Sehne "schneiden" sich im "Fernpunkt" von g_v. (s. Abb. 5.10)

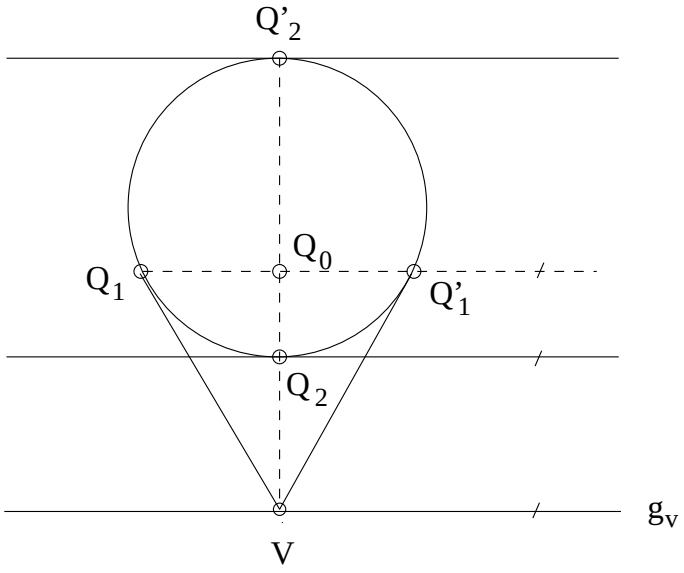


Abbildung 5.10: Urbilder konjugierter Durchmesser einer Ellipse

(7) Die Sehnen $\overline{Q_1Q_1'}$ und $\overline{Q_2Q_2'}$ werden auf **konjugierte** Durchmesser der Bildellipse abgebildet. Also ist der Schnittpunkt Q_0 dieser Sehnen das Urbild des **Mittelpunktes** der Bildellipse. Es ist

$$Q_0 : (\gamma_1, \gamma_2)/\gamma_3 \qquad (\text{in } \xi - \eta-\text{Koordinaten}).$$

Die Bildellipse ist also durch die Bilder der Punkte Q_0, Q_1, Q_2 eindeutig bestimmt.

(8) Die $x - y - z-$Koordinaten der Punkte Q_0, Q_1, Q_2 ergeben sich aus ihren $\xi - \eta-$Koordinaten:

$$Q_i : \mathbf{p}_0 + \mathbf{f}_1\xi_i + \mathbf{f}_2\eta_i \qquad i = 0, 1, 2.$$

(9) Bestimmung der Anfangs- und Endparameter t_3, t_4:

(9.1) Beschreibung der Punkte $Q_i : \mathbf{q}_i$ bzgl. der Basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{n}_0\}$:

$$Q_i : \bar{\mathbf{q}}_i := (\bar{x}_i, \bar{y}_i, \bar{z}_i)$$

mit

$$\bar{x}_i = (\mathbf{q}_i - \mathbf{h}) \cdot \mathbf{e}_1,\ \bar{y}_i = (\mathbf{q}_i - \mathbf{h}) \cdot \mathbf{e}_2,\ \bar{z}_i = (\mathbf{q}_i - \mathbf{h}) \cdot \mathbf{n}_0.$$

(9.2) Die "Perspektivität"

$$\sigma : (\bar{x}, \bar{y}, \bar{z}) \to (\frac{\bar{x}}{1 - \bar{z}/\delta}, \frac{\bar{y}}{1 - \bar{z}/\delta}, \frac{\bar{z}}{1 - \bar{z}/\delta})$$

des $\mathbb{R}^3 \setminus \varepsilon_v$ bildet die Ellipse E auf eine Ellipse $\sigma(E)$ ab, deren Mittelpunkt $\sigma(Q_0)$ ist, und $\sigma(Q_1), \sigma(Q_2)$ sind zwei konjugierte Punkte von $\sigma(E)$. Die Bildkoordinaten eines Punktes $P : (\bar{x}, \bar{y}, \bar{z})$ bzgl. der Zentralprojektion Φ erhält man aus den Koordinaten von $\sigma(P)$ durch weglassen der 3. Koordinate, d.h. Φ ist die Hintereinanderausführung der Perspektivität σ und der Parallelprojektion auf die $\bar{x} - \bar{y}-$Ebene.

(9.3) Anfangs- und Endpunkte P_3, P_4 des gegebenen Ellipsenbogens werden durch σ auf $\sigma(P_3)$, $\sigma(P_4)$ abgebildet.

Wie in `pp_ellipse_arc` berechnet man aus $\sigma(P_3)$ und $\sigma(P_4)$ die Anfangs- und Endparameter t_3 und t_4, die durch die anschließende Parallelprojektion (lineare Abbildung !) nicht mehr verändert werden.

Das folgende Unterprogramm `cp_ellipse_arc` führt die hier beschriebenen Rechnungen durch und zeichnet schließlich das Bild des durch $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4$ gegebenen Ellipsenbogens. `cp_ellipse` projiziert eine ganze Ellipse.

```
procedure cp_ellipse_arc(p0,p1,p2,p3,p4 : vt3d; style : integer);
  {Projektion eines Ellipsenbogens VOR der Verschwindungsebene.}
 var pm0,pm1,pm2,pm3,pm4,qm0,qm1,qm2,qm3,qm4,p34,f1,f2 : vt3d;
    qq0,qq1,qq2 : vt2d;     xi0,xi1,xi2,eta0,eta1,eta2 : real;
    g1,g2,g3,g21,g321,w21,w321,fac,ct3,st3,t3,ct4,st4,t4 : real;
    i : integer;    error : boolean;
 begin
   transf_to_e1e2n0_base(p0, pm0);
   transf_to_e1e2n0_base(p1, pm1);    diff3d(pm1,pm0, f1);
   transf_to_e1e2n0_base(p2, pm2);    diff3d(pm2,pm0, f2);
 {Bestimmung der Urbilder von Mittelp. und zweier konj. Punkte der Bildellipse}
   g1:= f1.z;     g2:= f2.z;
   if (abs(g1)<eps4) and (abs(g2)<eps4) then     {Ellipse parall. z. Bildt.}
      begin
      xi0:=0; eta0:= 0;  xi1:= 1; eta1:= 0;  xi2:= 0;  eta2:= 1;
      end
     else
      begin
      g3:= distance - pm0.z;    g21:= g2*g2 + g1*g1;     g321:= g3*g3/g21 - 1;
      if g321<=0 then
         writeln('Ellipse schneidet die Verschwindungsebene !!')
        else
         begin
         w321:= sqrt(g321);            w21 := sqrt(g21);
         xi0 := g1/g3;                 eta0:= g2/g3;
         xi1 := (g1+g2*w321)/g3;       eta1:= (g2-g1*w321)/g3;
         xi2 := g1/w21;                eta2:= g2/w21;
         end;    {if}
      end;    {if}
 {Urbilder von Mittelpkt und zwei konj. Punkte in e1,e2,n0-Koord.:}
   lcomb3vt3d(1,pm0, xi0,f1, eta0,f2, qm0);
   lcomb3vt3d(1,pm0, xi1,f1, eta1,f2, qm1);
   lcomb3vt3d(1,pm0, xi2,f1, eta2,f2, qm2);
 {Perspektivitaet im R3 mit Zentrum (0,0,distance):}
   fac:= 1/(1-qm0.z/distance);    scale3d(fac,qm0, qm0);
   fac:= 1/(1-qm1.z/distance);    scale3d(fac,qm1, qm1);
   fac:= 1/(1-qm2.z/distance);    scale3d(fac,qm2, qm2);
 {Bestimmung von Anfangs- und Endparameter:}
   t3:= 0 ;   t4:= 0 ;     diff3d(p4,p3, p34);
   if (abs3d(p34)<eps4)  then  t4:= pi2     {ganze Ellipse}
      else
       begin
       transf_to_e1e2n0_base(p3, pm3);
```

```
      transf_to_e1e2n0_base(p4, pm4);
      fac:= 1/(1-pm3.z/distance);    scale3d(fac,pm3, qm3);
      fac:= 1/(1-pm4.z/distance);    scale3d(fac,pm4, qm4);
      diff3d(qm1,qm0, f1);              diff3d(qm2,qm0,f2);
      ptco_plane3d(qm0,f1,f2,qm3 ,ct3,st3, error);  t3:= polar_angle(ct3,st3);
      ptco_plane3d(qm0,f1,f2,qm4 ,ct4,st4, error);  t4:= polar_angle(ct4,st4);
      end;
 {Parallelproj. in e1,e2,n0-Koordinaten = Abschneiden der 3.Koordinate:}
   qq0.x:= qm0.x;   qq0.y:= qm0.y;
   qq1.x:= qm1.x;   qq1.y:= qm1.y;
   qq2.x:= qm2.x;   qq2.y:= qm2.y;
   ellipse_arc2d(qq0,qq1,qq2,t3,t4,style);
 end;    { cp_ellipse_arc }
{**************}
procedure cp_ellipse(p0,p1,p2 : vt3d; style : integer);
  {Projektion einer Ellipse VOR der Verschwindungsebene.}
 begin  cp_ellipse_arc(p0,p1,p2,p1,p1,style);  end;
{*************}
```

5.6.2 Das Unterprogramm cp_ellipse_before_plane

Das zum Unterprogramm pp_ellipse_before_plane analoge Unterprogramm läßt sich bis auf den Schluß (Bestimmung der Anfangs- und Endparameter und Zeichnen) fast unverändert übernehmen. Da eine Zentralprojektion keine lineare Abbildung mehr ist, muß man Anfangs- und Endpunkte zunächst bestimmen und dann den Ellipsenbogen mit cp_ellipse_arc projizieren.

```
procedure cp_ellipse_before_plane(p0,p1,p2: vt3d; nv: vt3d; d: real; side,style: integer);
 {Projiziert den Teil der durch p bestimmten Ellipse, f"ur den side*(nv*x - d) > 0 ist und,
 falls style = 10, strichelt den Rest.}
 var f1,f2,p3,p4,nvv: vt3d;     nis,i : integer;     g1,g2,g3,xi1,xi2,eta1,eta2,test,dd : real;
 begin
   scale3d(side,nv, nvv);    dd:= side*d;
   diff3d(p1,p0, f1);        diff3d(p2,p0, f2);
   g1:= scalarp3d(nvv,f1);   g2:= scalarp3d(nvv,f2);    g3:= dd - scalarp3d(nvv,p0);
   if (abs(g1)<eps3) and (abs(g2)<eps3)
     then                                    {Ellipse parallel zur Ebene !}
        begin
        if g3<=0                   then cp_ellipse(p0,p1,p2,0);
        if (style=10) and (g3>0)   then cp_ellipse(p0,p1,p2,1);
        end
     else
        begin
        is_unitcircle_line(g1,g2,g3, xi1,eta1,xi2,eta2,nis);
        if nis<2  then
           begin
           if g3<=0                   then  cp_ellipse(p0,p1,p2,0);
           if (style=10) and (g3>0)   then  cp_ellipse(p0,p1,p2,1);
           end
          else      {nis=2}
           begin
           lcomb3vt3d(1,p0, xi1,f1, eta1, f2, p3);
           lcomb3vt3d(1,p0, xi2,f1, eta2, f2, p4);
           if abs(g3)>eps5 then
              begin
```

```
              test:= xi1*eta2-eta1*xi2;
              if (g3*test<0) then  change3d(p3,p4);
              end
             else
              if (-eta1*g1 + xi1*g2 - g3 <0) then  change3d(p3,p4);
           cp_ellipse_arc(p0,p1,p2,p3,p4,style);
           end;    {nis=2}
        end;
 end;  {cp_ellipse_before_plane}
{**************}
```

5.6.3 Zentralprojektion von Hyperbel und Parabel

Die Zentralprojektion von Hyperbel und Parabel führen wir durch Berechnen der Punkte mit `hyperb_pts3d` bzw. `parab_pts3d` und anschließender Projektion der "Kurven" mit `cp_curve` aus.

5.7 Ergänzungen zu Ellipse, Hyperbel und Parabel

5.7.1 Schnitt Hyperbel - Gerade

Für die Hauptachsentransformation einer Hyperbel im nächsten Abschnitt benötigen wir das folgende Unterprogramm `is_unithyperbola_line`, das die Schnittpunkte der Hyperbel $x^2 - y^2 = 1$ mit der Gerade $ax + by = c$ berechnet. Dieses Unterprogramm entsteht durch geringfügige Abänderungen aus dem Unterprogramm `is_circle_line` (Schnitt Kreis-Gerade, vgl. Kapitel 2).

```
procedure is_unithyperbola_line(a,b,c : real; var x1,y1,x2,y2 : real;
                                              var nis : integer);
 {Schnitt Hyperbel der  x*x - y*y = 1  mit der Gerade  a*x + b*y = c,
 Schnittpunkte: (x1,y1),(x2,y2).Es ist x1<=x2, ns Anzahl der Schnittpunkte.}
```

5.7.2 Hauptachsentransformationen

Unter Hauptachsentransformation wollen wir hier das Auffinden der Scheitel bzw. Nebenscheitel einer Ellipse, Parabel oder Hyperbel und die anschließende Beschreibung dieser Kurven mit Hilfe ihrer Scheitel bzw.. Nebenscheitel verstehen. Wir werden sehen, daß sich alle drei Fälle leicht in **einem** Unterprogramm behandeln lassen.

Hauptachsentransformation einer Ellipse:

Gegeben: Ellipse $E = \{\mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi | \ldots\}$ (vgl. 5.1).
Gesucht: Scheitel der Ellipse E.
Die zu einem Halbmesser $\mathbf{q}(\varphi_0) = \mathbf{f}_1 \cos\varphi_0 + \mathbf{f}_2 \sin\varphi_0$ konjugierten Halbmesser sind wegen $\mathbf{q}'(\varphi_0) = -\mathbf{f}_1 \sin\varphi_0 + \mathbf{f}_2 \cos\varphi_0$ (Tangentenvektor) $\mathbf{q}(\varphi_0 \pm \pi/2)$, und es ist

$$E = \{\mathbf{q}(\varphi_0)\cos\varphi + \mathbf{q}(\varphi_0 \pm \pi/2)\sin\varphi | \ldots\}$$

Deshalb genügt es, **einen** Scheitel $S_1 : \bar{\mathbf{f}_1} = \mathbf{f}_1\xi_1 + \mathbf{f}_2\eta_1$ der Ellipse zu bestimmen. Die restlichen Scheitel sind dann

$$S_2 : \bar{\mathbf{f}_2} = -\mathbf{f}_1\eta_1 + \mathbf{f}_2\xi_1, \quad S_3 : -\bar{\mathbf{f}_1}, \quad S_4 : -\bar{\mathbf{f}_2}.$$

Ein Punkt $Q : \mathbf{q}(\varphi) = \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi$ ist genau dann ein Scheitel, wenn der Tangentenvektor $\mathbf{q}'(\varphi) = -\mathbf{f}_1 \sin\varphi + \mathbf{f}_2 \cos\varphi$ auf $\mathbf{q}(\varphi)$ senkrecht steht, d.h. wenn $\mathbf{q}(\varphi) \cdot \mathbf{q}'(\varphi) = \mathbf{f}_1 \cdot \mathbf{f}_2(\cos^2\varphi -$

$\sin^2\varphi) + (\mathbf{f}_2^2 - \mathbf{f}_1^2)\sin\varphi\cos\varphi = 0$ ist. Die Koordinaten ξ, η eines Scheitels bezüglich der Basis $\{\mathbf{f}_1, \mathbf{f}_2\}$ genügen also dem Gleichungssystem:

$$\begin{array}{lrcll} (1) & a(\xi^2-\eta^2)+b\xi\eta & = & 0 & \\ (2) & \xi^2+\eta^2 & = & 1 & \text{mit} \quad a := \mathbf{f}_1\cdot\mathbf{f}_2, \quad b := \mathbf{f}_2^2 - \mathbf{f}_1^2. \end{array}$$

Falls $a = 0$ ist, ist der Punkt Q ein Scheitel. Falls $a \neq 0$ ist, ist das Gleichungssystem zu dem folgenden äquivalent:

$$\begin{array}{lrcll} (1') & \gamma_1\xi+\gamma_2\eta & = & 0 & \text{(... entsteht durch Auflösen von (1) nach } \xi/\eta). \\ (2) & \xi^2+\eta^2 & = & 1 & \text{mit} \quad \gamma_1 := 2a, \quad \gamma_2 := b \pm \sqrt{b^2+4a^2}. \end{array}$$

Da wir nur einen Scheitel berechnen müssen, können wir uns bei γ_2 auf das "+"-Zeichen beschränken. Mit `is_circle_line` findet man dann eine Lösung (ξ_1, η_1) und $\bar{\mathbf{f}}_1 = \mathbf{f}_1\xi_1 + \mathbf{f}_2\eta_1, \bar{\mathbf{f}}_2 = -\mathbf{f}_1\eta_1 + \mathbf{f}_2\xi_1$ sind zwei zueinander senkrechte konjugierte Halbmesser. Es ist

$$E = \{\bar{\mathbf{f}}_1\cos\varphi + \bar{\mathbf{f}}_2\sin\varphi | \ldots\}$$

Hauptachsentransformation einer Hyperbel:

Gegeben : Hyperbel $H = \{\mathbf{f}_1\cosh t + \mathbf{f}_2\sinh t | \ldots\}$ (vgl. 5.2).
Gesucht : Haupt- und Nebenscheitel. Die überlegungen für eine Ellipse lassen sich fast wörtlich übernehmen. Man muß nur beachten, daß zu dem Halbmesser $\mathbf{q}(t_0) = \mathbf{f}_1\cosh t_0 + \mathbf{f}_2\sinh t_0$ der Vektor $\mathbf{q}'(t_0) = \mathbf{f}_1\sinh t_0 + \mathbf{f}_2\cosh t_0$ konjugiert ist. D.h. hier muß man zur Bestimmung eines Scheitels das Gleichungssystem

$$\begin{array}{lrcll} (1) & a(\xi^2+\eta^2)+b\xi\eta & = & 0 & \\ (2) & \xi^2-\eta^2 & = & 1 & \text{mit} \quad a := \mathbf{f}_1\cdot\mathbf{f}_2, \quad b := \mathbf{f}_1^2 - \mathbf{f}_2^2. \end{array}$$

bzw.

$$\begin{array}{lrcll} (1') & \gamma_1\xi+\gamma_2\eta & = & 0. & \\ (2) & \xi^2-\eta^2 & = & 1 & \text{mit} \quad \gamma_1 := 2a, \quad \gamma_2 := b \pm \sqrt{b^2-4a^2}. \end{array}$$

lösen. Hierfür verwendet man das Unterprogramm `is_hyperbola_line`.

Hauptachsentransformation einer Parabel:

Gegeben : Parabel $P = \{\mathbf{f}_1\xi + \mathbf{f}_2\xi^2 | \ldots\}$ (vgl. 5.3).
Gesucht : Scheitel von P.
Der Punkt $Q : \mathbf{q}(\xi_0) = \mathbf{f}_1\xi_0 + \mathbf{f}_2\xi_0^2$ ist genau dann der Scheitel von P, wenn

$$\mathbf{q}'(\xi_0)\cdot\mathbf{f}_2 = (\mathbf{f}_1 + 2\mathbf{f}_2\xi_0)\cdot\mathbf{f}_2 = \mathbf{f}_1\cdot\mathbf{f}_2 + 2\mathbf{f}_2\xi_0 = 0,$$

d.h. wenn

$$\xi_0 = -\mathbf{f}_1\cdot\mathbf{f}_2/2\mathbf{f}_2^2$$

ist. Es gilt

$$\mathbf{q}(\xi_0) + \mathbf{q}'(\xi_0)\xi + \mathbf{f}_2\xi^2 = \mathbf{f}_1(\xi_0+\xi) + \mathbf{f}_2(\xi_0+\xi)^2 = \mathbf{q}(\xi_0+\xi)$$

und damit

$$P = \{\mathbf{q}(\xi_0) + \bar{\mathbf{f}}_1\xi + \mathbf{f}_2\xi^2. | \ldots\} \quad \text{mit } \bar{\mathbf{f}}_1 = \mathbf{q}'(\xi_0)$$

5.7.3 Schnitt Kreis-Ellipse, Kreis-Hyperbel

Eine beliebige Ellipse läßt sich mit Hilfe der Hauptachsentransformation, der Verschiebung des Mittelpunktes in den Nullpunkt und einer geeigneten Drehung so transformieren, daß ihre Punkte einer Gleichung $x^2/a^2 + y^2/b^2 = 1$ genügen. Da bei der Hauptachsentransformation (s.o.) das Koordinatensystem nicht geändert wird und bei einer Translation und Drehung ein Kreis in einen Kreis

übergeht, können wir uns auf die folgende spezielle Situation beschränken.

Gegeben:	Kreis	$(x-x_0)^2+(y-y_0)^2$	$=$	r^2 ,	(1)
	Ellipse	$x^2/a^2+y^2/b^2$	$=$	1 ,	(2)
Gesucht:	Schnittpunkte				

Elimination von y^2 aus (1) mit Hilfe von (2) liefert

$$x^2(b^2/a^2-1)+2x_0x+2y_0y=b^2-r^2+x_0^2+y_0^2. \qquad (1')$$

Falls $y_0=0$ ist, erhält man die quadratische Gleichung

$$x^2(1-b^2/a^2)+2x_0x+b^2-r^2+x_0^2=0. \qquad (1'')$$

Die Lösungen von $(1'')$ ergeben mit (2) die Schnittpunkte.
Falls $y_0\neq 0$ ist, löst man $(1')$ nach y auf:

$$y=cx^2+dx+e \qquad (1''')$$

mit

$$c:=(1-b^2/a^2)/(2y_0),\quad d:=-x_0/y_0,\quad e:=(b^2-r^2+x_0^2+y_0^2)/(2y_0)$$

und setzt $(1''')$ in (2) ein :

$$c^2x^4+2cdx^3+(2ce+d^2+b^2/a^2)x^2+2dex+e^2-b^2=0$$

Die Lösung dieser Gleichnung **4.Grades** berechnet man entweder mit dem Unterprogramm `equation_degree4` (s. Anhang 2) oder mit Hilfe einer Newton-Iteration. (Bei Verwendung einer Newton-Iteration sollte man beachten, daß Lösungen nur für $x\in[x_0-r,x_0+r]$ in Frage kommen.) Mit (2) erhält man schließlich die Schnittpunkte.

Die Schnittpunkte eines Kreises mit einer Hyperbel werden analog bestimmt. Dabei kann man die Rechnung für den Schnitt mit einer Ellipse übernehmen, wenn man b^2 durch $-b^2$ ersetzt.

5.7.4 Schnitt Kreis-Parabel

Eine beliebige Parabel läßt sich durch eine geeignete Transformation in eine Parabel mit der Gleichung $y=ax^2$ überführen, so daß ein Kreis in einen Kreis übergeht. Wir können uns also auch hier auf eine vereinfachte Situation beschränken.

Gegeben:	Kreis	$(x-x_0)^2+(y-y_0)^2$	$=$	r^2	(1)
	Parabel	y	$=$	ax^2	(2)
Gesucht:	Schnittpunkte				

Einsetzen der Parabelgleichung in die Kreisgleichung führt auf die Gleichung

$$a^2x^4+(1-2ay_0)x^2-2x_0x+x_0^2+y_0^2-r^2=0.$$

Diese Gleichung löst man mit `equation_degree4` (s. Anhang) und berechnet dann mit (2) die y-Koordinaten der Schnittpunkte.

5.7.5 Gleichung einer Ellipse bzw. Hyperbel bzw. Parabel

Gegeben: Ellipse $E : \mathbf{x} = \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi$
Gesucht: Gleichung $f(x, y) = 0$ von E (implizite Darstellung von E).

Löst man das Gleichungssystem

$$\begin{array}{rcll} x & = & f_{1x}\cos\varphi + f_{2x}\sin\varphi & \text{(Es ist } \mathbf{x} = (x, y),\ \mathbf{f}_i = (f_{ix}, f_{iy}).) \\ y & = & f_{1y}\cos\varphi + f_{2y}\sin\varphi & \end{array}$$

nach $\cos\varphi$ und $\sin\varphi$ auf, so erhält man aus $(\cos\varphi)^2 + (\sin\varphi)^2 - 1 = 0$:

$$f(x, y) := (xf_{2y} - yf_{2x})^2 + (yf_{1x} - xf_{1y})^2 - (f_{1x}f_{2y} - f_{1y}f_{2x})^2 = 0.$$

Dies ist die **Gleichung** der Ellipse E.

Analog erhält man die
Gleichung der **Hyperbel** $\mathbf{x} = \pm\mathbf{f}_1 \cosh t + \mathbf{f}_2 \sinh t$ aus der Beziehung $(\cosh t)^2 - (\sinh t)^2 - 1 = 0$:

$$f(x, y) := (xf_{2y} - yf_{2x})^2 - (yf_{1x} - xf_{1y})^2 - (f_{1x}f_{2y} - f_{1y}f_{2x})^2 = 0,$$

Gleichung der **Parabel** $\mathbf{x} = \mathbf{f}_1 t + \mathbf{f}_2 t^2$ aus der Beziehung $t \cdot t - t^2 = 0$:

$$f(x, y) := (xf_{2y} - yf_{2x})^2 - (yf_{1x} - xf_{1y})(f_{1x}f_{2y} - f_{1y}f_{2x}) = 0.$$

Kapitel 6

EBENE KURVEN

6.1 Parametrisierte Kurven im $\mathbb{R}^2$ und $\mathbb{R}^3$

Unter einer parametrisierten Kurve Γ wollen wir eine Kurve verstehen, deren Punkte die Bilder einer Abbildung eines reellen Intervalls $[t_1, t_2]$ in den $\mathbb{R}^2$ bzw. $\mathbb{R}^3$ sind:
$\Gamma = \{\mathbf{c}(t) | t_1 \leq t \leq t_2\}$

Beispiele:
a) Ellipse, Hyperbel, Parabel gemäß Kapitel 5
b) Epi- bzw. Hypozykloiden: $\mathbf{c}(t) = (x(t), y(t))$ mit

$$\begin{aligned} x(t) &= (a+b)\cos t - \lambda a \cos((a+b)t/a) \\ y(t) &= (a+b)\sin t - \lambda a \sin((a+b)t/a) \quad b > 0, \quad a+b > 0, \quad \lambda > 0. \end{aligned}$$

Diese Kurven entstehen durch Abrollen eines Kreises mit Radius a auf bzw. in einem größeren Kreis mit Radius b .
Für $\lambda \neq 1$ entstehen *verlängerte* bzw. *verkürzte* Zykloiden.

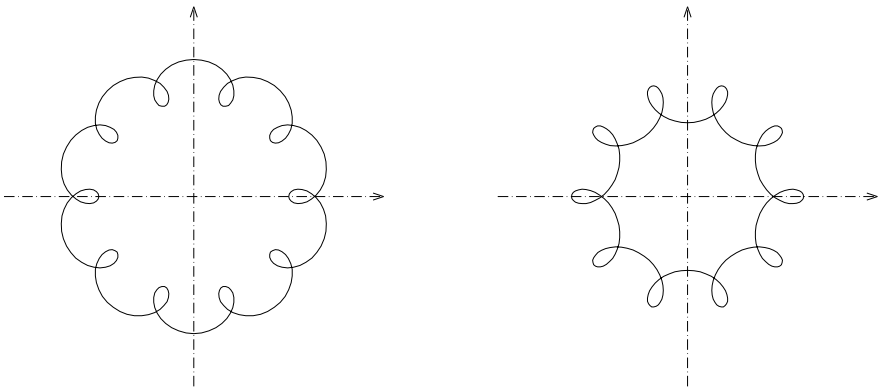


Abbildung 6.1: Zykloiden

Indem man das Parameterintervall in (nicht notwendig gleichlangen) Schritten durchläuft, lassen sich beliebig viele Punkte einer parametrisierten Kurve berechnen und anschließend
a) im Fall einer ebenen Kurve durch einen Polygonzug mit `curve2d` verbinden oder,
b) im Fall einer Kurve im $\mathbb{R}^3$, mit `pp_curve` projizieren.

Da der Parameter t i.a. nicht die Bogenlänge ist, können die Abstände benachbarter Punkte stark differieren. Dies läßt sich vermeiden, wenn die Kurve $\mathbf{c}(t)$ differenzierbar ist und man die Schrittweite im Parameterbereich in Abhängigkeit von $\dot{\mathbf{c}}(t)$ wählt. Soll der Abstand zweier Punkte (ungefähr) s sein, so erhält man aus der Taylorentwicklung von $\mathbf{c}(t)$:

$$\mathbf{c}(t_{i+1}) \approx \mathbf{c}(t_i) + \dot{\mathbf{c}}(t_i)\Delta t_i, \qquad \Delta t_{i+1} = t_{i+1} - t_i,$$

Es ist $\|\mathbf{c}(t_{i+1}) - \mathbf{c}(t_i)\| \approx s$, wenn man $\Delta t_i = s/\|\dot{\mathbf{c}}(t_i)\|$ setzt, falls $\|\dot{\mathbf{c}}(t_i)\| \neq 0$ ist.
D.h. man erhält damit eine relativ **gleichmäßige Verteilung der** berechneten **Punkte.**

Die folgenden Überlegungen führen zu einer **krümmungsabhängigen Verteilung** der Punkte. Hierfür verlangen wir, daß die Winkel zwischen je zwei benachbarten Tangentenvektoren $\dot{\mathbf{c}}(t_i), \dot{\mathbf{c}}(t_{i+1})$ einen betragsmäßig nahezu konstanten Winkel $|\alpha|$ einschließen. Mit Hilfe von

$$\dot{\mathbf{c}}(t_{i+1}) \approx \dot{\mathbf{c}}(t_i) + \ddot{\mathbf{c}}(t_i)\Delta t_i, \quad \Delta t_{i+1} = t_{i+1} - t_i, \quad \text{und} \quad \frac{\|\dot{\mathbf{c}}(t_{i+1}) \times \dot{\mathbf{c}}(t_i)\|}{|\dot{\mathbf{c}}(t_{i+1}) \cdot \dot{\mathbf{c}}(t_i)|} = |\tan\alpha| \approx |\alpha|$$

für "kleine" Winkel α erhält man

$$|\alpha| \approx \frac{\|\dot{\mathbf{c}}(t_i) \times \ddot{\mathbf{c}}(t_i)\|\,|\Delta t_i|}{|\dot{\mathbf{c}}(t_i)^2 + \dot{\mathbf{c}}(t_i) \cdot \ddot{\mathbf{c}}(t_i)\Delta t_i|} \approx \frac{\|\dot{\mathbf{c}}(t_i) \times \ddot{\mathbf{c}}(t_i)\|\,|\Delta t_i|}{\dot{\mathbf{c}}(t_i)^2}.$$

Also schließen bei der krümmungsabhängigen Wahl

$$|\Delta t_i| = \frac{|\alpha|\dot{\mathbf{c}}(t_i)^2}{\|\dot{\mathbf{c}}(t_i) \times \ddot{\mathbf{c}}(t_i)\|}\ , \qquad \|\cdots\| \neq 0.$$

benachbarte Tangenten ungefähr den gleichen Winkel α ein.

a) $\Delta\, t_i = const.$ b) $\|\mathbf{c}(t_{i+1}) - \mathbf{c}(t_i)\| \approx const.$ c) Winkel $\approx const.$

Abbildung 6.2: Punktverteilungen der Kurve $\mathbf{c}(t) = (t, t^2)$

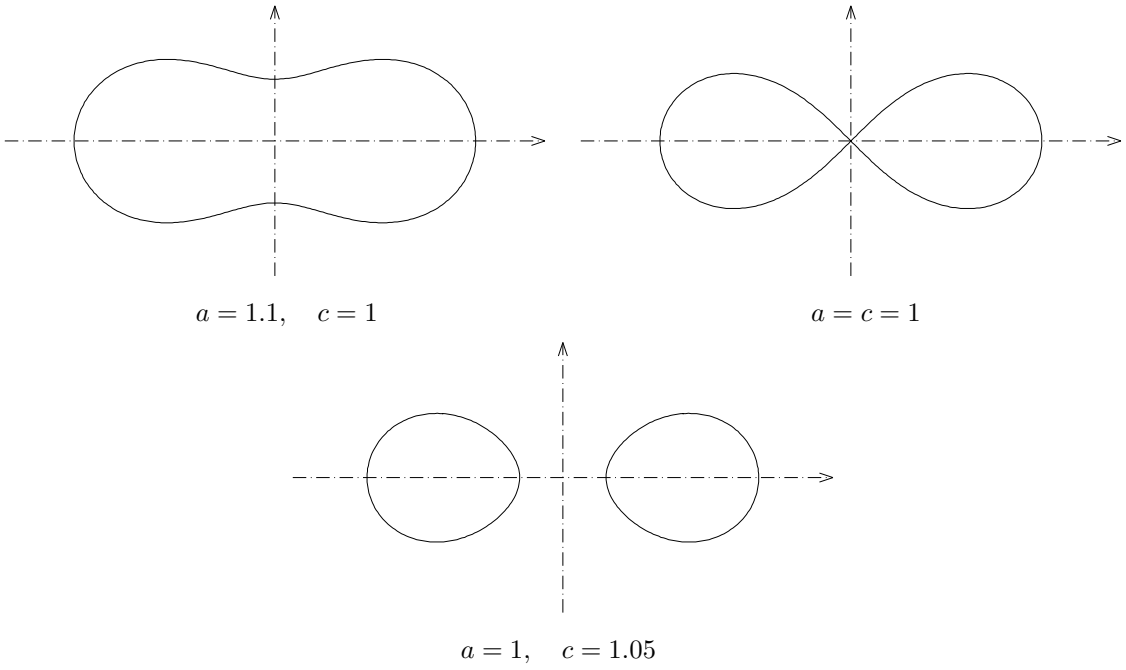
Bemerkung:
a) Falls Γ eine ebene Kurve ist, ergänze man die dritte Komponente zu 0, damit das Vektorprodukt ausgeführt werden kann.
b) Beschreibt $\mathbf{c}$(t) einen Kreis, so ist $\Delta t_i = \alpha = const.$
c) Wegen des wesentlich höheren Rechenaufwands ist die krümmungsabhängige Schrittweitensteuerung nur in begründeten Fällen zu empfehlen. Z.B. wenn die berechneten Punkte gespeichert werden sollen und die Kurve deshalb durch möglichst wenig Punkte gut repräsentiert werden soll.

6.2 Implizite Kurven

Unter einer impliziten Kurve Γ wollen wir eine Punktmenge des $\mathbb{R}^2$ verstehen, die einer Gleichung $f(x, y) = 0$ genügt :

$$\Gamma = \{\mathbf{x} \in D \,|\, f(\mathbf{x}) = 0\} \qquad f : D \to \mathbb{R}, \qquad D \subset \mathbb{R}^2.$$

Beispiele:
a) Kreis $x^2 + y^2 - r^2 = 0$, Hyperbel $xy - 1 = 0$
b) Cassini-Kurven $(x^2 + y^2)^2 - 2c^2(x^2 - y^2) - (a^4 - c^4) = 0, \quad a > 0, \quad c > 0.$
Für $a = c$ ergeben sich Lemniskaten.



$a = 1.1, \quad c = 1$ $\qquad$ $a = c = 1$

$a = 1, \quad c = 1.05$

Abbildung 6.3: Cassini-Kurven

Die **Berechnung von Punkten** einer imzpliziten Kurve ist nicht so einfach wie bei parametrisierten Kurven. Es werden hier zwei Methoden zur Erzeugung von Kurvenpunkten besprochen. Der **Verfolgungsalgorithmus** started mit einem Punkt in der Nähe der Kurve und berechnet sukzessive weitere auf demselben Zweig. Der **Raster-Newton-Algorithmus** started mit einem Raster des zu untersuchenden Bereichs im $\mathbb{R}^2$ und sucht Kurvenpunkte in der Nähe von Rasterpunkten. Der erste Algoritmus ist schnell und erzeugt einen zusammenhängenden Polygonzug. Der zweite ist langsamer, erzeugt nur einzelne Punkte, entdeckt aber (bis auf eine rasterabhängige Genauigkeit) alle Komponenten.

6.2.1 Der Verfolgungsalgorithmus

Es sind im wesentlichen zwei Teilprobleme zu lösen:
(1) Finden eines ersten Punktes P_1.
(2) Von P_1 ausgehend weitere Punkte der Kurve zu finden.

Idee des Algorithmus:
Zu (1): Man wählt einen Startpunkt $Q_0 = (x_0, y_0)$ in der Nähe der Kurve. Diesen faßt man als Grundriß des Punktes $\overline{Q}_0 = (x_0, y_0, f(x_0, y_0))$ auf der Fläche $z = f(x, y)$ auf. Ist $f(x_0, y_0) > 0$ bzw. < 0, so „läuft“ man in Richtung des steilsten Abstiegs bzw. Aufstiegs auf die Niveaulinie $f(x, y) = 0$ zu. „Laufen“ bedeutet hier, immer wieder Newton-Schritte auszuführen. Man erhält so den ersten Punkt P_1.
Zu (2): Von P_1 aus geht man ein Stück entlang der Tangente zu einem neuen Punkt Q_0 und wiederholt (1), ...

Da Teil (1) auch beim Raster–Newton–Verfahren und beim Lotefällen benutzt wird, schreiben wir hierfür eine eigenständige Prozedur `curvepoint`:

(CP1) Es sei $Q_0 = (x_0, y_0)$ ein Punkt in der Nähe der Kurve.

(CP2) Iteration entlang des steilsten Weges: Es sei $\mathbf{q}_i = (x_i, y_i)$.
$\mathbf{q}_{i+1} = \mathbf{q}_i - \frac{f(\mathbf{q}_i)}{\nabla f(\mathbf{q}_i)^2} \nabla f(\mathbf{q}_i)$.

(CP3) Wiederhole (CP2) bis $\|\mathbf{q}_{i+1} - \mathbf{q}_i\|$ klein genug ist.
(oder andere Abbruchbedingung.)

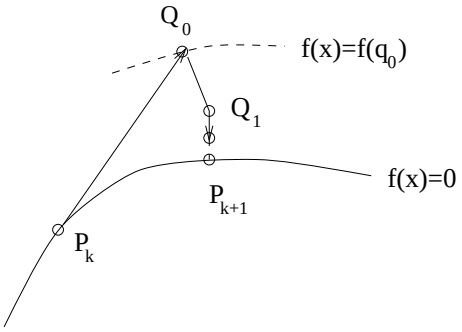


Abbildung 6.4: Berechnung von Punkten einer impliziten Kurve

Durchführung des Verfolgungsalgorithmus:

1) Wahl eines geeigneten Startpunktes $Q_0 : \mathbf{q}_0 = (x_0, y_0)$ und einer Schrittweite s.

2) Erster Kurvenpunkt ist $P_1 : \mathbf{p}_1 = \texttt{curvepoint}(\mathbf{q}_0)$.

3) Kurvenpunkt $P_{k+1} : \mathbf{p}_{k+1}$ aus $P_k : \mathbf{p}_k$:
$\mathbf{p}_{k+1} = \texttt{curvepoint}(\mathbf{p}_k + s\ \mathbf{t}_k)$, wobei $\mathbf{t}_k = (-f_y(\mathbf{p}_k), f_x(\mathbf{p}_k))/\| \cdots \|$ Einheitstangente im Punkt P_k ist.

4) Führe 3) sooft durch, bis die gewünschte Anzahl von Punkten berechnet ist
oder $P_{k+1} \approx P_0$ (geschlossene Kurve)
oder $\cdots$ (andere Abbruchbedingung)

Bemerkung:
Da der Verfolgungsalgorithmus relativ robust ist, läßt er sich auch auf Kurven mit einzelnen Singularitäten anwenden. Beim Überschreiten einer Singularität kann allerdings eine Richtungsumkehr stattfinden. Um zu verhindern, daß der Algorithmus "hängen bleibt", sollte man in Schritt 3) zunächst die Richtung von $\mathbf{t}_k$ überprüfen:
Falls $\mathbf{t}_k \cdot (\mathbf{p}_k - \mathbf{p}_{k-1}) < 0$ ist, ersetze $\mathbf{t}_k$ durch $-\mathbf{t}_k$.

Aufgabe 6.1 *Schreibe ein Programm, das Cassini-Kurven:*

$$(x^2+y^2)^2 - 2c^2(x^2-y^2) - (a^4-c^4) = 0, \qquad c>0, \quad a>0,$$

zeichnet.

Bemerkung:
Eine **krümmungsabhängige Schrittweitensteuerung** ergibt sich aus den folgenden Überlegungen.
Setzt man eine Parameterdarstellung $\mathbf{c}(s)$ der impliziten Kurve $f(x,y)=0$ in die Funktion f ein und entwickelt die so entstandene Funktion $f(\mathbf{c}(s))$ nach s, so erhält man

$$0 = f(\mathbf{c}(s)) = f(\mathbf{c}(0)) + s\nabla f(\mathbf{c}(0))\cdot\mathbf{c}'(0) + \frac{s^2}{2}(\nabla f(\mathbf{c}(0))\cdot\mathbf{c}''(0) + \mathbf{c}'(0)H_f(\mathbf{c}(0))\mathbf{c}'(0)^T) + \cdots$$

wobei H_f die Hesse-Matrix von f ist.
Die Koeffizienten bei s^0 und s^1 liefern keine neuen Erkenntnisse (denn es ist $f(\mathbf{c}(0))=0$ und $\mathbf{c}'(0)$ ist ein Tangentenvektor der Kurve). Der Koeffizient bei s^2 gibt uns die Möglichkeit $\mathbf{c}''(0)$ zu berechnen. Hierzu machen wir die Annahme, daß der Parameter s die **Bogenlänge** ist. Denn dann gilt

(1) $\mathbf{c}'(0) = (-f_y, f_x)/\|\cdots\|$, wobei $(f_x, f_y) := \nabla f(\mathbf{c}(0)$ ist,

(2) $\mathbf{c}''(0)\cdot\mathbf{c}'(0) = 0$ d.h. $\mathbf{c}''(0)$ hat die Richtung von $\nabla f(\mathbf{c}(0))$,

(3) $\nabla f(\mathbf{c}(0))\cdot\mathbf{c}''(0) = -\mathbf{c}'(0)H_f(\mathbf{c}(0))\mathbf{c}'(0)^T$.

Aus (2) und (3) ergibt sich

$$\mathbf{c}''(0) = -\frac{\mathbf{c}'(0)H_f(\mathbf{c}(0))\mathbf{c}'(0)^T}{\nabla f(\mathbf{c}(0))^2}\nabla f(\mathbf{c}(0)) \quad \text{und die Krümmung}$$

$$\|\mathbf{c}''(0)\| = |\frac{f_y^2 f_{xx} - 2f_x f_y f_{xy} + f_x^2 f_{yy}}{(f_x^2+f_y^2)^{3/2}}| \text{ an der Stelle } \mathbf{c}(0).$$

Die Schrittweite s wählt man jetzt **krümmungsabhängig**. Z.B. bewirkt

$$|s| = \frac{1}{5\|\mathbf{c}''(0)\|}, \quad \text{falls} \quad \|\mathbf{c}''(0)\| \neq 0,$$

daß benachbarte Tangenten ungefähr einen Winkel von 11 Grad einschließen. Natürlich muß man in der Praxis sicherstellen, daß die Schrittweite bei kleiner Krümmung nicht zu groß ausfällt.
Damit ergibt sich die 0-te Näherung $Q_0 : \mathbf{q}_0$ von $P_{k+1} : \mathbf{q}_0 = \mathbf{p}_k + s\mathbf{c}'(0) + \frac{s^2}{2}\mathbf{c}''(0)$.

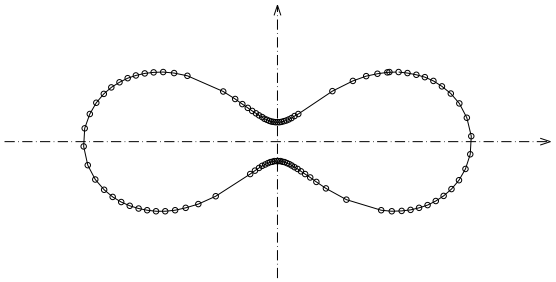


Abbildung 6.5: Krümmungsabhängige Berechnung von Punkten einer impliziten Kurve

6.2.2 Der Raster–Newton–Algorithmus

Idee des Algorithmus:
Man started hier mit einem achsenparallelen Raster, in dem man Kurvenpunkte vermutet. Mit Hilfe einer Abschätzung der Distanz der Rasterpunkte zur Kurve findet man Rasterpunkte in der Nähe der Kurve. Die so gefundenen Rasterpunkte dienen als Startpunkte für eine *Iteration entlang des steilsten Weges* (Prozedur `curvepoint` des Verfolgungsalgorithmus). Auf diese Weise erhält man eine Ansammlung von Punkten, die bei hinreichend feinem Raster den Kurvenverlauf wiedergeben.

Durchführung:
Gegeben: implizite Kurve Γ mit der Gleichung $f(x, y,) = 0$.

1) Wähle ein x-Intervall $[x_{min}, x_{max}]$ und ein y-Intervall $[y_{min}, y_{max}]$, sowie die Anzahlen n_x und n_y der Unterteilungen dieser Intervalle für ein achsenparalleles Raster mit den Rasterabständen $d_x = (x_{max} - x_{min})/n_x$ bzw. $d_y = (y_{max} - y_{min})/n_y$.
$d_{max} = \max(d_x, d_y)$ ist der maximale Rasterabstand.

2) Für jeden Rasterpunkt (x_i, y_k) werden $f(x_i, y_k)$ und $\nabla f(x_i, y_k)$ berechnet.
Ist $\nabla f(x_i, y_k) = (0, 0)$, so wird dieser (singuläre) Rasterpunkt ignoriert,
Im anderen Fall wird der Abstand von (x_i, y_k) zur Kurve durch

$$\delta_{ik} := |f(x_i, y_k)|/\|\nabla f(x_i, y_k)\|, \ \textit{Distanz 1. Ordnung}$$

abgeschätzt.

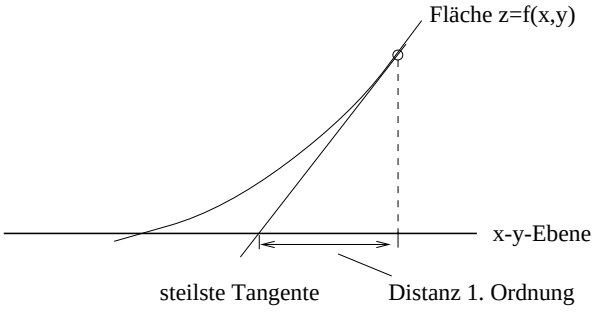


Abbildung 6.6: Distanz 1. Ordnung eines Punktes zur impliziten Kurve

3) Für jeden Punkt (x_i, y_k), für den $\delta_{ik} < \gamma d_{max}$, $\gamma > 0.5$, ist, sucht man mit `curvepoint` (s. Verfolgungsalgorithmus) einen Kurvenpunkt.

Bemerkung:
Der Algorithmus läßt sich wesentlich beschleunigen, wenn man nicht für jeden Rasterpunkt die Distanz zur Kurve abschätzt, sondern, abhängig von der Distanz des zuletzt betrachteten Punktes zur Kurve, einen Rastersprung zuläßt. Dabei sollte man sicher stellen, daß dieser Sprung nicht zu groß ausfällt. Es könnten sonst Lücken im Kurvenbild entstehen.

Beispiel 6.1 *Für das folgende Beispiel ist*
$f(x, y) = 0.004 + 0.11x - 0.177y - 0.174x^2 + 0.224xy - 0.303y^2 - 0.168x^3 + 0.327x^2y - 0.087xy^2 - 0.013y^3 + 0.235x^4 - 0.667x^3y + 0.745x^2y^2 - 0.029xy^3 + 0.072y^4$
(aus TAU'94).

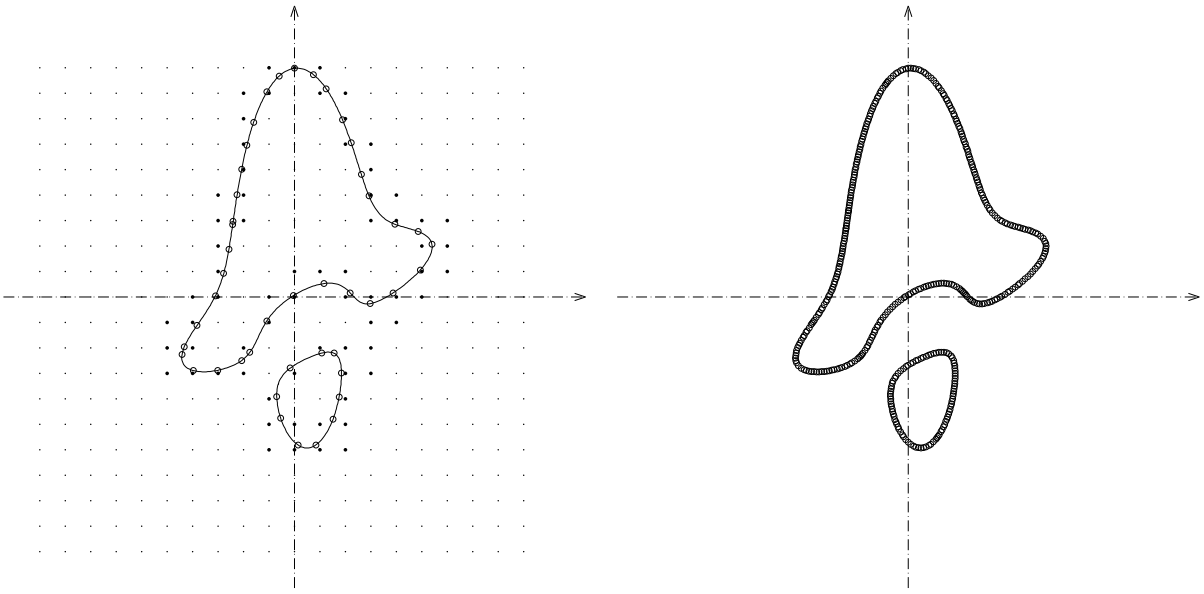


Abbildung 6.7: zum Raster–Newton–Algorithmus. Links: $n_x = n_y = 20$, rechts: $n_x = n_y = 200$

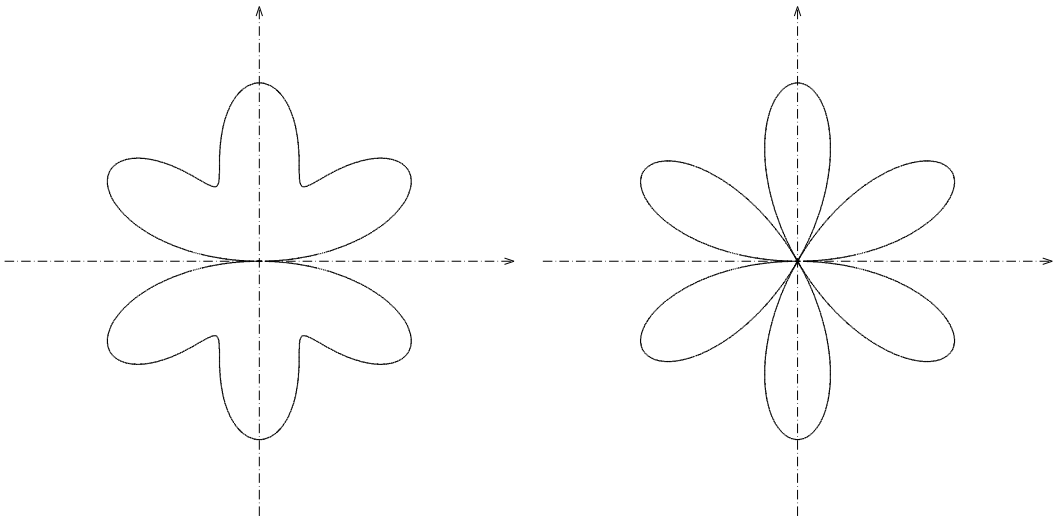


Abbildung 6.8: Zu Beispiel 6.2. Es ist $n_x = n_y = 500$.

Beispiel 6.2 *a)* $f(x,y) = (8x^4 - 4x^2y^2 + y^4)y^2 - (x^2+y^2)^4$,
b) $f(x,y) = (3x^2 - y^2)^2y^2 - (x^2+y^2)^4$.

Bemerkung:
Mit Hilfe des Raster–Newton–Algorithmus lassen sich auch Schnittpunkte impliziter Kurven bestimmen. Sind die beiden Kurven $f_1(x,y) = 0, f_2(x,y) = 0$ gegeben, so sucht man mit dem Algorithmus Nullstellen der Funktion $f(x,y) := f_1(x,y)^2 + f_2(x,y)^2$.

Beispiel 6.3 $f_1(x,y) = (3x^2 - y^2)^2y^2 - (x^2+y^2)^4, \quad f_2(x,y) = (x-0.5)^2 + y^2 - 1 = 0$

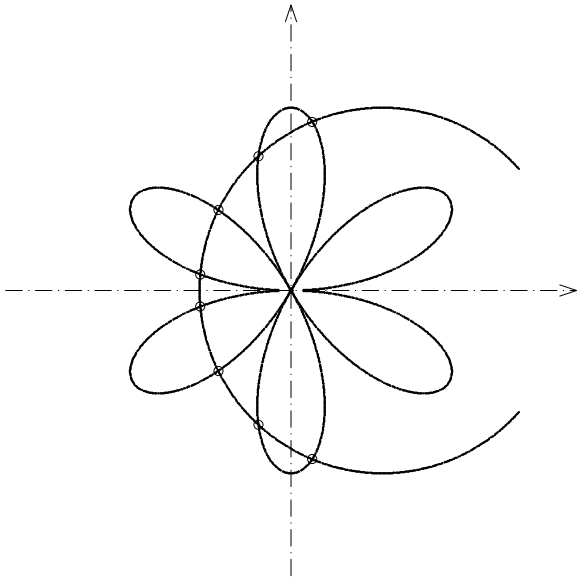


Abbildung 6.9: Zu Beispiel 6.3

6.3 Ebene Kurven im Raum

Sind von einer parametrisierten oder impliziten ebenen Kurve die Punkte $\overline{P}_0 = (\xi_0, \eta_0), \cdots, \overline{P}_n = (\xi_n, \eta_n) \in \mathbb{R}^2$ und von einer Ebene ε im $\mathbb{R}^3$ ein Punkt $Q_0 : \mathbf{q}_0$ und eine ON-Basis $\mathbf{b}_1, \mathbf{b}_2$ bekannt, so sind

$$P_i : \mathbf{p}_i = \mathbf{q}_0 + \mathbf{b}_1\xi_i + \mathbf{b}_2\eta_i, \qquad i = 0, \cdots, n,$$

Kurvenpunkte in der Ebene ε .

In den Beispielen der Aufgabe 3.2 e) wurden Punkte einer Zykloide im $\mathbb{R}^2$ berechnet und auf jeder Seitenfläche eines Polyeders der Mittelpunkt M und eine ON-Basis $\mathbf{b}_1, \mathbf{b}_2$ mit Hilfe von Kantenvektoren und der Flächennormalen $\mathbf{n}$ bestimmt:
Sind $Q_1 : \mathbf{q}_1, \cdots Q_m : \mathbf{q}_m$ Punkte einer Fläche, dann ist $M : \mathbf{m} = (\mathbf{q}_1 + \cdots + \mathbf{q}_m)/m$ der Mittelpunkt und $\mathbf{b}_1 := (\mathbf{q}_2 - \mathbf{q}_1)/\|\mathbf{q}_2 - \mathbf{q}_1\|, \quad \mathbf{b}_2 := (\mathbf{b}_1 \times \mathbf{n})/\| \cdots \|$ eine ON-Basis.

Man beachte: Da eine **Parallelprojektion** Φ im wesentlichen eine lineare Abbildung ist, genügt es in diesem Fall die Vektoren $\mathbf{q}_0, \mathbf{b}_1, \mathbf{b}_2$ zu projizieren, d.h. die Bildpunkte der Kurvenpunkte in ε sind:

$$\Phi(P_i) : \Phi(\mathbf{p}_i) = \Phi(\mathbf{q}_0) + \Phi(\mathbf{b}_1)\xi_1 + \Phi(\mathbf{b}_2)\eta_i, \quad i = 0, \cdots, n$$

6.4 Bézier-Kurven

Da Bézier-Kurven in CAD eine große Rolle spielen und wir diese im Kapitel über Rotationsflächen verwenden wollen, sei hier eine kleine Einführung gegeben. Für weitergehende Informationen zu diesem Thema sei der Leser z. B. auf die Bücher von FARIN (FA'90) oder HOSCHEK/LASSER (HO,LA '89) verwiesen.

Numerisch einfache Kurven in der Ebene sind solche, die mit Hilfe einer Parameterdarstellung $\mathbf{x}(t) = (x(t), y(t)),\ t_1 \le t \le t_2$, wobei $x(t)$ und $y(t)$ Polynome in t sind, beschrieben werden. Ist $x(t) :=$

$a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n$ und $y(t) := b_0 + b_1 t + b_2 t + \cdots + b_n t^n$, so ist

$$\begin{aligned} \mathbf{x}(t) &= (a_0, b_0) + (a_1, b_1)t + \cdots + (a_n, b_n)t^n. \\ &= \mathbf{a}_0 + \mathbf{a}_1 t + \cdots + \mathbf{a}_n t^n \end{aligned}$$

mit den Vektoren $\mathbf{a}_i := (a_i, b_i)$.
I.a. sagen die Punkte $A_i : \mathbf{a}_i$ nicht viel über den Kurvenverlauf aus. Dies ändert sich, wenn man die Polynome $x(t), y(t)$ nicht in der „Monom-Basis“ $\{1, t, t^2, \cdots, t^n\}$ sondern in der folgenden **Bernsteinbasis** $\{B_0^n(t), B_1^n(t), \cdots, B_n^n(t)\}$ darstellt:

$$B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}, \quad 0 \le i \le n.$$

Es sei nun $n > 0$ festgewählt und die Vektoren $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n$ beschreiben ein Polygon. Dann ist $\mathbf{x}(t) := \mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \cdots + \mathbf{b}_n B_n^n(t), \quad 0 \le t \le 1$, eine **Bézier-Kurve** vom (maximalen)Grad n. Die Punkte $\mathbf{b}_0, \cdots, \mathbf{b}_n$ heißen **Kontrollpunkte** der Bézierkurve.

Eigenschaften der Bernstein-Polynome:

(1) $B_0^n(t) + B_1^n(t) + \cdots + B_n^n(t) = 1$,

(2) $B_0^n(0) = 1,\ B_i^n(0) = 0$ für $i > 0, \quad B_n^n(1) = 1,\ B_i^n(1) = 0$ für $i < n$.

(3) Das Bernstein-Polynom B_i^n hat genau ein Maximum und zwar an der Stelle $t = i/n$. D.h. eine leichte Veränderung des Punktes $\mathbf{b}_i$ hat nur eine wesentliche Veränderung der Kurve in der Umgebung von $\mathbf{x}(i/n)$ zur Folge.

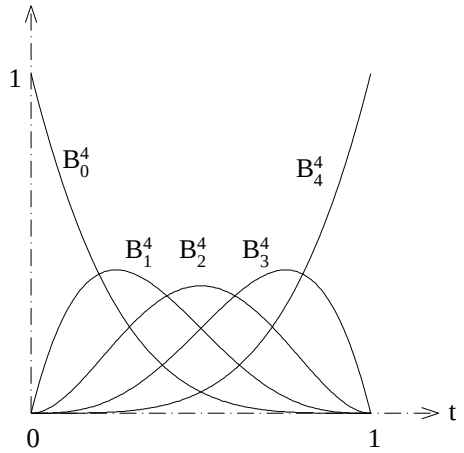


Abbildung 6.10: Bernsteinpolynome B_i^4

Eigenschaften einer Bézier-Kurve:

(1) $\mathbf{b}_0$ ist der Anfangs- , $\mathbf{b}_n$ der Endpunkt

(2) $\mathbf{b}_1 - \mathbf{b}_0$ ist die Richtung der Tangente im Punkt $\mathbf{b}_0 = \mathbf{x}(0)$, $\mathbf{b}_n - \mathbf{b}_{n-1}$ ist die Richtung der Tangente im Punkt $\mathbf{b}_n = \mathbf{x}(1)$.

(3) Das Polygon $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n$ gibt einen ungefähren Verlauf der Kurve an.

Da die Komponenten $x(t), y(t)$ einer Bézierkurve reelle Linearkombinationen von Bernstein-Polynomen sind, können wir zur Berechnung von Punkten einer Bézierkurve ein Unterprogramm verwenden, das solche Linearkombinationen berechnet. Wir werden hier das in FA'90 , S. 48, angegebene Unterprogramm `bezier_comp` (dort mit `hornbez` bezeichnet) verwenden. Es berechnet nach einer Art Horner-Schema den Ausdruck

$$a_0 B_0^n(t) + a_1 B_1^n(t) + \cdots + a_n B_n^n(t), \qquad a_i \in \mathbb{R},$$

bei Vorgabe des Grades n , der Koeffizienten $a_0, \cdots, a_n$ und des Parameters t .

```
function bezier_comp(degree: integer; coeff : r_array; t: real) : real;
{Berechnet eine Komponente einer Bezier-Kurve. (Aus FARIN: Curves and Surfaces...)}
 var i,n_choose_i : integer;    fact,t1,aux : real;
 begin
   t1:= 1-t;  fact:=1;  n_choose_i:= 1;
   aux:= coeff[0]*t1;
   for i:= 1 to degree-1 do
       begin
       fact:= fact*t;
       n_choose_i:= n_choose_i*(degree-i+1) div i;
       aux:= (aux + fact*n_choose_i*coeff[i])*t1;
       end;
   aux:= aux + fact*t*coeff[degree] ;
   bezier_comp:= aux;
 end;   bezier_comp
{*************}
```

Bemerkung:
Das Unterprogramm `bezier_comp` kann man natürlich auch zum Berechnen von Punkten einer Bézier-Kurve $(x(t), y(t), z(t))$ im Raum verwenden.

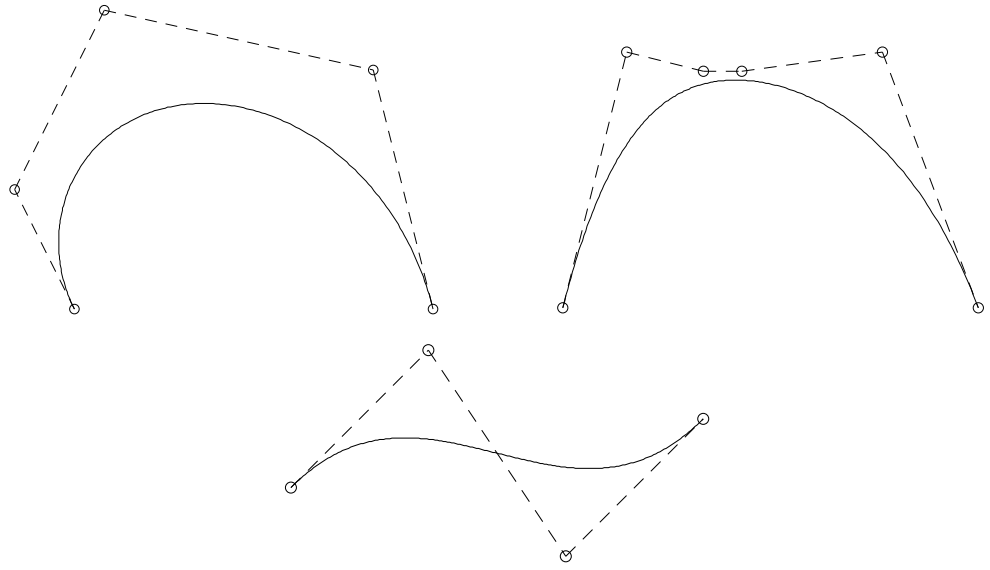


Abbildung 6.11: Bézierkurven mit ihren Kontrollpolygonen

6.5 Schnittpunkte zweier Kurven

Die Schnittpunkte zweier Geraden oder zweier Kreise führt auf leicht zu lösende Gleichungen 1.bzw. 2. Grades. Will man zwei beliebige Kegelschnitte miteinander schneiden so muß man i.a. eine Gleichung 4. Grades lösen, was auch noch direkt möglich ist. Beim Schnitt von allgemeineren Kurven ist man aber auf Iterationsverfahren zum Lösen der entsprechenden Gleichungen angewiesen. Das am meisten angewandte Verfahren ist die **Newton-Iteration:**

Gegeben: Funktion $\mathbf{F} : D \to \mathbb{R}^n, D \subseteq \mathbb{R}^n$, und ein Startpunkt $\mathbf{x}_0$ für die Iteration.
Gesucht: Ein Punkt $\mathbf{x}^*$ in der „Nähe“ von $\mathbf{x}_0$ mit $\mathbf{F}(\mathbf{x}^*) = 0$.

Algorithmus:

(1) Für $\nu = 0, 1, 2, \ldots$ löse man das lineare Gleichungssystem
$\mathbf{F}'(\mathbf{x}_\nu)\mathbf{d}_\nu = -\mathbf{F}(\mathbf{x}_\nu)$, wobei $\mathbf{F}' := (\frac{\partial F_i}{\partial x_k})$ und $\mathbf{F} = (F_1, F_2, \ldots, F_n)$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ ist.

(2) Setze $\mathbf{x}_{\nu+1} = \mathbf{x}_\nu + \mathbf{d}_\nu$

(3) Falls $\|\mathbf{x}_{\nu+1} - \mathbf{x}_\nu\|$ klein „genug“ (oder andere Abbruchbedingung) setze $\mathbf{x}^* = \mathbf{x}_{\nu+1}$.

6.5.1 Schnitt einer parametrisierten mit einer impliziten Kurve

Gegeben: Kurven $\Gamma_1 : \mathbf{x} = \mathbf{c}(t), \quad a \le t \le b$ und $\Gamma_2 : f(\mathbf{x}) = 0, \quad \mathbf{x} \in D \subseteq \mathbb{R}^2$.
Gesucht: Schnittpunkte $\Gamma_1 \cap \Gamma_2$.

Der **Algorithmus:**

(1) Bestimmung eines Startpunktes :
Wähle eine Unterteilung $t_1, t_2, \ldots$ des Intervalls $[a, b]$ (eventuell äquidistant).
Beim Durchlaufen der Kurve Γ_1 auf Punkten $\mathbf{x}_i := \mathbf{c}(t_i), \quad i = 1, 2, \ldots$ prüft man $f(\mathbf{x}_i)$ auf Vorzeichenwechsel. Findet nach $s_0 := t_i$ ein solcher Vorzeichenwechsel statt, so wählt man s_0 als Startwert für die folgende Newton-Iteration.

(2) Bestimme eine Nullstelle t^* der Funktion $F(t) := f(\mathbf{c}(t))$ mit Hilfe der **Newton-Iteration** (s.o.) für $n = 1$ und dem Startwert $t := s_0$ aus (1). Der Punkt $\mathbf{x}^* := \mathbf{c}(t^*)$ ist ein Schnittpunkt der beiden Kurven.

(3) Führe (2) für alle Stellen mit einem Vorzeichenwechsel durch.

Aufgabe 6.2 *Bestimme die Schnittpunkte der Kurven*
$\Gamma_1 : \mathbf{x} = \mathbf{c}(t) := (t, t^3), \quad 0 \le t \le 3,$
$\Gamma_2 : f(x, y) := (x-1)^2 + (y-1)^2 - 10 = 0$

6.5.2 Schnitt zweier impliziter Kurven

Gegeben: $\Gamma_1 : f(\mathbf{x}) = 0$ und $\Gamma_2 : g(\mathbf{x}) = 0, \quad \mathbf{x} \in D \subseteq \mathbb{R}^2$.
Gesucht: Schnittpunkte $\Gamma_1 \cap \Gamma_2$.
Der Algorithmus:

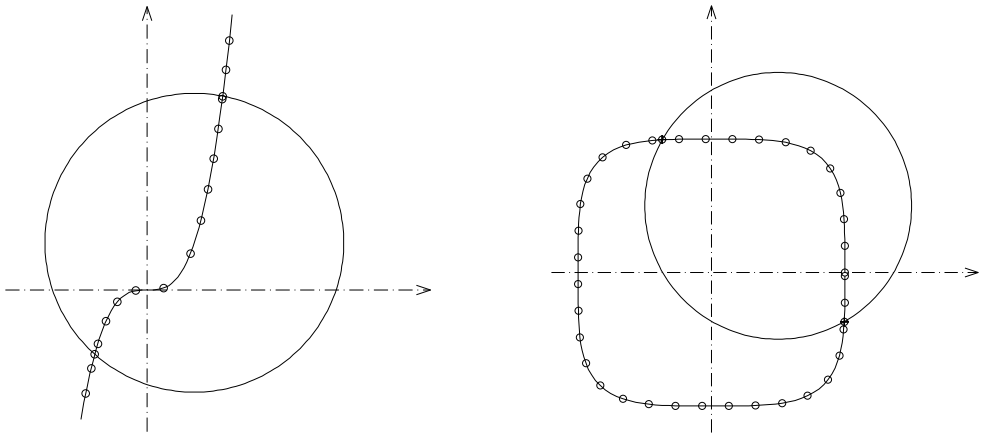


Abbildung 6.12: Links: zu Aufgabe 6.2, rechts: zur Aufgabe 6.3

(1) Bestimmung eines Startpunktes:

Berechnen von Punkten $\mathbf{p}_1, \mathbf{p}_2, ...$ gemäß Abschnitt 6.2.1. Beim Durchlaufen der Kurve Γ_1 auf Punkten $\mathbf{p}_1, \mathbf{p}_2, ...$ prüft man $g(\mathbf{p}_i)$ auf Vorzeichenwechsel. Findet nach $\mathbf{x}_0 := \mathbf{p}_i$ ein solcher Vorzeichenwechsel statt, so wählt man $\mathbf{x}_0$ als Startwert für die folgende Newton-Iteration.

(2) Bestimme eine Nullstelle $\mathbf{x}^*$ der Funktion $\mathbf{F}(\mathbf{x}) := (f(\mathbf{x}), g(\mathbf{x}))$ mit Hilfe der **Newton-Iteration** (s.o.) für $n = 2$ und dem Startpunkt $\mathbf{x}_0$ aus (1).

Der Punkt x ist ein Schnittpunkt der beiden Kurven.

(3) Führe (2) für alle Stellen mit einem Vorzeichenwechsel durch.

Aufgabe 6.3 *Bestimme die Schnittpunkte der Kurven*
$\Gamma_1 : f(x, y) := x^4 + y^4 - 1 = 0$ *und*
$\Gamma_2 : g(x, y) := (x - 0.5)^2 + (y - 0.5)^2 - 1 = 0.$

6.5.3 Schnitt zweier parametrisierter Kurven

Gegeben: Kurven $\Gamma_1 : \mathbf{x} = \mathbf{c}_1(s), s \in [a, b],$ *und* $\Gamma_2 : \mathbf{x} = \mathbf{c}_2(t), t \in [c, d]$.
Gesucht: Schnittpunkte $\Gamma_1 \cap \Gamma_2$.

Der **Algorithmus:**

(1) Bestimmung von Startwerten im Parameterbereich:
Wähle Unterteilungen des Intervalls $[a, b]$ und des Intervalls $[c, d]$ (eventuell äquidistant) und berechne die zugehörigen Polygonzüge $P_1, P_2, ...$ und $Q_1, Q_2, ...$ Bestimme mit dem Algorithmus aus dem nächsten Abschnitt Näherungen für mögliche Schnittpunkte. Die Parameterwerte dieser Näherungen dienen als Startwerte.

(2) Bestimme für Startwerte aus (1) eine Nullstelle der Funktion $\mathbf{F}(s, t) := \mathbf{c}_1(s) - \mathbf{c}_2(t)$ mit Hilfe einer **Newton–Iteration** (s.o.). (Ein Newtonschritt läßt sich geometrisch als Schnitt zweier Tangenten interpretieren.)

(3) Führe (2) für alle Startwerte aus (1) durch.

6.6 Schnitt zweier Polygone

Da oft Kurven als Polygonzüge vorliegen, ist der folgende Algorithmus eine schnelle und universelle Methode um Schnittpunkte von Kurven zu bestimmen. Zur Darstellung der Schnittpunkte auf dem Bildschirm oder Plotter genügt i.a. die Genauigkeit, die man duch diesen Algorithmus erreicht.

Gegeben: Zwei Polygone Π und Ψ durch ihre Punkte $P_1, P_2, ...$ und $Q_1, Q_2, ...$
Gesucht: Schittpunkte $\Pi \cap \Psi$.

Der Algorithmus:

(1) Wähle Schrittweiten und unterteile die Polygone gemäß dieser Schrittweiten in Teilpolygone $\Pi_1, \Pi_2, ...$ bzw. $\Psi_1, \Psi_2, ...$
Bestimme zu jedem Teilpolygon das zugehörige Fenster, d.h. die minimalen bzw. maximalen x- und y-Koordinaten.

(2) Schneide für jedes Paar i, k das Fenster von Π_i mit dem Fenster von Ψ_k.

(3) Haben die Fenster von Π_i und Ψ_k einen nichtleeren Durchschnitt, so

(3.1) bestimme zu jeder in Π_i bzw. Ψ_k enthaltene Strecke das zugehörige Fenster.

(3.2) Schneidet das Fenster der Strecke σ_m von Π_i das Fenster von Ψ_k, so

(3.2.1) schneide das Fenster von σ_m mit den Fenstern der Strecken τ_n von Ψ_k.

(3.2.2) Schneidet das Fenster von σ_m das Fenster von τ_n, so schneide die Strecken σ_m und τ_n. Haben diese Strecken einen Schnittpunkt, so ist dies ein Schnittpunkt der Polygone.

(4) Führe (3) für alle schneidenden Fenster von Teilpolygonen Π_i und Ψ_i durch.

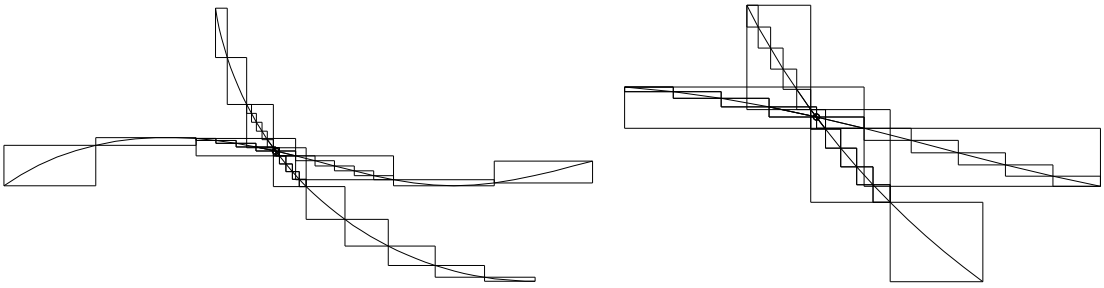


Abbildung 6.13: Zum Schnitt zweier Polygone

Beispiel 6.4 *Schnitt eines Polygons auf einem Kreis mit einem Polygon auf einer Cassini–Kurve (Abb. 6.6).*

6.7 Lotfußpunkt auf einer parametrisierten Kurve, Kurveninversion

Gegeben: Kurve $\Gamma : \mathbf{x} = \mathbf{c}(t), t \in [a, b]$, Punkt $P : \mathbf{p}$.
Gesucht: Lotfußpunkt L des Lotes von P auf Γ.
Vorauss.: $\mathbf{c}(t)$ ist (2-mal) differenzierbar, es existiert ein eindeutiges Lot von P auf Γ.

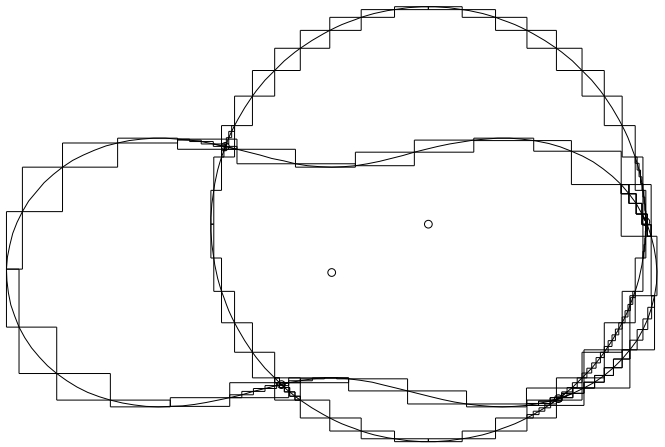


Abbildung 6.14: zum Beispiel 6.4

Ist Γ eine Gerade $\mathbf{x} = \mathbf{q} + t\mathbf{r}$, so ist

$$L : \mathbf{l} = \mathbf{q} + \frac{(\mathbf{p} - \mathbf{q}) \cdot \mathbf{r}}{\mathbf{r}^2}\mathbf{r}.$$

Im allgemeinen Fall suchen wir einen Punkt $\mathbf{c}(t)$ der Kurve Γ mit der Eigenschaft:

$$f(t) := (\mathbf{c}(t) - \mathbf{p}) \cdot \dot{\mathbf{c}}(t) = 0.$$

Der **Algorithmus:**

(1) Bestimmung eines Startparameters:

Man durchläuft mit einer festen oder variablen Schrittweite (im Parameterbereich) die Kurve auf Punkten $Q_1, Q_2, ...$ und sucht einen Punkt $L_0 := Q_i$, der von P minimalen Abstand hat. t_0 sei der zugehörige Parameterwert, d.h. es ist $L_0 : \mathbf{c}(t_0)$.

(2) Mit einem in (1) gefundenen Startwert und einer **Newton–Iteration** löst man die Gleichung $f(t) := (\mathbf{c}(t) - \mathbf{p}) \cdot \dot{\mathbf{c}}(t) = 0$.
Dabei ist $f'(t) = (\dot{\mathbf{c}}(t))^2 + (\mathbf{c}(t) - \mathbf{p}) \cdot \ddot{\mathbf{c}}(t)$.
(Es werden 2. Ableitungen verwendet !)

Oder: Man führt die Lösung auf sukzessives **Lotefällen auf Tangenten** zurück. (Es werden nur 1. Ableitungen verwendet.)

(2') Es sei L_i die i-te Näherung des Lotfußpunktes und t_i der zugehörige Parameter. $\mathbf{x} = \mathbf{c}(t_i) + \Delta t\dot{\mathbf{c}}(t_i)$ ist die Tangente im Punkt L_i (lineare Approximation von $\mathbf{c}(t)$!). Man fällt nun das Lot von P auf die Tangente (s.o.). Für den Lotfußpunkt auf der Tangente ist $\Delta t = (\mathbf{p} - \mathbf{c}(t_i))\dot{\mathbf{c}}(t_i)/\dot{\mathbf{c}}(t_i)^2$. Mit $t_{i+1} = t_i + \Delta t$ erhält man $L_{i+1} : \mathbf{c}(t_{i+1})$.

(3') Wiederhole (2') solange, bis $\|\mathbf{c}(t_{i+1}) - \mathbf{c}(t_i)\|$ klein genug ist. (Oder ein anderes Abbruchkriterium.)

Bemerkung: Dieser Algorithmus kann auch dazu verwendet werden, um zu einem Punkt $P_0 : \mathbf{p}_0$, von dem man weiß, daß er auf der Kurve $\Gamma : \mathbf{c}(t)$ liegt, den zugehörigen Kurvenparameter t_0 (mit $\mathbf{p}_0 = \mathbf{c}(t_0)$) zu bestimmen. In diesem Fall wird die Länge von $\mathbf{c}(t) - \mathbf{p}_0$ klein und man kann auf jeden Fall (2') und (3') verwenden.

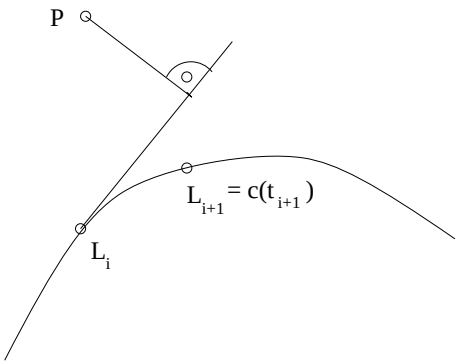


Abbildung 6.15: Iterationsschritt zur Lotfußpunktbestimmung

Die Zuordnung $\mathbf{p}_0 \to t_0$ heißt **Kurveninversion**.
(Man beachte, daß zu einem Punkt $\mathbf{p}_0$ auch mehrere Parameter gehören können!)

Bemerkung: Der obige Algorithmus kann analog für das Fällen eines Lotes auf eine **räumliche Kurve** bzw. zur **Kurveninversion im** $\mathbb{R}^3$ verwendet werden.

6.8 Lotfußpunkt auf einer impliziten Kurve

Gegeben: Kurve $\Gamma : f(\mathbf{x}) = 0, \mathbf{x} \in D \subseteq \mathbb{R}^2$ und Punkt $P : \mathbf{p}$. f 2-mal diff., $\nabla f \neq 0$.
Gesucht: Lotfußpunkt L des Lotes von P auf Γ.

Der **Algorithmus**:

(1) Mit Hilfe der Prozedur `curvepoint` aus Abschnitt 6.2 findet man eine erste Näherung $L_0 : \mathbf{x}_0$ für den Lotfußpunkt.

(2) Der Lotfußpunkt ergibt sich als Lösung des Systems

$$(\mathbf{x} - \mathbf{p}) \cdot (f_y(\mathbf{x}), -f_x(\mathbf{x})) = 0, \quad f(\mathbf{x}) = 0.$$

(Dabei ist $(f_y(\mathbf{x}),\ -f_x(\mathbf{x}))$ ein Tangentenvektor im Kurvenpunkt $\mathbf{x}$.)

Dieses System löst man mit Hilfe einer **Newton–Iteration** unter Verwendung des Startpunktes $\mathbf{x}_0$ aus (1).
(Es werden 2. Ableitungen verwendet !)

Oder: Man führt die Lösung auf sukzessives Lotefällen auf Tangenten zurück. (Es werden nur 1. Ableitungen vewendet.)

(2') wiederhole $\mathbf{t}_{i+1} = \mathbf{p} - \dfrac{(\mathbf{p} - \mathbf{x}_i) \cdot \nabla f(\mathbf{x}_i)}{\nabla f(\mathbf{x}_i)^2} \nabla f(\mathbf{x}_i)$ (Lotfußpunkt auf der Tangente),
$\mathbf{x}_{i+1} = \texttt{curvepoint}(\mathbf{t}_{i+1})$.
bis $\|\mathbf{x}_{i+1} - \mathbf{x}_i\|$ "klein genug" ist.
$L = \mathbf{x}_{i+1}$.

6.9 Die Normalform einer ebenen Kurve

Analog zur HESSE-Normalform einer Gerade (im $\mathbb{R}^2$) werden wir eine Normalform für fast beliebige ebene Kurven definieren. Obwohl sich diese Normalform nur in wenigen Fällen explizit angeben läßt (z.B.: Gerade, Kreis) werden wir sehen, daß dies ein starkes Hilfsmittel in CAGD ist. Wir werden die Normalform zunächst für parametrisierte Kurven definieren und anschließend verallgemeinern insbesondere auf implizite Kurven.

6.9.1 Die Normalform einer parametrisierten Kurve

Es sei

- $\Gamma : \mathbf{x} = \mathbf{c}(t) = (c_1(t), c_2(t)),\ t \in [a, b]$ eine glatte ebene Kurve,
- $\mathbf{n}(t) := (c_2(t), -c_1(t))/\|...\|$ die Einheitsnormale
- $D \subset \mathbb{R}^2$ so, daß für jedes $\mathbf{x} \in D$ die Funktion $d_{\mathbf{x}}(t) := \|\mathbf{x} - \mathbf{c}(t)\|$ genau ein Minimum besitzt,
- $t_m(\mathbf{x})$ der Parameter des Minimums, $\mathbf{c}(t_m(\mathbf{x}))$ der *Lotfußpunkt* (zu $\mathbf{x}$),
- $h(\mathbf{x}) := sign\big((\mathbf{x} - \mathbf{c}(t_m(\mathbf{x}))) \cdot \mathbf{n}(t_m(\mathbf{x}))\big)\ d_{\mathbf{x}}(t_m(\mathbf{x}))$ die *orientierte Distanz* von $\mathbf{x}$ zur Kurve Γ.

Die Gleichung $h(\mathbf{x}) = 0$ ist eine implizite Darstellung von Γ und heißt die *Normalform* von Γ.

Bemerkung:
Die Normalform einer Gerade ist die wohlbekannte HESSE–Normalform
$h(\mathbf{x}) := \mathbf{n} \cdot (\mathbf{x} - \mathbf{x}_1) = 0$, wobei $\mathbf{n}$ eine Einheitsnormale und $\mathbf{x}_1$ ein Punkt der Gerade ist.

Wesentliche Eigenschaften der orientierten Distanzfunktion:

(1) Höhenlinien der orientierten Distanzfunktion h sind Offsetkurven von Γ. (Eine Offsetkurve zur Distanz δ ist die Menge aller Punkte, die zu Γ den Abstand δ haben.)

(2) h ist differenzierbar: $\nabla h(\mathbf{x}) = \mathbf{n}(t_m(\mathbf{x}))$ (Einheitsnormale im Lotfußpunkt zu $\mathbf{x}$).

Die zweite Eigenschaft kann man sich so klar machen: Die Richtung der maximalen Zunahme von h ist die Richtung der Normalen im Lotfußpunkt. Die maximale Zunahme hat den Betrag 1.

Die **Berechnung** eines Funktionswertes $h(\mathbf{x})$ erfordert also "nur", den Lotfußpunkt von $\mathbf{x}$ auf Γ zu bestimmen (s. 6.7). $\nabla h(\mathbf{x})$ bekommt man dann praktisch geschenkt. Damit hat man zwar eine implizite Darstellung der Offsetkurven einer parametrisierten Kurve, was kein großer Gewinn ist. Denn die Offsetkurven einer parametrisierten Kurve lassen sich leicht auch wieder parametrisiert darstellen. Mit der Normalform werden aber insbesondere "Blenden" von ebenen Kurven (s. ???) die Konstruktion von "Voronoi–Kurven" (s. 6.9.4.2) sehr erleichtert.

6.9.2 Verallgemeinerung

Es ist nicht nötig, daß die Kurve Γ des vorigen Abschnitts parametrisiert gegeben ist. Die einzigen notwendigen Bedingungen sind:

(NF1) Γ muß glatt sein und

(NF2) es gibt ein Gebiet $D \subset \mathbb{R}^2$ in der Umgebung von Γ so, daß es für jeden Punkt $\mathbf{x} \in D$ möglich ist, einen (eindeutigen) *Lotfußpunkt* (Punkt auf Γ mit minimaler Distanz zu $\mathbf{x}$) zu bestimmen.

Folgerung:
Jede ebene Kurve Γ mit den Eigenschaften (NF1), (NF2) läßt sich als implizite Kurve $h(\mathbf{x}) = 0$ auffassen, wobei $h(\mathbf{x})$ die orientierte Distanz des Punktes $\mathbf{x} \in D$ zur Kurve Γ und $\nabla h(\mathbf{x})$ die Einheitsnormale im Lotfußpunkt zu $\mathbf{x}$ ist. Die Gleichung $h(\mathbf{x}) = 0$ heißt die *Normalform* von Γ.

Also läßt sich jeder Algorithmus für implizite Kurven, der nur $h(\mathbf{x})$ und $\nabla h(\mathbf{x})$ für $\mathbf{x}$ verwendet auf Γ anwenden, z..B. der Verfolgungsalgorithmus zur Erzeugung von Punkten, die Newton–Iteration für die Bestimmung von Schnittpunkten.

Bemerkung:
a) Die Bestimmung des Bereichs D (vgl. (NF2)) ist i.a. schwierig.
b) Die Konvergenz des Lotfußpunkt–Algorithmus wird in der Nähe des Randes von D kritisch.

6.9.3 Die Normalform einer implziten Kurve

Es sei $\Gamma_0 : \quad f = 0$ eine glatte implizite Kurve. Normalerweise hat der Funktionswert $f(\mathbf{x})$ keine konkrete geometrische Bedeutung. Der Vorteil der Normalform $h = 0$ einer Kurve liegt in der einfachen geometrischen Bedeutung des Funktiondwertes $h(\mathbf{x})$ (orientierte Distanz) und der Höhenlinien $h(\mathbf{x}) = c$ (Offsetkurven). Also: Falls die Distanz eines Punktes zur Kurve oder Offsetkurven eine Rolle spielen, sollte man auf die Normalform zurückgreifen.
Die **Berechnung** eines Funktionswertes $h(\mathbf{x})$ verlangt die Bestimmung des Lotfußpunktes von $\mathbf{x}$ auf Γ (s. 6.8).

6.9.4 Anwendungen der Normalform

In diesem Abschnitt ist die Anwendung der Normalform auf Offsetkurven und Voronoi–Kurven enthalten. Den großen Vorteil, den die Normalform für das Blenden von beliebigen Kurven bietet, werden wir in Kap. ???? sehen. Man sollte auf jeden Fall beachten, daß die Normalform fast jede Kurve zu einer impliziten Kurve macht, solange nur die Funktionswerte und der Gradient der orientierten Distanzfunktion benötigt werden.

6.9.4.1 Offsetkurven von impliziten Kurven

Offsetkurven einer Kurve sind Höhenlinien der orientierten Distanzfunktion: $h = c$.

Beispiel 6.5 *In Abb. 6.16 ist $f(\mathbf{x}) = x^4+y^4-1 = 0$ (dicke Kurve) und einige Offsetkurven zu sehen. Man beachte, daß einige der inneren Offsetkurven Singularitäten auf Symmetriegeraden besitzen.*

6.9.4.2 Voronoi–Kurven zweier Kurven

Die *Voronoi–Kurven* zweier Kurven Γ_1, Γ_2 bestehen aus den Punkten des $\mathbb{R}^2$, die zu Γ_1 und Γ_2 denselben Abstand haben.
Die Voronoi–Kurven von Geraden sind ihre Winkelhalbierenden.

Es seien Γ_1, Γ_2 zwei glatte parametrisierte oder implizite Kurven und $h_1 = 0, h_2 = 0$ ihre Normalformen. Dann sind die Gleichungen

$$h_1(\mathbf{x}) - h_2(\mathbf{x}) = 0, \quad h_1(\mathbf{x}) + h_2(\mathbf{x}) = 0$$

implizite Darstellungen der Voronoi–Kurven von Γ_1 und Γ_2). Punkte von Voronoi–Kurven können also mit Hilfe des Verfolgungsalgorithmus für implizite Kurven berechnet werden.

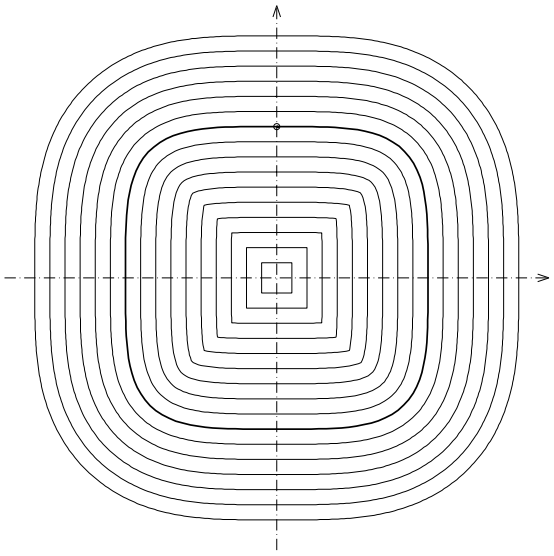


Abbildung 6.16: Offsetkurven der impliziten Kurve $f(\mathbf{x}) = x^4 + y^4 - 1 = 0$

Beispiel 6.6 *Abb. 6.17 zeigt die Voronoi–Kurven zweier Bézierkurven.*

Arten die beiden Kurven zu zwei Punkte aus, so ist die Voronoi–Kurve ihre Mittelsenkrechte. Den Fall, daß eine Kurve zu einem Punkt ausartet zeigt das folgende Beispiel:

Beispiel 6.7 *Abb. 6.18 zeigt die Voronoi–Kurve*

a) der Ellipse $x^2 + \frac{y^2}{4} = 1$ *und dem Punkt* $(-0.5, 1)$

b) der Kurve $x^4 + y^4 = 1$ *und dem Punkt* $(-0.4, 0.6)$.

Bei dem letzten Beispiel muß man allerdings beachten, daß die Normalform der Ellipse bzw. der Kurve 4. Grades Singularitäten auf den Symmetriegeraden besitzt.

Weitere Informationen hierzu sind in FA,JO'94a, FA,JO'94b enthalten.

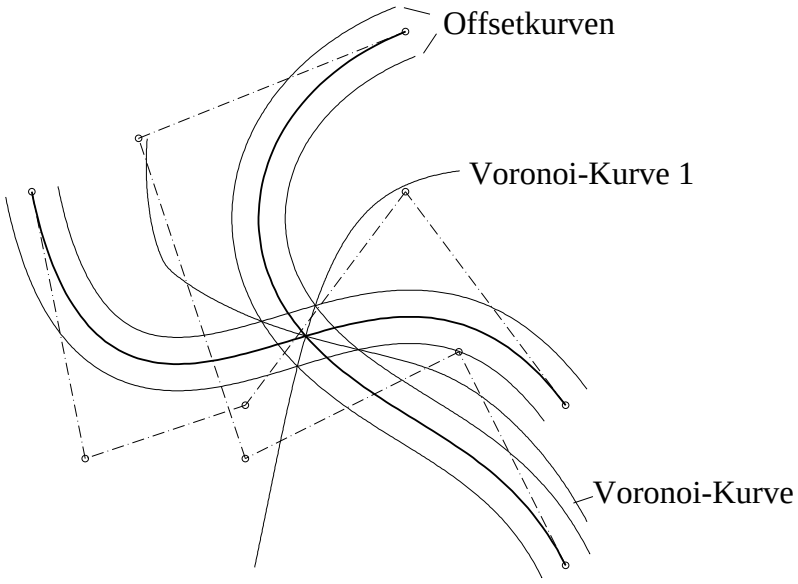


Abbildung 6.17: Voronoi–Kurven zweier Bézierkurven (fett)

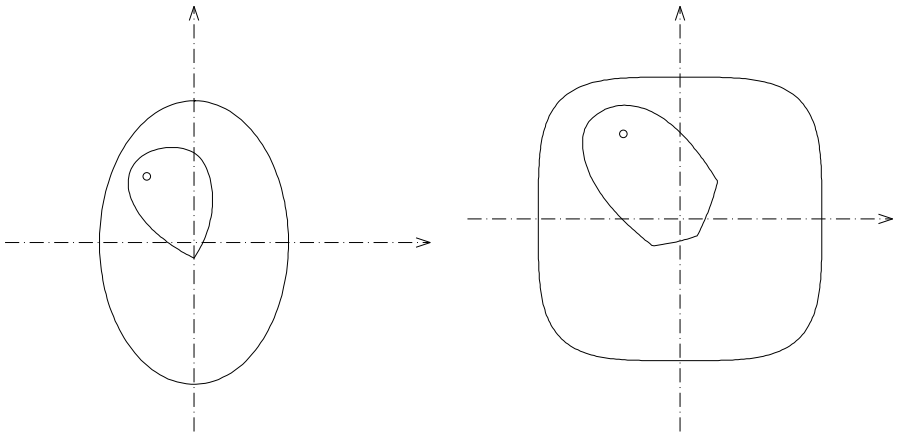


Abbildung 6.18: Voronoi–Kurve eines Punktes und a) einer Ellipse b) Kurve 4. Grades

Kapitel 7

ELLIPSOID UND PARABOLOID

Quadriken, insbesondere Kegel, Kugel, Zylinder, spielen in der Darstellenden Geometrie eine wesentliche Rolle. Wir wollen hier nicht nur diese Standardquadriken, sondern deren affinen Bilder und die affinen Bilder einiger weiterer Quadriken, wie das einschalige Hyperboloid (Kühlturm) und das Rotationsparaboloid, behandeln. Ein großer Vorteil aller Quadriken ist, daß ihre Umrisse immer in einer Ebene liegen. Dies hat zur Folge, daß sich die Sichtbarkeit von Kurven auf Quadriken besonders einfach klären läßt. Denn ein Quadrikenpunkt "hinter"der Umrißebene kann nie sichtbar sein.

7.1 Ellipsoid

Die Definition eines Ellipsoids erfolgt analog zur Definition einer Ellipse (s. Kapitel 5.1).
Jedes affine Bild der Einheitskugel $K := \{(x, y, z) \mid x^2 + y^2 + z^2 = 1\}$ heißt **Ellipsoid**.
Beschreibt man die Einheitskugel K_1 mit Hilfe der "Längen"- und "Breiten"-winkel (wie bei der Erdkugel):

$$K_1 = \{(\cos\beta\cos\alpha, \cos\beta\sin\alpha, \sin\beta) \mid 0 \le \alpha \le 2\pi, -\pi/2 \le \beta \le \pi/2\},$$

so gibt es zu jedem Ellipsoid E einen Vektor $\mathbf{q}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$, so daß

$$E = \{\mathbf{q}_0 + \mathbf{f}_1\cos\beta\cos\alpha + \mathbf{f}_2\cos\beta\sin\alpha + \mathbf{f}_3\sin\beta \mid \ldots\}$$

ist. Umgekehrt ist für einen Vektor $\mathbf{q}_0$, und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ die Punktmenge $E = \{\ldots \mid \ldots\}$ ein Ellipsoid.

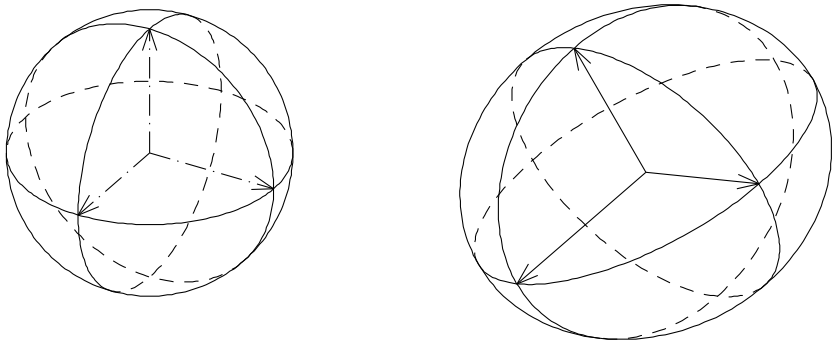


Abbildung 7.1: Ellipsoid als affines Bild der Einheitskugel

Die Koordinaten ξ, η, ζ eines Punktes von E bezüglich des Koordinatensystems mit Nullpunkt Q_0: $\mathbf{q}_0$ und der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ genügen also der Gleichung

$$\xi^2 + \eta^2 + \zeta^2 = (\cos\beta\cos\alpha)^2 + (\cos\beta\sin\alpha)^2 + (\sin\beta)^2 = \ldots = 1,$$

d. h. E ist bezüglich des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ eine "Einheitskugel".

$Q_0 : \mathbf{q}_0$ heißt Mittelpunkt des Ellipsoids E.
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ heißen **konjugierte Halbmesser** von E und
$Q_1 : \mathbf{q}_0 + \mathbf{f}_1, Q_2 : \mathbf{q}_0 + \mathbf{f}_2, Q_3 : \mathbf{q}_0 + \mathbf{f}_3$ **konjugierte Punkte** von E.
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sind i.a. nicht orthogonal. Aber es gilt:
Die Tangentialebene in Q_1 ist parallel zur Ebene, die von $\mathbf{f}_2, \mathbf{f}_3$ aufgespannt wird, usw.
Falls $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ paarweise senkrecht zueinander stehen, nennt man die Punkte $\mathbf{q}_0 \pm \mathbf{f}_1, \mathbf{q}_0 \pm \mathbf{f}_2, \mathbf{q}_0 \pm \mathbf{f}_3$ **Scheitel** von E.
Stehen $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ paarweise zu einander senkrecht und gilt $|f_i| = r,\ i = 1, 2, 3$, so ist E eine Kugel mit Radius r.

Die **Gleichung** $f(x, y, z) = 0$ des Ellipsoids $\mathbf{x} = \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta,\ \xi^2 + \eta^2 + \zeta^2 = 1$ ergibt sich aus dem Gleichungssystem

$$\begin{array}{rcll} x & = & f_{1x}\xi + f_{2x}\eta + f_{3x}\zeta & \quad (\text{mit} \quad \mathbf{x} := (x, y, z),\ \mathbf{f}_i = (f_{ix}, f_{iy}, f_{iz})) \\ y & = & f_{1y}\xi + f_{2y}\eta + f_{3y}\zeta & \\ z & = & f_{1z}\xi + f_{2z}\eta + f_{3z}\zeta. & \end{array}$$

Löst man nach ξ, η und ζ auf, so erhält man aus $\xi^2 + \eta^2 + \zeta^2 = 1$ die Gleichung

$$f(x, y, z) := (\det(\mathbf{x}, \mathbf{f}_2, \mathbf{f}_3))^2 + (\det(\mathbf{f}_1, \mathbf{x}, \mathbf{f}_3))^2 + (\det(\mathbf{f}_1, \mathbf{f}_2, \mathbf{x}))^2 - (\det(\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3))^2 = 0,$$

wobei $\det(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ die 3×3-Determinate mit den Spaltenvektoren $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$ ist.

7.1.1 Ebene Schnitte einer Kugel

Der ebene Schnitt einer Kugel ist entweder leer, besteht aus einem Punkt oder aus einem Kreis. Es ist nützlich ein Unterprogramm zu haben, das für eine beliebige Kugel und eine beliebige Ebene deren Schnitt berechnet. Wir werden sehen, daß man dieses Unterprogramm nicht nur für Kugeln sondern auch für Ellipsoide verwenden kann. Insbesondere zur Bestimmung des Umrisses eines Ellipsoids in Parallel- und Zentralprojektion werden wir auf das unten angegebene Unterprogramm zurückgreifen.
Gegeben: Kugel $K : x^2 + y^2 + z^2 = r^2, \quad r > 0,$
Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$.
Gesucht: Schnitt $\varepsilon \cap K$. Der Abstand der Ebene ε zum Mittelpunkt der Kugel ist $\delta = \frac{|d|}{\|\mathbf{n}\|}$.
Falls $\delta > r$ ist, ist der Schnitt **leer**.
Falls $\delta = r$ ist, besteht der Schnitt aus einem **Punkt**.
Falls $\delta < r$ ist, gibt es einen Schnitt**kreis**.

a) Der Radius des Schnittkreises ist $\rho = \sqrt{r^2 - \delta^2}$.

b) Der Mittelpunkt ist $\mathbf{p}_0 := \frac{d}{\mathbf{n}^2}\mathbf{n}$.

c) Konjugierte Punkte des Kreises:
Im Folgenden sei $\mathbf{n}_0 = (n_x, n_y, n_z) := \frac{\mathbf{n}}{\|\mathbf{n}\|}$ und
$\mathbf{f}_1 := \rho(n_y, -n_x, 0)/\sqrt{n_x^2 + n_y^2}$, falls $|n_x| > 0.5$ oder $|n_y| > 0.5$ ist, andernfalls sei
$\mathbf{f}_1 := \rho(0, n_z, -n_y)/\sqrt{n_y^2 + n_z^2}$.

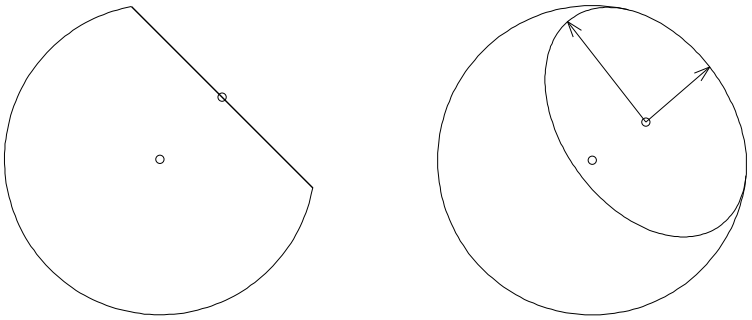


Abbildung 7.2: ebener Schnitt einer Kugel

Ein zu $\mathbf{f}_1$ konjugierter Radiusvektor ist $\mathbf{f}_2 := \mathbf{n}_0 \times \mathbf{f}_1$. Also sind
$P_1 : \mathbf{p}_1 = \mathbf{p}_0 + \mathbf{f}_1$ und $P_2 : \mathbf{p}_2 = \mathbf{p}_0 + \mathbf{f}_2$ konjugierte Punkte des Schnittkreises.

Das folgende Unterprogramm `is_sphere_plane` bestimmt die ebenen Schnitte einer Kugel mit Mittelpunkt $\mathbf{p}_m$ und Radius r.

```
procedure is_sphere_plane(pm: vt3d; r: real; nv: vt3d; d: real; var p0,p1,p2: vt3d;
                                                                 var nis: integer);
 {Berechnet gegebenenfalls (nis=2) pc des Schnittes der Kugel mit Mittelpkt pm,
  Radius r und der Ebene nv*x=d.}
```

Da wir oft ebene Schnitte der Einheitskugel bestimmen müssen, geben wir noch das entsprechende Uterprogramm `is_unitsphere_plane` an.

```
procedure is_unitsphere_plane(a,b,c,d: real; var pc0,pc1,pc2: vt3d; var nis: integer);
 var nv : vt3d;
 begin
   put3d(a,b,c, nv);
   is_sphere_plane(null3d,1, nv,d, pc0,pc1,pc2, nis);
 end;    {is_unitsphere_plane}
```

7.1.2 Ebene Schnitte eines Ellipsoids

Gegeben: a) Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$,
b) Ellipsoid $E : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \qquad \xi^2 + \eta^2 + \zeta^2 = 1.$
Gesucht: Schnitt $\varepsilon \cap E$.

Durch Einsetzen der Parameterdarstellung des Ellipsoids in die Gleichung der Ebene erhält man das Gleichungssystem:

$$\begin{array}{lcl} \gamma_1\xi + \gamma_2\eta + \gamma_3\zeta & = & \gamma_0, \quad \text{mit } \gamma_i = \mathbf{n} \cdot \mathbf{f}_i,\ i = 1,2,3,\ \gamma_0 = d - \mathbf{n} \cdot \mathbf{q}_0, \\ \xi^2 + \eta^2 + \zeta^2 & = & 1 \end{array}$$

Löst man dieses System mit Hilfe von `is_unitsphere_plane`, so erhält man gegebenenfalls Mittelpunkt und konjugierte Punkte der Schnittellipse zunächst in $\xi-\eta-\zeta-$Koordinaten. Durch Einzetzen in $\mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta$ ergeben sich die zugehörigen $x - y - z-$Koordinaten.

```
procedure is_ellipsoid_plane(q0,q1,q2,q3,nv: vt3d; d: real; var pc0,pc1,pc2: vt3d;
                                                            var nis : integer);
 {Berechnet den Schnitt des Ellipsoids mit Mittelpunkt q0 und den konjugierten Punkten
  q1,q2,q3  mit der Ebene  nv*x=d.}
```

7.1.3 Parallelprojektion eines Ellipsoids

Gegeben: a) Ellipsoid $E : \mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 + \zeta^2 = 1.$
b) Parallelprojektion durch $\mathbf{n}_0$ (Normalenvektor auf Bildtafel).
Gesucht: Umriß von E.

Ein Punkt Q von E ist ein Umrißpunkt, wenn die Normale in Q zur Projektionsrichtung und damit zum Vektor $\mathbf{n}_0$ senkrecht steht. Die Normale $\mathbf{n}(\alpha,\beta)$ auf E in einem Punkt $\mathbf{q}(\alpha,\beta) = \mathbf{q}_0 + \mathbf{f}_1\cos\beta\cos\alpha + \mathbf{f}_2\cos\beta\sin\alpha + \mathbf{f}_3\sin\beta, \quad \beta \neq \pm\,\pi/2$, hat die Richtung

$$\mathbf{q}_\alpha(\alpha,\beta) \times \mathbf{q}_\beta(\alpha,\beta) = \cos\beta(\mathbf{f}_2 \times \mathbf{f}_3\cos\beta\cos\alpha + \mathbf{f}_3 \times \mathbf{f}_1\cos\beta\sin\alpha + \mathbf{f}_1 \times \mathbf{f}_2\sin\beta).$$

Da die Normale im Punkt $\mathbf{q}(\alpha,\pm\,\pi/2)$ die Richtung von $\mathbf{f}_1 \times \mathbf{f}_2$ hat (s.o.), können wir als Normale in einem beliebigen Punkt $\mathbf{q}(\alpha,\beta)$ den folgenden Vektor verwenden:

$$\mathbf{n}(\alpha,\beta) = \mathbf{f}_2 \times \mathbf{f}_3\cos\beta\cos\alpha + \mathbf{f}_3 \times \mathbf{f}_1\cos\beta\sin\alpha + \mathbf{f}_1 \times \mathbf{f}_2\sin\beta.$$

Der Punkt $\mathbf{q}(\alpha,\beta)$ ist genau dann Umrißpunkt, wenn $\mathbf{n}_0 \cdot \mathbf{n}(\alpha,\beta) = 0$ ist:

$$\mathbf{n}_0 \cdot \mathbf{n}(\alpha,\beta) = \mathbf{n}_0 \cdot (\mathbf{f}_2 \times \mathbf{f}_3)\cos\beta\cos\alpha + \mathbf{n}_0 \cdot (\mathbf{f}_3 \times \mathbf{f}_1)\cos\beta\sin\alpha + \mathbf{n}_0 \cdot (\mathbf{f}_1 \times \mathbf{f}_2)\sin\beta = 0.$$

Die $\xi-\eta-\zeta-$Koordinaten eines Umrißpunktes bzgl. des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ ergeben sich also aus dem Gleichungssystem

$$\begin{array}{lcl} \gamma_1\xi + \gamma_2\eta + \gamma_3\zeta & = & 0, \quad \text{mit} \quad \gamma_1 = \mathbf{n}_0 \cdot (\mathbf{f}_2 \times \mathbf{f}_3),\ \gamma_2 = \mathbf{n}_0 \cdot (\mathbf{f}_3 \times \mathbf{f}_1),\ \gamma_3 = \mathbf{n}_0 \cdot (\mathbf{f}_1 \times \mathbf{f}_2), \\ \xi^2 + \eta^2 + \zeta^2 & = & 1 \end{array}$$

mit Hilfe von `is_unitsphere_plane` erhält man die $\xi-\eta-\zeta-$Koordinaten konjugierter Punkte der Umrißellipse und mit $\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta$ schließlich die $x-y-z-$Koordinaten.
Das folgende Unterprogramm `ellipsoid_pp_cont` berechnet den Mittelpunkt und zwei konjugierte Punkte der Umrißellipse des Ellipsoids mit Mittelpunkt $Q_0 : \mathbf{q}_0$ und den konjugierten Punkten $Q_1 : \mathbf{q}_i,\ i = 1,2,3$. (Die Vektoren $\mathbf{f}_i$ ergeben sich aus $\mathbf{f}_i = \mathbf{q}_i - \mathbf{q}_0$.)

```
procedure ellipsoid_pp_cont(q0,q1,q2,q3: vt3d; var qc0,qc1,qc2 : vt3d);
  {Berechnet den Umriss des durch q0, q1, q2, q3 bestimmten Ellipsoids.}
```

Das Unterprogramm `pp_ellipsoid` projiziert den mit `ellipsoid_pp_cont` berechneten Umriß des durch Q_0, Q_1, Q_2, Q_3 bestimmten Ellipsoids.

```
procedure pp_ellipsoid(q0,q1,q2,q3: vt3d);
 {Projektion des durch q0,q1,q2,q3 (s. ellipsoid_pp_cont) bestimmten Ellipsoids.}
```

Beispiel 7.1 *Ellipsoid mit Längen- und Breiten-"kreise".*

```
{************************************************}
{***  Ellipsoid mit L"angen- und Breiten-"kreisen"  ***}
{************************************************}
program ellipso;
uses geograph;
 var q0,q1,q2,q3,pc0,pc1,pc2,qc0,qc1,qc2,nv,p1,p2 : vt3d;
     rx,ry,rz,d,dw,cw,sw,cdw,sdw,ccw,am,amd,z0 : real;
     nb,nl,isi,inds,i,inz,ipld : integer;     error : boolean;
{******************}
 begin  {Hauptprogramm}
```

```
   writeln('pld-Datei ?');   readln(ipld);
   graph_on(ipld);
   writeln(' *** Ellipsoid mit L"angen- und Breiten-"kreisen" ***');
   writeln(' Halbachsen des Ellipsoids (in x-, y-, z-Richtung)?');
   readln(rx,ry,rz);
   writeln(' Anzahl der Laengen-"kreise" ?');                readln(nl);
   writeln(' Anzahl der Breiten-"kreise" ?');                readln(nb);
   put3d(0,0,0, q0); put3d(rx,0,0, q1); put3d(0,ry,0, q2); put3d(0,0,rz, q3);
   repeat
     init_parallel_projection;                  inds:= 0;
     writeln(' Unsichtbare Kurven stricheln ? (Ja = 1)');   readln(isi);
     if isi=1 then       inds:= 10;
     am:= max(ry,rz);      am:= max(rx,am);      amd:= 2*am;
     draw_area(amd+10,amd+10,am+5,am+5,1);
     ellipsoid_pp_cont(q0,q1,q2,q3, qc0,qc1,qc2);
     plane_equ(qc0,qc1,qc2, nv,d,error);     { Umriss-Ebene }
{Umrissellipse: }        new_color(yellow);
     pp_ellipse(qc0,qc1,qc2,0);
{ Nordpol,Suedpol : }
     if nl<2 then
        begin
        put3d(0,0,rz, p1); put3d(0,0,-rz, p2); pp_point(p1, 0); pp_point(p2, 0);
        end;
{ Laengen-"kreise" : }
     pc0:= null3d;   put3d(0,0,rz, pc1);
     if nl>0 then
        begin
        dw:= pi/nl;    cdw:= cos(dw);  sdw:= sin(dw);    cw:= 1;  sw:= 0;
        for i:= 1 to nl do
            begin
            put3d(rx*cw,ry*sw,0, pc2);
            pp_ellipse_before_plane(pc0,pc1,pc2,nv,d,1,inds);
            ccw:= cw;  cw:= cw*cdw - sw*sdw;  sw:= sw*cdw + ccw*sdw;
            end; {for}
        end; {if nl>0}
{ Breiten-"kreise" : }
     if nb>0 then
        begin
        dw:= pi/(nb+1);   cdw:= cos(dw);  sdw:= sin(dw);   sw:= -cdw;  cw:= sdw;
        for i:= 1 to nb do
            begin
            z0:= rz*sw;
            put3d(0,0,z0, pc0);  put3d(rx*cw,0,z0, pc1);  put3d(0,ry*cw,z0, pc2);
            pp_ellipse_before_plane(pc0,pc1,pc2,nv,d,1,inds);
            ccw:= cw;   cw:= cw*cdw - sw*sdw;  sw:= sw*cdw + ccw*sdw;
            end; {for}
        end;  { if nb>0 }
     draw_end;
     writeln(' Noch eine Zeichnung aus anderer Sicht ? (Ja = 1)');  readln(inz);
   until inz=0;
   graph_off;
 end.
```

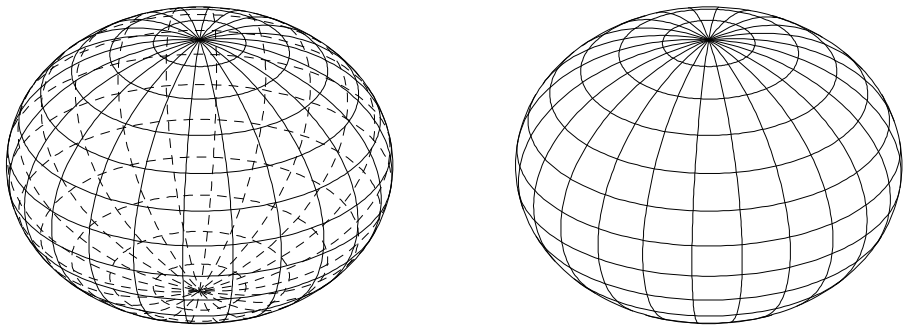


Abbildung 7.3: zu Beispiel 7.1

Aufgabe 7.1 *Ebener Schnitt eines Ellipsoids mit Längen- und Breiten-"kreise".*

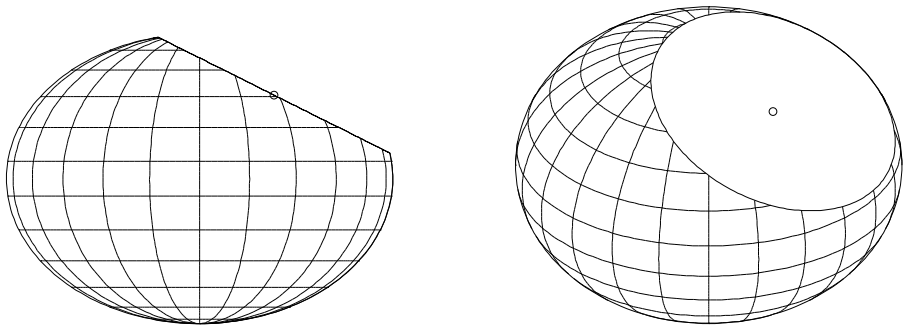


Abbildung 7.4: zu Aufgabe 7.1

Aufgabe 7.2 *a) Ellipsoid mit Kontinente,* *b) Fußball (vgl. Aufgabe 3.2)*

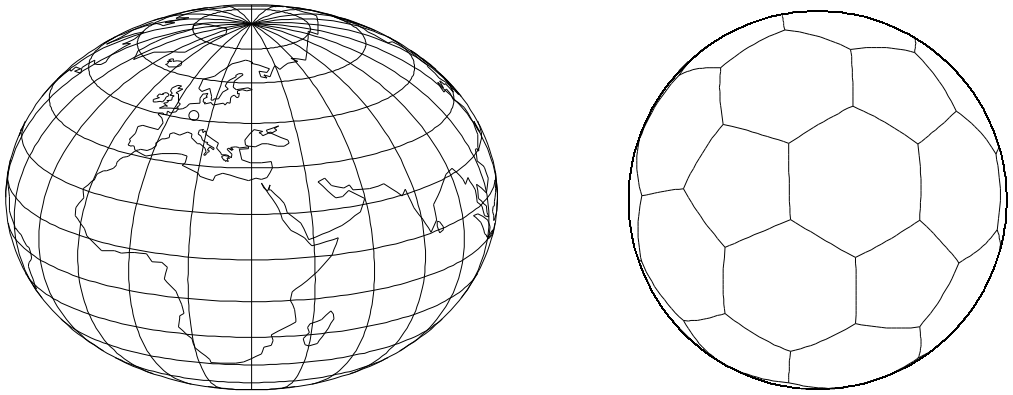


Abbildung 7.5: zu Aufgabe 7.2

7.1.4 Zentralprojektion eines Ellipsoids

Für das Ellipsoid: $\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \qquad \xi^2 + \eta^2 + \zeta^2 = 1.$
lautet die Umrißbedingung: $\gamma_1\xi + \gamma_2\eta + \gamma_3\zeta = \gamma_0, \quad \xi^2 + \eta^2 + \zeta^2 = 1,$
mit $\gamma_0 = \mathbf{f}_1 \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_1 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_2 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_3 \times \mathbf{f}_1), \quad \gamma_3 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_1 \times \mathbf{f}_2).$

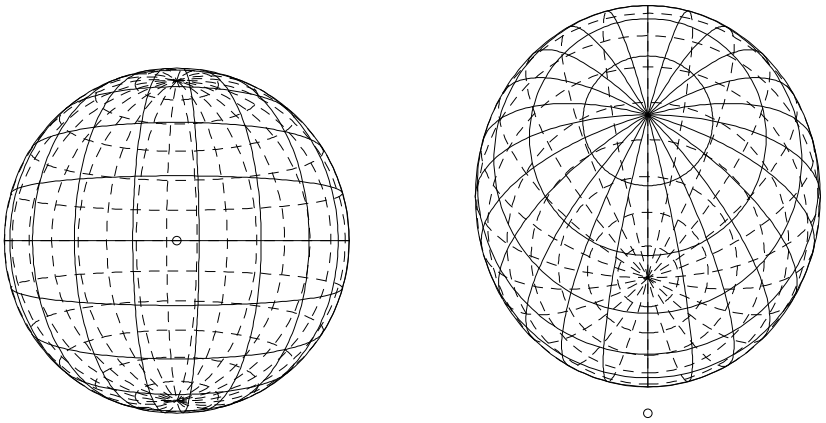


Abbildung 7.6: Zentralprojektion einer Kugel

7.2 Paraboloid

Durch Rotation der Einheitsparabel $z = x^2$ um die $z-$Achse erhält man das Einheitsparaboloid $z = x^2 + y^2$.

Ein affines Bild des Einheitsparaboloids $P_1 := \{(x, y, z) \in \mathbb{R}^3 \,|\, z = x^2 + y^2\}$ heißt **Paraboloid**.
Beschreibt man P_1 durch die Parameterdarstellung: $P_1 = \{(s, t, s^2 + t^2) \,|\, s, t \in \mathbb{R}\}$, so gilt

Zu jedem Paraboloid P gibt es einen Vektor $\mathbf{q}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$, so daß

$$P = \{\mathbf{q}_0 + \mathbf{f}_1 s + \mathbf{f}_2 t + \mathbf{f}_3(s^2 + t^2) \,|\, \ldots\}$$

ist. Umgekehrt ist für einen Vektor $\mathbf{q}_0$, und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ die Punktmenge $P = \{\ldots \,|\, \ldots\}$ ein Paraboloid.
Die Koordinaten ξ, η, ζ eines Punktes von P bezüglich des Koordinatensystems mit Nullpunkt $Q_0 : \mathbf{q}_0$ und der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ genügen also der Gleichung $\zeta = \xi^2 + \eta^2$ d. h. P ist bezüglich des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ das "Einheitsparaboloid".

$Q_0 : \mathbf{q}_0$ ist ein Punkt von P.
$Q_1 : \mathbf{q}_0 + \mathbf{f}_1, Q_2 : \mathbf{q}_0 + \mathbf{f}_2$ sind Punkte der Tangentialebene in Q_0.
$\mathbf{f}_3$ ist parallel zur Achse von P.
$Q_{13} : \mathbf{q}_0 + \mathbf{f}_1 + \mathbf{f}_3$ und $Q_{23} : \mathbf{q}_0 + \mathbf{f}_2 + \mathbf{f}_3$ sind Punkte von P.
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sind i.a. nicht orthogonal. Aber es gilt:
Die Tangentialebene in Q_0 ist parallel zur Ebene, die von $\mathbf{f}_1, \mathbf{f}_2$ aufgespannt wird.
Falls $\mathbf{f}_1, \mathbf{f}_2$ zu $\mathbf{f}_3$ senkrecht stehen, nennt man den Punkt Q_0 den **Scheitel** von P.

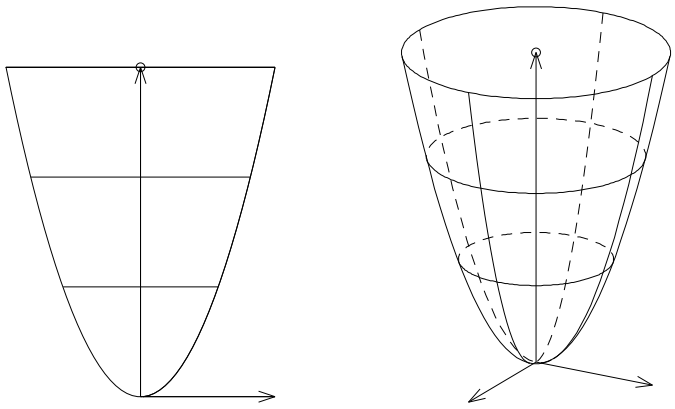


Abbildung 7.7: Paraboloid

Für die **Gleichung** $f(x,y,z) = 0$ des Paraboloids $x = \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 - \zeta = 0$, in $x-y-z-$Koordinaten erhält man (vgl. Ellipsoid)

$$f(x,y,z) := (\det(\mathbf{x},\mathbf{f}_2,\mathbf{f}_3))^2 + (\det(\mathbf{f}_1,\mathbf{x},\mathbf{f}_3))^2 - (\det(\mathbf{f}_1,\mathbf{f}_2,\mathbf{x}))(\det(\mathbf{f}_1,\mathbf{f}_2,\mathbf{f}_3)) = 0,$$

wobei $\det(\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3)$ die $3\times 3-$Determinate mit den Spaltenvektoren $\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3 \in \mathbb{R}^3$ ist.

7.2.1 Ebene Schnitte des Einheits-Paraboloids

Auch hier bestimmen wir zunächst die Schnitte des Einheits-Paraboloids

$$P_1 := \{(x,y,z) \in \mathbb{R}^3 \mid z = x^2 + y^2\}$$

mit der Ebene $\varepsilon : ax + cz = d$.
Analoge Überlegungen wie beim Kegel liefern das folgende Ergebnis:

Fall I: $c = 0$
Die Schnittkurve ist eine **Parabel** mit (vgl. Kap. 5.3)

$$\begin{array}{rcl} P_0 & = & (\frac{d}{a}, 0, \frac{d^2}{a^2}) \qquad \text{(Scheitel)} \\ P_1 & = & (\frac{d}{a}, 1, \frac{d^2}{a^2}) \ , \ P_2 = (\frac{d}{a}, 0, 1 + \frac{d^2}{a^2}). \end{array}$$

Fall II: $c \neq 0$

IIa) Falls $4cd + a^2 < 0$ ist, ist der Schnitt **leer**.

IIb) Falls $4cd + a^2 = 0$ ist, besteht der Schnitt aus dem **Punkt**.

$$P_0 = (-\frac{a}{2c}, 0, \frac{2cd + a^2}{2c^2})$$

IIc) Falls $4cd + a^2 > 0$ ist, ist die Schnittkurve eine **Ellipse**.

$$P_0 = (-\frac{a}{2c}, 0, \frac{2cd + a^2}{2c^2}) \text{ ist der Mittelpunkt.}$$

$$P_1 = (x_1, 0, \frac{d - ax_1}{c}), \ P_1' = (x_1', 0, \frac{d - ax_1'}{c})$$

mit $x_1 = -a + \frac{\sqrt{4cd+a^2}}{2c},\ x_1' = -a - \frac{\sqrt{\cdots}}{2c},$

$$P_2 = (-\frac{a}{2c}, \frac{\sqrt{\cdots}}{2c}, \frac{2cd+a^2}{2c^2}), \quad P_2' = (\ldots, -\frac{\sqrt{\cdots}}{\ldots}, \ \ldots)$$

sind ihre Scheitel.

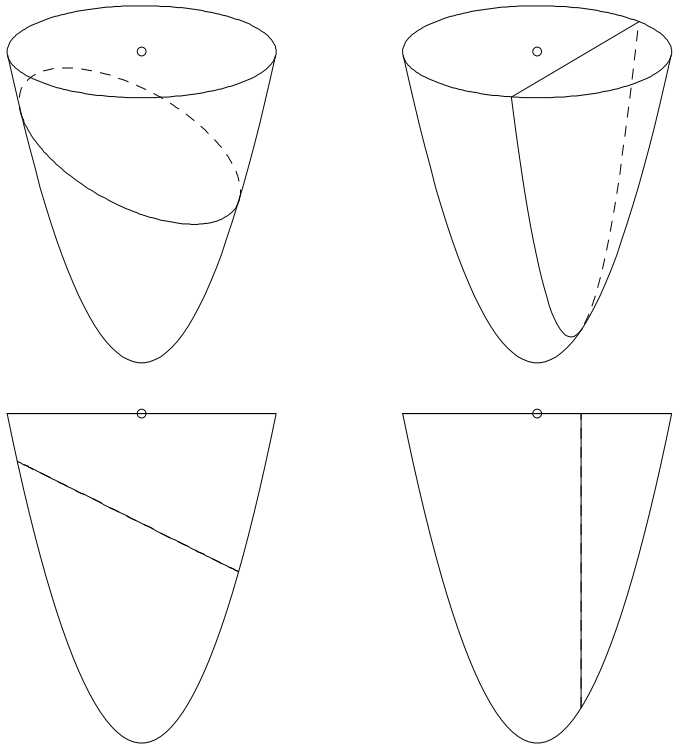


Abbildung 7.8: ebene Schnitte eines Paraboloids

Das Unterprogramm `is_unitparaboloid_plane0` berechnet für den jeweiligen Fall die Punkte P_0, P_1, P_2 und liefert eine Zahl `ind`, mit der man die einzelnen Fälle unterscheiden kann. Und zwar ist der Schnitt

eine	Parabel,	falls	`ind` =	0,
eine	Ellipse,	falls	`ind` =	1,
ein	Punkt,	falls	`ind` =	100,
	leer,	falls	`ind` =	-100.

```
procedure is_unitparaboloid_plane0(a,c,d: real; var pc0,pc1,pc2: vt3d; var ind: integer);
{Berechnet den Schnitt des Paraboloids z = x*x + y*y mit der Ebene a*x + c*z = d.
 ind=-100 : leer,  ind=100 : Punkt,  ind=0 : Parabel,  ind=1 : Ellipse.}
 var da,da2,dis,rad,pc1x : real;
 begin
   if c=0 then                                              {Fall I}
      begin
      da:= d/a;           da2:= da*da;
      put3d(da,0,  da2, pc0);   put3d(da,1,  da2, pc1);   put3d(da,0,1+da2, pc2);
      ind:= 0;
```

```
     end
    else   c<>0                                              {Fall II}
     begin
     dis:= 4*c*d + a*a;
     ind:= -100;                                             {Fall IIa}
     if dis>=0 then
        begin
        put3d(-a/(2*c), 0,(2*c*d+a*a)/(2*c*c), pc0);
        if dis=0 then     ind:= 100                         {Fall IIb}
          else   dis>0
           begin
           ind:= 1;                                          {Fall IIc}
           rad:= sqrt(dis)/(2*c);  pc1x:= pc0.x + rad;
           put3d( pc1x,   0, (d-a*pc1x)/c, pc1);
           put3d(pc0.x, rad,        pc0.z, pc2);
           end;
        end;  {dis>=0}
  end;  {c<>0}
end;     {is_paraboloid_plane0}
{**************}
```

Mit Hilfe von `is_unitparaboloid_plane0` werden wir jetzt die Schnitte des Einheitsparaboloids mit einer beliebigen Ebene $ax + by + cz = d$ bestimmen. Hierzu wird zunächst eine Rotation um die z-Achse durchgeführt, sodaß die Ebene die in `is_unitparaboloid_plane0` geforderte Form erhält.

```
procedure is_unitparaboloid_plane(a,b,c,d : real; var pc0,pc1,pc2:vt3d;
                                                  var ind : integer);
 var aa,cw,sw : real;      p0,p1,p2 : vt3d;
 begin
   aa:= sqrt(a*a+b*b); cw:= a/aa;  sw:= b/aa;
   is_paraboloid_plane0(aa,c,d, p0,p1,p2,ind);
   rotorz(cw,sw,p0, pc0);   rotorz(cw,sw,p1, pc1);   rotorz(cw,sw,p2,pc2);
 end;  { is_unitparaboloid_plane }
```

7.2.2 Ebene Schnitte eines Paraboloids

Gegeben: a) Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$,
b) Paraboloid $P : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 = \zeta$.
Gesucht: Schnitt $\varepsilon \cap P$.

Durch Einsetzen der Parameterdarstellung des Paraboloids in die Gleichung der Ebene erhält man das Gleichungssystem:

$$\begin{aligned} \gamma_1\xi + \gamma_2\eta + \gamma_3\zeta &= \gamma_0, \text{ mit } \gamma_i = \mathbf{n} \cdot \mathbf{f}_i,\ i = 1,2,3,\ \gamma_0 = d - \mathbf{n} \cdot \mathbf{q}_0, \\ \xi^2 + \eta^2 &= \zeta \end{aligned}$$

Löst man dieses System mit Hilfe von **is_unitparaboloid_plane**, so erhält man die Daten der Schnittkurve (s.o.) zunächst in $\xi - \eta - \zeta-$Koordinaten. Durch Einsetzen in $\mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta$ ergeben sich die zugehörigen $x - y - z-$Koordinaten.

```
procedure is_paraboloid_plane(q0,q1,q2,q3,nv: vt3d; d: real;
                                  var pc0,pc1,pc2: vt3d; var ind: integer);
 {Berechnet den Schnitt des Paraboloids mit Mittelpunkt q0 und den
 konjugierten Punkten q1,q2,q3  mit der Ebene  nv*x=d}.
```

7.2.3 Parallelprojektion eines Paraboloids

Die Normale in einem Punkt Q des Paraboloids

$$P = \{\mathbf{q}_0 + \mathbf{f}_1 s + \mathbf{f}_2 t + \mathbf{f}_3(s^2 + t^2) \mid \ldots\}$$

läßt sich durch $\mathbf{n}(s,t) = \mathbf{q}_s \times \mathbf{q}_t = \mathbf{f}_1 \times \mathbf{f}_2 - 2\mathbf{f}_2 \times \mathbf{f}_3 s - 2\mathbf{f}_3 \times \mathbf{f}_1 t$ beschreiben. $Q : \mathbf{q}(s,t)$ ist ein Umrißpunkt, wenn $\mathbf{n}(s,t)$ auf der (negativen) Projektionsrichtung $\mathbf{n}_0$ senkrecht steht:

$$\mathbf{n}_0 \cdot \mathbf{n}(s,t) = \mathbf{n}_0 \cdot (\mathbf{f}_1 \times \mathbf{f}_2) - 2\mathbf{n}_0 \cdot (\mathbf{f}_2 \times \mathbf{f}_3)s - 2\mathbf{n}_0 \cdot (\mathbf{f}_3 \times \mathbf{f}_1)t = 0.$$

In $\xi - \eta - \zeta-$Koordinaten führt diese Gleichung auf das System

$$\begin{array}{rcl} 2\gamma_1\xi + 2\gamma_2\eta & = & \gamma_0, \quad \text{mit } \gamma_1 = \mathbf{n}_0 \cdot (\mathbf{f}_2 \times \mathbf{f}_3),\ \gamma_2 = \mathbf{n}_0 \cdot (\mathbf{f}_3 \times \mathbf{f}_1),\ \gamma_0 = \mathbf{n}_0 \cdot (\mathbf{f}_1 \times \mathbf{f}_2), \\ \xi^2 + \eta^2 & = & \zeta \end{array}$$

Falls $\gamma_{=}\gamma_2 = 0$ ist, ist der Umriß die "Deckelelleipse". Andernfalls ist der Umriß ein Parabelbogen und man erhält mit `is_unitparaboloid_plane` die $\xi - \eta - \zeta-$Koordinaten der Punkte, die die Umrißparabel beschreiben. Einsetzen in $\mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta$ liefert die zugehörigen $x - y - z-$Koordinaten.

```
procedure paraboloid_pp_cont(q0,q1,q2,q3: vt3d; var qc0,qc1,qc2 : vt3d);
{Berechnet den Umriss des durch q0, q1, q2, q3 bestimmten Paraboloids.}
```

Das folgende Unterprogramm stellt das Rotationsparaboloid mit der Gleichung $z = a(x^2 + y^2)$ und der Höhe h dar.

```
procedure pp_paraboloid_rev(a,h: real; style: integer);
{Projektion des Paraboloids  z = a*(x*x + y*y).}
```

Aufgabe 7.3 *a) Schreibe ein Programm, das die obigen Zeichnungen zu "Ebene Schnitt eines Paraboloids" erstellt.*
b) Stelle ein Rotationsparaboloid mit Kreisen und Parabeln dar.

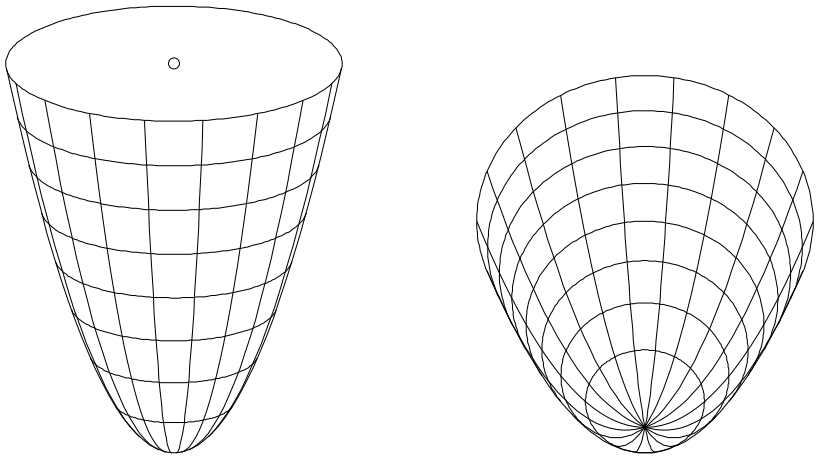


Abbildung 7.9: zu Aufgabe 7.3

7.2.4 Zentralprojektion eines Paraboloids

Für $\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 = \zeta$
lautet die Umrißbedingung: $\quad 2\gamma_1\xi + 2\gamma_2\eta - \gamma_0\zeta = \gamma_3, \qquad \xi^2 + \eta^2 = \zeta,$
mit $\quad \gamma_0 = \mathbf{f}_1 \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_1 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_2 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_3 \times \mathbf{f}_1), \quad \gamma_3 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_1 \times \mathbf{f}_2).$
Da γ_0 immer ungleich 0 ist, ist der Umriß eines Paraboloids bei Zentralprojektion immer eine Ellipse.

Kapitel 8

ZYLINDER UND KEGEL

8.1 Zylinder

Ein affines Bild des Einheitszylinders $Z_1 := \{(x, y, z) \mid x^2 + y^2 = 1\}$ heißt **(elliptischer) Zylinder**. Beschreibt man den Einheitszylinder Z_1 durch die Parameterdarstellung:

$$Z_1 = \{(\cos\varphi, \sin\varphi, \alpha) \mid 0 \leq \varphi \leq 2\pi, \alpha \in \mathbb{R}\},$$

so gibt es zu jedem Zylinder Z einen Vektor $\mathbf{q}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$, so daß

$$Z = \{\mathbf{q}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi + \mathbf{f}_3\alpha \mid \ldots\}$$

ist. Umgekehrt ist für einen Vektor $\mathbf{q}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ die Punktmenge $Z = \{\ldots \mid \ldots\}$ ein elliptischer Zylinder.

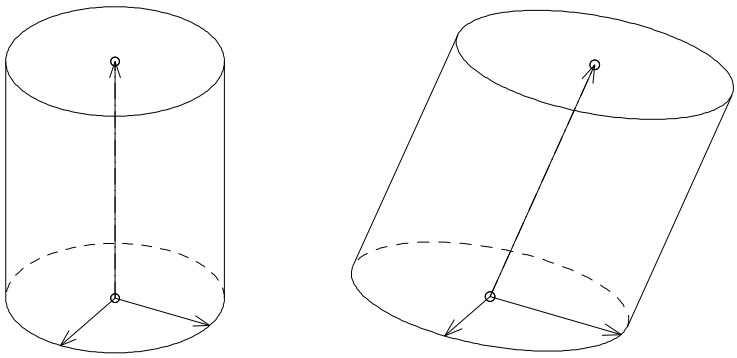


Abbildung 8.1: senkrechter bzw- schiefer elliptischer Zylinder

Die Koordinaten ξ, η, ζ eines Punktes von Z bezüglich des Koordinatensystems mit Nullpunkt $Q_0 : \mathbf{q}_0$ und der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ genügen also der Gleichung

$$\xi^2 + \eta^2 = \cos\varphi^2 + \sin\varphi^2 = 1,$$

d. h. Z ist bezüglich des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ der Einheitszylinder.
$E_0 := \{\mathbf{q}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi \mid \ldots\}$ heißt die *Basisellipse von Z*.
$\mathbf{f}_3$ heißt *Achsenvektor von Z*.

$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sind i.a. nicht orthogonal. Aber es gilt:
Ist die Basisellipse ein Kreis und $\mathbf{f}_3$ senkrecht zum Basiskreis, so heißt Z *senkrechter Kreiszylinder.*

Die **Gleichung** $f(x, y, z) = 0$ des Zylinders $Z : \mathbf{x} = \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi + \mathbf{f}_3\alpha$ in $x-y-z-$Koordinaten erhält man aus dem Gleichungssystem

$$\begin{aligned} x &= f_{1x}\cos\varphi + f_{2x}\sin\varphi + f_{3x}\alpha \qquad (\text{mit} \quad \mathbf{x} := (x, y, z),\ \mathbf{f}_i := (f_{ix}, f_{iy}, f_{iz})) \\ y &= f_{1y}\cos\varphi + f_{2y}\sin\varphi + f_{3y}\alpha \\ z &= f_{1z}\cos\varphi + f_{2z}\sin\varphi + f_{3z}\alpha. \end{aligned}$$

Löst man nach $\cos\varphi$ und $\sin\varphi$ auf, so erhält man aus $(\cos\varphi)^2 + (\sin\varphi)^2 = 1$ die Gleichung

$$f(x, y, z) := (\det(\mathbf{x}, \mathbf{f}_2, \mathbf{f}_3))^2 + (\det(\mathbf{f}_1, \mathbf{x}, \mathbf{f}_3))^2 - (\det(\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3))^2 = 0,$$

wobei $\det(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ die $3 \times 3-$Determinate mit den Spaltenvektoren $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$ ist.

8.1.1 Ebene Schnitte eines Zylinders

Gegeben: a) Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$,
b) Zylinder $Z : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \qquad \xi^2 + \eta^2 = 1.$
Gesucht: Schnitt $\varepsilon \cap Z$.

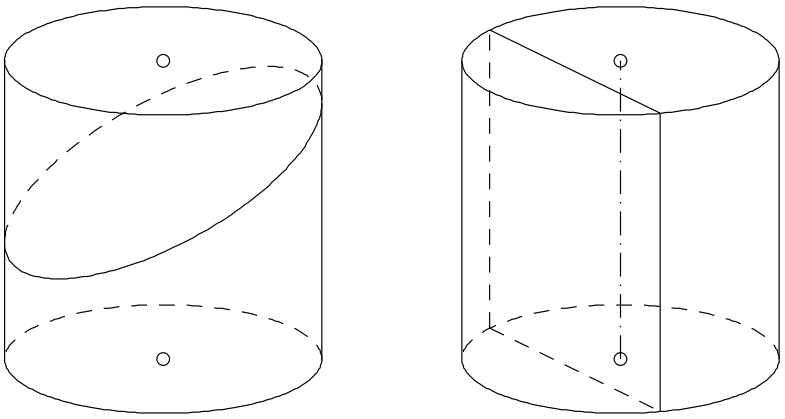


Abbildung 8.2: ebene Schnitte eines Zylinders

Durch Einsetzen der Parameterdarstellung des Zylinders in die Gleichung der Ebene erhält man das Gleichungssystem:

$$\gamma_1\xi + \gamma_2\eta + \gamma_3\zeta = \gamma_0, \quad \text{mit} \quad \gamma_i = \mathbf{n} \cdot \mathbf{f}_i, \quad i = 1, 2, 3, \qquad \gamma_0 = d - \mathbf{n} \cdot \mathbf{q}_0, \quad \xi^2 + \eta^2 = 1$$

Falls $\gamma_3 = 0$ ist, ist die Ebene parallel zur Zylinderachse und der Schnitt ist leer oder ein (eventuell zusammenfallendes) Geradenpaar, das parallel zur Zylinderachse ist. Im zweiten Fall genügt es, die Schnittpunkte dieser Geraden mit der Basisellipse anzugeben. Die $\xi-\eta-$Koordinaten dieser Punkte P_1, P_2 ergeben sich mit `is_unitcircle_line` aus $\gamma_1\xi + \gamma_2\eta = \gamma_0,\ \xi^2 + \eta^2 = 1$.
Falls $\gamma_3 \neq 0$ ist, ist der Schnitt eine Ellipse mit Mittelpunkt auf der Achse.
Die $\xi - \eta - \zeta-$Koordinaten von Mittelpunkt und zwei konjugierten Punkten sind:

$$\begin{aligned} &\xi_0 = 0, \quad &&\eta_0 = 0, \quad &&\zeta_0 = \gamma_0/\gamma_3, \\ &\xi_1 = 1, \quad &&\eta_1 = 0, \quad &&\zeta_1 = (\gamma_0 - \gamma_1)/\gamma_3, \\ &\xi_2 = 0, \quad &&\eta_2 = 1, \quad &&\zeta_2 = (\gamma_0 - \gamma_2)/\gamma_3. \end{aligned}$$

(Man beachte: Die Schnittellipse ist ein affines Bild der Basisellipse.)
Aus den $\xi-\eta-\zeta-$Koordinaten ergeben sich mit $\mathbf{x}=\mathbf{q}_0+\mathbf{f}_1\xi+\mathbf{f}_2\eta+\mathbf{f}_3\zeta$ die $x-y-z-$Koordinaten der Punkte P_0, P_1, P_2. Das Unterprogramm hierzu ist

```
procedure is_cylinder_plane(q0,q1,q2,q3,nv: vt3d; d: real; var pc0,pc1,pc2: vt3d;
                                                           var ind: integer);
 {Schnitt des durch q0,q1,q2,q3 bestimmten Zylinders mit der Ebene nv*x=d.
 q0: Mittelpunkt, q1,q2 konj. Punkte der Basisellipse, q3: Achsenpunkt.
 ind=1: Schnitt ist Ellipse, ind=11: Schnitt ist Geradenpaar.}
```

8.1.2 Parallelprojektion eines Zylinders

In der Praxis ist der Achsenparameter α nicht unbeschränkt, sondern bewegt sich in bestimmten Grenzen. Wir wollen hier annehmen, daß α über ein Intervall $[0,h]$ läuft. D.h. der Zylinder wird durch die Basisellipse E_0 und eine dazu parallele Ellipse begrenzt. Der Umriß eines solchen elliptischen Zylinders besteht i.a. aus Ellipsenbögen und Strecken. Um die Umrißstrecken zeichnen zu können, berechnen wir zunächst die Umrißpunkte auf der Basisellipse.
Es sei

$$Z := \{\mathbf{q}_0+\mathbf{f}_1\cos\varphi+\mathbf{f}_2\sin\varphi+\mathbf{f}_3\alpha \mid 0\le\varphi\le 2\pi,\ 0\le\alpha\le h\}$$

ein beschränkter elliptischer Zylinder. Eine Normale im Punkt $\mathbf{q}(\varphi,\alpha)=\mathbf{p}_0+\mathbf{f}_1\cos\varphi+\mathbf{f}_2\sin\varphi+\mathbf{f}_3\alpha$ ist

$$\mathbf{n}(\varphi)=\mathbf{q}_\varphi(\varphi,\alpha)\times\mathbf{q}_\alpha(\varphi,\alpha)=(-\mathbf{f}_1\sin\varphi+\mathbf{f}_2\cos\varphi)\times\mathbf{f}_3.$$

Ist $\mathbf{n}_0$ die negative Projektionsrichtung, so ist $\mathbf{q}(\varphi,\alpha)$ genau dann Umrißpunkt, wenn

$$\mathbf{n}_0\cdot\mathbf{n}(\varphi)=\mathbf{n}_0\cdot(\mathbf{f}_2\times\mathbf{f}_3)\cos\varphi+\mathbf{n}_0\cdot(\mathbf{f}_3\times\mathbf{f}_1)\sin\varphi=0$$

ist. D.h. $\xi:=\cos\varphi,\ \eta:=\sin\varphi$ (Koordinaten bzgl. $\mathbf{f}_1,\mathbf{f}_2$) ergeben sich aus dem Gleichungssystem

$$\begin{aligned}\gamma_1\xi+\gamma_2\eta &= 0, \quad \text{wobei} \quad \gamma_1:=\mathbf{n}_0\cdot(\mathbf{f}_2\times\mathbf{f}_3), \quad \gamma_2:=\mathbf{n}_0\cdot(\mathbf{f}_3\times\mathbf{f}_1), \quad \text{ist}\\ \xi^2+\eta^2 &= 1\end{aligned}$$

Falls $\gamma_1=\gamma_2=0$ ist, ist der Umriß die Projektion der Basisellipse.
Falls $\gamma_1\neq 0$ oder $\gamma_2\neq 0$ ist, löst man das Gleichungssystem mit dem Unterprogramm `is_unitcircle_line`. Sind $(\xi_1,\eta_1),\ (\xi_2,\eta_2)$ Lösungen dieses Gleichungssystems, so sind

$$C_1:\ \mathbf{c}_1=\mathbf{p}_0+\mathbf{f}_1\xi_1+\mathbf{f}_2\eta_1 \quad \text{und} \quad C_2:\ \mathbf{c}_2=\mathbf{p}_0+\mathbf{f}_1\xi_2+\mathbf{f}_2\eta_2$$

Umrißpunkte auf der Basisellipse von Z.

In dem folgenden Unterprogramm `cylinder_pp_contpts` beschreiben die Vektoren $\mathbf{q}_0,\mathbf{q}_1,\mathbf{q}_2$ die Basisellipse und $\mathbf{q}_3$ ist der Mittelpunkt der "Deckelellipse" auf der Achse eines elliptischen Zylinders. Falls die Zylinderachse projizierend ist, ist `contlines` = false. Im anderen Fall sind $C_1:\mathbf{c}_1,\ C_2:\mathbf{c}_2$ die Umrißpunkte auf der Basisellipse, $C_{11}:\mathbf{c}_{11}=\mathbf{c}_1+\mathbf{f}_3,\ C_{22}:\mathbf{c}_{22}=\mathbf{c}_2+\mathbf{f}_3$ die Umrißpunkte auf der Deckelellipse und es ist `contlines`=true.

```
procedure cylinder_pp_contpts(q0,q1,q2,q3: vt3d; var qc1,qc2,qc11,qc22: vt3d;
                                                  var contlines: boolean);
 {Berechnet den Umriss des durch q0, q1, q2, q3 bestimmten Zylinders.}
```

Mit Hilfe von `cylinder_pp_contpts` schreiben wir eine Prozedur `pp_cylinder`, die einen elliptischen Zylinder mit (durch $\mathbf{q}_0,\mathbf{q}_1,\mathbf{q}_2$) vorgegebener Basisellipse und vorgegebenem Mittelpunkt ($\mathbf{q}_3$) der zur Basisellipse parallelen "Deckel"- Ellipse.

a) mit **nur** sichtbaren Ellipsenbögen, falls **style** $= 0$ ist,

b) mit gestrichelten unsichtbaren Ellipsenbögen, falls **style** $= 10$ ist, projiziert.

```
procedure pp_cylinder(q0,q1,q2,q3: vt3d; style: integer);
 { Projektion des durch q0, q1, q2, q3  bestimmten Zylinders. }
 var  f1,f2,f3,q00,q11,q22,nv,pc1,pc2,pc11,pc22,nvc : vt3d;
     error,contlines : boolean;      side : integer;      test,dc : real;
 begin
  diff3d(q1,q0, f1);    diff3d(q2,q0, f2);   diff3d(q3,q0, f3);
  cylinder_pp_contpts(q0,q1,q2,q3, pc1,pc2,pc11,pc22,contlines);
  if not contlines  then  pp_ellipse(q0,q1,q2,0)
     else
     begin
     sum3d(q0,f3, q00);  sum3d(q1,f3, q11);  sum3d(q2,f3, q22);
{ Koeffizienten der Umrissebene }
     plane_equ(pc1,pc11,pc2, nvc,dc,error);
     if scalarp3d(nvc,n0vt)<0 then side:= -1 else side:= 1;
     vectorp(f1,f2, nv);   {Normale auf Bodenebene}
     test:= scalarp3d(nv,f3) * scalarp3d(n0vt,nv);
{ Bodenkreis : }
     if test<0  then  pp_ellipse(q0,q1,q2,0)
                else  pp_ellipse_before_plane(q0,q1,q2,nvc,dc,side,style);
{ Deckelkreis : }
     if test>=0 then  pp_ellipse(q00,q11,q22,0)
                else  pp_ellipse_before_plane(q00,q11,q22,nvc,dc,side,style);
     pp_point(q0,0);       pp_point(q00,0);
{ Umrissgeraden : }
     pp_line(pc1,pc11,0);  pp_line(pc2,pc22,0);
     end;  { if }
end;  { pp_cylinder }
{*************}
```

Da wir oft **senkrechte** elliptische Zylinder einer vorgegebenen Länge projizieren müssen, schreiben wir hierfür eine Prozedur `pp_cylinder_orth`. Darin ist `clength` die Länge des Zylinders.

```
procedure pp_cylinder_orth(q0,q1,q2: vt3d; clength: real; style: integer);
 {Projiziert einen senkrechten elliptischen Zylinder der Laenge clengeth.}
```

Beispiel 8.1 *: Schräg abgeschnittener Zylinder.*

```
{*******************************}
{***  ebener Zylinderschnitt  ***}
{*******************************}
program zylinds;
uses geograph;
 var r,zo,zu,rb,a,c,d,dc : real;
     inz,side,ind : integer;  contlines,error : boolean;
     pc0,pc1,pc2,pc3,pc4,nv,nvc,q0,q1,q2,q3,cp1,cp2,cp11,cp22 : vt3d;
{*****************}
begin {Hauptprogramm}
  graph_on(0);
  writeln('*** Ebener Zylinderschnitt ***');
  writeln(' Zylinderradius r ?');                              readln(r);
```

```
  writeln(' Oberster  Punkt des Schnittes ist (-r,0,zo).');
  writeln(' Unterster Punkt des Schnittes ist (r,0,zu).');
  writeln('zo,zu?');                                          readln(zo,zu);
  q0:= null3d;    put3d(r,0,0, q1);    put3d(0,r,0, q2);    put3d(0,0,zo, q3);
  a:= -(zu-zo)/(2*r);    d:= (zu+zo)/2;    put3d(a,0,1, nv);
  repeat
    init_parallel_projection;
    draw_area(180,180,90,50,1);
    pp_axes(20);
{Umriss:}
    cylinder_pp_contpts(q0,q1,q2,q3,cp1,cp2,cp11,cp22,contlines);
    side:= -1;
    pp_line_before_plane(cp1,cp11,nv,d,side,0);
    pp_line_before_plane(cp2,cp22,nv,d,side,0);
    plane_equ(cp1,cp11,cp2, nvc,dc,error);
{Basiskreis:}
    if scalarp3d(q3,n0vt) < 0
       then pp_ellipse(q0,q1,q2,0)
       else pp_ellipse_before_plane(q0,q1,q2,nvc,dc,1,10);
    point2d(null2d,0);
{Deckelellipse:}
    is_cylinder_plane(q0,q1,q2,q3, nv,d, pc0,pc1,pc2,ind);
    if scalarp3d(nv,n0vt) > 0
       then pp_ellipse(pc0,pc1,pc2,0)
       else pp_ellipse_before_plane(pc0,pc1,pc2,nvc,dc,1,10);
    pp_point(pc0,0);
    draw_end;
    writeln(' Noch eine Zeichnung ? (nein: 0) ');    readln(inz);
  until inz=0;
  graph_off;
 end.
```

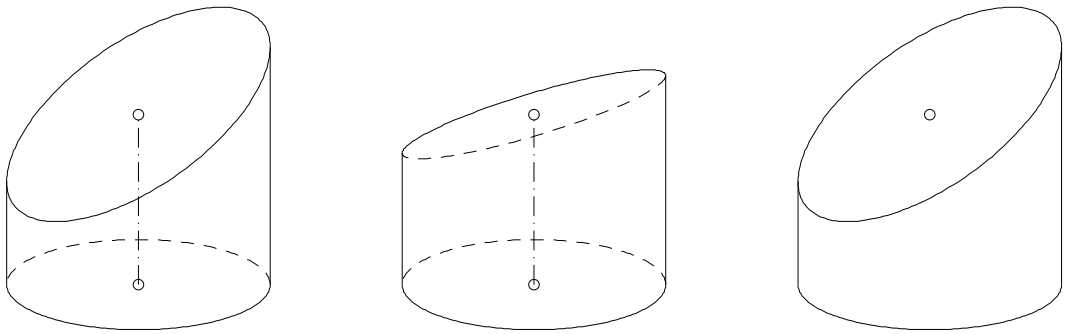


Abbildung 8.3: schräg abgeschnittener Zylider

Aufgabe 8.1 *a) Erweitere das Programm aus Beispiel 8.1 so, daß die schräge Ebene auch in die Bodenellipse schneiden darf.*
b) Stelle den Durchschnitt zweier sich senkrecht schneidender Vollzylinder dar.

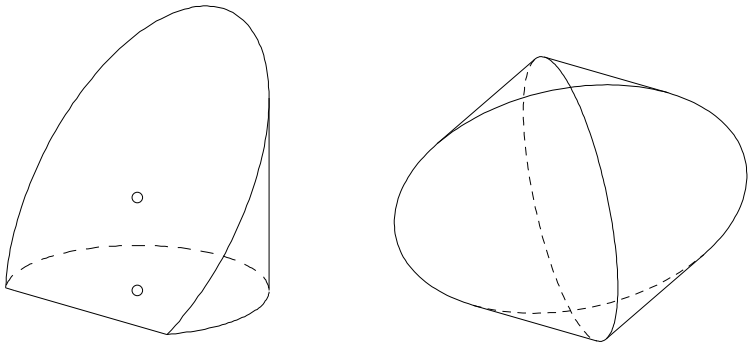


Abbildung 8.4: zu Aufgabe 8.1

8.1.3 Zentralprojektion eines Zylinders

Für den Zylinder $\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\alpha, \qquad \xi^2 + \eta^2 = 1.$
lautet die Umrißbedingung: $\gamma_1\xi + \gamma_2\eta = \gamma_0, \quad \xi^2 + \eta^2 = 1,$
mit $\gamma_0 = \mathbf{f}_1 \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_1 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_2 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_3 \times \mathbf{f}_1).$

Beispiel:

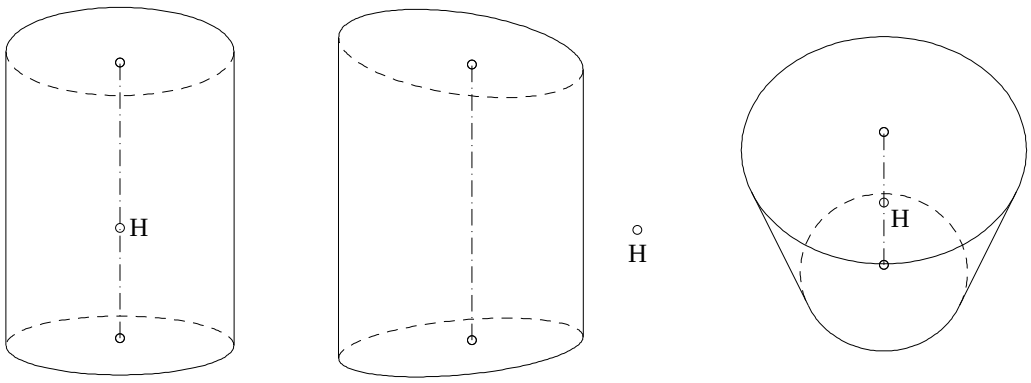


Abbildung 8.5: Zentralprojektion eines Zylinders

8.2 Kegel

Ein affines Bild des Einheitskegels $K_1 := \{(x, y, z) \mid x^2 + y^2 = z^2\}$ heißt **(elliptischer) Kegel**. Beschreibt man den Einheitskegel K_1 durch die Parameterdarstellung

$$K_1 = \{(\alpha\cos\varphi, \alpha\sin\varphi, -\alpha) \mid 0 \le \varphi \le 2\pi, \alpha \in \mathbb{R}\}, \quad \text{sogilt :}$$

Zu jedem Kegel K gibt es einen Vektor $\mathbf{s}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$, so daß

$$K = \{\mathbf{s}_0 + \mathbf{f}_1\alpha\cos\varphi + \mathbf{f}_2\alpha\sin\varphi - \mathbf{f}_3\alpha \mid \ldots\} \quad \text{ist.}$$

Umgekehrt ist für einen Vektor $\mathbf{s}_0$, und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ die Punktmenge $K = \{\ldots \mid \ldots\}$ ein elliptischer Kegel.
Die Koordinaten ξ, η, ζ eines Punktes von K bezüglich des Koordinatensystems mit Nullpunkt $S_0 : \mathbf{s}_0$ und der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ genügen also der Gleichung

$$\xi^2 + \eta^2 = (\alpha\cos\varphi)^2 + (\alpha\sin\varphi)^2 = \alpha^2 = \zeta^2,$$

d.h. K ist bezüglich des Koordinatensystems $(S_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ der Einheitskegel.
$E_0 := \{\mathbf{s}_0 - \mathbf{f}_3 + \mathbf{f}_1\cos\varphi + \mathbf{f}_2\sin\varphi \mid \ldots\}$ heißt die Basisellipse von K.
$Q_0 : \mathbf{q}_0 := \mathbf{s}_0 - \mathbf{f}_3$ ist der Mittelpunkt von E_0.
Es ist zweckmäßig den Kegel K mit Hilfe seiner Basisellipse und der Kegelspitze zu beschreiben:

$$K = \{\mathbf{q}_0 + \mathbf{f}_1\alpha\cos\varphi + \mathbf{f}_2\alpha\sin\varphi - \mathbf{f}_3(1-\alpha) \mid \ldots\}$$

Bezüglich des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ genügen die $\xi - \eta - \zeta-$Koordinaten der Gleichung $\xi^2 + \eta^2 = (\zeta - 1)^2$.
$\mathbf{f}_3$ heißt Achsenvektor von K. $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sind i.a. nicht orthogonal.
Ist die Basisellipse ein Kreis und $\mathbf{f}_3$ senkrecht zum Basiskreis, so heißt K **senkrechter Kreiskegel**.

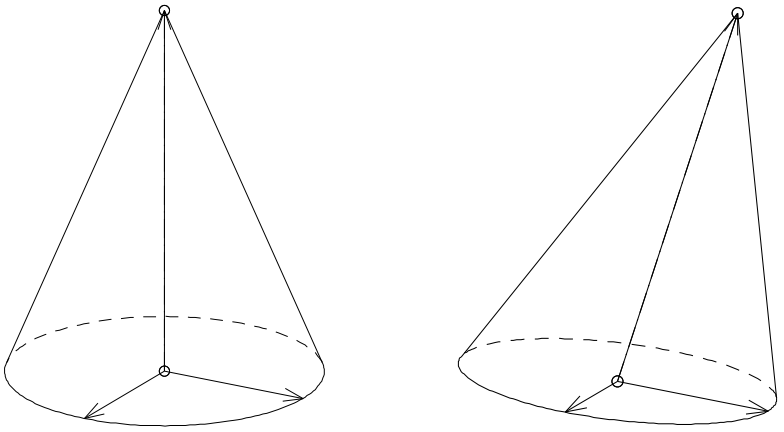


Abbildung 8.6: senkrechter Kreiskegel bzw. schiefer elliptischer Kegel

Die Gleichung $f(x, y, z) = 0$ des Kegels $K : \mathbf{x} = \alpha(\mathbf{f}_1\cos\varphi + \mathbf{f}_2\sin\varphi - \mathbf{f}_3)$ in $x - y - z-$Koordinaten erhält man aus dem Gleichungssystem:

$$\begin{array}{rcll} x & = & f_{1x}\alpha\cos\varphi + f_{2x}\alpha\sin\varphi - f_{3x}\alpha & \quad (\text{mit } \mathbf{x} := (x, y, z),\ \mathbf{f}_i := (f_{ix}, f_{iy}, f_{iz})) \\ y & = & f_{1y}\alpha\cos\varphi + f_{2y}\alpha\sin\varphi - f_{3y}\alpha & \\ z & = & f_{1z}\alpha\cos\varphi + f_{2z}\alpha\sin\varphi - f_{3z}\alpha & \end{array}$$

nach $\alpha\cos\varphi, \alpha\sin\varphi$ und α auf, so erhält man aus $(\alpha\cos\varphi)^2 + (\alpha\sin\varphi)^2 - \alpha^2 = 0$ die Gleichung

$$f(x, y, z) := (\det(\mathbf{x}, \mathbf{f}_2, \mathbf{f}_3))^2 + (\det(\mathbf{f}_1, \mathbf{x}, \mathbf{f}_3))^2 - (\det(\mathbf{f}_1, \mathbf{f}_2, \mathbf{x}))^2 = 0,$$

wobei $\det(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ die 3×3-Determinate mit den Spaltenvektoren $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$ ist.

8.2.1 Ebene Schnitte des Einheitskegels

Die Schnittkurve einer Ebene mit einem Kegel kann, wie wir sehen werden, eine Ellipse, eine Hyperbel, eine Parabel, ein Geradenpaar, eine Gerade oder ein Punkt sein.
Gegeben: Ebene $\varepsilon : ax + cz = d$,
Kegel $K_1 : x^2 + y^2 = z^2$.
Gesucht: Schnitt $\varepsilon \cap K_1$.

Fall I: $c = 0$
In diesem Fall ist $a \neq 0$ und $x = d/a$. Eliminiert man x aus der Kegelgleichung, so erhält man $z^2 - y^2 = d^2/a^2$.

Fall Ia: $d = 0$
In diesem Fall besteht der Schnitt aus dem **Geradenpaar**

$$(0,0,0) + t(0,1,\pm 1),\ t \in \mathbb{R}.$$

Fall Ib: $d \neq 0$
Die obige Gleichung beschreibt jetzt eine Hyperbel. Also ist auch die Schnittkurve eine **Hyperbel** und

$P_0 = (d/a, 0, 0)$	ist der Mittelpunkt,
$P_1 = (d/a, 0, \pm d/a)$	sind die Scheitel und
$P_2 = (d/a, \pm d/a, 0)$	sind die Nebenscheitel der Schnitthyperbel.

Fall II: $c \neq 0$
Eliminiert man z aus der Kegelgleichung mit Hilfe der Ebenengleichung, so erhält man das Gleichungssystem

$$(1)\quad (c^2 - a^2)x^2 + 2adx + c^2y^2 = d^2, \qquad (2)\quad ax + cz = d.$$

Fall IIa: $d = 0$
Gleichung (1) hat jetzt die Gestalt $(c^2 - a^2)x^2 + c^2y^2 = 0$.
Für $c^2 > a^2$ ist der Schnitt der **Punkt** $P_0 = (0,0,0)$.
Für $c^2 = a^2$ ist der Schnitt die **Gerade** $(0,0,0) + t(c,0,-a),\ t \in \mathbb{R}$.
Für $c^2 < a^2$ ist der Schnitt das **Geradenpaar**

$$(0,0,0) + t(c/\pm\sqrt{a^2 - c^2}, 1, -a/\pm\sqrt{a^2 - c^2}),\ t \in \mathbb{R}.$$

(Die letzten beiden Fälle und der Fall Ia können durch

$$(0,0,0) + t(c, \pm\sqrt{a^2 - c^2}, -a),\ t \in \mathbb{R}$$

gemeinsam beschrieben werden.)

Fall IIb: $d \neq 0$
Für $c^2 = a^2$ geht (1) in die **Parabel**gleichung

$$x = -\frac{c^2}{2ad}y^2 + \frac{d}{2a}$$

über.

$P_0 = (d/2a, 0, d/2c)$	ist der Scheitel der Schnittparabel.
$P_1 = (d/2a, d/c, d/2c)$	ist ein Punkt der Scheiteltangente.
$P_2 = (0, 0, d/c)$	der Punkt auf der Achse, für den
	$P_{12} : \mathbf{p}_0 + \mathbf{p}_1 + \mathbf{p}_2$ ein Parabelpunkt ist. (s. Abschn. 5.3)

Für $c^2 \neq a^2$ formen wir (1) um in

$$\frac{(c^2-a^2)^2}{d^2c^2}(x+\frac{ad}{c^2-a^2})^2+\frac{c^2-a^2}{d^2}y^2=1$$

Für $c^2 > a^2$ ergibt sich als Schnittkurve eine **Ellipse** und $P_0 = (\frac{-ad}{c^2-a^2}, 0, \frac{dc}{c^2-a^2})$ ist ihr Mittelpunkt

$$\begin{aligned} P_1 &= (\frac{d}{a-c}, 0, -\frac{d}{a-c}), & P_1' &= (\frac{d}{a+c}, 0, -\frac{d}{a+c}), \\ P_2 &= (-\frac{ad}{c^2-a^2}, \frac{d}{\sqrt{c^2-a^2}}, \frac{dc}{c^2-a^2}), & P_2' &= (-\frac{ad}{c^2-a^2}, \frac{-d}{\sqrt{c^2-a^2}}, \frac{dc}{c^2-a^2}), \end{aligned}$$

sind ihre Scheitel. Dabei sind P_1, P_2 bzw. P_1', P_2' zueinander konjugiert.

Für $c^2 < a^2$ ergibt sich eine **Hyperbel** und die Punkte P_0, P_1 und P_1' können aus dem Ellipsenfall übernommen werden. Bei P_2 und P_2' muß man nur $\sqrt{c^2-a^2}$ durch $\sqrt{a^2-c^2}$ ersetzen. P_2, P_2' sind hier allerdings Nebenscheitel. Ferner sei noch bemerkt, daß die Lösung des letzten Falles auf den Fall $c = 0$ (Fall I) fortgesetzt werden kann.

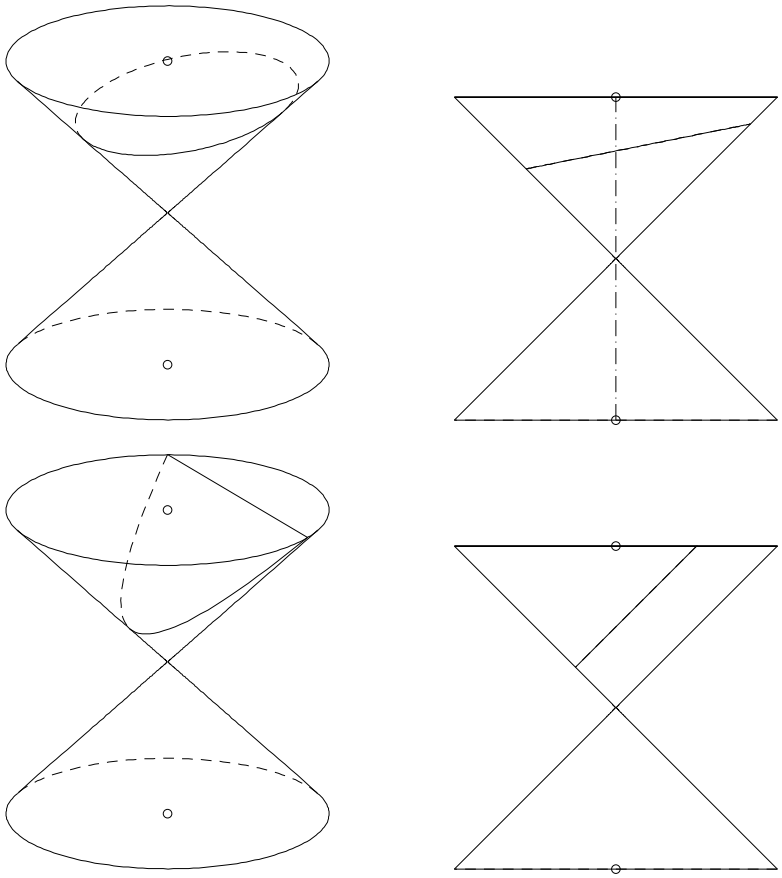


Abbildung 8.7: ebene Schnitte eines Kegels

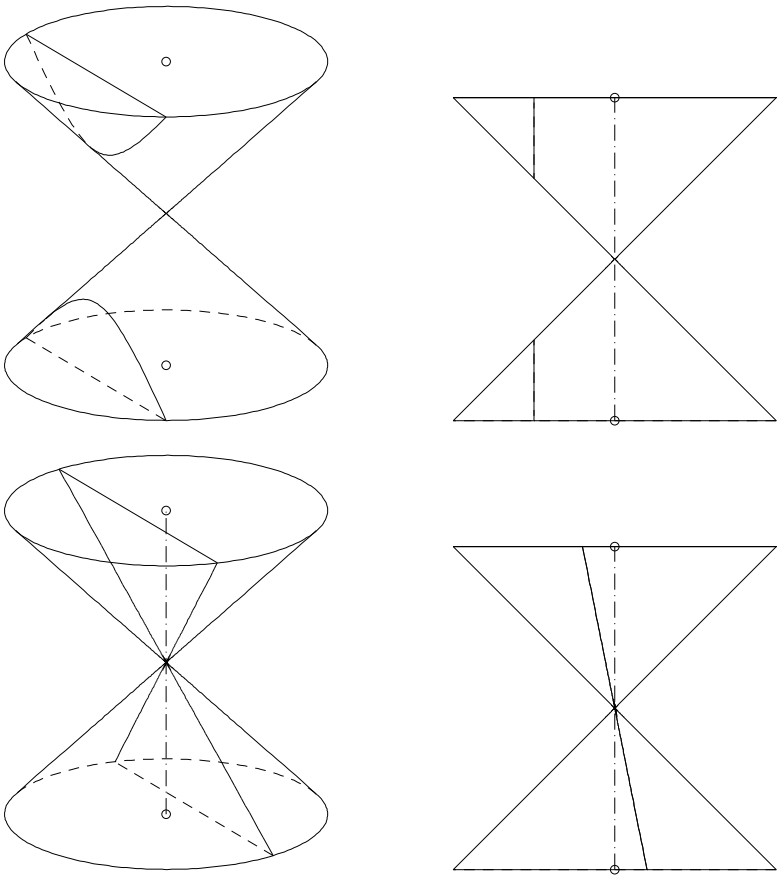


Abbildung 8.8: ebene Schnitte eines Kegels (Forts.)

In dem folgenden Unterprogramm werden für die einzelnen Fälle die Punkte P_0, P_1, P_2 berechnet. In den Fällen, in denen als Schnitt ein Punkt, eine Gerade bzw. ein Geradenpaar auftritt, ist $P_0 = (0,0,0)$ und

$$P_1 = (c, \sqrt{c^2-a^2}, -a), \quad P_2 = (c, -\sqrt{c^2-a^2}, a), \quad \text{falls} \quad c^2 > a^2.$$

Der Index **ind** gibt an, um welchen Fall es sich handelt.

```
procedure is_unitcone_plane0(a,c,d: real; var p0,p1,p2: vt3d;
                                          var ind: integer);
 {Berechnet den Schnitt des Kegels  z*z = x*x + y*y  mit der
  Ebene  a*x + c*z = d .
  ind=100: Punkt  , ind=10: Gerade , ind=11: Geradenpaar
  ind=  0: Parabel, ind= 1: Ellipse, ind=-1: Hyperbel }
 var a2,c2,wu,ca2: real;
 begin
   a2:= a*a ;   c2:= c*c ;  ca2:= c2-a2 ;
   if abs(d)<eps5  then
      if c2>a2  then  ind:= 100
         else
          begin
```

```
          ind:= 11  ;  wu:= sqrt(a2-c2);
          if abs(wu)<eps5 then  ind:= 10;
          put3d(0,  0, 0, p0);
          put3d(c, wu,-a, p1);
          put3d(c,-wu,-a, p2);
          end
       else
        if abs(c2-a2)<eps5  then
          begin
          ind:= 0;
          put3d(d/(2*a),  0 , d/(2*c), p0);
          put3d(   p0.x, d/c,    p0.z, p1);
          put3d(      0,  0 ,    p1.y, p2);
          end
         else
          begin
          ind:= 1 ;  if ca2<0 then  ind:= -1  ;
          put3d(-a*d/ca2, 0 , d*c/ca2 , p0);
          put3d( d/(a-c), 0 , -d/(a-c), p1);
          put3d( p0.x, d/sqrt(abs(ca2)), p0.z, p2);
          end;
 end;   { is_unitcone_plane0 }
{****************}
```

Mit Hilfe von `is_unitcone_plane0` werden wir jetzt die Schnitte des Einheitskegels mit einer beliebigen Ebene $ax + by + cz = d$ bestimmen. Hierzu wird zunächst eine Rotation um die z-Achse durchgeführt, sodaß die Ebene die in `is_unitcone_plane0` geforderte Form erhält.

```
procedure is_unitcone_plane(a,b,c,d: real; var p0,p1,p2: vt3d; var ind: integer);
 {Berechnet den Schnitt des Kegels  z*z = x*x + y*y  mit der Ebene  a*x + b*y + c*z = d.}
 var aa,cw,sw : real;
 begin
   aa:= sqrt(a*a + b*b);
   is_unitcone_plane0(aa,c,d, p0,p1,p2,ind);
   if aa>eps4 then
      begin cw:= a/aa;  sw:= b/aa;
      rotorz(cw,sw,p0, p0); rotorz(cw,sw,p1, p1); rotorz(cw,sw,p2, p2);
      end;
 end;  { is_unitcone_plane }
{****************}
```

8.2.2 Ebene Schnitte eines Kegels

Gegeben: a) Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$,
b) Kegel $K : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\alpha\cos\varphi + \mathbf{f}_2\alpha\sin\varphi + \mathbf{f}_3(1-\alpha)$.
Gesucht: Schnitt $\varepsilon \cap K$.

Mit der Kegelspitze $S_0 : \mathbf{s}_0 = \mathbf{q}_0 + \mathbf{f}_3$ läßt sich der Kegel wie folgt beschreiben
b') $K : \mathbf{x} = \mathbf{s}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 = \zeta^2.$

Durch Einsetzen von b') in die Gleichung der Ebene ergibt sich das Gleichungssystem:

$$\begin{aligned} \gamma_1\xi + \gamma_2\eta + \gamma_3\zeta &= \gamma_0, \quad \text{mit} \quad \gamma_i = \mathbf{n}\cdot\mathbf{f}_i, \quad i = 1,2,3, \quad \gamma_0 = d - \mathbf{n}\cdot\mathbf{s}_0 \\ \xi^2 + \eta^2 &= \zeta^2 \end{aligned}$$

Löst man dieses System mit Hilfe von `is_unitcone_plane`, so erhält man die Daten der Schnittkurve (s.o.) zunächst in $\xi-\eta-\zeta-$Koordinaten. Durch Einsetzen in $\mathbf{x}=\mathbf{s}_0+\mathbf{f}_1\xi+\mathbf{f}_2\eta+\mathbf{f}_3\zeta$ ergeben sich die zugehörigen $x-y-z-$Koordinaten.

```
procedure is_cone_plane(q0,q1,q2,peak,nv: vt3d; d: real; var pc0,pc1,pc2 : vt3d;
                                                          var ind : integer);
 {Berechnet den Schnitt des Kegels mit q0,q1,q2: Mittelpunkt und konj. Punkte der
  Basisellipse, peak: Spitze mit der Ebene  nv*x=d.}
```

8.2.3 Parallelprojektion eines Kegels

Auch einen Kegel müssen wir zu seiner Darstellung begrenzen. Wir werden hier immer den Teil eines Kegels darstellen, der von seiner Basisellipse und seiner Spitze begrenzt wird. D. h. wir gehen aus von

$$K=\{\mathbf{q}_0+\mathbf{f}_1\alpha\cos\varphi+\mathbf{f}_2\alpha\sin\varphi+\mathbf{f}_3(1-\alpha)\,|\,0\le\varphi\le 2\pi,\quad 0\le\alpha\le 1\}$$

Die Normale in einem Punkt $\mathbf{q}(\varphi,\alpha)$ ergibt sich aus

$$\begin{aligned}\mathbf{q}_\varphi(\varphi,\alpha)\times\mathbf{q}_\alpha(\varphi,\alpha) &= (-\mathbf{f}_1\alpha\sin\varphi+\mathbf{f}_2\alpha\cos\varphi)\times(\mathbf{f}_1\cos\varphi+\mathbf{f}_2\sin\varphi-\mathbf{f}_3)\\ &= \alpha(\mathbf{f}_1\times\mathbf{f}_2-\mathbf{f}_2\times\mathbf{f}_3\cos\varphi-\mathbf{f}_3\times\mathbf{f}_1\sin\varphi).\\ \mathbf{n}(\varphi,\alpha) &= \mathbf{f}_1\times\mathbf{f}_2+\mathbf{f}_2\times\mathbf{f}_3\cos\varphi+\mathbf{f}_3\times\mathbf{f}_1\sin\varphi.\end{aligned}$$

Der Punkt $Q:\mathbf{q}(\varphi,\alpha)$ ist genau dann Umrißpunkt, wenn für die (negative) Projektionsrichtung $\mathbf{n}_0$ und die Normale $\mathbf{n}(\varphi,\alpha)$ in Q gilt :

$$\mathbf{n}_0\cdot\mathbf{n}(\varphi,\alpha)=\mathbf{n}_0\cdot(\mathbf{f}_1\times\mathbf{f}_2)+\mathbf{n}_0\cdot(\mathbf{f}_2\times\mathbf{f}_3)\cos\varphi+\mathbf{n}_0\cdot(\mathbf{f}_3\times\mathbf{f}_1)\sin\varphi=0.$$

$\xi:=\cos\varphi$ und $\eta:=\sin\varphi$ müssen also Lösungen des folgenden Gleichungssystems sein:

$$\gamma_1\xi+\gamma_2\eta=-\gamma_3,\quad \xi^2+\eta^2=1,$$

wobei

$$\gamma_1:=\mathbf{n}_0\cdot(\mathbf{f}_2\times\mathbf{f}_3),\quad \gamma_2:=\mathbf{n}_0\cdot(\mathbf{f}_3\times\mathbf{f}_1),\quad \gamma_3:=\mathbf{n}_0\cdot(\mathbf{f}_1\times\mathbf{f}_2)$$

ist.
Falls $\gamma_1\neq 0$ oder $\gamma_2\neq 0$ ist, löst man das Gleichungssystem mit dem Unterprogramm `is_unitcircle_line`. Sind $(\xi_1,\eta_1),(\xi_2,\eta_2)$ Lösungen dieses Gleichungssystems, so sind

$$C_1:\ \mathbf{c}_1=\mathbf{q}_0+\mathbf{f}_1\xi_1+\mathbf{f}_2\eta_1\quad\text{und}\quad C_2:\ \mathbf{c}_2=\mathbf{q}_0+\mathbf{f}_1\xi_2+\mathbf{f}_2\eta_2$$

Umrißpunkte auf der Basisellipse von K.
Falls $\gamma_1=\gamma_2=0$ ist, ist die Projektionsrichtung gleich der Achsenrichtung und die Mantellinien tragen nichts zum Umriß bei.

Analog zu den Prozeduren für Zylinder erhalten wir hier:

```
procedure cone_pp_contpts(q0,q1,q2,peak: vt3d; var qc1,qc2 : vt3d;  var contlines: boolean);
 {Berechnet den Umriss des durch q0, q1, q2, peak bestimmten Kegels.}
 {*************}
procedure pp_cone(q0,q1,q2,peak: vt3d; style: integer);
 {Projektion des durch q0, q1, q2, peak  bestimmten Kegels.
  q0: Mittelpunkt der Basis-Ellipse, q1,q2,: konj. Punkte; peak: Spitze}
 {*************}
procedure pp_cone_orth(p0,p1,p2 : vt3d; clength : real; style : integer);
 {Projiziert einen senkrechten elliptischen Kegel der Laenge clengeth.
  p0: Mittelpunkt, p1,p2 konj. Punkte der Basis-Ellipse.}
```

Aufgabe 8.2 *Schreibe ein Programm, das die folgenden abgeschnittetenen Kegel darstellt. a) Ellipsenschnitt b) Hyperbelschnitt c) Parabelschnitt)*

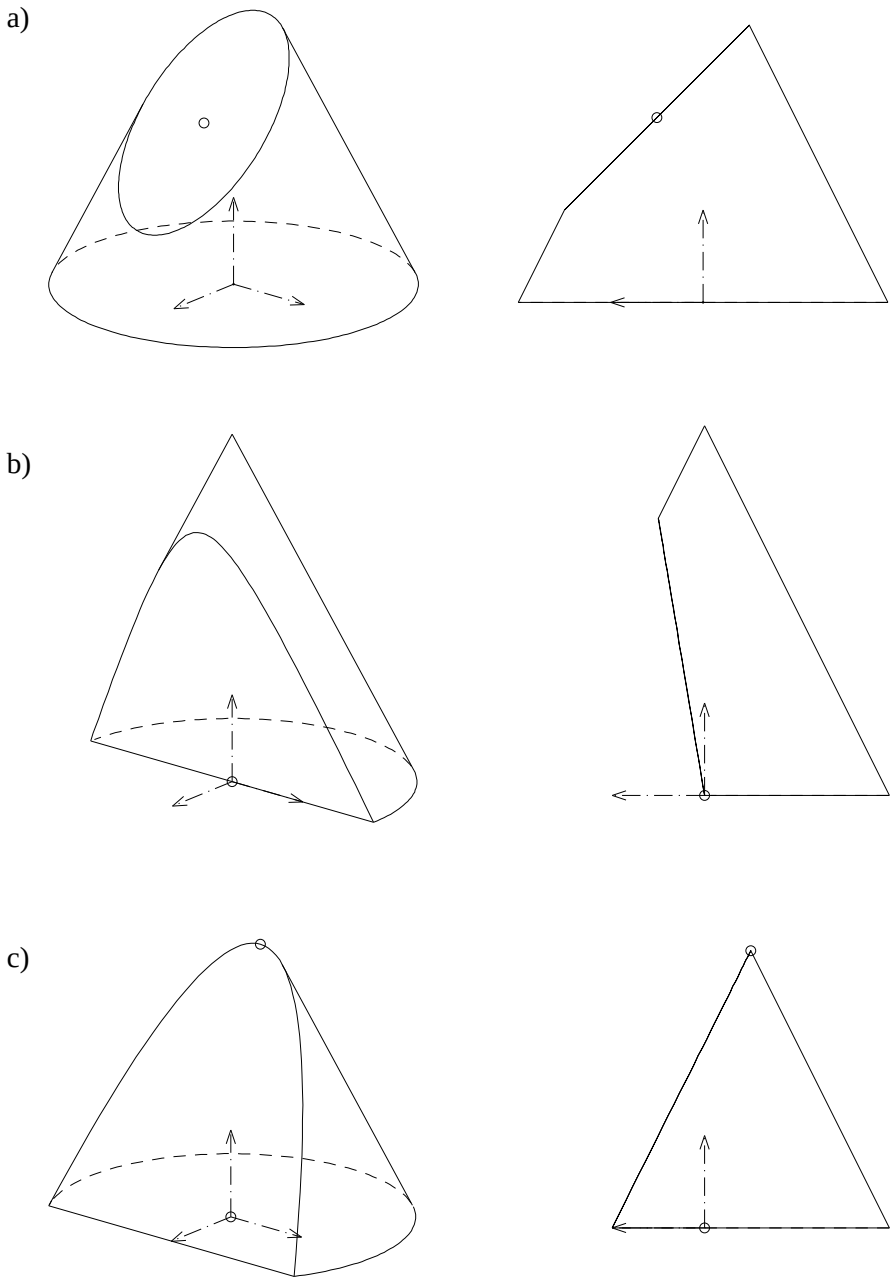


Abbildung 8.9: zu Aufgabe 8.2

8.2.4 Zentralprojektion eines Kegels

Für den Kegel $\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\alpha\xi + \mathbf{f}_2\alpha\eta + \mathbf{f}_3(1-\alpha), \quad \xi^2 + \eta^2 = 1.$
lautet die Umrißbedingung: $\gamma_1\xi + \gamma_2\eta = -\gamma_0, \qquad \xi^2 + \eta^2 = 1,$
mit $\gamma_0 = (\mathbf{z} - \mathbf{q}_0 - \mathbf{f}_3) \cdot (\mathbf{f}_1 \times \mathbf{f}_2), \quad \gamma_1 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_2 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_3 \times \mathbf{f}_1).$

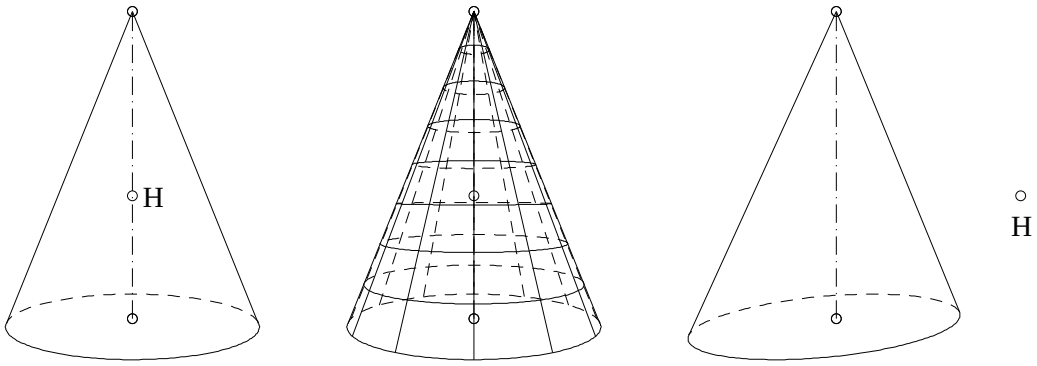


Abbildung 8.10: Zentralprojektion eines Kegels

Kapitel 9

HYPERBOLOID

9.1 Einschaliges Hyperboloid

Läßt man die Hyperbel $x^2 - z^2 = 1$ der $x - z-$Ebene um ihre Nebenachse rotieren, so entsteht ein einschaliges Hyperboloid. Rotation der Hyperbel um ihre Hauptachse liefert ein zweischaliges Hyperboloid. Einschalige Hyperboloide benutzt man z. B. beim Bau von Kühltürmen.Wir wollen uns hier zunächst mit dem einschaligen Hyperboloiden beschäftigen.

Ein affines Bild des Einheitshyperboloids

$$H_1 := \{(x, y, z) \in \mathbb{R}^3 \,|\, x^2 + y^2 - z^2 = 1\}$$

heißt **einschaliges Hyperboloid**.
Beschreibt man H_1 (analog dem Ellipsoid) durch die folgende Parameterdarstellung:

$$H_1 = \{(\cosh\beta\cos\alpha, \cosh\beta\sin\alpha, \sinh\beta) \,|\, 0 \leq \alpha \leq 2\pi, \quad \beta \in \mathbb{R}\},$$

so gilt

Zu jedem Hyperboloid H gibt es einen Vektor $\mathbf{q}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$, so daß

$$H = \{\mathbf{q}_0 + \mathbf{f}_1 \cosh\beta\cos\alpha + \mathbf{f}_2 \cosh\beta\sin\alpha + \mathbf{f}_3 \sinh\beta \,|\, \ldots\}$$

ist.
Umgekehrt ist für einen Vektor $\mathbf{q}_0$, und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ die Punktmenge $H = \{\ldots|\ldots\}$ ein Hyperboloid.

Die Koordinaten ξ, η, ζ eines Punktes von H bezüglich des Koordinatensystems mit Nullpunkt $Q_0 : \mathbf{q}_0$ und der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ genügen also der Gleichung

$$\xi^2 + \eta^2 - \zeta^2 = (\cosh\beta\cos\alpha)^2 + (\cosh\beta\sin\alpha)^2 - (\sinh\beta)^2 = \ldots = 1,$$

d. h. H ist bezüglich des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ das "Einheitshyperboloid".

$Q_0 : \mathbf{q}_0$ heißt Mittelpunkt des Hyperboloids H.
$E_0 := \{\mathbf{q}_0 + \mathbf{f}_1\cos\alpha + \mathbf{f}_2\sin\alpha \,|\, \ldots\}$ heißt die *Basisellipse* von H.
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ heißen **konjugierte Halbmesser** von H und
$Q_1 : \mathbf{q}_0 + \mathbf{f}_1, \quad Q_2 : \mathbf{q}_0 + \mathbf{f}_2, \quad Q_3 : \mathbf{q}_0 + \mathbf{f}_3$ **konjugierte Punkte** von H.

$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sind i.a. nicht orthogonal. Aber es gilt:

Die Tangentialebene in Q_1 ist parallel zur Ebene, die von $\mathbf{f}_2, \mathbf{f}_3$ aufgespannt wird, die Tangentialebene in Q_2 ist parallel zu $\mathbf{f}_1, \mathbf{f}_3$.
(Q_3 ist kein Punkt von H!)

Falls $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ paarweise senkrecht zueinander stehen, nennt man die Punkte $\mathbf{q}_0 \pm \mathbf{f}_1$, $\mathbf{q}_0 \pm \mathbf{f}_2$ **Scheitel** und $\mathbf{q}_0 \pm \mathbf{f}_3$ **Nebenscheitel** von H.

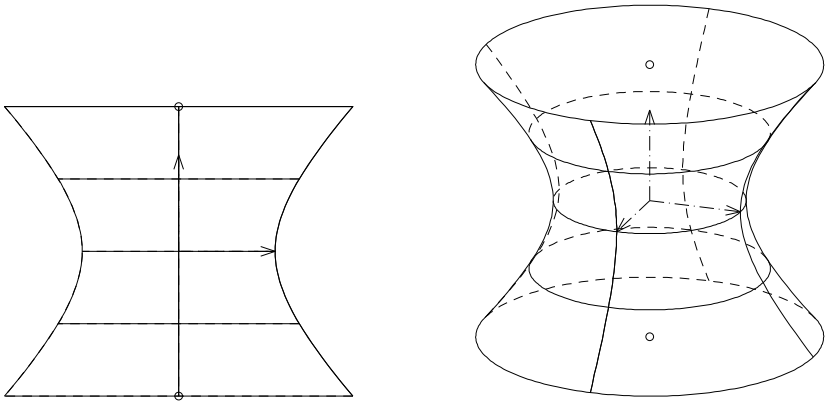


Abbildung 9.1: einschaliges Hyperboloid

Für die **Gleichung** $f(x, y, z) = 0$ des Hyperboloids $\mathbf{x} = \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 - \zeta^2 = 1$, in $x - y - z-$Koordinaten ergibt sich (vgl. Ellipsoid):

$$f(x, y, z) := (\det(\mathbf{x}, \mathbf{f}_2, \mathbf{f}_3))^2 + (\det(\mathbf{f}_1, \mathbf{x}, \mathbf{f}_3))^2 - (\det(\mathbf{f}_1, \mathbf{f}_2, \mathbf{x}))^2 - (\det(\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3))^2 = 0,$$

wobei $\det(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ die $3 \times 3-$Determinate mit den Spaltenvektoren $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$ ist.

9.1.1 Ebene Schnitte des Einheits-Hyperboloids

Wir bestimmen zunächst die Schnitte des "Einheits"-Hyperboloids $H_1 := \{(x, y, z) \in \mathbb{R} \,|\, x^2 + y^2 - z^2 = 1\}$ mit der Ebene $\varepsilon : ax + cz = d$. Mit Überlegungen, wie wir sie für einen Kegel durchgeführt haben, erhält man das folgende Ergebnis:

Fall I: $0 \neq c^2 - a^2 \neq -d^2$

Ia): Falls $c^2 - a^2 > 0$ ist, ist die Schnittkurve eine **Ellipse**,

Ib): Falls $c^2 - a^2 < -d^2$ ist, ist die Schnittkurve eine **Hyperbel**
und $P_0 = (-ad/(c^2 - a^2), 0, cd/(c^2 - a^2))$ ist der Mittelpunkt,

$$\begin{aligned}
P_1 &= (-ad/(c^2 - a^2), \quad \sqrt{|(c^2 - a^2 + d^2)/(c^2 - a^2)|}, cd/(c^2 - a^2)) \\
P_1' &= (-ad/(c^2 - a^2), -\sqrt{|(c^2 - a^2 + d^2)/(c^2 - a^2)|}, cd/(c^2 - a^2)) \\
P_2 &= ((-ad + c\sqrt{|c^2 - a^2 + d^2|})/(c^2 - a^2), 0, (dc - a\sqrt{|c^2 - a^2 + d^2|})/(c^2 - a^2)) \\
P_2' &= ((-ad - c\sqrt{|c^2 - a^2 + d^2|})/(c^2 - a^2), 0, (dc - a\sqrt{|c^2 - a^2 + d^2|})/(c^2 - a^2))
\end{aligned}$$

sind die Scheitel bzw. Nebenscheitel.

Ic): Falls $-d^2 < c^2 - a^2 < 0$ ist, ist die Schnittkurve eine **Hyperbel** und P_0 (aus Ib)) ihr Mittelpunkt, P_2, P_2' ihre Scheitel und P_1, P_1' ihre Nebenscheitel.

Fall II: $c^2 - a^2 = -d^2 \neq 0$
Die Schnittkurve besteht aus zwei **sich schneidenden Geraden**:

$$\mathbf{p}_0 + t\mathbf{p}_1, t \in \mathbb{R}, \quad \mathbf{p}_0 + t\mathbf{p}_2, t \in \mathbb{R},$$

$$\mathbf{p}_0 = (a/d, 0, -c/d),\ \mathbf{p}_1 = (c, d, -a),\ \mathbf{p}_2 = (c, -d, -a)$$

Fall III: $c^2 - a^2 = 0$

IIIa) Falls $d = 0$ ist, ergibt sich das **parallele Geradenpaar**:

$$\mathbf{p}_1 + t\mathbf{p}_0, t \in \mathbb{R}, \quad \mathbf{p}_2 + t\mathbf{p}_0, t \in \mathbb{R},$$

$$\mathbf{p}_0 = (-c, 0, a),\ \mathbf{p}_1 = (0, 1, 0),\ \mathbf{p}_2 = (0, -1, 0)$$

IIIb) Falls $d \neq 0$ ist, erhält man eine **Parabel** mit

$$\begin{array}{lcll} P_0 & = & ((d/a + a/d)/2, 0, (d/c - c/d)/2) & \text{(Scheitel)} \\ P_1 & = & (\ldots, \sqrt{1 + (d/a)^2}, \ldots) & \text{(s.Kap. 5.3)} \\ P_2 & = & (0, 0, d/a) & (\text{ “ }). \end{array}$$

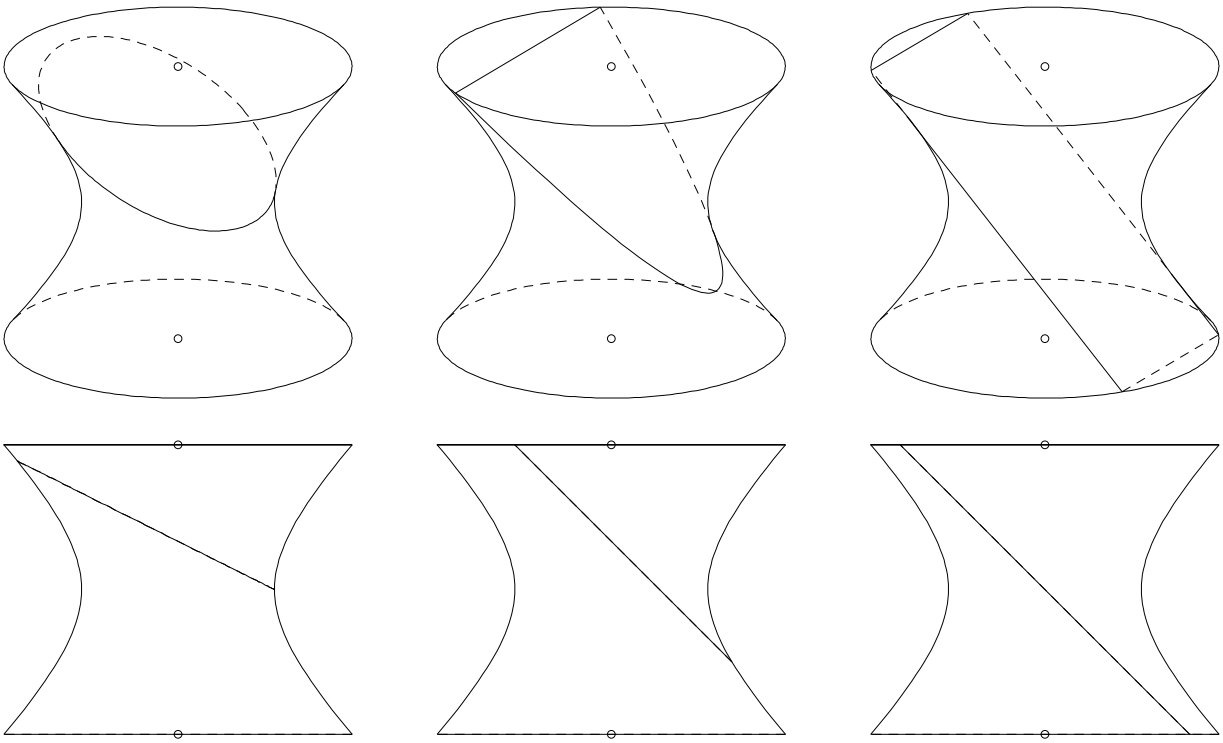


Abbildung 9.2: Ebene Schnitte eines einschaligen Hyperboloids

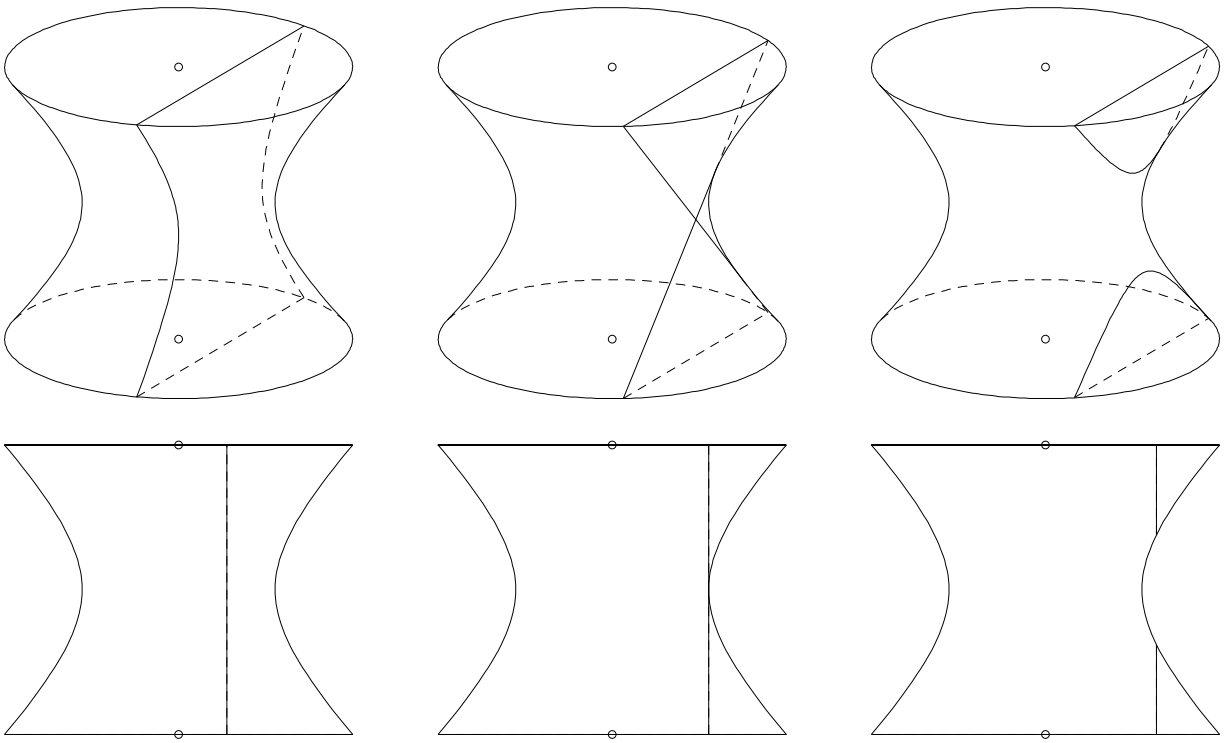


Abbildung 9.3: Ebene Schnitte eines einschaligen Hyperboloids (Forts.)

Das Unterprogramm `is_unithyperboloid_plane0` führt die obigen Rechnungen durch und bestimmt für den jeweiligen Fall die Punkte P_0, P_1, P_2 und liefert eine Zahl **ind**, mit der man die einzelnen Fälle von einander unterscheiden kann. Und zwar ist die Schnittkurve

eine	Ellipse	falls `ind` $= 1$,				
eine	Hyperbel	falls `ind` $= -1$,	(Fall 1b)	bzw.	`ind` $= -2$	(Fall 1c),
eine	Parabel	falls `ind` $= 0$,				
ein	Geradenpaar	falls `ind` $= 10$,	(parallel)	bzw.	`ind` $= 11$	(schneidend).

```
procedure is_unithyperboloid_plane0(a,c,d : real; var pc0,pc1,pc2 : vt3d;
                                                   var ind : integer);
{ Berechnet den Schnitt der Ebene a*x+c*z=d mit dem Hyperboloid x*x+y*y-z*z=1
  ind=1  : Ellipse,   ind=0  : Parabel,   ind=-1, -2 : Hyperbel,
  ind=10 : paralleles Geradenpaar p1+t*p0, p2+s*p0,
  ind=11 : nicht paralleles Geradenpaar p0+t*p1, p0+s*p2. }
 var a2,c2,d2,ca2,wu : real;      i1,i2 : integer;
 begin
   a2 := sqr(a);   c2 := sqr(c);   d2 := sqr(d);
   ca2:= c2 - a2;   if abs(ca2)<eps8 then ca2:= 0;
   if ca2 <> 0 then
      begin
      if abs(ca2+d2)<eps8 then                          {Fall II}
         begin
         ind := 11;
         put3d(a/d, 0,-c/d, pc0);
         put3d(  c, d,  -a, pc1);
```

```
         put3d(  c,-d,  -a, pc2);
         end
        else
         begin                                                  {Fall I }
         put3d(-a*d/ca2, 0, c*d/ca2, pc0);
         pc1.x:= pc0.x;                          {Fall Ia,Ib}
         pc1.y:= sqrt(abs((ca2 + d2)/ca2));
         pc1.z:= pc0.z;
         wu := sqrt(abs(ca2+d2));
         pc2.x:= (-a*d+c*wu)/ca2;
         pc2.y:= 0;
         pc2.z:= ( d*c-a*wu)/ca2;
         if (-d2 < ca2) and (ca2 < 0) then    {Fall Ic }
            begin
            change3d(pc1,pc2);
            ind:= -2;
            end;
         if ca2 > 0 then         ind:= 1;      {Fall Ia }
         if ca2 <= -d2 then      ind:=-1;      {Fall Ib }
         end
      end
     else      {ca2=0 :}                       {Fall III }
      begin
      if d=0 then                               {Fall IIIa }
         begin
         ind:= 10;
         put3d(-c, 0,a, pc0);
         put3d( 0, 1,0, pc1);
         put3d( 0,-1,0, pc2);
         end
        else                                    {Fall IIIb }
         begin
         ind:= 0;
         put3d((d/a+a/d)/2,                0, (d/c-c/d)/2, pc0);
         put3d(      pc0.x, sqrt(1+sqr(d/a)),       pc0.z, pc1);
         put3d( 0, 0, d/a, pc2);
         end;
     end;
 end;  { is_hyperboloid_plane0 }
{************}
```

In dem Unterprogramm `is_unithyperboloid_plane` werden die Schnitte des Einheitshyperboloids mit einer beliebigen Ebene $ax + by + cz = d$ bestimmt. Hierzu wird zunächst eine Rotation um die $z-$Achsae durchgeführt, sodaß die Ebene die in `is_unithyperboloid_plane0` geforderte Form erhält.

```
procedure is_unithyperboloid_plane(a,b,c,d: real; var p0,p1,p2: vt3d; var ind: integer);
```

9.1.2 Ebene Schnitte eines einschaligen Hyperboloids

Gegeben: a) Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$,
b) Hyperboloid $H : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 - \zeta^2 = 1.$
Gesucht: Schnitt $\varepsilon \cap H$.

Durch Einsetzen von b) in die Gleichung der Ebene erhält man das Gleichungssystem:

$$\begin{array}{rcl} \gamma_1\xi+\gamma_2\eta+\gamma_3\zeta & = & \gamma_0, \quad \text{mit } \gamma_i=\mathbf{n}\cdot\mathbf{f}_i,\ i=1,2,3,\ \gamma_0=d-\mathbf{n}\cdot\mathbf{q}_0, \\ \xi^2+\eta^2-\zeta^2 & = & 1 \end{array}$$

Löst man dieses System mit Hilfe von `is_unithyperboloid_plane`, so erhält man die Daten der Schnittkurve (s.o.) zunächst in $\xi-\eta-\zeta-$Koordinaten. Durch Einsetzen in $\mathbf{x}=\mathbf{q}_0+\mathbf{f}_1\xi+\mathbf{f}_2\eta+\mathbf{f}_3\zeta$ ergeben sich die zugehörigen $x-y-z-$Koordinaten.

```
procedure is_hyperboloid_plane(q0,q1,q2,q3,nv: vt3d; d: real; var pc0,pc1,pc2: vt3d;
                                                              var ind: integer);
{Berechnet den Schnitt des Hyperboloids mit Mittelpunkt q0 und den konjugierten
Punkten q1,q2,q3  mit der Ebene  nv*x=d .}
```

9.1.3 Parallelprojektion eines Hyperboloids

Wir bestimmen zunächst Umrißpunkte des Hyperboloids

$$\begin{array}{rcl} H & = & \{\mathbf{q}_0+\mathbf{f}_1\cosh\beta\cos\alpha+\mathbf{f}_2\cosh\beta\sin\alpha+\mathbf{f}_3\sinh\beta\,|\,\ldots\} \\ & = & \{\mathbf{q}_0+\mathbf{f}_1\xi+\mathbf{f}_2\eta+\mathbf{f}_3\zeta\,|\,\xi^2+\eta^2-\zeta^2=1\} \end{array}$$

Analog zu den überlegungen für das Ellipsoid ergibt sich hier:
Eine Normale im Punkt $Q:\mathbf{q}(\alpha,\beta)$ ist

$$\mathbf{n}(\alpha,\beta)=\mathbf{f}_2\times\mathbf{f}_3\cosh\beta\cos\alpha+\mathbf{f}_3\times\mathbf{f}_1\cosh\beta\sin\alpha-\mathbf{f}_1\times\mathbf{f}_2\sinh\beta.$$

Wenn wir von Selbstüberdeckungen (ein Hyperboloid ist nicht konvex !) absehen, ist Q ein Umrißpunkt, falls $\mathbf{n}(\alpha,\beta)$ auf der (negativen) Projektionrichtung $\mathbf{n}_0$ senkrecht steht. In $\xi-\eta-\zeta-$Koordinaten liefert die Bedingung $\mathbf{n}_0\cdot\mathbf{n}(\alpha,\beta)=0$ das Gleichungssystem:

$$\begin{array}{rcl} \gamma_1\xi+\gamma_2\eta+\gamma_3\zeta & = & \gamma_0, \quad \text{mit } \gamma_1=\mathbf{n}_0\cdot(\mathbf{f}_2\times\mathbf{f}_3),\ \ \gamma_2=\mathbf{n}_0\cdot(\mathbf{f}_3\times\mathbf{f}_1),\ \ \gamma_3=\mathbf{n}_0\cdot(\mathbf{f}_1\times\mathbf{f}_2), \\ \xi^2+\eta^2-\zeta^2 & = & 1 \end{array}$$

Die Prozedur `is_unithyperboloid_plane` liefert die Daten für den Umriß zunächst in $\xi-\eta-\zeta-$Koordinaten. Der Umriß ist entweder eine Hyperbel oder eine Ellipse oder ein paralleles Geradenpaar, das parallel zu $\mathbf{n}_0$ ist. Mit Hilfe von $\mathbf{x}=\mathbf{q}_0+\mathbf{f}_1\xi+\mathbf{f}_2\eta+\mathbf{f}_3\zeta$ erhält man die den Umriß definierenden Punkte in $x-y-z-$Koordinaten.

```
procedure hyperboloid_pp_cont(q0,q1,q2,q3: vt3d; var qc0,qc1,qc2: vt3d;
                                                  var ind: integer);
{Berechnet den Umriss des durch q0, q1, q2, q3 bestimmten Hyperboloids.}
```

Das folgende Unterprogramm stellt ein Rotationshyperboloid mit der Gleichung $(x^2+y^2)/a^2-z^2/c^2=1$ im Bereich $z_1\le z\le z_2$ dar.

```
procedure pp_hyperboloid_rev(a,c,z1,z2: real; style: integer);
{Projektion des ROTATIONSHYPERBOLOIDs  (x*x+y*y)/a*a - z*z/c*c = 1 , z1<z<z2,
 fr den Fall : abs(tan(v))<c/a  (Umriss ist Hyperbel).}
```

Aufgabe 9.1 *a) Auf einem Hyperboloid gehen durch jeden Punkt genau zwei Geraden, die ganz auf dem Hyperboloid liegen. Schreibe ein Programm, das die folgenden Bilder erzeugt. (Auf dem Rotationshyperboloid* $(x^2+y^2)/a^2-z^2/c^2=1$ *gehen z. B. durch den Punkt* $\mathbf{x}_0:=(a,0,0)$ *die Geraden* $\mathbf{x}=\mathbf{x}_0+t(0,a,\pm c)$*.)*
b) Ergänze das Programm aus a) so, daß auch Kreise und Hyperbeln auf dem Hyperboloid dargestellt werden können.

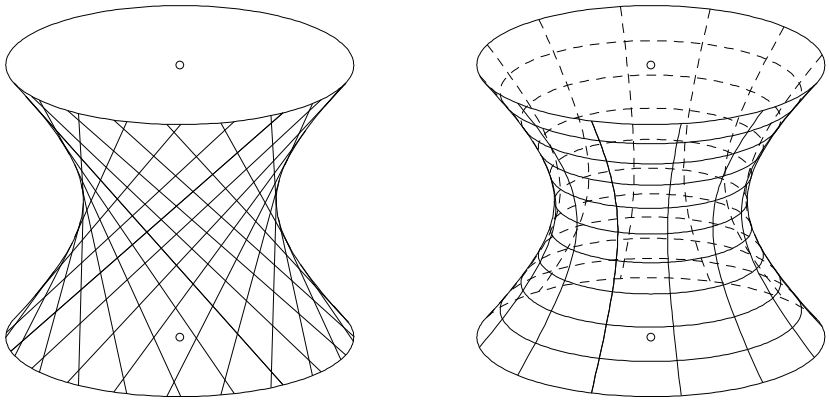


Abbildung 9.4: zu Aufgabe 9.1

9.1.4 Zentralprojektion eines Hyperboloids

Für $\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad \xi^2 + \eta^2 - \zeta^2 = 1$
lautet die Umrißbedingung: $\gamma_1\xi + \gamma_2\eta - \gamma_3\zeta = \gamma_0, \qquad \xi^2 + \eta^2 - \zeta^2 = 1,$
mit $\gamma_0 = \mathbf{f}_1 \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_1 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_2 \times \mathbf{f}_3), \quad \gamma_2 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_3 \times \mathbf{f}_1), \quad \gamma_3 = (\mathbf{z} - \mathbf{q}_0) \cdot (\mathbf{f}_1 \times \mathbf{f}_2).$

9.2 Zweischaliges Hyperboloid

Läßt man die Hyperbel $-x^2 + z^2 = 1$ der $x - z-$Ebene um ihre Hauptachse rotieren, so entsteht ein zweischaliges Hyperboloid.
Wie beim einschaligen Hyperboloid definiert man

Ein affines Bild des Einheitshyperboloids $H := \{(x, y, z) \in \mathbb{R}^3 \mid -x^2 - y^2 + z^2 = 1\}$ heißt *2-schaliges Hyperboloid*.

Eine Parameterdarstellung von H_1 ist:

$$H_1 = \{(\sinh\beta\cos\alpha, \sinh\beta\sin\alpha, \pm\cosh\beta) \mid 0 \le \alpha \le 2\pi, \beta \in \mathbb{R}\}.$$

Damit gilt
Zu jedem 2-schaligen Hyperboloid H gibt es einen Vektor $\mathbf{q}_0$ und drei linear unabhängige Vektoren $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$, so daß

$$H = \{\mathbf{q}_0 + \mathbf{f}_1 \sinh\beta\cos\alpha + \mathbf{f}_2 \sinh\beta\sin\alpha \pm \mathbf{f}_3 \cosh\beta \mid \ldots\}$$

ist.
Die Koordinaten ξ, η, ζ eines Punktes von H bezüglich des Koordinatensystems mit Nullpunkt $Q_0 : \mathbf{q}_0$ und der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ genügen also der Gleichung $-\xi^2 - \eta^2 + \zeta^2 = 1$, d.h. H ist bezüglich des Koordinatensystems $(Q_0; \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3)$ das "Einheitshyperboloid".

$Q_0 : \mathbf{q}_0$ heißt Mittelpunkt des Hyperboloids H.
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ heißen **konjugierte Halbmesser** von H und
$Q_1 : \mathbf{q}_0 + \mathbf{f}_1, Q_2 : \mathbf{q}_0 + \mathbf{f}_2, Q_3 : \mathbf{q}_0 + \mathbf{f}_3$, **konjugierte Punkte** von H. Nur Q_3 liegt auf H.
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sind i.a. nicht orthogonal.

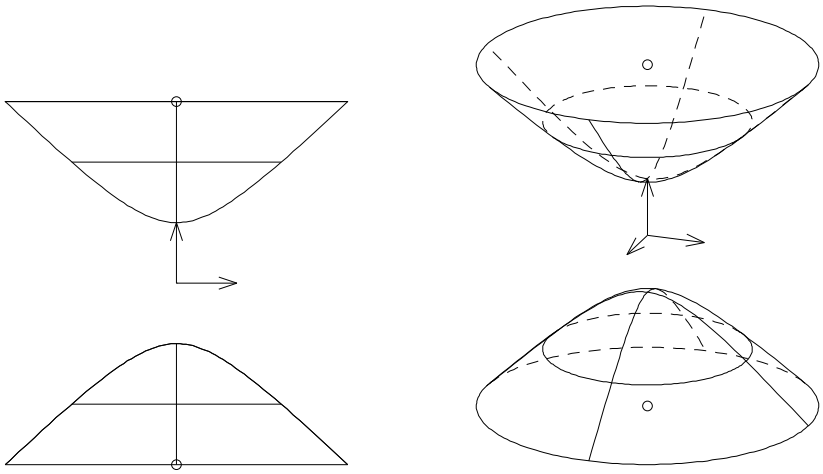


Abbildung 9.5: zweischaliges Hyperboloid

Falls $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ paarweise senkrecht zueinander stehen, nennt man die Punkte $\mathbf{q}_0 \pm \mathbf{f}_1, \mathbf{q}_0 \pm \mathbf{f}_2$ **Nebenscheitel** von H und $\mathbf{q}_0 \pm \mathbf{f}_3$ **Scheitel** von H.

Für die **Gleichung** $f(x,y,z) = 0$ des Hyperboloids $\mathbf{x} = \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \quad -\xi^2 - \eta^2 + \zeta = 1$, in $x-y-z-$Koordinaten ergibt sich (vgl. Ellipsoid):

$$f(x,y,z) := -(\det(\mathbf{x},\mathbf{f}_2,\mathbf{f}_3))^2 - (\det(\mathbf{f}_1,\mathbf{x},\mathbf{f}_3))^2 + (\det(\mathbf{f}_1,\mathbf{f}_2,\mathbf{x}))^2 - (\det(\mathbf{f}_1,\mathbf{f}_2,\mathbf{f}_3))^2 = 0,$$

wobei $\det(\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3)$ die $3\times 3-$Determinate mit den Spaltenvektoren $\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3 \in \mathbb{R}^3$ ist.

9.2.1 Ebene Schnitte des Einheits-Hyperboloids

Wir bestimmen zunächst die Schnitte des "Einheits"-Hyperboloids

$$H_1 := \{(x,y,z) \in \mathbb{R} \mid -x^2 - y^2 + z^2 = 1\}$$

mit der Ebene $\varepsilon : ax + cz = d$.
Mit überlegungen, wie wir sie für einen Kegel durchgeführt haben, erhält man das folgende Ergebnis:
Fall I: Für $c^2 - a^2 > d^2$ und $c^2 - a^2 = d^2 = 0$ ist der Schnitt **leer**.
Fall II: Für $c^2 - a^2 = d^2 > 0$ besteht der Schnitt aus dem **Punkt** $P = (-a/d, 0, c/d)$.
Fall III: Für $0 \neq c^2 - a^2 < d^2$ ist die Schnittkurve eine **Ellipse**, falls $0 < c^2 - a^2$ bzw. eine **Hyperbel**, falls $c^2 - a^2 < 0$ ist. $P_0 = (\frac{-ad}{c^2-a^2}, 0, \frac{cd}{c^2-a^2})$ ist der Mittelpunkt,

$$\begin{aligned}
P_1 &= (-ad + c\tfrac{\sqrt{|c^2-a^2-d^2|}}{c^2-a^2}, 0, dc - a\tfrac{\sqrt{|c^2-a^2-d^2|}}{c^2-a^2}) \\
P_1' &= (-ad - c\tfrac{\sqrt{|c^2-a^2-d^2|}}{c^2-a^2}, 0, dc + a\tfrac{\sqrt{|c^2-a^2-d^2|}}{c^2-a^2}) \\
P_2 &= (\tfrac{-ad}{c^2-a^2}, \sqrt{\tfrac{|c^2-a^2-d^2|}{|c^2-a^2|}}, \tfrac{cd}{c^2-a^2}) \\
P_2' &= (\tfrac{-ad}{c^2-a^2}, -\sqrt{\tfrac{|c^2-a^2-d^2|}{|c^2-a^2|}}, \tfrac{cd}{c^2-a^2})
\end{aligned}$$

sind die Scheitel bzw. Nebenscheitel.

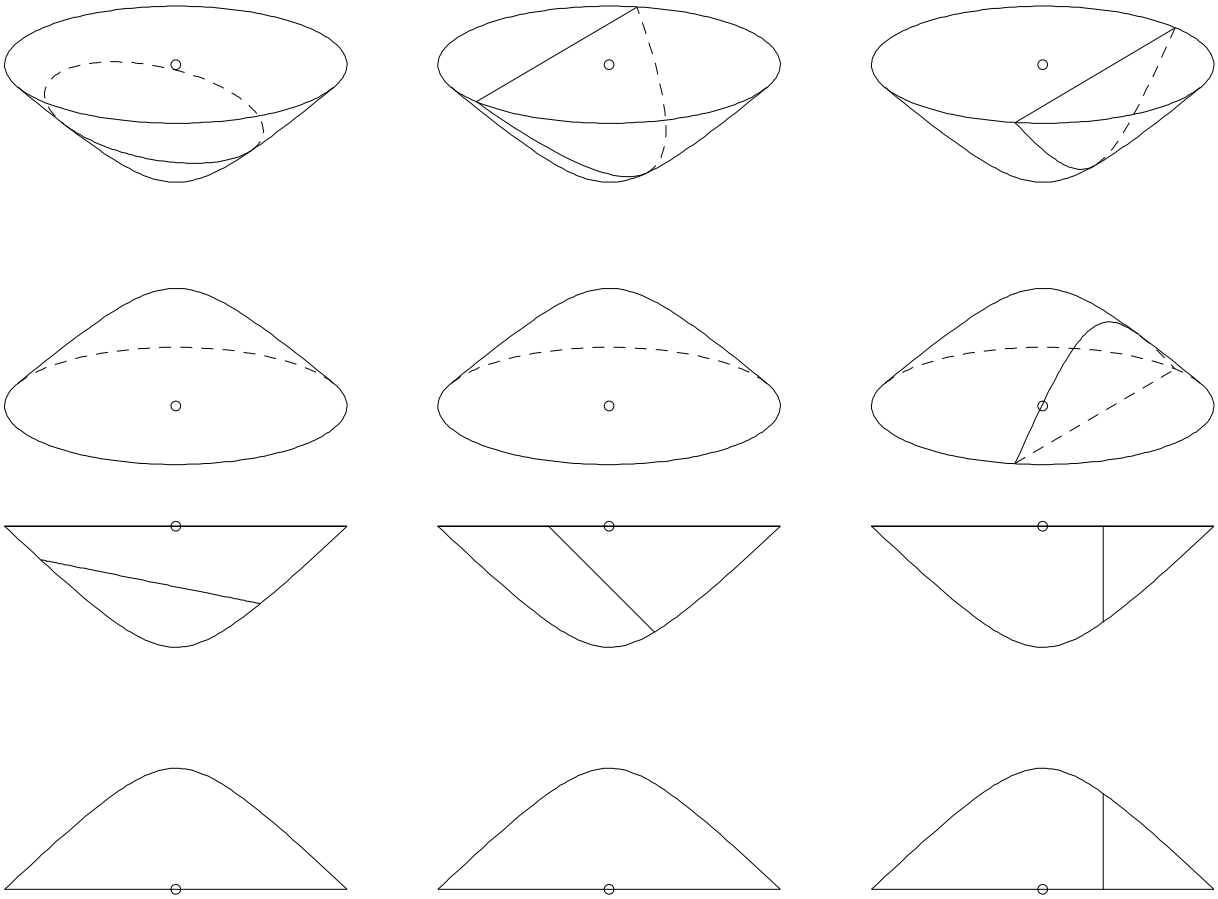


Abbildung 9.6: ebene Schnitte eines zweischaligen Hyperboloids

Fall IV: Für $c^2 - a^2 = 0$ und $d \neq 0$ ist die Schnittkurve eine **Parabel**.

$$\begin{array}{lcll} P_0 & = & (\frac{d^2-a^2}{2ad}, 0, \frac{d^2+c^2}{2cd}) & (Scheitel) \\ P_1 & = & (\frac{d^2-a^2}{2ad}, \sqrt{1+(\frac{d}{a})^2}, \frac{d^2+c^2}{2cd}) & (s.Kap.5) \\ P_2 & = & (\frac{-a}{d}, 0, \frac{c}{d}+\frac{d}{c}) & (s.Kap.5). \end{array}$$

Den Schnitt eines 2-schaligen Hyperboloids mit einer beliebigen Ebene berechnet man wie im Fall des Kegels.

9.2.2 Ebene Schnitte eines 2-schaligen Hyperboloids

Gegeben:

a) Ebene $\varepsilon : \mathbf{n} \cdot \mathbf{x} = d$,

b) Hyperboloid $H : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \qquad -\xi^2 - \eta^2 + \zeta^2 = 1.$

Gesucht: Schnitt $\varepsilon \cap H$.

Durch Einsetzen von b) in die Gleichung der Ebene erhält man das Gleichungssystem:

$$\begin{array}{rcl} \gamma_1\xi + \gamma_2\eta + \gamma_3\zeta & = & \gamma_0, \quad \text{mit} \quad \gamma_i = \mathbf{n}\cdot\mathbf{f}_i, \quad i = 1,2,3, \quad \gamma_0 = d - \mathbf{n}\cdot\mathbf{q}_0, \\ -\xi^2 - \eta^2 + \zeta^2 & = & 1 \end{array}$$

Dieses Gleichungssystem beschreibt den Schnitt des zweischaligen Einheitshyperboloids mit einer Ebene (s.o.). Man erhält auch hier zunächst die Schnittdaten bzgl. des $\xi-\eta-\zeta-$Koordinatensystems. Durch Einsetzen in $\mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta$ ergeben sich schließlich die zugehörigen $x - y - z-$Koordinaten.

9.2.3 Parallelprojektion eines 2-schaligen Hyperboloids

Wir bestimmen zunächst Umrißpunkte des Hyperboloids

$$\begin{array}{rcl} H & = & \{\mathbf{q}_0 + \mathbf{f}_1 \sinh\beta\cos\alpha + \mathbf{f}_2 \sinh\beta\sin\alpha \pm \mathbf{f}_3\cosh\beta | \ldots\}. \\ & = & \{\mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta \mid -\xi^2 - \eta^2 + \zeta^2 = 1\} \end{array}$$

Analog zu den Überlegungen für das Ellipsoid ergibt sich hier:
Eine Normale im Punkt

$$Q : \mathbf{q}(\alpha,\beta) \quad \text{ist} \quad \mathbf{n}(\alpha,\beta) = \mathbf{f}_2 \times \mathbf{f}_3 \sinh\beta\cos\alpha + \mathbf{f}_3 \times \mathbf{f}_1 \sinh\beta\sin\alpha - \mathbf{f}_1 \times \mathbf{f}_2 \cosh\beta.$$

Wenn wir von Selbstüberdeckungen (ein Hyperboloid ist nicht konvex!) absehen, ist Q ein Umrißpunkt, falls $\mathbf{n}(\alpha,\beta)$ auf der (negativen) Projektionrichtung $\mathbf{n}_0$ senkrecht steht. In $\xi-\eta-\zeta-$Koordinaten liefert die Bedingung $\mathbf{n}_0 \cdot \mathbf{n}(\alpha,\beta) = 0$ das Gleichungssystem:

$$\begin{array}{rcl} \gamma_1\xi + \gamma_2\eta - \gamma_3\zeta & = & 0 \quad \text{mit} \qquad \gamma_1 = \mathbf{n}_0\cdot(\mathbf{f}_2\times\mathbf{f}_3), \qquad \gamma_2 = \mathbf{n}_0\cdot(\mathbf{f}_3\times\mathbf{f}_1), \qquad \gamma_3 = \mathbf{n}_0\cdot(\mathbf{f}_1\times\mathbf{f}_2), \\ -\xi^2 - \eta^2 + \zeta^2 & = & 1 \end{array}$$

Dieses Gleichungssystem löst man analog zum Fall des einschaligen Hyperboloids.

Aufgabe 9.2 *Schreibe ein Programm das ein zweischaliges Rotations-Hyperboloid mit Höhenkreisen und Hyperbeln darstellt.*

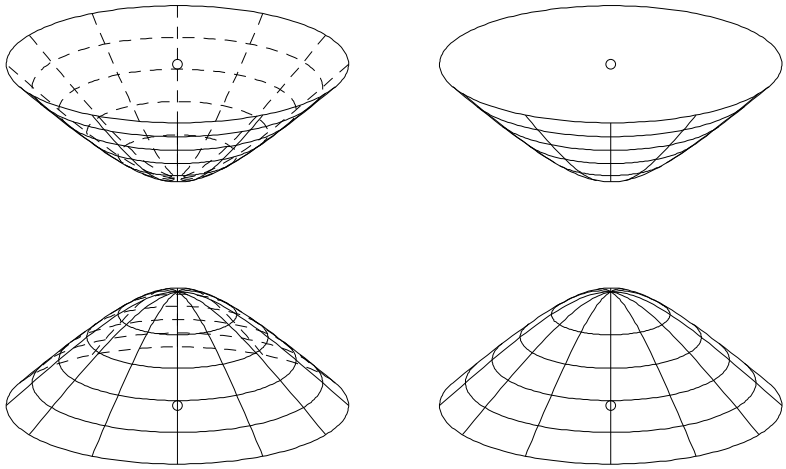


Abbildung 9.7: zur Aufgabe 9.2

Kapitel 10

HIDDENLINE-ALGORITHMUS MIT SEHSTRAHLEN

Will man mehrere sich z.T. verdeckende Quadriken gleichzeitig darstellen, so kann man das dabei auftretende Hiddenline-Problem lösen, indem man für jeden Punkt einer darzustellenden Kurve den zugehörigen Projektionsstrahl, kurz **Sehstrahl**, untersucht, ob er ein weiteres Objekt davor trifft. Für Quadriken ist dieses Schnittproblem Sehstrahl-Quadrik relativ leicht zu lösen, da man hierfür nur eine quadratische Gleichung lösen muß. Für Flächen höherer Ordnung, wie z. B. den Torus, muß man sich etwas anderes einfallen lassen (s. u.). Wir wollen hier zunächst den Sehstrahltest für die Quadriken Ellipsoid, Zylinder und Kegel besprechen. Der Einfachheit halber setzen wir allerdings voraus, daß Ellipsoid, Zylinder und Kegel in "Hauptachsenform" (s.u.) vorliegen.

10.1 Sehstrahltest für ein Ellipsoid

Gegeben: Ellipsoid $E : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1 \cos\beta\cos\alpha + \mathbf{f}_2 \cos\beta\sin\alpha + \mathbf{f}_3 \sin\beta$,
(vgl. Kap. 7). $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ sei eine **Orthogonal**basis.
Gerade $g : \mathbf{x} = \mathbf{p} + t\mathbf{v}$
(für den Sehstrahltest ist $\mathbf{v}$ die negative Projektionsrichtung $\mathbf{n}_0$).

Der Parameter t und die Koordinaten ξ, η, ζ eines Punktes des Schnittes $E \cap g$ bezüglich der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ erfüllen das folgende Gleichungssystem:

$$\begin{array}{lrcl} (1) & \mathbf{p} + t\mathbf{v} & = & \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta \\ (2) & 1 & = & \xi^2 + \eta^2 + \zeta^2 \end{array}$$

Durch skalare Multiplikation der ersten Gleichung mit $\mathbf{f}_1$ bzw. $\mathbf{f}_2$ bzw. $\mathbf{f}_3$ erhält man

$$\xi = \varepsilon_1 + t\delta_1, \qquad \eta = \varepsilon_2 + t\delta_2, \qquad \zeta = \varepsilon_3 + t\delta_3$$

mit

$$\varepsilon_i := (\mathbf{p} - \mathbf{q}_0) \cdot \mathbf{f}_i / \mathbf{f}_i^2, \qquad \delta_i := \mathbf{v} \cdot \mathbf{f}_i / \mathbf{f}_i^2, \qquad i = 1, 2, 3.$$

Mit Gleichung (2) folgt die quadratische Gleichung

$$t^2(\delta_1^2 + \delta_2^2 + \delta_3^2) + 2t(\varepsilon_1\delta_1 + \varepsilon_2\delta_2 + \varepsilon_3\delta_3) + \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 - 1 = 0$$

Das Unterprogramm `equation_degree2` liefert die möglichen reellen Lösungen t_1, t_2. Die Schnittpunkte sind dann $\mathbf{p} + t_1\mathbf{v}$ und $\mathbf{p} + t_2\mathbf{v}$.

Bei der Untersuchung, ob Punkte einer Kurve durch ein Ellipsoid verdeckt werden, werden immer wieder die Vektoren $\mathbf{f}_i/\mathbf{f}_i^2$, die nur vom Ellipsoid abhängen, benutzt. Für einen effektiven Hiddenline-Algorithmus ist es zweckmäßig diese Vektoren nur einmal in einer Hilfsprozedur zu berechnen. Dies geschieht in `aux_hidden_by_ellipsoid`. Die eigentliche Sichtbarkeitsuntersuchung findet in `pt_hidden_by_ellipsoid`. Bei Parallelprojektion kann man auch die Berechnung der Zahlen δ_i in der Hilfsprozedur durchführen. Bei Zentralprojektion hängt die Richtung `ray_vt` des Sehstrahls vom Punkt $\mathbf{p}$ ab.

```
procedure aux_hidden_by_ellipsoid(q0,q1,q2,q3 : vt3d; var ff1,ff2,ff3: vt3d);
 {Berechnet die Vektoren ffi:= fi/scalarp3d(fi,fi), i=1,2,3.}
 var f1,f2,f3 : vt3d;    f12,f22,f32 : real;
 begin
   diff3d(q1,q0, f1);         diff3d(q2,q0, f2);          diff3d(q3,q0, f3);
   f12:= scalarp3d(f1,f1);   f22:= scalarp3d(f2,f2);   f32:= scalarp3d(f3,f3);
   scale3d(1/f12,f1, ff1);   scale3d(1/f22,f2, ff2);   scale3d(1/f32,f3, ff3);
 end;   { aux_hidden_by_ellipsoid }
{**************}
function pt_hidden_by_ellipsoid(p,ray_vt,q0,ff1,ff2,ff3: vt3d): boolean;
 var a,b,c,ep1,ep2,ep3,de1,de2,de3,t1,t2 : real;   p0 : vt3d;  ns : integer;
 begin
   pt_hidden_by_ellipsoid:= false;       diff3d(p,q0, p0);
   ep1:= scalarp3d(p0,ff1);   de1:= scalarp3d(ray_vt,ff1);
   ep2:= scalarp3d(p0,ff2);   de2:= scalarp3d(ray_vt,ff2);
   ep3:= scalarp3d(p0,ff3);   de3:= scalarp3d(ray_vt,ff3);
   a:=   sqr(de1) + sqr(de2) + sqr(de3);
   b:= 2*(ep1*de1 + ep2*de2  + ep3*de3);
   c:=   sqr(ep1) + sqr(ep2) + sqr(ep3) -1;
   equation_degree2(a,b,c,t1,t2,ns);
   if ns=2 then if t2>eps5 then pt_hidden_by_ellipsoid:= true;
 end;   { pt_hidden_by_ellipsoid }
```

Aufgabe 10.1 *Stelle eine Kugel vor einem Kegel dar.*

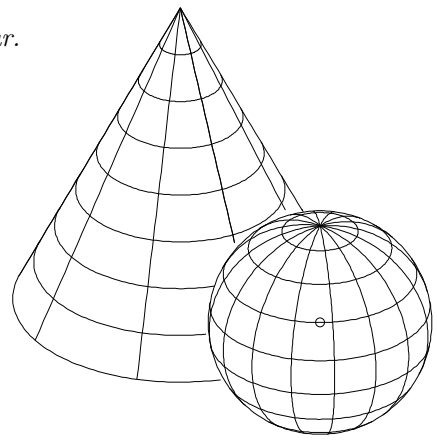


Abbildung 10.1: Kugel vor einem Kegel

10.2 Sehstrahltest für einen Zylinder

Gegeben: Zylinder $Z : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi + \mathbf{f}_3\alpha$, (vgl. Kap. 8) .
$\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ sei eine **Orthogonal**basis.

Gerade $g : \mathbf{x} = \mathbf{p} + t\mathbf{v}$.

Der Parameter t und die $\xi - \eta - \zeta-$Koordinaten eines Punktes des Schnittes $Z \cap g$ bezüglich der Basis $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ erfüllen das folgende Gleichungssystem:

$$\begin{array}{lrcl} (1) & \mathbf{p} + t\mathbf{v} & = & \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\alpha, \\ (2) & 1 & = & \xi^2 + \eta^2. \end{array}$$

Durch skalare Multiplikation der ersten Gleichung mit $\mathbf{f}_1$ bzw. $\mathbf{f}_2$ erhält man

$$\begin{array}{llll} \xi = \varepsilon_1 + t\delta_1, & \eta = \varepsilon_2 + t\delta_2, & \text{mit} \\ \varepsilon_i := (\mathbf{p} - \mathbf{q}_0) \cdot \mathbf{f}_i / \mathbf{f}_i^2, & \delta_i := \mathbf{v} \cdot \mathbf{f}_i / \mathbf{f_i^2}, & i = 1, 2. \end{array}$$

Mit Gleichung (2) folgt die quadratische Gleichung

$$t^2(\delta_1^2 + \delta_2^2) + 2t(\varepsilon_1\delta_1 + \varepsilon_2\delta_2) + \varepsilon_1^2 + \varepsilon_2^2 - 1 = 0$$

Das Unterprogramm `equation_degree2` liefert die möglichen reellen Lösungen t_1, t_2. Die Schnittpunkte sind dann $\mathbf{p} + t_1\mathbf{v}$ und $\mathbf{p} + t_2\mathbf{v}$.

Die Unterprogramme für den Sehstrahltest mit einem Zylinder *endlicher* Höhe $h = \|\mathbf{f}_3\|$, d.h. mit $0 \le \alpha \le 1$, sind

```
procedure aux_hidden_by_cylinder(q0,q1,q2 : vt3d; h: real; var ff1,ff2,ff3: vt3d);
 {Berechnet die Hilfsvektoren ff1,ff2,ff3 fuer senkrechten elliptischen Zylinder (Hoehe h)}
{*************}
function pt_hidden_by_cylinder(p,ray_vt,q0,ff1,ff2,ff3: vt3d): boolean;
```

Aufgabe 10.2 *Stelle einen Zylinder vor einem Kegel dar.*

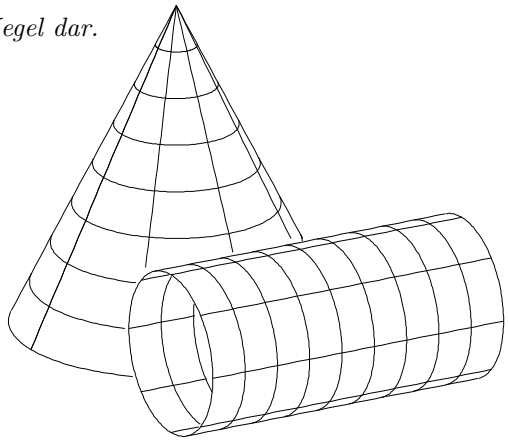


Abbildung 10.2: Zylinder vor einem Kegel

10.3 Sehstrahltest für einen Kegel

Gegeben: Kegel $K : \mathbf{x} = \mathbf{q}_0 + \mathbf{f}_1\alpha\cos\varphi + \mathbf{f}_2\alpha\sin\varphi + \mathbf{f}_3(1-\alpha)$, (vgl. Kap. 8).
$\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ sei eine **Orthogonal**basis.
Gerade $g : \mathbf{x} = \mathbf{p} + t\mathbf{v}$.
Der Parameter t und $\xi := \alpha\cos\varphi, \quad \eta := \alpha\sin\varphi, \quad \zeta := 1 - \alpha$ eines Punktes des Schnittes $K \cap g$ erfüllen das folgende Gleichungssystem:

$$\begin{array}{lrcl} (1) & \mathbf{p} + t\mathbf{v} & = & \mathbf{q}_0 + \mathbf{f}_1\xi + \mathbf{f}_2\eta + \mathbf{f}_3\zeta, \\ (2) & \xi^2 + \eta^2 & = & (\zeta - 1)^2 \end{array}$$

Durch skalare Multiplikation der ersten Gleichung mit $\mathbf{f}_1$ bzw. $\mathbf{f}_2$ bzw. $\mathbf{f}_3$ erhält man

$$\begin{array}{lll} \xi = \varepsilon_1 + t\delta_1, & \eta = \varepsilon_2 + t\delta_2, & \zeta - 1 = \varepsilon_3 + t\delta_3 \quad \text{mit} \\ \varepsilon_i := (\mathbf{p} - \mathbf{q}_0) \cdot \mathbf{f}_i / \mathbf{f}_i^2,\ i = 1, 2, & \delta_i := \mathbf{v} \cdot \mathbf{f}_i / \mathbf{f}_i^2,\ i = 1, 2, 3, & \text{und} \\ \varepsilon_3 := (\mathbf{p} - \mathbf{q}_0) \cdot \mathbf{f}_3 / \mathbf{f}_3^2 - 1, & & \end{array}$$

Aus $\xi^2 + \eta^2 = (\zeta - 1)^2$ folgt die quadratische Gleichung

$$t^2(\delta_1^2 + \delta_2^2 - \delta_3^2) + 2t(\varepsilon_1\delta_1 + \varepsilon_2\delta_2 - \varepsilon_3\delta_3) + \varepsilon_1^2 + \varepsilon_2^2 - \varepsilon_3^2 = 0$$

Das Unterprogramm `equation_degree2` liefert die möglichen reellen Lösungen t_1, t_2. Die Schnittpunkte sind dann $\mathbf{p} + t_1\mathbf{v}$ und $\mathbf{p} + t_2\mathbf{v}$.

Die Unterprogramme für den Sehstrahltest mit einem Kegel der Höhe $h = \|\mathbf{f}_3\|$, d.h. mit $\alpha \leq 1$, sind

```
procedure aux_hidden_by_cone(q0,q1,q2: vt3d; h: real; var ff1,ff2,ff3: vt3d);
 {Berechnet die Hilfsvektoren ff1,ff2,ff3 fuer senkrechten elliptischen Kegel (Hoehe h)}
{***********}
function pt_hidden_by_cone(p,ray_vt,q0,ff1,ff2,ff3: vt3d): boolean;
```

Aufgabe 10.3 *Stelle einen Kegel vor einem Zylinder dar.*

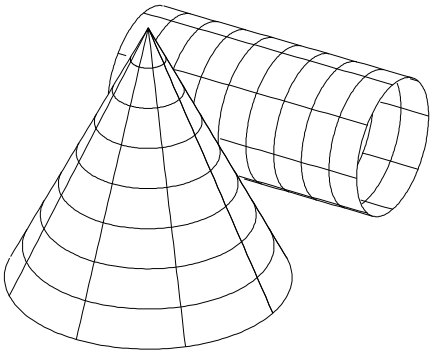


Abbildung 10.3: Kegel vor einem Zylinder

10.4 Sehstrahltest für implizite Flächen, Beispiel Torus

Bei allgemeineren Flächen bestimmt man die Schnittpunkte des Sehstrahls mit der/den Fläche(n) nicht exakt, sonderen es genügt die Existenz von Schnittpunkten mit Hilfe geeigneter Kriterien (Vorzeichenwechsel einer Funktion,...) nachzuweisen oder innerhalb von Fehlerschranken auszuschließen. Da wir es immer mit räumlich begrenzten Objekten zu tun haben, kann man die Suche nach Schnittpunkten auf ein endliches Intervall des Sehstrahls beschränken. Im Folgenden wird das Prinzip der Sehstrahlmethode am Beispiel des Torus erläutert. Diese Methode ist auf Flächen anwendbar, denen man mit Hilfe eines Kriteriums zwei "Seiten", z. B. innen-außen oder oben-unten, zuordnen kann.

Gegeben: Torus $\tau : f(x, y, z) = (x^2 + y^2 + z^2 + r^2 - a^2)^2 - 4r^2(x^2 + y^2) = 0$,
Punkt $P : \mathbf{p}$
Gesucht: Entscheidung, ob P durch den Torus verdeckt wird oder nicht.

Durchführung des Tests:

(0) Auswahl einer den Torus umgebenden einfachen Fläche. Wir wählen hier den Rand des Kugelteils σ

$$x^2 + y^2 + z^2 = (r + a + \varepsilon)^2,\ |z| \leq a + \varepsilon \text{ mit } \varepsilon > 0.$$

(1) Bestimmung des Parameters t_0 >0 des Schnittpunktes des Sehstrahls $\mathbf{x} = \mathbf{p} + t_0\mathbf{n}_0, wobei \mathbf{n}_0$ die negative Projektionsrichtung ist, mit dem Kugelteil σ. (Die Parameter möglicher Schnittpunkte des Sehstrahls mit dem Torus müssen kleiner als t_0 sein.)

(2) Als Kriterium, ob ein Punkt innerhalb oder außerhalb des Torus liegt, nehmen wir das Vorzeichen der Funktion f an der Stelle des Testpunktes.

(3) Parameter von Testpunkten auf dem Sehstrahlabschnitt mit $0 < t < t_0$:
Beim 1-ten Durchgang nehmen wir $t_0/2$.
Beim n-ten Durchgang liegen die Parameter im Abstand $t_0/2^n$.
Es werden nur neu hinzukommende Testpunkte getestet.

(4) Der Test bricht ab, sobald

a) ein Seitenwechsel festgestellt wird oder

b) der Parameterabstand eine Schranke unterschreitet.
Im Fall a) ist der Punkt P verdeckt,
im Fall b) wird der Punkt P als sichtbar erklärt.

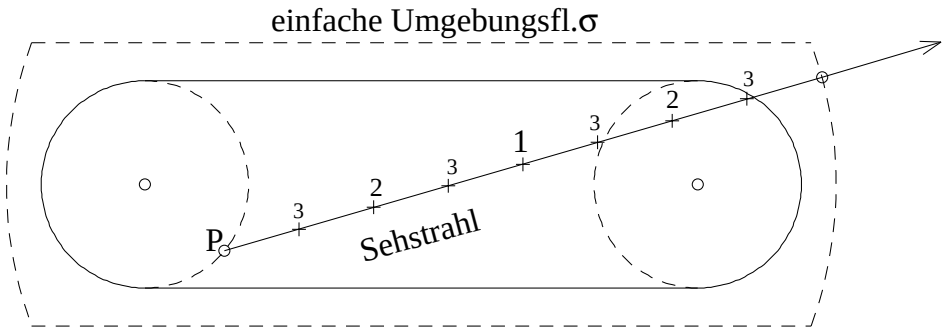


Abbildung 10.4: Sehstrahltest für einen Torus

Nach dem Sichtbarkeitstest der Punkte einer Kurve werden benachbarte sichtbare Punkte projiziert und durch eine Strecke verbunden. (Zur Bestimmung des Umrisses eines Torus s. nächstes Kapitel)

```
procedure vis_pts_by_torus(p: vts3d; n1,n2: integer; var vis: b_array);
 {Setzt vis[j]:= true, wenn der Punkt p[j] vom Torus nicht verdeckt wird.}
 var stepmin,step1,step2,t0,tt : real;  j,s0,stt : integer;    pj : vt3d;
 {***}
function fvalue(p: vt3d) : real;
begin
  fvalue:= sqr(sqr(p.x)+sqr(p.y)+sqr(p.z)+r2-a2) - 4*r2*(sqr(p.x)+sqr(p.y));
end;
 {***}
function max_param(p: vt3d) : real;
 {Berechnet den maximalen Parameter auf dem Sehstrahl eines Punktes.
  Hier: Flaeche ist TORUS, Ersatzflaeche ist Kugelteil um den Torus.}
 var rsrs2,c1,c2,z0,t0 : real;
```

```
 begin
   rs := r+a+1;   {Radius der den Torus einhuellenden Kugel}
   rs2:= sqr(rs);
   c1 := scalarp3d(p,n0vt);     c2:= length3d(p);
   t0 := -c1 + sqrt(abs(c1*c1-c2+rs2));   {Parameter des Schnittp. mit der Kugel}
   z0 := p.z + t0*n0vt.z;
   if z0>a+1 then  t0:= (a+1-p.z)/n0vt.z;     {Schnittp. mit der Ebene z=a+1}
   max_param:= t0;
 end;   { max_param }
 {***}
function side_of_surface(p: vt3d; t: real) : integer;
 var pt : vt3d; fxyz : real;
 begin
   lcomb2vt3d(1,p, t,n0vt, pt);
   fxyz:= fvalue(pt);
   if fxyz>0 then side_of_surface:= 1 else side_of_surface:= -1;
 end;   { side_of_surface }
 {***}
 begin    {vis_pts_by_torus}
   stepmin:= 3;        {minimale Schrittweite}
   for j:= n1 to n2 do
       begin
       pj:= p[j];
       t0:= max_param(pj);
       s0:= side_of_surface(pj,t0);      {Testpunkt: pj + t0*n0vt}
       step1:= t0;   step2:= 0.5*step1;
       repeat
         tt:= t0-step2;
         repeat
           stt:= side_of_surface(pj,tt);
           tt := tt - step1;
         until (stt<>s0) or (tt<0) ;
         step1:= step2;  step2:= 0.5*step1;
       until (stt<>s0) or (step2<stepmin);
       if (stt=s0)  then  vis[j] := true else vis[j] := false;
       end;    {for}
end;    {vis_pts_by_torus}
{******************}
```

Bemerkung: Zur Beschleunigung des Sichtbarkeitstests kann man schnellere Tests, wie z. B. den Normalentest, der Sehstrahlmethode vorschalten.

Aufgabe: s. nächstes Kapitel.

Kapitel 11

ROTATIONSFLÄCHEN UND SCHRAUBFLÄCHEN

11.1 Rotationsflächen

Wie wir gesehen haben, lassen sich Kegel, Ellipsoid, Hyperboloid ... relativ leicht darstellen, da ihre Umrisse (bis auf Randkurven) immer ebene Schnitte d.h. Ellipsen, Hyperbeln, Parabeln oder Strecken sind. Dies gilt i. a. nicht mehr, wie das folgende Beispiel zeigt:

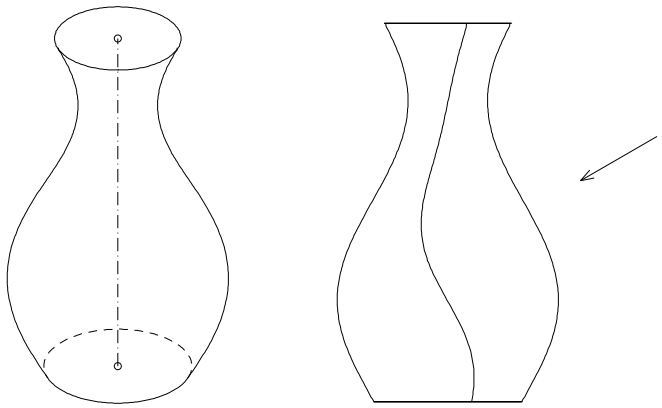


Abbildung 11.1: Umrißkurve einer Rotationsfläche

(Die rechte Zeichnung zeigt die Umrißlinie, die bei der Parallelprojektion links entsteht.)

In diesem Kapitel wollen wir zeigen, wie man für die 4 wichtigsten Typen von Rotationsflächen potentielle Umrißpunkte bestimmt.
Eine Rotationsfläche Φ entsteht durch Rotation einer Kurve Γ um eine Gerade a, der Rotationsachse. Eine Rotationsfläche trägt zwei ausgezeichnete Kurvenscharen: die Breitenkreise (ebene Schnitte $\perp$ zur Achse a) und die Meridiane (ebene Schnitte durch a). Entsteht eine Rotationsfläche Φ durch Rotation eines Meridians μ_0, so heißt μ_0 der Hauptmeridian von Φ. Im Folgenden werden wir von einem Meridian in der $r-z-$Ebene bezüglich Zylinderkoordinaten ausgehen und 4 Typen unterscheiden:

1) $\mu_0 : r = f(z)$, 2) $\mu_0 : z = f(r)$, 3) $\mu_0 : f(r,z) = 0$, 4) $\mu_0 : r = r(\beta), z = z(\beta)$

Die dabei auftretenden Funktionen seien stets differenzierbar.

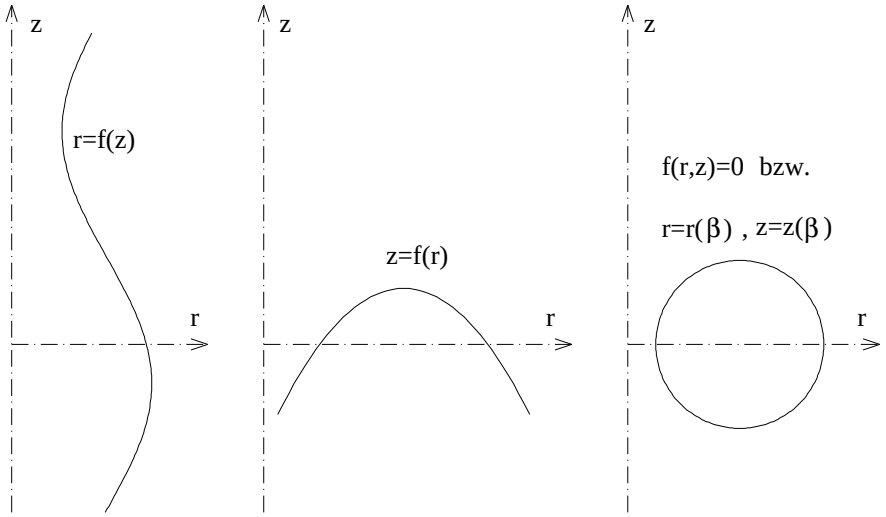


Abbildung 11.2: Typen von Hauptmeridianen

11.1.1 Meridian $r = f(z)$

In $x-y-z-$Koordinaten läßt sich die Rotationsfläche durch

$$F(x,y,z) := x^2 + y^2 - f(z)^2 = 0$$

beschreiben. Eine Normale in einem Punkt der Rotationsfläche Φ erhält man aus $\nabla F(x,y,z) = (2x, 2y, -2f'(z)f(z)) = 2r(\cos t, \sin t, -f'(z))$, wobei $x = r\cos t, y = r\sin t$ (Zylinderkoordinaten!) gesetzt wurde. Für Punkte, die nicht auf der Rotationsachse liegen, können wir also als Normale $\mathbf{n}(z,t) = (\cos t, \sin t, -f'(z))$ wählen.
Ein Punkt $Q : \mathbf{q}(z,t) = (f(z)\cos t, f(z)\sin t, z)$ ist ein potentieller Umrißpunkt bei **Parallelprojektion**, wenn die (negative) Projektionsrichtung $\mathbf{n}_0$ und $\mathbf{n}(z,t)$ senkrecht zu einander stehen:

$$\mathbf{n}_0 \cdot \mathbf{n}(z,t) = n_{0x}\cos t + n_{0y}\sin t - n_{0z}f'(z) = 0.$$

Diese Gleichung ist äquivalent zu dem System

$$n_{0x}\xi + n_{0y}\eta = n_{0z}f'(z), \quad \xi^2 + \eta^2 = 1.$$

Für festes $z = z_0$ lassen sich mit Hilfe von `is_unitcircle_line` mögliche Lösungen $(\xi_1, \eta_1), (\xi_2, \eta_2)$ bestimmen. Mit $\mathbf{x}_i = (f(z_0)\xi_i, f(z_0)\eta_i, z_0), i = 1, 2$ ergeben sich dann die zugehörigen Umrißpunkte des Breitenkreises mit $z = z_0$.
Ist von μ_0 nur die Punktreihe $(r(1), z(1)), \ldots, (r(n), z(n))$ bekannt, so kann man $f'(z(i))$ durch $(r(i+1) - r(i))/(z(i+1) - z(i))$ annähern.

Das folgende Unterprogramm `pp_surface_rev0` berechnet nach dieser Methode den Umriß einer Rotationsfläche und projiziert diesen. Man beachte, daß dieses Unterprogramm nur dann korrekt arbeitet, wenn es auf jedem Höhenkreis Umrißpunkte gibt.

```
procedure pp_surface_rev0(rm,zm: r_array; n2,style : integer);
{Berechnet Umrisspunkte einer Rotationsflaeche mit den Meridiankurvenpunkten
 pm[0],...,pm[n2]) , und projiziert den Umriss (Polygon). Unsichtbare Kurven
 werden nicht (style=0) oder gestrichelt (style=10) gezeichnet.}
 var p1,p2 : vts3d;      pc0,pc1,pc2 : vt3d;
     i,nis : integer;    ci,cw1,sw1,cw2,sw2,ri : real;
 begin
   if abs(cos_v) > eps6 then
      begin
      for i:= 0 to n2-1 do
          begin
          ci:= (rm[i+1]-rm[i])/(zm[i+1]-zm[i]);
          ri:= rm[i];
          is_unitcircle_line(n0vt.x,n0vt.y,n0vt.z*ci, cw1,sw1,cw2,sw2,nis);
          put3d(ri*cw1,ri*sw1,zm[i], p1[i]);
          put3d(ri*cw2,ri*sw2,zm[i], p2[i]);
          end; { for i }
      p1[n2]:= p1[n2-1];      p1[n2].z:= zm[n2];
      p2[n2]:= p2[n2-1];      p2[n2].z:= zm[n2];
      pp_curve(p1,0,n2,0);    pp_curve(p2,0,n2,0);
{ Deckel- und Bodenkreise : }
      put3d(   0, 0,zm[0], pc0);
      put3d(rm[0],0,zm[0], pc1);
      put3d(0,rm[0],zm[0], pc2);
      if v_angle>0 then pp_ellipse_arc(pc0,pc1,pc2,p2[0],p1[0],style)
                   else pp_ellipse(pc0,pc1,pc2,style);
      put3d(    0, 0,zm[n2], pc0);
      put3d(rm[n2],0,zm[n2], pc1);
      put3d(0,rm[n2],zm[n2], pc2);
      if v_angle<0 then pp_ellipse_arc(pc0,pc1,pc2,p2[n2],p1[n2],style)
                   else pp_ellipse(pc0,pc1,pc2,style);
      end; { if }
 end;  { pp_surface_rev0 }
{***************}
```

Dieses Unterprogramm ist in `proc_prt.pas` enthalten.

Als **Testfunktion** bei der **Sehstrahlmethode** kann man

$$F(x,y,z) := x^2 + y^2 - f(z)^2$$

verwenden. Falls μ_0 nur als Polygonzug (s.o.) gegeben ist, kann man als Testfunktion für einen Testpunkt $T = (x,y,z)$

$$F(x,y) := x^2 + y^2 - r_i^2$$

wählen, wobei i der Bedingung $z_i \le z < z_{i+1}$ genügt.

Aufgabe 11.1 *Vase 1 Meridian* $r(z) = a\cos(2\pi(z-b)/v_l) + c$ *(s.o.)*

Aufgabe 11.2 *Faß. Der Hauptmeridian ist ein Kreisbogen.*

Aufgabe 11.3 *Vase 2. Der Meridian ist eine BEZIER-Kurve (s. Kap. 6).*

Bei **Zentralprojektion** lautet für die Rotationsfläche mit Hauptmeridian $r = f(z)$ die Umrißbedingung: $z_x\xi + z_y\eta = f(z) + (z_z - z)f'(z), \qquad \xi^2 + \eta^2 = 1,$
wobei z_x, z_y, z_z die Koordinaten des Projektionszentrums sind.

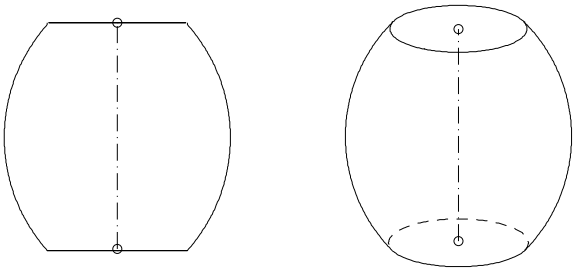


Abbildung 11.3: Faß

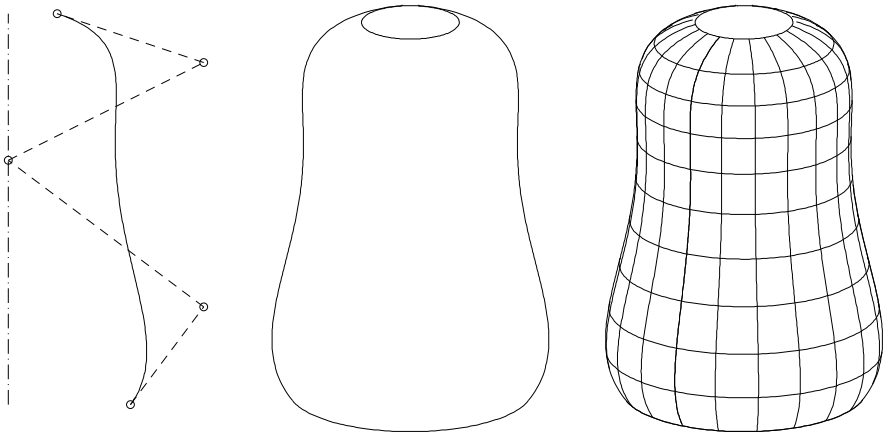


Abbildung 11.4: Rotationsfläche mit Bézierkurve als Meridian

11.1.2 Meridian $z = f(r)$

In $x-y-z-$Koordinaten läßt sich die Rotationsfläche durch

$$F(x,y,z) := f(\sqrt{x^2+y^2}) - z = 0$$

beschreiben. Eine Normale in einem Punkt der Rotationsfläche erhält man aus

$$\nabla F(x,y,z) = (f'(r)x/r, f'(r)y/r, -1), \text{ wobei } r = \sqrt{x^2+y^2}$$

ist. Eine Normale im Punkt $\mathbf{q}(r,t) = (r\cos t, r\sin t, f(r))$ ist also

$$\mathbf{n}(r,t) = (f'(r)\cos t, f'(r)\sin t, -1). \qquad \text{(Zylinderkoordinaten!)}$$

Ein Punkt $\mathbf{q}(r,t)$ ist ein potentieller Umrißpunkt bei **Parallelprojektion**, wenn

$$\mathbf{n}_0 \cdot \mathbf{n}(r,t) = n_{0x} f'(r)\cos t + n_{0y} f'(r)\sin t - n_{0z} = 0$$

ist. Diese Gleichung ist äquivalent zu dem System

$$n_{0x} f'(r)\xi + n_{0y} f'(r)\eta = n_{0z}, \quad \xi^2 + \eta^2 = 1.$$

Für festes $r = r_0$ lassen sich mit Hilfe von `is_unitcircle_line` mögliche Lösungen $(\xi_1, \eta_1), (\xi_2, \eta_2)$ bestimmen. Mit $\mathbf{x}_i = (r_0\xi_i, r_0\eta_i, f(r_0)),\ i = 1, 2$ ergeben sich dann die zugehörigen Umrißpunkte des Breitenkreises mit $r = r_0$ und $z = f(r_0)$.

Als **Testfunktion** für die **Sehstrahlmethode** (vgl. Kap. 10) kann man

$$F(x, y, z) := f(\sqrt{x^2 + y^2}) - z$$

verwenden.

Aufgabe 11.4 *Es sei* $0 < a < R, 0 < z_0$ *und der Meridian:*
$z = f(r) = z_0 - z_0(r - R)^2/a^2, R - a < r < r + a$, *(Der Meridian ist eine Parabel.)*

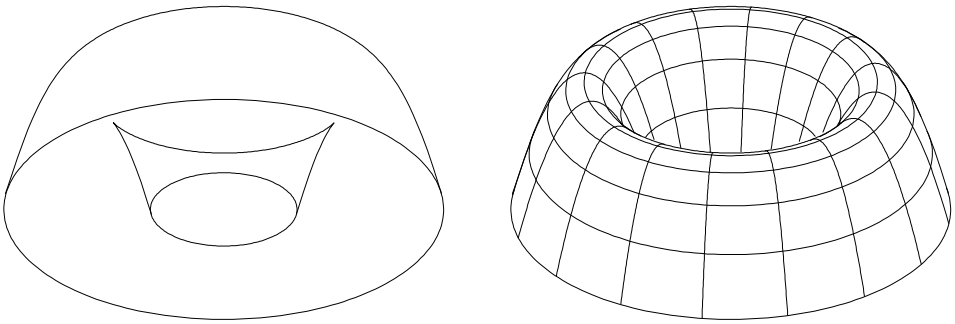


Abbildung 11.5: Rotationsfläche mit Parabelbogen als Meridian

Bei **Zentralprojektion** lautet für die Rotationsfläche mit Hauptmeridian $z = f(r)$ die Umrißbedingung: $z_x f'(r)\xi + z_y f'(r)\eta = z_z + rf'(r), \qquad \xi^2 + \eta^2 = 1.$

11.1.3 Meridian $f(r, z) = 0$

In $x - y - z-$Koordinaten läßt sich die Rotationsfläche durch

$$F(x, y, z) := f(\sqrt{x^2 + y^2}, z) = 0$$

beschreiben. Eine Normale in einem Punkt der Rotationsfläche erhält man aus

$$\nabla F(x, y, z) = (f_r(r, z)x/r, f_r(r, z)y/r, f_z(r, z)), \qquad \text{wobei } r = \sqrt{x^2 + y^2}$$

ist. (Dabei ist f_r bzw. f_z die partielle Ableitung von f nach r bzw. z.) Eine Normale im Punkt $\mathbf{q} = (x, y, z)$ mit $F(x, y, z) = 0$ ist also

$$\mathbf{n}(x, y, z) = (f_r(r, z)x, f_r(r, z)y, f_z(r, z)r), \qquad \text{wobei } r = \sqrt{x^2 + y^2}$$

ist.
Ein Punkt $\mathbf{q}$ mit $F(x, y, z) = 0$ ist also ein potentieller Umrißpunkt bei **Parallelprojektion**, wenn

$$\mathbf{n}_0 \cdot \mathbf{n}(x, y, z) = n_{0x} f_r(r, z)x + n_{0y} f_r(r, z)y + n_{0z} f_z(r, z)r = 0$$

ist. Mit der üblichen Substitution $x = r\xi,\ y = r\eta$ ergibt sich das System

$$n_{0x} f_r(r, z)\xi + n_{0y} f_r(r, z)\eta = -n_{0z} f_z(r, z), \quad \xi^2 + \eta^2 = 1.$$

Berechnet man sich von dem Meridian mit fem Verfolgungsalgorithmus für implizite Kurven Punkte $(r_1, z_1), ...(r_n, z_n)$, so läßt sich für r_i, z_i das Gleichungssystem mit `is_unitcircle_line` lösen. Mit Hilfe der Transformation $(r\xi, r\eta, z)$ erhält man dann die Umrißpunkte, falls das obige System Lösungen besitzt.
Als Testfunktion für die Sehstrahlmethode kann man

$$F(x, y, z) := f(\sqrt{x^2 + y^2}, z)$$

verwenden.

Beispiel 11.1 *Durch Rotation von Cassini-Kurven (vgl. Kap. 6) entstehen die folgenden Flächen:*

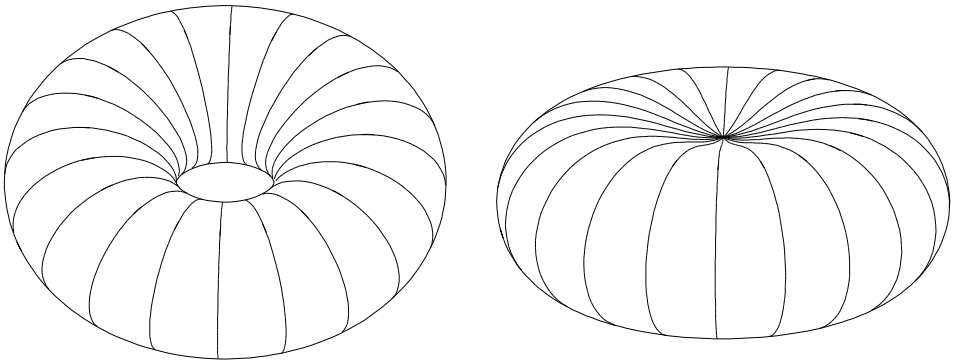


Abbildung 11.6: Rotationsfläche mit Cassini-Kurven als Meridian

Bei **Zentralprojektion** lautet für die Rotationsfläche mit Hauptmeridian $f(r, z) = 0$ die Umrißbedingung: $z_x f_r(r, z)\xi + z_y f_r(r, z)\eta = r f_r(r, z) + (z - z_z) f_z(r, z), \qquad \xi^2 + \eta^2 = 1.$

11.1.4 Meridian $r = r(\beta), z = z(\beta)$

In diesem Fall erhalten wir nur eine Parameterdarstellung der Rotationsfläche:
$\mathbf{x}(\alpha, \beta) = (r(\beta)\cos\alpha, r(\beta)\sin\beta, z(\beta))$.
Eine Normale $\mathbf{n}(\alpha, \beta)$ im Punkt $\mathbf{x}(\alpha, \beta)$ erhält man aus

$$\begin{aligned} \mathbf{x}_\alpha(\alpha,\beta) \times \mathbf{x}_\beta(\alpha,\beta) &= (-r(\beta)\sin\alpha, r(\beta)\cos\alpha, 0) \times (r'(\beta)\cos\alpha, r'(\beta)\sin\alpha, z'(\beta)) \\ &= r(\beta)(z'(\beta)\cos\alpha, z'(\beta)\sin\alpha, -r'(\beta)). \\ \mathbf{n}(\alpha,\beta) &:= (z'(\beta)\cos\alpha, z'(\beta)\sin\alpha, -r'(\beta)). \end{aligned}$$

Ein Punkt $Q : \mathbf{q}(\alpha, \beta)$ ist ein potentieller Umrißpunkt bei **Parallelprojektion**, wenn

$$(1) \qquad \mathbf{n}_0 \cdot \mathbf{n}(\alpha, \beta) = n_{0x} z'(\beta)\cos\alpha + n_{0y} z'(\beta)\sin\alpha - n_{0z} r'(\beta) = 0$$

ist. Setzt man wieder $\xi := \cos\alpha, \eta := \sin\alpha$, erhalten wir das System

$$(2) \qquad n_{0x} z'(\beta)\xi + n_{0y} z'(\beta)\eta - n_{0z} r'(\beta) = 0, \quad \xi^2 + \eta^2 = 1.$$

Für festes $\beta = \beta_0$ lassen sich mit Hilfe von `is_unitcircle_line` mögliche Lösungen $(\xi_1, \eta_1), (\xi_2, \eta_2)$ bestimmen. Mit $\mathbf{x}_i = (r(\beta_0)\xi_i, r(\beta_0)\eta_i, z(\beta_0)), \ i = 1, 2$ ergeben sich dann die zugehörigen Umrißpunkte des Breitenkreises mit $r = r(\beta_0)$ und $z = z(\beta_0)$.

Eine Testfunktion für die Sehstrahlmethode läßt sich hier i.a. nicht so leicht angeben.

Beispiel: Torus
Es sei $0 < a < R$ und $r(\beta) = R + a\cos\beta, z(\beta) = a\sin\beta$. Dann beschreibt

$$\mathbf{x}(\alpha,\beta) = (r(\beta)\cos\alpha, r(\beta)\sin\beta, z(\beta)) = ((R + a\cos\beta)\cos\alpha, (R + a\cos\beta)\sin\alpha, a\sin\beta)$$

den Torus mit der Gleichung

$$f(x,y,z) = (x^2 + y^2 + z^2 + R^2 - a^2)^2 - 4R^2(x^2 + y^2) = 0.$$

In diesem Fall ist $r'(\beta) = -a\sin\beta, z'(\beta) = a\cos\beta$ und die Gleichung (1) läßt sich auf die folgende Form bringen:

$$(n_{0x}\cos\alpha + n_{0y}\sin\alpha)\cos\beta + n_{0z}\sin\beta = 0.$$

Setzt man hier (im Gegensatz zu oben) $\xi := \cos\beta, \eta := \sin\beta$ so erhält man das Gleichungssystem:

$$(3) \qquad \begin{aligned} \gamma_1\xi + \gamma_2\eta &= 0 \qquad \text{mit } \gamma_1 := n_{0x}\cos\alpha + n_{0y}\sin\alpha, \quad \gamma_2 := n_{0z}. \\ \xi^2 + \eta^2 &= 1 \end{aligned}$$

Dieses System hat für festes $\alpha = \alpha_0$ **immer** zwei Lösungen (sofern $(\gamma_1, \gamma_2) \neq (0,0)$ ist), im Gegensatz zum obigen System (2). Die Lösungen findet man mit `is_unitcircle_line`. Es ergeben sich für jedes α genau zwei Umrißpunkte (innere bzw. äußere Kurve).

Der Umriß eines beliebigen Torus mit der Parameterdarstellung $\mathbf{x}(\alpha,\beta) = \mathbf{q}_0 + \mathbf{f}_1(R + a\cos\beta)\cos\alpha + \mathbf{f}_2(R + a\cos\beta)\sin\alpha + \mathbf{f}_3 a\sin\beta$, wobei $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ eine Orthonormalbasis ist, erhält man aus dem zu (3) analogen System

$$\begin{aligned} \gamma_1\xi + \gamma_2\eta &= 0 \qquad \text{mit } \gamma_1 := \mathbf{n}_0 \cdot \mathbf{f}_1\cos\alpha + \mathbf{n}_0 \cdot \mathbf{f}_2\sin\alpha, \quad \gamma_2 := \mathbf{n}_0 \cdot \mathbf{f}_3. \\ \xi^2 + \eta^2 &= 1 \end{aligned}$$

In dem folgenden Unterprogramm `torus_pp_contpts` werden Punkte der Umrißkurven eines Torus für den Fall, daß $\gamma_2 \neq 0$ ist, berechnet und **error**=false gesetzt. Im Fall $\gamma_2 = 0$ besteht der Umriß aus zwei senkrechten Kreisen und zwei horizontalen Kreisen, die in der Projektion als Strecken erscheinen.

```
procedure torus_pp_contpts(qt0,f1,f2,f3: vt3d; r,a: real; var p1,p2 : vts3d;
                                          var n2: integer;  var error: boolean);
{Umrisspunkte eines Torus in allgemeiner Lage.}
 var   dal,cdal,sdal,sal,cal,ccal,fac,cc,c1,c2,c3,xi1,xi2,xi3,
       cbe1,sbe1,cbe2,sbe2 : real;
       i,nis : integer;
 begin
{Umriss: }
   c1:= scalarp3d(n0vt,f1);  c2:= scalarp3d(n0vt,f2);  c3:= scalarp3d(n0vt,f3);
   n2:= 200;
   dal := pi2/n2;    cdal:= cos(dal);     sdal:= sin(dal);
   cal := 1 ;          sal:= 0;
   if abs(scalarp3d(n0vt,f3))>eps5 then
      begin
      for i:= 0 to n2 do
          begin
          cc:= c1*cal + c2*sal;
          is_unitcircle_line(cc,c3,0, cbe1,sbe1,cbe2,sbe2,nis);
```

```
          fac := r+a*cbe1;  xi1:= fac*cal;  xi2:= fac*sal;  xi3:= a*sbe1;
          lcomb4vt3d(1,qt0, xi1,f1, xi2,f2, xi3,f3, p1[i]);  { innen }
          fac:= r+a*cbe2;   xi1:= fac*cal;  xi2:= fac*sal;  xi3:= a*sbe2;
          lcomb4vt3d(1,qt0, xi1,f1, xi2,f2, xi3,f3, p2[i]);  { aussen }
          ccal:= cal;
          cal := cal*cdal-sal*sdal;  sal := sal*cdal+ccal*sdal;
          end; { for }
       error:= false;
       end
      else
       error:= true;
 end;   { torus_pp_contpts }
{****************}
```

Dieses Unterprogramm ist in `proc_prt.pas` enthalten.

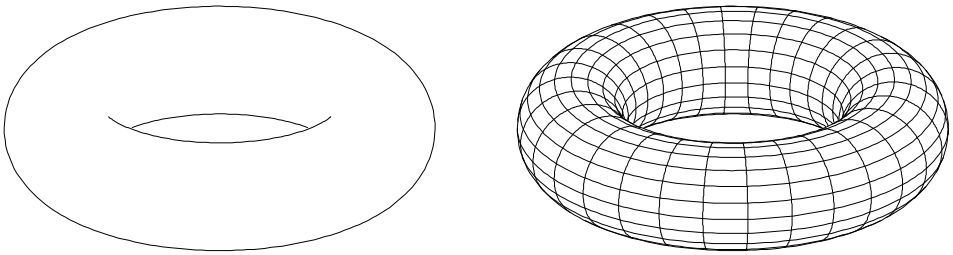


Abbildung 11.7: a) Umriß eines Torus, b) Torus mit Parameterkurven

Aufgabe 11.5 *ZWEI TORI.*

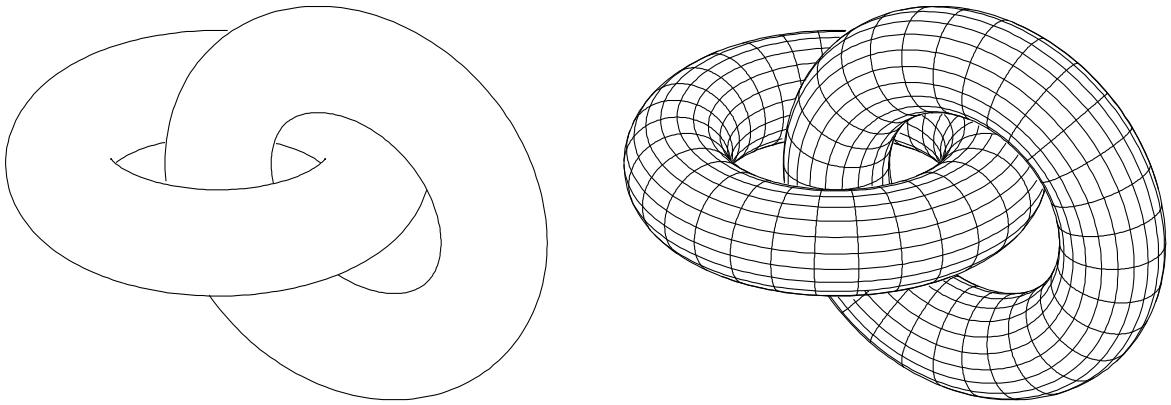


Abbildung 11.8: Zwei Tori

Bei **Zentralprojektion** lautet für die Rotationsfläche mit Hauptmeridian $r = r(\beta), z = z(\beta)$ die Umrißbedingung: $z_x z'(\beta)\xi + z_y z'(\beta)\eta = r(\beta)z'(\beta) + (z_z - z(\beta))r'(\beta), \qquad \xi^2 + \eta^2 = 1.$

Speziell für den Torus: $\mathbf{q}(\alpha,\beta) = ((R + a\cos\beta)\cos\alpha, (R + a\cos\beta)\sin\alpha, a\sin\beta)$ lautet die Umrißbedingung: $(z_x\cos\alpha + z_y\sin\alpha - R)\cos\beta + z_z\sin\beta = a$

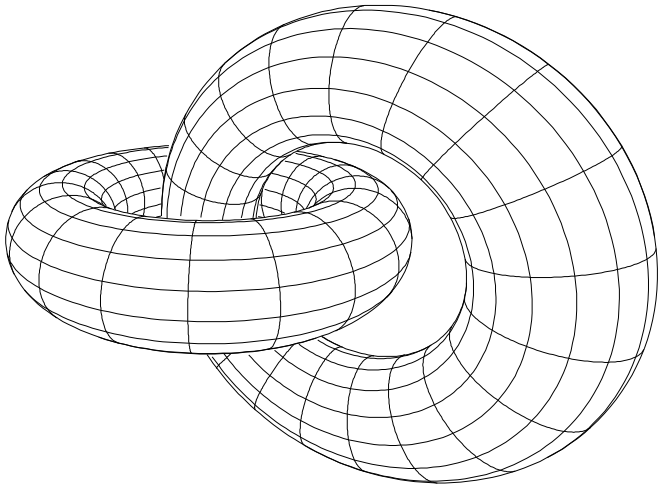


Abbildung 11.9: Zwei Tori

11.2 Schraublinien und Schraubflächen

11.2.1 Schraublinien

Es sei $Z := \{\mathbf{q}_0 + \mathbf{f}_1\cos\varphi + \mathbf{f}_2\sin\varphi + \alpha\mathbf{f}_3 \mid 0 \le \varphi \le 2\pi, \quad \alpha \in \mathbb{R}\}$ ein senkrechter Kreiszylinder, d.h. $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ bilden eine Orthogonalbasis mit $|\mathbf{f}_1| = |\mathbf{f}_2|$. Ferner sei $|\mathbf{f}_3| = 1$. Eine Kurve k auf Z mit der Parameterdarstellung

$$\mathbf{x}(\varphi) = \mathbf{q}_0 + \mathbf{f}_1\cos\varphi + \mathbf{f}_2\sin\varphi + \mathbf{f}_3\, c\varphi, \quad \text{wobei} \quad 0 \neq c \in \mathbb{R}$$

eine Konstante ist, heißt *Schraublinie*.

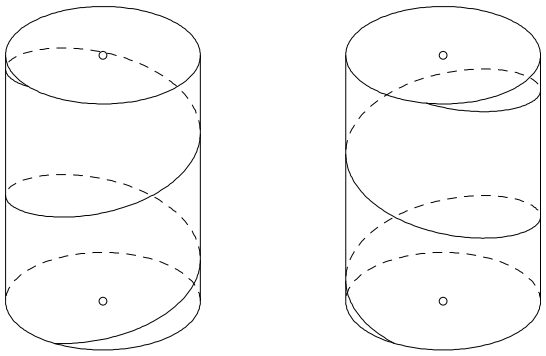


Abbildung 11.10: Schraublinien, rechts- bzw. linksgängig

Die Konstante c bezeichnet man als Schubparameter und die zu einer vollen Umdrehung gehörige Schiebstrecke $h = 2\pi c$ als *Ganghöhe*. Falls $c > 0$ bzw. $c < 0$ ist, sagt man k ist *rechts-* bzw. *linksgängig*.

Beispiele:

(1) $\mathbf{x}(\varphi) = (r\cos\varphi, r\sin\varphi, c\varphi)$, r, c Konstanten, ist eine Schraublinie auf dem Zylinder $x^2+y^2 = r^2$ mit der Ganghöhe $h = 2\pi c$.

(2) $\mathbf{x}(\varphi) = (x_0\cos\varphi - y_0\sin\varphi, x_0\sin\varphi + y_0\cos\varphi, z_0 + c\varphi)$ entsteht durch Verschraubung des Punktes (x_0, y_0, z_0) um die z-Achse. (Drehung des Punktes (x_0, y_0, z_0) um den Winkel φ um die z-Achse und anschließende Verschiebung um $c\varphi$ in z-Richtung.)

Bemerkung:
Die Projektion einer Schraublinie ist das affine Bild einer Zykloide $\mathbf{x}(t) = (t - \lambda\sin t, 1 - \lambda\cos t)$.

11.2.2 Schraubflächen

Es sei $\mu_0 : \mathbf{x}(\beta) = (u(\beta), v(\beta), w(\beta)) \quad \beta_1 \le \beta \le \beta_2$, eine Kurve im $\mathbb{R}^3$.
Verschraubt man den Punkt $\mathbf{x}(\beta)$ um die **z-Achse** so erhält man die Parameterdarstellung der dadurch erzeugten *Schraubfläche* Φ:

$$\mathbf{x}(\alpha,\beta) = (u(\beta)\cos\alpha - v(\beta)\sin\alpha, u(\beta)\sin\alpha + v(\beta)\cos\alpha, w(\beta) + c\alpha), \quad \alpha_1 \le \alpha \le \alpha_2, \quad \beta_1 \le \beta \le \beta_2.$$

(Drehung des Punktes $\mathbf{x}(\beta)$ um die z-Achse um den Winkel α und anschließende Verschiebung in z-Richtung um $c\alpha$.)
Die Kurve μ_0 heißt *Erzeugende* von Φ.
Eine Ebene durch die z-Achse nennt man *Meridian-Ebene*. Der Schnitt einer Meridian-Ebene mit der Schraubfläche Φ heißt *Meridian* von Φ. Den Schnitt von Φ mit der $x-z-$Ebene heißt *Hauptmeridian*.

Parallelprojektion einer Schraubfläche

Da jede Schraubfläche auch durch eine in der Hauptmeridianebene liegende Kurve μ_0 erzeugt werden kann, wollen wir dies hier voraussetzen.
Es sei also $\mu_0 : \mathbf{x}(\beta) = (u(\beta), 0, w(\beta)), \quad \beta_1 \le \beta \le \beta_2$,
Dann hat Φ die Parameterdarstellung

$$\mathbf{x}(\alpha,\beta) = (u(\beta)\cos\alpha, u(\beta)\sin\alpha, w(\beta) + c\alpha).$$

Aus $\mathbf{x}_\alpha, \mathbf{x}_\beta$ und der (negativen) Projektionsrichtung $\mathbf{n}_0 = (n_{0x}, n_{0y}, n_{0z})$ ergibt sich die folgende Umrißbedingung:

$$(*) \qquad \mathbf{n}_0 \cdot (\mathbf{x}_\alpha \times \mathbf{x}_\beta) = cu'(n_{0y}\cos\alpha - n_{0x}\sin\alpha) + uw'(n_{0x}\cos\alpha + n_{0y}\sin\alpha) - n_{0z}uu' = 0$$

Beispiele:

Meridiankreis-Schraubfläche:
$\mu_0 : \mathbf{x}(\beta) = (R + a\cos\beta, 0, a\sin\beta)$ ist ein Kreis in der $x - z-$Ebene mit Mittelpunkt $(R, 0, 0)$ und Radius a. Die von μ_0 erzeugte Schraubfläche Φ heißt Meridiankreis-Schraubfläche.

Umriß:
Mit $u'(\beta) = -a\sin\beta$ und $w'(\beta) = a\cos\beta$ erhält (*) die Form

$$\begin{aligned} -ca\sin\beta(n_{0y}\cos\alpha - n_{0x}\sin\alpha) \quad &+ \quad (R + a\cos\beta)a\cos\beta(n_{0x}\cos\alpha + n_{0y}\sin\alpha) \\ &+ \quad n_{0z}(R + a\cos\beta)a\sin\beta = 0 \end{aligned}$$

Setzt man $\xi := \cos\beta, \eta := \sin\beta$, so läßt sich diese Gleichung durch das folgende nichtlineare System ausdrücken:

$$\begin{aligned} \gamma_1\xi^2 + \gamma_2\xi + \gamma_3\eta + \gamma_4\xi\eta &= 0 \\ \xi^2 + \eta^2 - 1 &= 0, \end{aligned}$$

mit

$$\begin{aligned} \gamma_1 &= a^2(n_{0x}\cos\alpha + n_{0y}\sin\alpha), & \gamma_2 &= aR(n_{0x}\cos\alpha + n_{0y}\sin\alpha), \\ \gamma_3 &= a(c(n_{0x}\sin\alpha - n_{0y}\cos\alpha) + n_{0z}R), & \gamma_4 &= a^2 n_{0z}. \end{aligned}$$

Lösungen dieses Systems bei festgehaltenem α lassen sich mit Hilfe einer Newton-Iteration bestimmen :

$$f(\xi,\eta) := \gamma_1\xi^2 + \gamma_2\xi + \gamma_3\eta + \gamma_2\xi\eta, \qquad g(\xi,\eta) := \xi^2 + \eta^2, \qquad \mathbf{F}(\xi,\eta) := (f(\xi,\eta), g(\xi,\eta)),$$

$$\mathbf{F}'(\xi,\eta) = \begin{pmatrix} f_\xi(\xi,\eta) & f_\eta(\xi,\eta) \\ g_\xi(\xi,\eta) & g_\eta(\xi,\eta) \end{pmatrix} = \begin{pmatrix} 2\gamma_1\xi + \gamma_2 + \gamma_4\eta & \gamma_3 + \gamma_4\xi \\ 2\xi & 2\eta \end{pmatrix}$$

Iteration: $\mathbf{F}'(\mathbf{x}_k)\cdot\mathbf{d}_k = -\mathbf{F}(\mathbf{x}_k) \to \mathbf{d}_k \to \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$.

Startwerte bei $\alpha = \alpha_1$ ergeben sich bei einem Durchlauf des Einheitskreises mit Vorzeichentest für $f(\xi,\eta) = \gamma_1\xi^2 + \gamma_2\xi + \gamma_3\eta + \gamma_4\xi\eta$. Als Startwerte bei $\alpha_{k+1} = \alpha_k + \Delta\alpha$ verwendet man zweckmäßigerweise die Lösungen $(\xi_k\eta_k)$ und (ξ'_k, η'_k) bei α_k.

Testfunktion für die **Sehstrahlmethode:**
(0) Als umgebende einfache Fläche wählt man einen Zylinder mit Radius $R + a + \varepsilon, \varepsilon > 0$. Es sei $T = (x_t, y_t, z_t)$ ein Testpunkt.
(1) Bestimme mit Hilfe von `polar_angle` den Polarwinkel α_0 zu (x_t, y_t). (α_0 liegt zwischen 0 und 2π !)
(2) Berechne $k_t \in \mathbb{N}$ so, daß $(2k_t - 1)c\pi \le z_t - \alpha_0 < (2k_t + 1)c\pi$ und setze $\alpha_t = \alpha_0 + k_t\pi$.
(3) Als Testfunktion kann man den Abstand von T zum Mittelpunkt M des um α_t verschraubten Erzeugerkreises nehmen. Besser, weil wurzelfrei, ist:

$$f(x,y,z,\alpha) := (x^2 + y^2 + (z - c\alpha)^2 + R^2 - a^2)^2 - 4R^2(x^2 + y^2).$$

Dies ist die Testfunktion des Torus, der denselben Erzeugerkreis enthält.

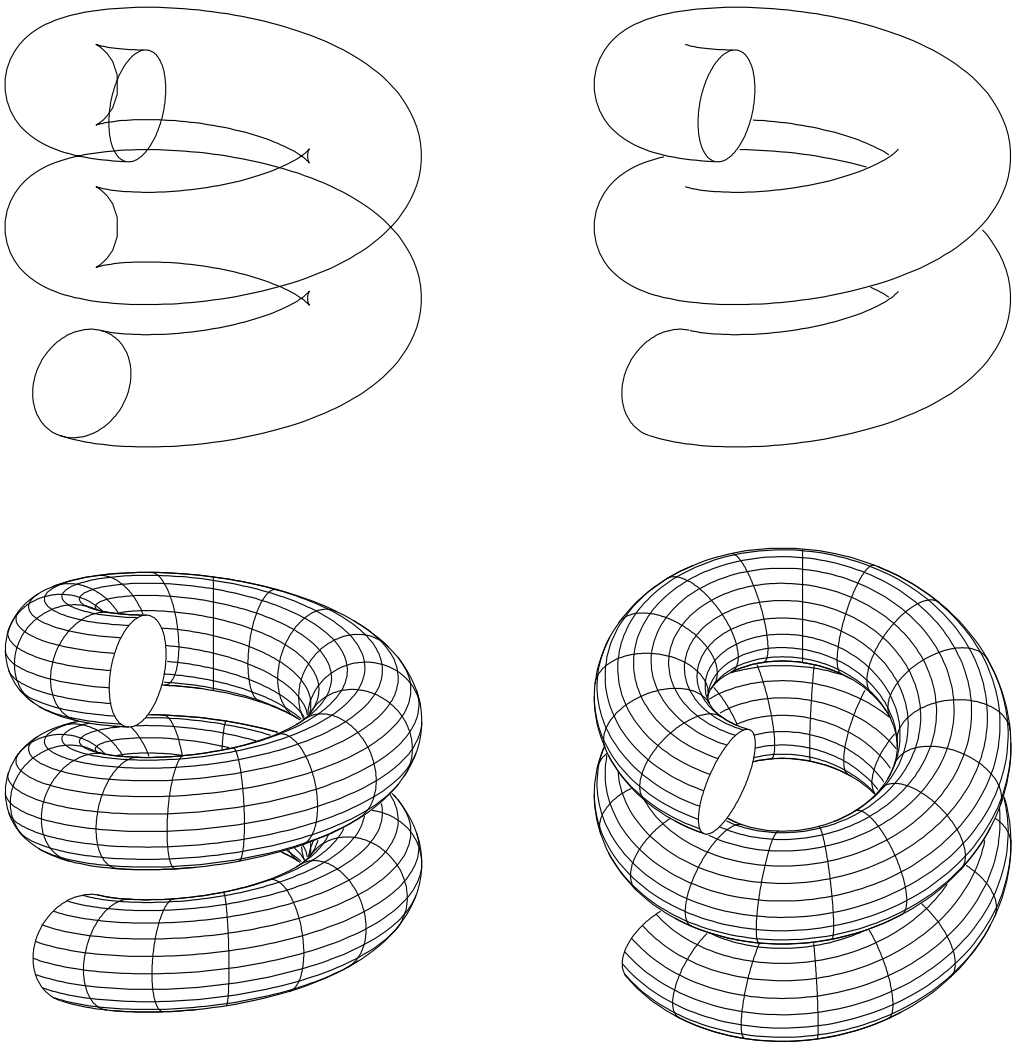


Abbildung 11.11: Meridiankreis-Schraubfläche

Wendel-Fläche

$\mu_0 : \mathbf{x}(r) = (r, 0, 0), -R \leq r \leq R$. (Hier wurde r anstatt β geschrieben.)
μ_0 ist hier eine horizontale Strecke durch die $z-$Achse.

Umriß:
Mit $u(r) = r, u'(r) = 1$ und $w = 0$ erhält man aus (*) die Bedingung:

$$\mathbf{n}_0 \cdot (\mathbf{x}_\alpha \times \mathbf{x}_r) = c(n_{0y} \cos\alpha - n_{0x} \sin\alpha) - n_{0z} r = 0$$

Diese Gleichung läßt sich im Fall $n_{0z} \neq 0$ nach r auflösen. Mit Hilfe dieser Funktion $r(\alpha)$ lassen sich dann mühelos Umrißpunkte berechnen.

Sichtbarkeitstest:
(0) Als umgebende Fläche wählt man einen Zylinder mit Radius $R + \varepsilon, \varepsilon > 0$.
Es sei $T = (x_t, y_t, z_t)$ ein Testpunkt.
(1) Aus $z_t = c\alpha$ erhält man $\alpha_t = z_t/c$ und damit die Lage der verschraubten Strecke in der Höhe $z = z_t$.
(2) Als Testfunktion wählt man hier den "Abstand" des Testpunktes T von der in gleicher Höhe liegenden verschraubten Strecke

$$\begin{aligned} \mathbf{x}(\alpha_t, r) &= (r\cos\alpha_t, r\sin\alpha_t, z_t), \quad -R \le r \le R; \\ f(x, y) &= x\sin\alpha_t - y\cos\alpha_t \end{aligned}$$

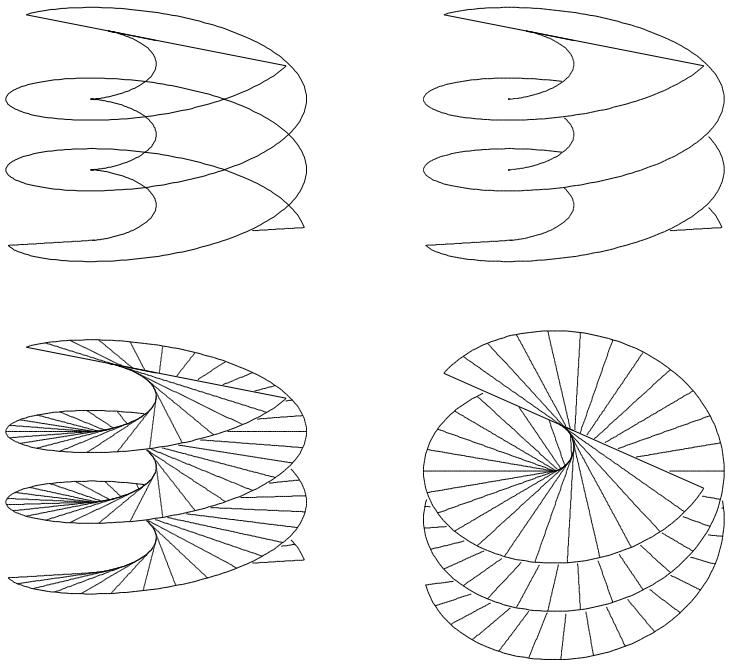


Abbildung 11.12: Wendelfläche

Kapitel 12

SCHNITT KURVE-FLÄCHE, SCHNITT FLÄCHE-FLÄCHE

12.1 Schnitt Kurve-Fläche

12.1.1 Schnitt Parametrisierte Kurve - implizite Fläche

Gegeben: Kurve $\Gamma : \mathbf{x} = \mathbf{c}(t), \quad t \in [a, b], \quad$ Fläche $\Phi : f(\mathbf{x}) = 0, \quad \mathbf{x} \in D \subset \mathbb{R}^3$.
$t \to \mathbf{c}(t)$ und $\mathbf{x} \to f(\mathbf{x})$ seien differenzierbar.
Gesucht: Schnittpunkte $\Gamma \cap \Phi$.

Der **Algorithmus:**

(1) Bestimmung von Startwerten:
Berechne ungefähr äquidistante Kurvenpunkte (vgl. Abschnitt 6.1). Setze diese Punkte in $f(\mathbf{x})$ ein und bestimme diejenigen Punktepaare zwischen denen f das Vorzeichen wechselt. Die Parameter solcher Punktepaare kann man als Startwerte für die folgende Newton-Iteration verwenden.

(2) Ist t_0 ein Startwert (aus (1)) in der Nähe eines Schnittpunktes, so führt man damit eine Newton-Iteration zur Lösung der Gleichung $F(t) := f(\mathbf{c}(t)) = 0$ durch (s. Kap. 4.5). Ist t^* eine Lösung so ist $\mathbf{x}^* := \mathbf{c}(t^*)$ ein Schnittpunkt aus $\Gamma \cap \Phi$.

(3) Führe (2) für alle in (1) bestimmten Startwerte durch.

12.1.2 Schnitt Implizite Kurve - implizite Fläche

In diesem Fall ist die Kurve als Schnittkurve zweier impliziter Flächen definiert.

Gegeben: Kurve $\Gamma : f_1(\mathbf{x}) = 0,\ f_2(\mathbf{x}) = 0,\ \mathbf{x} \in D_{12} \subset \mathbb{R}^3, \quad$ Fläche $\Phi : f(\mathbf{x}) = 0,\ \mathbf{x} \in D \subset \mathbb{R}^3$.
f_1, f_2 und f seien differenzierbar.
Gesucht: Schnittpunkte $\Gamma \cap \Phi$.

Der **Algorithmus:**

(1) Bestimmung von Startpunkten:
Berechne ungefähr äquidistante Punkte (vgl. Abschnitt 12.2.2) der Durchdringungskurve der

beiden Flächen $f_1(\mathbf{x}) = 0,\ f_2(\mathbf{x}) = 0$. Die Punkte dieses Polygonzuges setzt man in $f(\mathbf{x})$ ein und bestimmt die Punktepaare, zwischen denen ein Vorzeichenwechsel auftritt. Ein Punkt eines jeden Punktepaares dient als Startpunkt der folgenden Newton-Iteration.

(2) Ist $\mathbf{x}_0$ ein Startpunkt (aus (1)) , so bestimmt man mit Hilfe einer Newton-Iteration (s.Kap. 6) eine Lösung $\mathbf{x}^*$ des Systems $\mathbf{F}(\mathbf{x}) := (f_1(\mathbf{x}), f_2(\mathbf{x}), f(\mathbf{x})) = 0$. $\mathbf{F}'$ ist hier die Matrix $\mathbf{F}' = (\nabla f_1, \nabla f_2, \nabla f)$. $P : \mathbf{x}^*$ ist ein Schnittpunkt aus $\Gamma \cap \Phi$.

(3) Führe (2) für alle in (1) bestimmten Startpunkte durch.

12.1.3 Schnitt Implizite Kurve - parametrisierte Fläche

Auch in diesem Fall ist die Kurve als Schnittkurve zweier impliziter Flächen definiert.

Gegeben: Kurve $\Gamma : f_1(\mathbf{x}) = 0,\ f_2(\mathbf{x}) = 0,\ \mathbf{x} \in D_{12} \subset \mathbb{R}^3$, Fläche $\Phi : \mathbf{x} = \mathbf{S}(u, v),\ (u, v) \in D \subset \mathbb{R}^2$.
f_1, f_2 und $\mathbf{S}$ seien differenzierbar.
Gesucht: Schnittpunkte $\Gamma \cap \Phi$.

Der **Algorithmus:**

(1) Schnitt der beiden Flächen $f_1(\mathbf{x}) = 0$ und $\mathbf{x} = \mathbf{S}(u, v)$. Die Schnittkurve ist im Parameterraum die implizite Kurve $F_1(u, v) := f_1(\mathbf{S}(u, v)) = 0$.

(2) Schnitt der beiden Flächen $f_2(\mathbf{x}) = 0$ und $\mathbf{x} = \mathbf{S}(u, v)$. Die Schnittkurve ist im Parameterraum die implizite Kurve $F_2(u, v) := f_2(\mathbf{S}(u, v)) = 0$.

(3) Die Schnittpunkte der beiden impliziten Kurven $F_1(u, v) = 0$ und $F_2(u, v) = 0$ (s. Abschnitt 6.5.2.) liefern die Parameter für Schnittpunkte aus $\Gamma \cap \Phi$.

Schnitt parametrisierte Kurve - parametrisierte Fläche

Gegeben: Kurve $\Gamma : \mathbf{x} = \mathbf{c}(t),\quad t \in [a, b]$, Fläche $\Phi : \mathbf{x} = \mathbf{S}(u, v),\quad (u, v) \in D \subset \mathbb{R}^2$
$\mathbf{c}$ und $\mathbf{S}$ seien differenzierbar.
Gesucht: Schnittpunkte $\Gamma \cap \Phi$.

Der **Algorithmus:**

(1) Bestimmung von Startpunkten:

(1.1) Berechne einen Polygonzug mit ungefähr äquidistanten Kurvenpunkten (vgl. Abschnitt 6.1).

(1.2) Trianguliere die Fläche (s. Abschn. 14.

(1.3) Bestimme mit Hilfe des Algoritmus in Abschn. 13.4.2 Paare σ_i, Δ_j die aus einer Strecke σ_i des Polygonzuges auf der Kurve und einem Dreieck Δ_j des zur Fläche gehörigen Polyeders bestehen und sich schneiden.

(2) Ist σ_i, Δ_j ein in (1) bestimmtes Paar, so wählt man den Parameter eines Endpunktes der Strecke σ_i als Startwert t_0, die Parameter eines Punktes des Dreiecks Δ_j als Startwerte u_0, v_0. Mit Hilfe dieser Startwerte führt man eine Newton-Iteration für die Funktion $\mathbf{F}(t, u, v) := \mathbf{c}(t) - \mathbf{S}(u, v)$ und die Gleichung $\mathbf{F}(t, u, v) = 0$ durch (s.Kap. 6.5). Eine Lösung (t^*, u^*, v^*)

liefert die Parameter eines Schnittpunktes der Kurve Γ mit der Fläche Φ.
Die bei der Iteration benötigte Matrix $\mathbf{F}'$ ist hier $\mathbf{F}' = (\dot{\mathbf{c}}, \mathbf{S}_u, \mathbf{S}_v)$.

(3) Führe (2) für alle in (1) bestimmten Paare durch.

12.2 Schnitt Fläche-Fläche

12.2.1 Schnitt zweier Quadriken

Der Schnitt zweier Quadriken, von denen eine **Geraden enthält** (z.B. Zylinder, Kegel oder einschaliges Hyperboloid), läßt sich relativ leicht ermitteln. Denn in diesem Fall muß man nur Geraden mit einer Quadrik schneiden (vgl. Absch. 10.3). In konkreten Fällen kann sich das Schneiden der auftretenden Geraden mit der zweiten Quadrik besonders einfach gestalten, so daß man, um Rechenzeit zu sparen, besser eine spezielle Prozedur erstellt. Die folgenden Beispiele sind von dieser Art.

Beispiel 12.1 *Schnitt KEGEL- ZYLINDER*

$$\begin{array}{lrcl} \Phi_1 : Kegel & (z-h)^2 & = & (h/r_k)^2(x^2+y^2), \quad 0 \le z \le h. \\ \Phi_2 : Zylinder & (y-y_a)^2+(z-z_a)^2 & = & r_z^2, \quad -l_z \le x \le l_z, \quad z_a \ge r_z. \end{array}$$

Wir unterscheiden 3 Schnittsituationen:

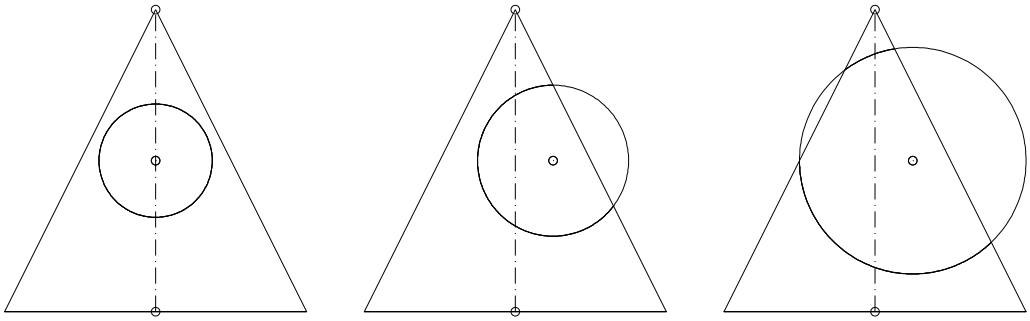


Abbildung 12.1: Schnitt Kegel-Zylinder

(1) Mit Hilfe von `is_circle_line` für die Mantellinien $z = \pm(h/r_k)y + h$ und den Kreis $(y-y_a)^2 + (z-z_a)^2 = r_z^2$ in der $y-z-$Ebene, bestimmt man, welcher Fall vorliegt.

(2) Durchdringungskurve:
Ist m die Mantellinie des Zylinders durch den Punkt $(0, y_a + r_z\cos\varphi, z_a + r_z\sin\varphi)$, so erhält man die gemeinsamen Punkte
$(\pm\sqrt{((h-z_i)r_k/h)^2 - y_i^2}, y_i, z_i)$ mit $y_i = y_a + r_z\cos\varphi,\ z_i = z_a + r_z\sin\varphi$,
falls $\sqrt{\cdots}$ existiert.
(s. auch Absch. 10.3)
Im **Fall I** erhält man Schnittpunkte für beliebige Winkel φ aus dem Intervall $[0, 2\pi]$. Im **Fall II** läuft φ über *ein* Teilintervall und im **Fall III** über *zwei* Teilintervalle von $[0, 2\pi]$.
Um alle Fälle gemeinsam behandeln zu können, schreibt man ein spezielles Unterprogramm is_cone_cylinder_spec$(w1, w2, dw, \mathbf{p}, np)$, das np Punkte ($\mathbf{p} : vts3d$) im Intervall $[w1, w2]$ im

Winkelabstand dw mit $x \leq 0$ berechnet. $w1$ und $w2$ ergeben sich in den Fällen II, III aus den in (1) berechneten Schnittpunkten. Die restlichen Kurventeile erhält man durch Spiegelung der schon berechneten an der $y-z-$Ebene.

(3) Die Umrisse des Kegels und des Zylinders projiziert man mit `pp_cone` und `pp_cylinder` oder verwendet die Sehstrahlmethode als Hiddenline-Algorithmus (s. Kap. 10) .

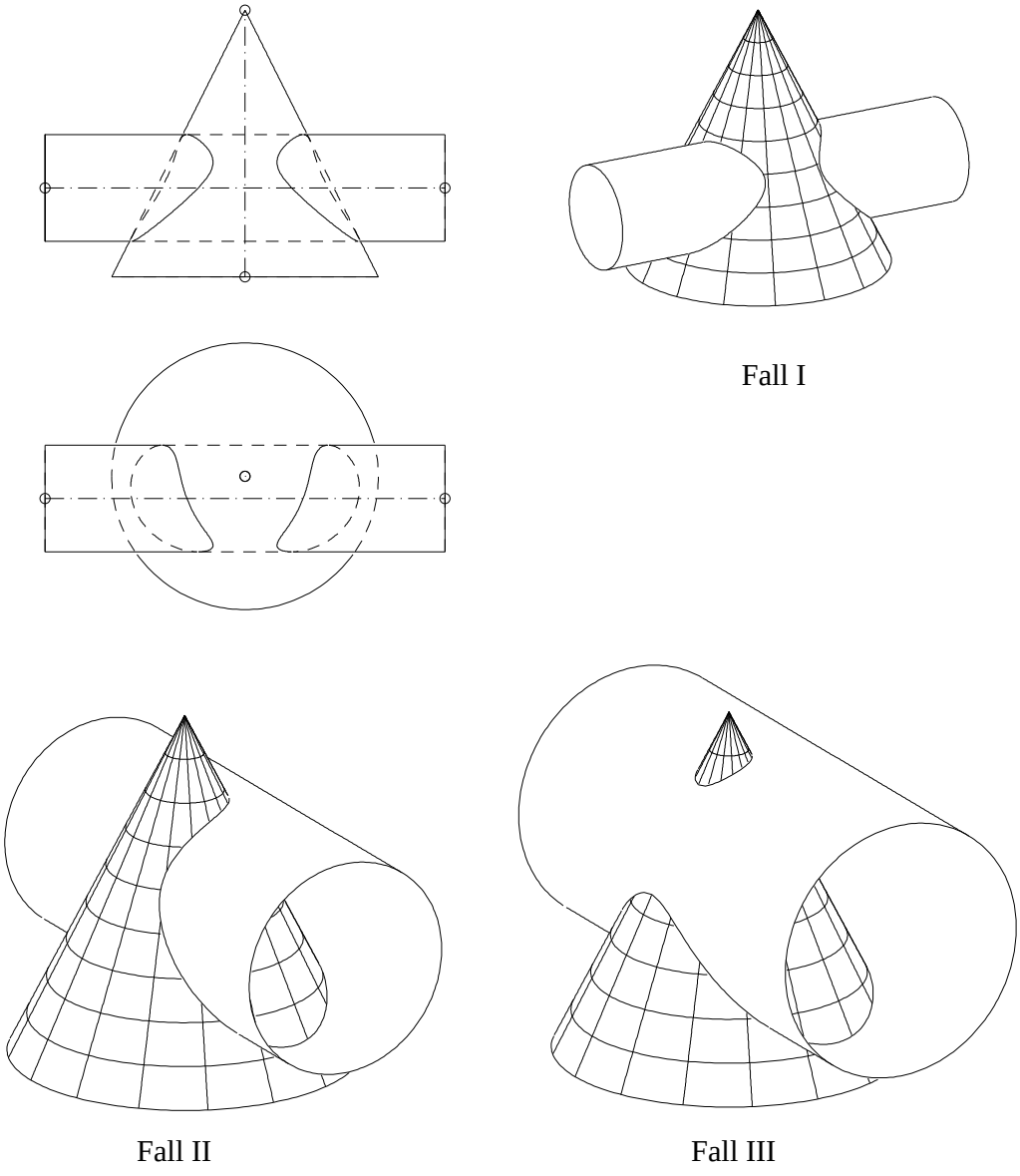


Abbildung 12.2: Durchdringung Kegel-Zylinder: Fall I, Fall II, Fall III

Aufgabe 12.1 *Schreibe ein Programm das den Schnitt der Flächen*

$$\begin{array}{lllcll} \Phi_1: & \textit{Kugel} & x^2+y^2+z^2 & = & r^2, & \\ \Phi_2: & \textit{Zylinder} & (y-y_a)^2+z^2 & = & r_z^2, & -l_z \le x \le l_z \end{array}$$

darstellt.

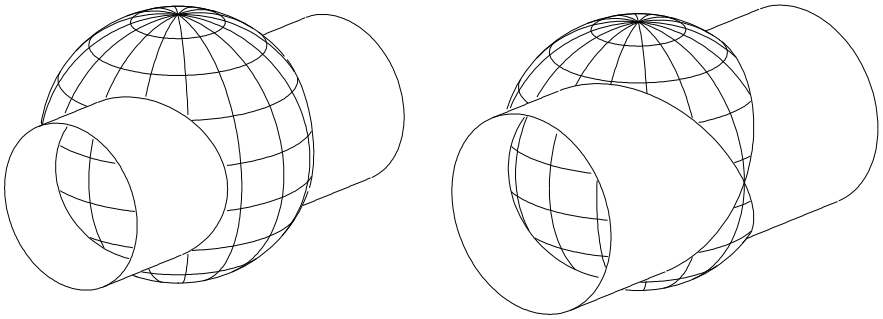


Abbildung 12.3: Durchdringung Kugel-Zylinder

Aufgabe 12.2 *Schnitt KEGEL-KUGEL:*

$$\begin{array}{lllcll} \Phi_1: & \textit{Kegel} & (z-h)^2 & = & (h/r_k)^2(x^2+y^2), & 0 \le z \le h, \\ \Phi_2: & \textit{Kugel} & (x-x_m)^2+y^2+(z-z_m)^2 & = & r^2, & r \le z_m. \end{array}$$

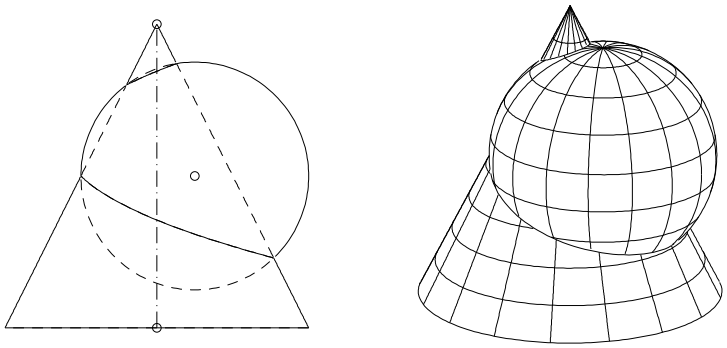


Abbildung 12.4: Durchdringung Kegel-Kugel

Aufgabe 12.3 *Bestimme die Schnittkurve eines Kegels mit einem geneigten Zylinder wie in den folgenden Abbildungen und wickle den Kegel und den Zylinder mit jeweils der Durchdringungskurve ab. (vgl. Kap. 16)*

Aufgabe 12.4 *Schnitt ZYLINDER - ZYLINDER*

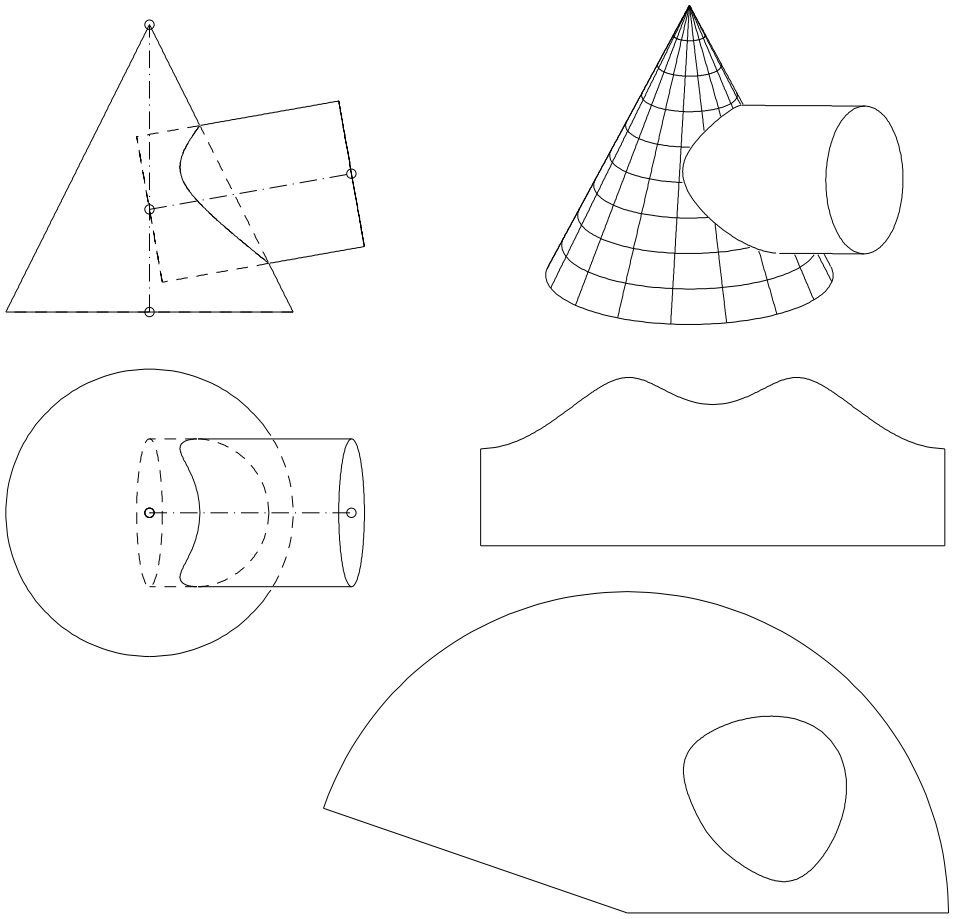


Abbildung 12.5: zu Aufgabe 12.3

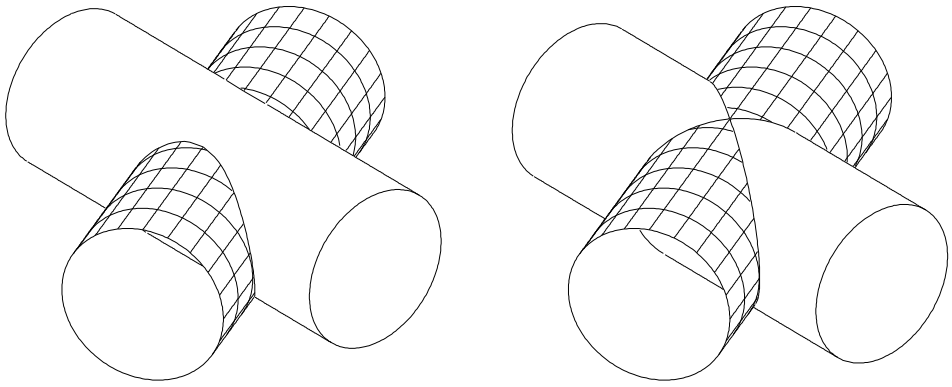


Abbildung 12.6: zu Aufgabe 12.4

In speziellen Situationen lassen sich auch Flächen höheren Grades mit Zylindern schneiden. Z.B. enthält der folgende Rohrabzweig die Durchdringungskurve zwischen einem Torus und einem Zylinder in spezieller Lage. Den Schnitt einer Zylindergeraden kann man hier auf den Schnitt einer Gerade mit zwei auf dem Torus liegenden Kreise zurückführen.

Aufgabe 12.5 *ROHRABZWEIG (Schnitt Torus-Zylinder)*

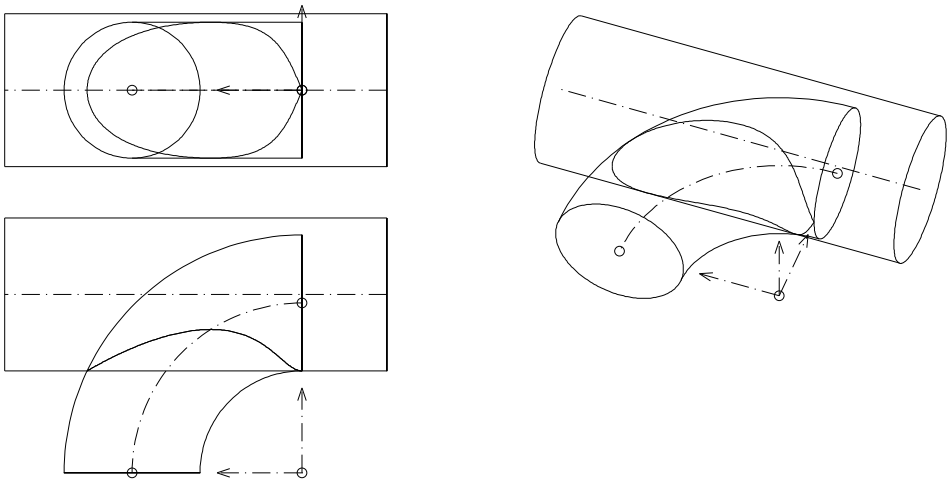


Abbildung 12.7: Rohrabzweig (Aufgabe 12.5)

12.2.2 Schnitt zweier impliziter Flächen

In BA,HO,LY,HO'88 wird ein Algorithmus für die sukzessive Bestimmung von Punkten der Schnittkurve zweier implizit gegebener Flächen beschrieben. Im Folgenden wird zunächst eine etwas vereinfachte Version dieses *Verfolgungsalgorithmus* beschrieben.

Es seien zwei Flächen durch Gleichungen $f_1(x,y,z) = 0,\ f_2(x,y,z) = 0$ gegeben. Für die Funktionen f_1, f_2 setzen wir voraus:

(1) f_1, f_2 sind differenzierbar.

(2) Die Gradienten $\nabla f_1, \nabla f_2$ sind auf und in der Nähe der Schnittkurve linear unabhängig.

Idee: Hat man einen Punkt P_i der Kurve innerhalb vorgegebener Toleranzgrenzen gefunden, so findet man eine Näherung Q_0 für den nächsten Punkt P_{i+1}, in dem man ein Stück entlang der Tangente in P_i läuft. Den Punkt Q_0 verbessert man mit Hilfe mehrerer Newton-Iterationen.
Durchführung:

A) Einen *Startpunkt* $P_0 : \mathbf{p}_0$ findet man, in dem man zunächst einen Punkt $Q_0 : \mathbf{q}_0$ in der "Nähe" der beiden Flächen bestimmt. Mit Hilfe der unten beschriebenen Iteration erhält man dann $\mathbf{p}_0$.

B) $P_{i+1} : \mathbf{p}_{i+1}$ aus $P_i : \mathbf{p}_i$:

(1) 0-te Näherung $\mathbf{q}_0$ für $\mathbf{p}_{i+1}$:
$\mathbf{q}_0 = \mathbf{p}_i + \delta\mathbf{t}$, wobei δ eine Schrittweite und $\mathbf{t} = (\nabla f_1(\mathbf{p}_i) \times \nabla f_2(\mathbf{p}_i))/\| \dots \|$.
der Tangenteneinheitsvektor ist.

(2) $\mathbf{q}_{k+1}$ aus $\mathbf{q}_k$:
Es sei

$$g_j(s,t) := f_j(\mathbf{q}_k + s\nabla f_1(\mathbf{q}_k) + t\nabla f_2(\mathbf{q}_k)),\ j = 1, 2.$$

Für die Gleichung $\mathbf{G}(s,t) := (g_1(s,t), g_2(s,t)) = (0,0)$ und den Startpunkt $(0,0)$ führen wir einen Newtonschritt aus, d.h. wir lösen das lineare Gleichungssystem

$$\begin{array}{lcll} s\nabla f_1(\mathbf{q}_k)^2 & + & t\nabla f_1(\mathbf{q}_k)\cdot\nabla f_2(\mathbf{q}_k) & = & -f_1(\mathbf{q}_k) \\ s\nabla f_1(\mathbf{q}_k)\cdot\nabla f_2(\mathbf{q}_k) & + & t\nabla f_2(\mathbf{q}_k)^2 & = & -f_2(\mathbf{q}_k) \end{array}$$

für s, t und setzen $\mathbf{q}_{k+1} = \mathbf{q}_k + s\nabla f_1(\mathbf{q}_k) + t\nabla f_2(\mathbf{q}_k)$.

(3) Ist der Abstand zwischen $\mathbf{q}_{k+1}$ und $\mathbf{q}_k$ klein "genug" (oder eine andere Abbruchbedingung), setzen wir $\mathbf{p}_{i+1} = \mathbf{q}_{k+1}$.

Aufgabe 12.6 *Bestimme die Schnittkurve der beiden impliziten Flächen (Abb. 12.8)*

$$\begin{array}{lcll} f_1(x,y,z) = x^4 + y^4 + z^4 - 1 & = & 0 & (\textit{"Hyperkugel"}) \\ f_2(x,y,z) = y^2 + (z - z_0)^2 - z_0^2 & = & 0 & (\textit{Zylinder}) \end{array}$$

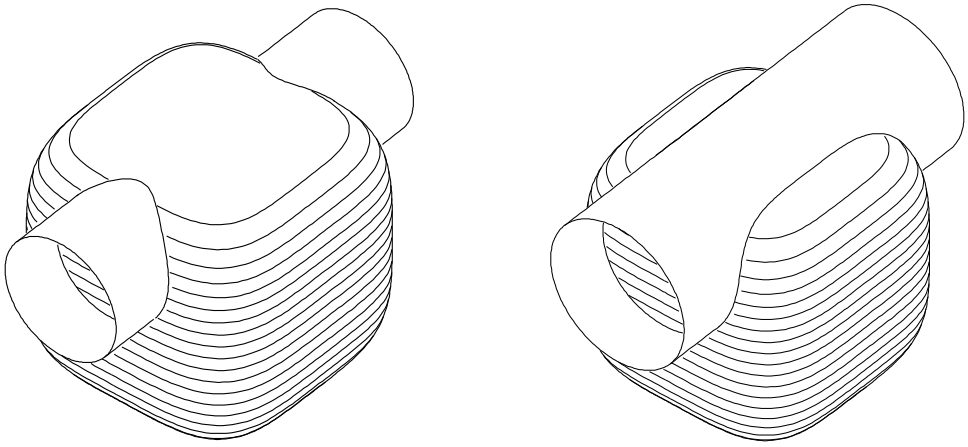


Abbildung 12.8: zu Aufgabe 12.6 , $z_0 = 0.47$ bzw. $z_0 = 0.6$

Bemerkung: Mit dem obigen Algorithmus lassen sich oft auch Fälle behandeln, bei denen die Schnittkurve **Singularität**en besitzt (s. Abb. 12.9. Der Grund dafür ist, daß man praktisch eine Singularität so gut wie nie trifft. Da aber in einer Singularität eine Richtungsumkehr eintreten kann, muß man dafür sorgen, daß der Algorithmus nicht um die Singularität pendelt und damit hängen bleibt.

Bemerkung zur Bestimmung von $\mathbf{q}_0$:
Die Bestimmung der $0-$ten Näherung $\mathbf{q}_0$ von $\mathbf{p}_{i+1}$ läßt sich durch die folgende Abänderung des Schrittes B1) *krümmungsabhängig* gestalten.

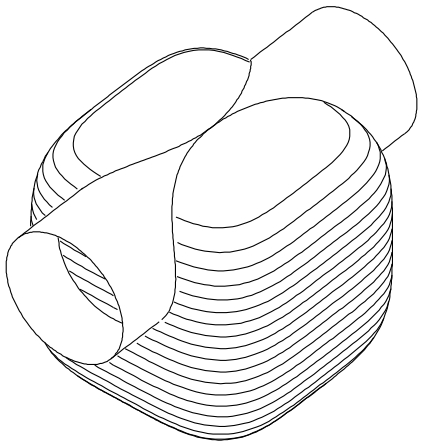


Abbildung 12.9: Schnittkurve mit einer Singularität , $z_0 = 0.5$

Setzt man eine Parameterdarstellung $\mathbf{c}(s)$ der Schnittkurve in die Funktionen f_1 und f_2 ein und entwickelt die so entstandenen Funktionen $f_1(\mathbf{c}(s))$ und $f_2(\mathbf{c}(s))$ nach s, so erhält man

$$0 = f_i(\mathbf{c}(s)) = f_i(\mathbf{c}(0)) + s\nabla f_i \cdot \mathbf{c}'(0) + \frac{s^2}{2}(\nabla f_i \cdot \mathbf{c}''(0) + \mathbf{c}'(0) \cdot H_i \cdot \mathbf{c}'(0)^T) + \cdots \qquad i = 1, 2,$$

wobei ∇f_i der Gradient und H_i die Hesse-Matrix an der Stelle $\mathbf{c}(0)$ sind.
Die Koeffizienten bei s^0 und s^1 liefern keine neuen Erkenntnisse. Der Koeffizient bei s^2 gibt uns die Möglichkeit $\mathbf{c}''(0)$ zu berechnen. Hierzu machen wir die Annahme, daß der Parameter s die *Bogenlänge* ist. Denn dann gilt

(1) $\mathbf{c}'(0) = (\nabla f_1 \times \nabla f_2)/\|\nabla f_1 \times \nabla f_2\|$, ($\mathbf{c}'(s)$ hat die Länge 1!)

(2) $\mathbf{c}''(0) \cdot \mathbf{c}'(0) = 0, \quad i = 1, 2$ ($\mathbf{c}''$ liegt in der von ∇f_1 und ∇f_2 aufgespannten Ebene)

(3) $\nabla f_i \cdot \mathbf{c}''(0) = -\mathbf{c}'(0) \cdot H_i \cdot \mathbf{c}'(0)^T, \quad i = 1, 2,$

$\nabla f_1, \nabla f_2$ und H_1, H_2 berechnet an der Stelle $\mathbf{c}(0)$. Wegen der linearen Unabhängigkeit von ∇f_1 und ∇f_2 und der Beziehung (2) lassen sich aus den zwei Gleichungen (3) Zahlen β und γ bestimmen mit

$$\mathbf{c}''(0) = \beta\nabla f_1 + \gamma\nabla f_2.$$

Die Schrittweite s wählt man jetzt krümmungsabhängig, z. B.

$$|s| \approx 1/5\|\mathbf{c}''(0)\|.$$

Damit ergibt sich die 0-te Näherung $\mathbf{q}_0$ von $\mathbf{p}_{i+1}$: $\mathbf{q}_0 = \mathbf{p}_i + s\mathbf{c}'(0) + \frac{s^2}{2}\mathbf{c}''(0)$.

12.2.3 Umriß impliziter Flächen

Der in 12.2 angegebene Verfolgungsalgorithmus läßt sich auch zur Berechnung des Umrisses einer implizit gegebenen Fläche verwenden. Sei $f(x, y, z) = 0$ die Gleichung einer Fläche Φ und ist f differenzierbar, so genügt ein potentieller Umrisspunkt (x_0, y_0, z_0) bei **Parallelprojektion** den beiden Bedingungen

$$f(x_0, y_0, z_0) = 0, \quad g(x_0, y_0, z_0) = \nabla f(x_0, y_0, z_0) \cdot \mathbf{n}_0 = 0,$$

wobei $\mathbf{n}_0$ die (negative) Projektionsrichtung ist. D.h. der Umriß läßt sich als Schnitt zweier implizit gegebener Flächen $f = 0$ und $g = 0$ auffassen. Da die Funktion g aber mittelbar von f abhängt, können wir die während des Algorithmus notwendige Berechnung von ∇g etwas formalisieren:

Es sei f 2-mal differenzierbar und $g := \nabla f \cdot \mathbf{n}_0$. Dann ist $\nabla g = H_f \mathbf{n}_0$, mit der HESSE-Matrix

$$H_f := \begin{pmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{pmatrix}.$$

Ist f von der speziellen Form $f(x, y, z) = h(x, y) - z$, so haben ∇f und H_f die Gestalt $\nabla f(x, y, z) = (\nabla h(x, y), -1)$ und

$$H_f = \begin{pmatrix} h_{xx} & h_{xy} & 0 \\ h_{yx} & h_{yy} & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Beispiel 12.2 *Bei der Darstellung der Funktion $f_1(x, y, z) = 0$ in Aufgabe 12.6 wurde der Umriß auf die hier beschriebene Art bestimmt.*

Aufgabe 12.7 *a) $f(x, y, z) = (1 - x)(1 + x)(1 - y)(1 + y)(1 - z)(1 + z) - c = 0, \qquad c > 0.$*
Die Fläche $f(x, y, z) = 0$ beschreibt (für festes c) eine "glatte und konvexe Approximation" des Würfels mit den Ecken $(\pm 1, \pm 1, \pm 1)$. (vgl. HA,FE'...) Stelle diese Approximation dar.
b) Gib analoge Approximationen eines Oktaeders an und stelle sie dar.

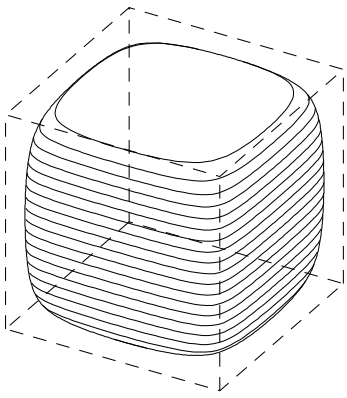
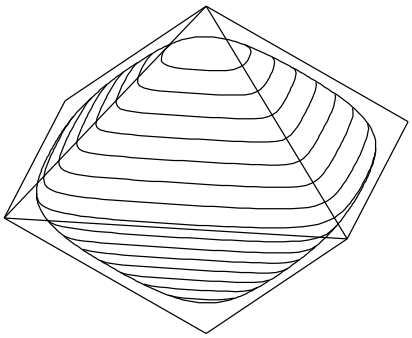


Abbildung 12.10: Approximation eines Würfels



Approximation eines Oktaeders (Aufgabe 12.7)

Aufgabe 12.8 *Die Gleichung*
a) $f(x, y, z) = (1 - x^2 - y^2 - z^2)z - c = 0,\ c > 0$, beschreibt eine glatte Approximation einer Halbkugel bzw.
b) $f(x, y, z) = (1 - x^2 - z^2)(1 - y^2 - z^2) - c = 0,\ c > 0$, beschreibt eine glatte Approximation des Durchschnitts zweier Zylinder. (vgl. HA,FE'...)
Stelle die Flächen für geeignete c dar.

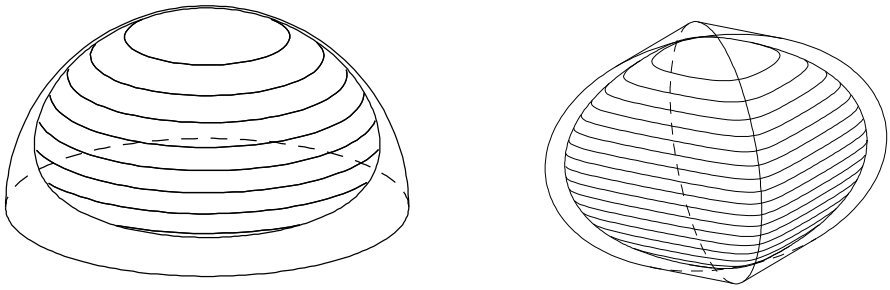


Abbildung 12.11: zu Aufgabe 12.8

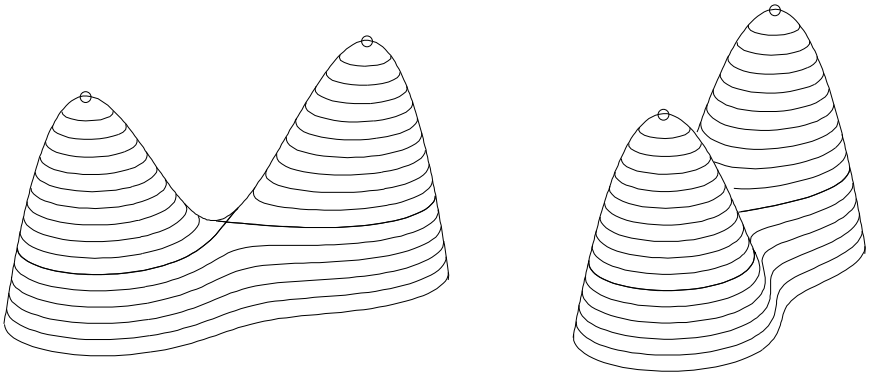


Abbildung 12.12: Cassini-Fläche

Aufgabe 12.9 *Die Gleichung* $z = -(x^2 + y^2)^2 + 2(x^2 - y^2) - 1$ *beschreibt eine Fläche, deren Höhenlinien Cassini-Kurven (s. 6.2) sind.*
Stelle die Fläche (incl. Umriß) dar.

Bei **Zentralprojektion** läßt sich der Umriß einer impliziten Fläche $f(\mathbf{x}) = 0, \mathbf{x} = (x, y, z)$, als Schnitt mit der impliziten Fläche $g(\mathbf{x}) := \nabla f(\mathbf{x}) \cdot (\mathbf{z} - \mathbf{x}) = 0$ auffassen. Zum Schnitt dieser Flächen benötigt man ∇g. Hier ergibt sich

$$\nabla g(\mathbf{x}) = H_f(\mathbf{x}) \cdot (\mathbf{z} - \mathbf{x}) - \nabla f(\mathbf{x}),$$

wobei H_f die HESSE-Matrix von f ist (vgl. 12.2.3).

Beispiel 12.3 *CASSINI-Fläche*

12.2.4 Schnitt einer impliziten mit einer parametrisierten Fläche

Es sei $f(x, y, z) = 0$ die Gleichung einer (impliziten) Fläche und $\mathbf{S}(s, t)$ die Parameterdarstellung einer zweiten Fläche. Schnittpunkte beider Flächen genügen dann der Gleichung $F(s, t) := f(\mathbf{S}(s, t)) = 0$. Mit dem Algorithmus für ebene implizite Kurven aus Kap. 6 lassen sich Parameter von Schnittpunkten und aus der gegebenen Parameterdarstellung $\mathbf{S}(s, t)$ Schnittpunkte selbst bestimmen.

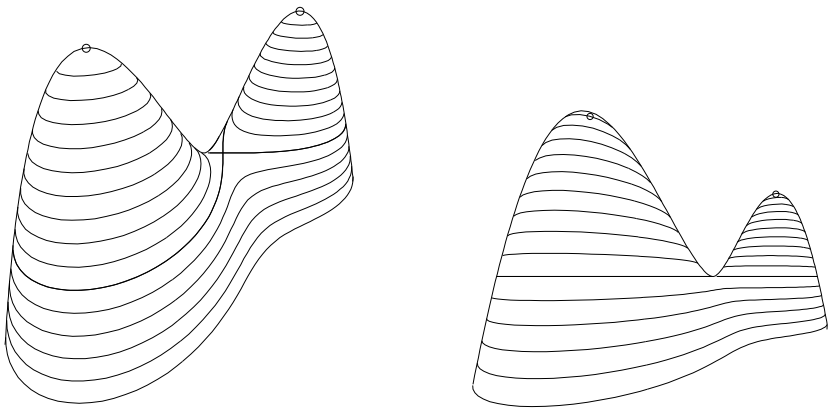


Abbildung 12.13: Zentralprojektion einer impliziten Fläche mit Umriß

Bemerkung:
Um eine *gleichmäßige Verteilung der Punkte* im Objektraum mit dem ungefähren Abstand δ zu erhalten, löst man das (quadratische) Gleichnungssystem

$$\begin{aligned} \mathbf{S}_s^2 \Delta s^2 + 2\mathbf{S}_s\mathbf{S}_t \Delta s \Delta t + \mathbf{S}_t^2 \Delta t^2 &= \delta^2 \\ (\nabla f \cdot \mathbf{S}_s)\Delta s + (\nabla f \cdot \mathbf{S}_t)\Delta t &= 0 \end{aligned}$$

und nimmt $(s_i + \Delta s, t_i + \Delta t)$ als neuen Startpunkt für die Methode des steilsten Weges (vgl. 6.2).

12.2.5 Umriß einer parametrisierten Fläche

Ist $\mathbf{S}(s,t)$ eine differenzierbare Parameterdarstellung einer Fläche, so ist $\mathbf{n}(s,t) := \mathbf{S}_s(s,t) \times \mathbf{S}_t(s,t)$ eine Flächennormale im Punkt $\mathbf{S}(s,t)$, falls $\mathbf{S}_s(s,t)$ und $\mathbf{S}_t(s,t)$ linear unabhängig sind. Beschreibt $\mathbf{n}_0$ die (negative) Projektionsrichtung, so ergeben sich potentielle Umrißpunkte bei **Parallelprojektion** aus der Gleichung $f(s,t) := \mathbf{n}(s,t) \cdot \mathbf{n}_0 = 0$. D.h. auch hier läßt sich der Verfolgungs-Algorithmus aus 6.2 verwenden, um zunächst im Parameterbereich den Umriß zu bestimmen.

Beispiel 12.4 *Für* $\mathbf{S}(s,t) = (s,t,1-s^4-t^4)$ *ist* $\mathbf{S}_s(s,t) = (1,0,-4s^3), \mathbf{S}_t(s,t) = (0,1,-4t^3), \mathbf{n}(s,t) = (4s^3, 4t^3, 1)$ *und damit* $f(s,t) = 4s^3 n_{0x} + 4t^3 n_{0y} + n_{0z}$.

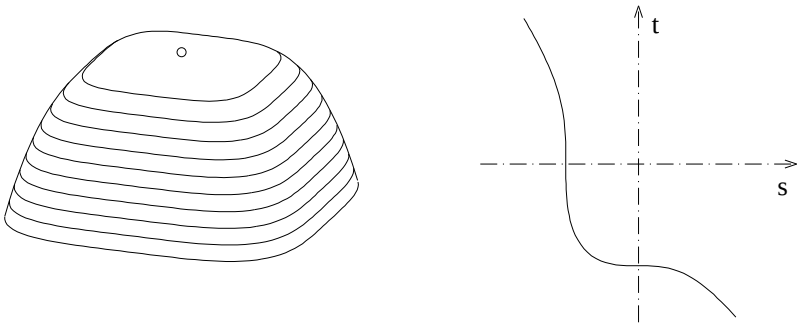
Bei **Zentralprojektion** ist ein Punkt $\mathbf{x}(s,t)$ ein Umrißpunkt, wenn

$$f(s,t) := \mathbf{n}(s,t) \cdot (\mathbf{z} - \mathbf{x}(s,t)) = 0$$

ist. Lösungen dieser Gleichung findet man mit dem Algorithmus für implizite Kurven in 6.2.

12.2.6 Schnitt zweier parametrisierter Flächen

Die Bestimmung der Schnittkurve von Flächen, die durch Parameterformen gegeben sind, ist schwieriger als im impliziten Fall, da nicht nur die Koordinaten von Punkten, sondern auch jeweils Parameter von Flächenpunkten bestimmt werden müssen. Der hier angegebene Verfolgungsalgorithmus für parametrisierte Flächen stammt im Wesentlichen aus BA,KE'90.
Ein wesentlicher Teil des Algorithmus ist die Bestimmung des Lotfußpunktes des Lotes von einem Punkt P auf eine Fläche Φ.



a) b)

Abbildung 12.14: Umriß a) Fläche mit Umriß b) im Parameterbereich ($f(s,t)=0$)

12.2.6.1 Lotfußpunkt und Flächeninversion.

Gegeben: $\Phi : \mathbf{x} = \mathbf{S}(s,t)$ sei eine differenzierbare Fläche mit stets linear unabhängigen Tangenten vektoren $\mathbf{S}_s, \mathbf{S}_t$.$P : \mathbf{p}$ sei ein Punkt und L_0, s_0, t_0 eine 0-te Näherung des Lotfußpunktes L und seiner Parameter.

Gesucht: Lotfußpunkt $L : \mathbf{l}$ des Lotes von P auf Φ.

Die Differenz $\mathbf{p} - \mathbf{l}$ muß senkrecht auf der Tangentialebene in L stehen, d.h. die Parameter von L müssen Lösung des folgenden Gleichungssytems sein:

$$f(s,t) := (\mathbf{S}(s,t) - \mathbf{p}) \cdot \mathbf{S}_s(s,t) = 0,\ g(s,t) := (\mathbf{S}(s,t) - \mathbf{p}) \cdot \mathbf{S}_t(s,t) = 0$$

Dieses Gleichungssystem in den zwei Variablen s,t läßt sich mit Hilfe der Startwerte s_0, t_0
entweder durch eine Newton-Iteration mit $\mathbf{F} := (f,g)$ und

$$\mathbf{F}' = \begin{pmatrix} \mathbf{S}_s^2 & \mathbf{S}_s \cdot \mathbf{S}_t \\ \mathbf{S}_s \cdot \mathbf{S}_t & \mathbf{S}_t^2 \end{pmatrix} + \begin{pmatrix} \mathbf{S}_{ss} \cdot (\mathbf{S} - \mathbf{p}) & \mathbf{S}_{st} \cdot (\mathbf{S} - \mathbf{p}) \\ \mathbf{S}_{ts} \cdot (\mathbf{S} - \mathbf{p}) & \mathbf{S}_{tt} \cdot (\mathbf{S} - \mathbf{p}) \end{pmatrix}$$

oder durch sukzessives Lotefällen auf Tangentialebenen lösen.

Sukzessives Lotefällen auf Tangentialebenen:

Es sei L_k, s_k, t_k die k-te Näherung für L.
Wir nähern die Fläche $\mathbf{S}(s,t)$ in der Umgebung von L_k durch die Tangentialebene

$$\varepsilon_k : \mathbf{x}(\Delta s, \Delta t) = \mathbf{S}(s_k, t_k) + \Delta s \mathbf{S}_s(s_k, t_k) + \Delta t \mathbf{S}_t(s_k, t_k)$$

in L_k an und bestimmen Δs und Δt so, daß $\mathbf{p} - \mathbf{x}(\Delta s, \Delta t)$ auf ε_k senkrecht steht: Setzen wir $\mathbf{e}_s := \mathbf{S}_s(s_k, t_k), \mathbf{e}_t := \mathbf{S}_t(s_k, t_k)$ so ergeben sich Δs und Δt aus dem linearen Gleichungssystem

$$\begin{aligned} (\mathbf{p} - \mathbf{S}(s_k, t_k)) \cdot \mathbf{e}_s &= \Delta s \mathbf{e}_s^2 + \Delta t \mathbf{e}_s \cdot \mathbf{e}_t \\ (\mathbf{p} - \mathbf{S}(s_k, t_k)) \cdot \mathbf{e}_t &= \Delta s \mathbf{e}_s \cdot \mathbf{e}_t + \Delta t \mathbf{e}_t^2 \end{aligned}$$

Wir setzen $s_{k+1} = s_k + \Delta s, t_{k+1} = t_k + \Delta t, \quad L_{k+1} = \mathbf{S}(s_{k+1}, t_{k+1})$. Ist der Abstand von L_{k+1} zu L_k klein "genug", so setzt man $L = L_{k+1}$.

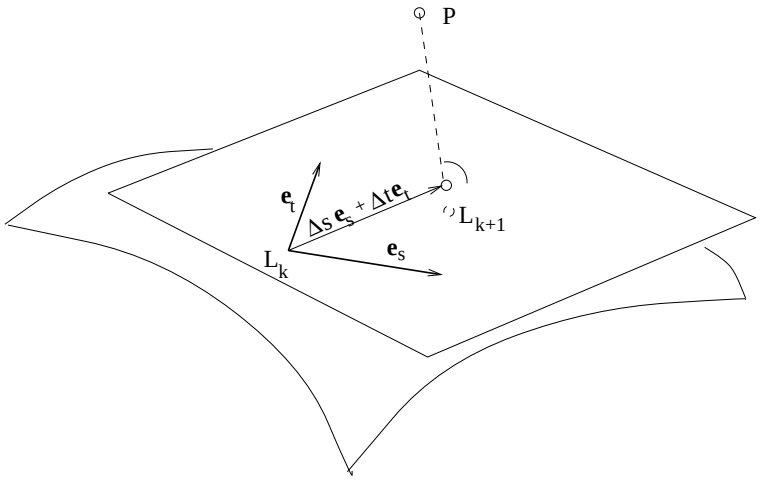


Abbildung 12.15: Lot auf eine Tangentialebene

Bemerkung:
Die Methode zur Bestimmung eines Lotfußpunktes läßt sich auch zur **Flächeninversion** verwenden. Bei der Flächeninversion sucht man zu einem Punkt $P_0 : \mathbf{p}_0$ der Fläche $\mathbf{S}(s,t)$ Parameter s_0, t_0 mit $\mathbf{p}_0 = \mathbf{S}(s_0, t_0)$. ($s_0, t_0$ müssen nicht eindeutig bestimmt sein.)

12.2.6.2 Der Verfolgungs-Algorithmus

Voraussetzungen:
Es seien $\mathbf{S}_1(s,t), \mathbf{S}_2(u,v)$ Parameterdarstellungen zweier Flächen mit folgenden Eigenschaften:
(i) $\mathbf{S}_1$ und $\mathbf{S}_2$ sind differenzierbar und
(ii) die Flächentangenten $\mathbf{S}_{1s}, \mathbf{S}_{1t}$ bzw. $\mathbf{S}_{2u}, \mathbf{S}_{2v}$ bzw. die Flächennormalen $\mathbf{n}_1 := \mathbf{S}_{1s} \times \mathbf{S}_{1t}, \mathbf{n}_2 := \mathbf{S}_{2u} \times \mathbf{S}_{2v}$ sind auf und in der "Nähe" der Schnittkurve linear unabhängig.

Durchführung:

(1) Bestimmung eines Startpunktes P_0 und seiner Parameter s_0, t_0, u_0, v_0:
Die Parameterbereiche der Flächen werden mit geeigneten Unterteilungsnetzen (äquidistant oder "angepaßt") überzogen. Jedem Rechteck dieser Netze werden Quader im $\mathbb{R}^3$ zugeordnet, die die zugehörigen Flächenpunkte enthalten. Schnitte der Quader der einen Fläche mit den Quadern der anderen Fläche liefern Näherungspunkte und Näherungsparameter für mögliche Schnittpunkte. Durch mehrmalige Iteration (s.u.) erhält man gegebenfalls einen Startpunkt Q_0 und seine Parameter.

(2) $P_{i+1}, s_{i+1}, t_{i+1}, u_{i+1}, v_{i+1}$ aus P_i, s_i, t_i, u_i, v_i:

(2.1) 0-te Näherung Q_0 für P_{i+1}:
Berechnung von $\mathbf{S}_{1s}(s_n,t_n), \mathbf{S}_{1t}(s_n,t_n), \mathbf{S}_{2u}(u_n,v_n), \mathbf{S}_{2v}(u_n,v_n)$, der Normalen $\mathbf{n}_1(s_n,t_n), \mathbf{n}_2(u_n,v_n)$ und der Kurveneinheitstangente $\mathbf{t} := \mathbf{n}_1 \times \mathbf{n}_2 / \| \dots \|$. $Q_0 = P_i + \delta\mathbf{t}, \delta$ Schrittweite.

(2.2) Lotfußpunkte A_0 bzw. B_0 der Lote von Q_0 auf die Flächen (s.o.).
(Als 0-te Näherung der Lotfußpunkte wählt man P_i und seine Parameter.)

(2.3) $Q_{k+1}, A_{k+1}, B_{k+1}$ aus Q_k, A_k, B_k:
Sind $\varepsilon_1, \varepsilon_2$ die Tangentialebenen in den Punkten A_k, B_k und ε_3 die Ebene durch den Mittelpunkt der Strecke A_kB_k mit der Normalen $\mathbf{n}_3 := \mathbf{n}_1 \times \mathbf{n}_2$ so ist Q_{k+1} der Schnittpunkt

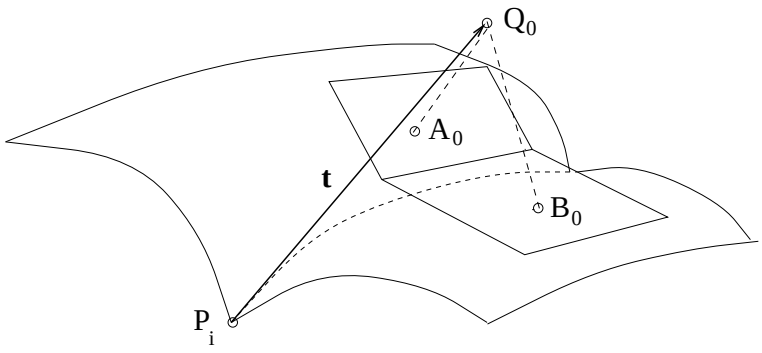


Abbildung 12.16: Zum Verfolgungs-Algorithmus

der Ebenen $\varepsilon_1, \varepsilon_2, \varepsilon_3$:

$$Q_{k+1} = \frac{(\mathbf{n}_1 \cdot A_k)(\mathbf{n}_2 \times \mathbf{n}_3) + (\mathbf{n}_2 \cdot B_k)(\mathbf{n}_3 \times \mathbf{n}_1) + (\mathbf{n}_3 \cdot \frac{A_k+B_k}{2})(\mathbf{n}_1 \times \mathbf{n}_2)}{\mathbf{n}_1 \cdot (\mathbf{n}_2 \times \mathbf{n}_3)}$$

A_{k+1}, B_{k+1} sind die Lotfußpunkte zu Q_{k+1} (s.o.).

(2.4) Ist der Abstand der Lotfußpunkte A_{k+1}, B_{k+1} klein genug, so setzt man $P_{i+1} = Q_{k+1}$. Als Parameter von P_{i+1} auf den Flächen nimmt man die Parameter der Lotfußpunkte A_{k+1} (1.Fläche) und B_{k+1} (2. Fläche).

Beispiel 12.5 $\mathbf{S}_1(s,t) = (s,t,s^2+t^2)$, $\mathbf{S}_2(u,v) = (u,v,2-u^4-v^4)$.

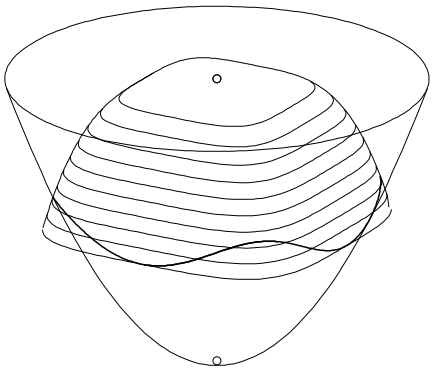


Abbildung 12.17: zu Beispiel 12.5

12.2.7 Die Normalform einer Fläche

Analog zur Normalform einer Kurve (s. Abschnitt 6.9) wollen wir hier die Normalform einer Fläche erklären. Sie ist eine Verallgemeinerung der HESSE Normalform für Ebenen.

Es sei Φ eine Fläche mit den folgenden Eigenschaften:

(NF1) Φ ist glatt und

(NF2) es gibt ein Gebiet $D \subset \mathbb{R}^3$ in der Umgebung von Φ so, daß es für jeden Punkt $\mathbf{x} \in D$ möglich ist, einen (eindeutigen) *Lotfußpunkt* (Punkt auf Φ mit minimaler Distanz zu $\mathbf{x}$) zu bestimmen.

Ist h die zugehörige orientierte Distanzfunktion, so ist die Gleichung $h(\mathbf{x}) = 0$ eine implizite Darstellung der Fläche Φ und heißt die *Normalform* von Φ.

Ist Φ eine parametrisierte Fläche, so verwendet man für die Berechnung von $h(\mathbf{x})$ und $\nabla h(\mathbf{x})$ den Lotfußpunkt-Algorithmus aus dem vorigen Abschnitt.

Der **Lotfußpunkt**–Algorithmus für **implizite** Flächen verwendet (analog zum Kurvenfall in Abschnitt 6.8) die Prozedur `surfacepoint`:

(SP1) Es sei $Q_0 = \mathbf{q}_0$ ein Punkt in der Nähe der impliziten Fläche $\Phi : f(\mathbf{x}) = 0$.

(SP2) Iteration entlang des steilsten Weges:
wiederhole $\mathbf{q}_{i+1} = \mathbf{q}_i - \dfrac{f(\mathbf{q}_i)}{\nabla f(\mathbf{q}_i)^2} \nabla f(\mathbf{q}_i)$ (Newtoschritt für $g(t) := f(\mathbf{q}_i + t\nabla f(\mathbf{q}_i))$)
bis $\|\mathbf{q}_{i+1} - \mathbf{q}_i\|$ klein genug ist.
(oder andere Abbruchbedingung.)

Bestimmung des Lotfußpunktes $\mathbf{x}_0$ (zu $\mathbf{x}$):

(0) $\mathbf{s}_0 = \texttt{surfacepoint}(\mathbf{x})$

(1) wiederhole $\mathbf{t}_{i+1} = \mathbf{x} - \dfrac{(\mathbf{x} - \mathbf{s}_i) \cdot \nabla f(\mathbf{s}_i)}{\nabla f(\mathbf{s}_i)^2} \nabla f(\mathbf{s}_i)$ (Lotfußpunkt auf der Tangentialebene),
$\mathbf{s}_{i+1} = \texttt{surfacepoint}(\mathbf{t}_{i+1})$ (Flächenpunkt)
bis $\|\mathbf{s}_{i+1} - \mathbf{s}_i\|$ klein "genug" ist.
$\mathbf{x}_0 = \mathbf{s}_{i+1}$.

Bemerkung:
Wie im Kurvenfall kann es bei den Lotfußpunkt-Algorithmen, die wiederholt Lote auf Tangentialebenen fällen, zu Konvergenzproblemen kommen, falls der Punkt $\mathbf{x}$, zu dem der Lotfußpunkt bestimmt werden soll, zu "weit" von der Fläche entfernt liegt und die Fläche in diesem Bereich "wesentlich" gekrümmt ist.

Eine erste Vereinfachung bietet die Normalform beim Schneiden von Flächen. Da bei Verwendung der Normalform jede Fläche als implizite Fläche aufgefaßt werden kann, genügt zum Schneiden von Flächen ein einziger Algorithmus, nämlich der für implizite Flächen.
Weitere Anwendungen sind in den Kapiteln 14 und 15 enthalten.

Kapitel 13

HIDDENLINE-ALGORITHMUS FÜR NICHTKONVEXE POLYEDER, DARSTELLUNG PARAMETRISIERTER FLÄCHEN

13.1 Der Hiddenline-Algorithmus

Der Hiddenline-Algorithmus `pp_convex_polyh` (vgl. Kap. 3) bzw. `cp_convex_polyh` (vgl. Kap. 4) ist nur auf konvexe Polyeder anwendbar. Um auch allgemeinere Konfigurationen von Kanten und ebenen Polygonen unter Berücksichtigung der Sichtbarkeit darstellen zu können, überlegen wir uns jetzt einen allgemeineren Algorithmus.

Voraussetzungen und **Vorgaben**:

Gegeben: a) Strecken (Kanten) $\{e_1, e_2, e_3, \ldots\}$
b) **ebene, konvexe** $N-$**Ecke** (Facetten) $\{f_1, f_2, f_3, \ldots\}$
Falls gewisse $N-$Ecke nicht konvex sind, zerlegt man sie in mehrere konvexe Teile. Die hierzu benötigten zusätzlichen Kanten ignoriert man dann beim Zeichnen. Die Kanten, die auf Sichtbarkeit untersucht werden sollen, müssen nicht unbedingt einem der $N-$Ecke angehören. (s. Haus mit Fenster und Tür am Ende dieses Paragraphen.)

Idee:

(1) Falls die Facetten orientiert sind, sondert man mit Hilfe des schnellen "Normalen-Tests" die unsichtbaren Facetten und deren unsichtbaren Kanten aus.

(2) Der einfache "Fenster-Test" unterdrückt den aufwendigeren Sichtbarkeitstest (3)-(5) bei "offensichtlichen" Fällen.

(3) In der Bildtafel wird untersucht, ob das Bild einer Kante teilweise innerhalb des Bildes einer Facette (konvexes Polygon) liegt.

(4) Falls dies der Fall ist, wird mit Hilfe eines Testpunktes (im Raum) festgestellt, ob die Kante auf derselben Seite der Ebene der Facette wie das Projektionszentrum liegt oder nicht.

(5) Wird die Kante teilweise von der Facette verdeckt, werden die sichtbaren Teile bestimmt und abgespeichert.

(6) Nachdem die Kante gegen alle Facetten auf diese Weise untersucht worden ist, werden die sichtbaren Teile gezeichnet.

Zur Verwirklichung dieser Idee benötigen wir die folgenden drei Unterprogramme:

a) `is_line_convex_polygon`: berechnet den Schnitt einer Strecke mit einem konvexen Polygon.

b) `intmint`: bestimmt die Differenz $[a,b]\backslash[c,d]$ zweier Intervalle von $\mathbb{R}$.

c) `cp_vts3d_vts2d_spez`: ist eine gerinfügige Erweiterung von `cp_vts3d_vts2d`. In einem zusätzlichen Parameter (`pdist`: r_array) werden die Abstände der Punkte von der Bildtafel ($z-$Koordinaten der Punkte im System $(H;\mathbf{e}_1,\mathbf{e}_2,\mathbf{n}_0)$) mitgeliefert.

```
procedure is_line_convex_polygon(p1,p2 : vt2d; p_pol : vts2d_pol; np : integer;
                                          var t1,t2 : real; var ind : integer);
{Berechnet die Parameter t1,t2 der Schnittpunkte der Strecke p1,p2 mit dem konvexen
 Polygon p_pol[0],...p_pol[np]. ind=0 bzw. 2 : Strecke  innerhalb bzw. ausserhalb,
 ind=1: sonst. vts2d_pol: array[0..npfmax] of vt2d. }
{***********}
procedure intmint(a,b,c,d: real; var e1,f1,e2,f2: real; var ind: integer);
 {Berechnet die Intervall-Differenz  [a,b] \ [c,d].
 ind=0: leer, ind=1: 1 Interv., ind=2: 2 Interv.}
{***********}
procedure cp_vts3d_vts2d_spez(var p: vts3d; n1,n2 : integer; var pp : vts2d;
                                                             var pdist : r_array);
 {Zentralprojektion (Koordinaten) einer Punktreihe.
 pdist[i] : Distanz des Punktes p[i]  von der Bildtafel.}
{***********}
```

Zur **Datenstruktur:**
Für jede **Facette** (Polygon) werden in dem *record* `face_dat` die folgenden Daten gespeichert:

1) `npf, nef`: Anzahl der Punkte bzw. Kanten.

2) `fp[1]`, ... `fp[npf]`: Erster, zweiter, ... Punkt der Facette,
`fe[1]`, ... `fe[nef]`: Erste, zweite, ... Kante der Facette.
(Falls die Facetten orientiert sind, müssen sie (von außen gesehen) gegen den Uhrzeiger angeordnet sein.)

3) `box`: enthält minimale und maximale $x-$Werte bzw. $y-$Werte der Bildpunkte der Facette.
`box.zmax`: maximaler Abstand der Punkte von der Bildtafel.

4) Den Normalenvektor `nv` und die Zahl `d` der Gleichung $\mathbf{n}_v \cdot \mathbf{x} = d$ der Ebene, die die Facette enthält.

5) `discentre`: Abstand des Zentrums von der Ebene, die die Facette enthält.

6) `vis`: `vis=true` bzw. `vis=false`, falls die orientierte Facette sichtbar bzw. unsichtbar ist.

Für jede **Kante** legen wir das *record* `edge_dat` an:

1) `ep1,ep2`: Erster bzw. zweiter Punkt der Kante.

2) `vis`: `vis=true` bzw. `vis=false`, falls die Kante einer sichtbaren bzw. unsichtbaren Facette angehört.

Alle records `face_dat` bzw. `edge_dat` sind zu dem *array* `face` bzw. `edge` zusammengefaßt.

Der Hiddenline-Algorithmus ist in dem folgenden Unterprogramm `cp_lines_before_convex_faces` enthalten.

Zum Unterprogramm `cp_lines_before_convex_faces (oriented_faces,is_permitted)`:
`oriented_faces=true`: Die Facetten sind orientiert. Es wird der schnelle Normalentest durchgeführt.
`is_permitted=true`: Die Kanten dürfen die Facetten schneiden. (langsam!)

(0) Im Hauptprogramm müssen zur Verfügung stehen:

a) `np, nf, ne` : Anzahl der Punkte, Facetten, Kanten,
b) die Koordinaten der Punkte $P_i : \mathbf{p}_i, \quad i = 1, \ldots, np$,
c) für jede Kante die Daten: Anfangs- und Endpunkt `ep1,ep2`,
d) für jede Facette die Daten:
`npf,nef` : Anzahl der Punkte bzw. Kanten der Facette,
`fp[1]`, ... `fp[npf]` : die Punkte der Facette, `fe[1]`, ... `fe[npf]` : die Kanten der Facette,
die Gleichung $\mathbf{n}_v\mathbf{x} = d$ der zugehörigen Ebene.

(1) Es werden

a) die Bildpunkte und deren Abstand zum Projektions-Zentrum berechnet,
b) für jede Facette
der "Abstand" (`discentre`) des Zentrums von dieser Ebene,
die Fensterdaten des Bildpolygons in `box (xmin, xmax, ymin, y max)`,
der maximale Abstand (`box.zmax`) der Facettenpunkte von der Bildebene berechnet.

b') der Normalentest für orientierte Facetten durchgeführt.
Ist die Facette sichtbar so wird sie und ihre Kanten als sichtbar erklärt.

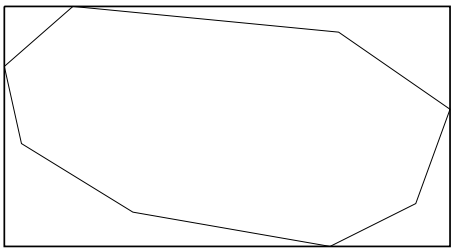


Abbildung 13.1: Fenster eines Polygons

(2) Untersuchung der i-ten Kante e_i, gegenüber der j-ten Facette f_j, falls beide sichtbar (vis=true) sind:

a) Berechnung des Fensters (`xemin, xemax, yemin, yemax`) und des minimalen Abstandes (`zemin`) von der Bildtafel der Kante.
b) Ist die Kante eine Kante der j-ten Facette ? (ja: $\rightarrow$ nächste Facette.)
c) Falls `box.zmax > zemin` und sich Kantenfenster und Facettenfenster überlappen: Schnitt der Bildkante mit dem (konvexen) Bildpolygon der j-ten Facette.

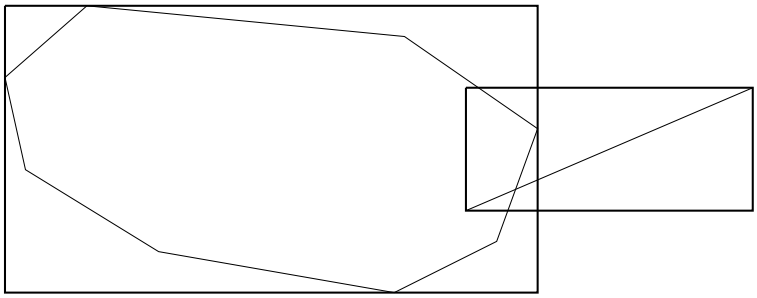


Abbildung 13.2: Fenstertest

d) Falls ein Teil der Bildstrecke in dem Polygon liegt:

d1) falls Schnitt Kante-Facette NICHT zugelassen ist (`is_permitted`=false):
Test mit Testpunkt $\mathbf{p}_t$, ob die Kante (im Raum) vor oder hinter der Ebene der j-ten Facette liegt.

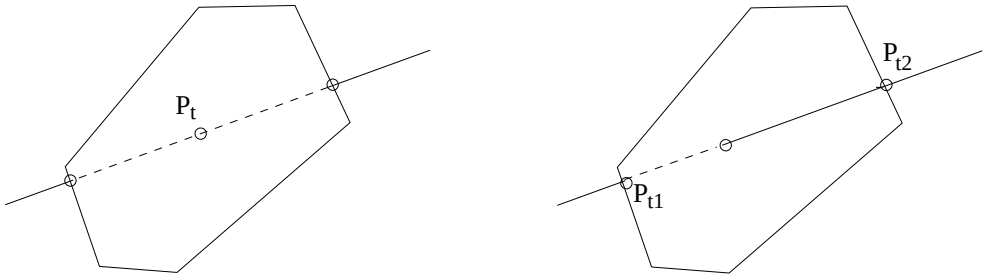


Abbildung 13.3: Fall d1) bzw. d2)

d2) falls Schnitt Kante-Facette zugelassen ist (`is_permitted`=true):
Mit Hilfe zweier Testpunkte $\mathbf{p}_{t1}, \mathbf{p}_{t2}$ wird die Lage der Kante relativ zur Facette festgestellt und der Teil der Kante, der vor der Facette liegt, bestimmt.

e) Falls die Kante teilweise hinter der Facette liegt:
Die Parameterintervalle $[par1(l), par2(l)]$ der noch sichtbaren Teilstrecken werden berechnet und abgespeichert.

(3) Ist die Kante e_i gegen alle Facetten getestet, werden die sichtbaren Teilstrecken gezeichnet.

```
procedure cp_lines_before_convex_faces(oriented_faces,is_permitted : boolean);
{Projiziert und zeichnet Kantenteile VOR (orientierten) ebenen n-Ecken.
 oriented_faces=true: die Flaechen sind orientiert,
 is_permitted=true: Kanten duerfen die Flaechen schneiden.
 Aus dem Hauptprogramm muessen bereitstehen:
 np, ne, nf: Anzahl der Punkte, Kanten, Flaechen,
 p[1],...,p[np] : Punkte,
 face[i].fp[k] (face[i].fe[k]): k-ter Punkt (k-te Kante) in i-ter Flaeche,
 face[i].npf (face[i].nef): Anzahl der Punkte (Kanten) in der i-ten Flaeche,
 edge[i].ep1 (edge[i].ep2): Anfangs-(End-)Punkt der i-ten Kante,
 face[i].nv,face[i].d: Koeffizienten der Ebenengleichung.}
 label 5,10;
```

```
 var  t1,t2,tt,dispt,xemin,xemax,yemin,yemax,zemin,cc,
      tt1,tt2,ts,disp1,disp2,dispt1,dispt2,test1,test2 : real;
      i,j,k,l,m,ip,i1,i2,nseg,nseg0,ind : integer;
      p1,p2,pd,pt, pt1,pt2,ps : vt3d;
      pp1,pp2,ppd,qq1,qq2,pps : vt2d;
      pp : vts2d;   error : boolean;
      par1,par2,pa1,pa2,pa3,pa4 : r_array_seg;  {r_array_seg : s. HP    }
      p_pol : vts2d_pol;                        {vts2d_pol   : s. HP    }
      nt : i_array_seg;                         {i_array_seg : s. HP    }
{******}
 begin
   if oriented_faces then
      begin
      for i:= 1 to nf do face[i].vis:= false;
      for i:= 1 to ne do edge[i].vis:= false;
      end;
{ Koordinaten der Bildpunkte: }
   cp_vts3d_vts2d_spez(p,1,np, pp,pdist);
{ Fenster und Normalentest fuer die Flaechen: }
   for i:= 1 to nf do
       begin
       with face[i] do
            begin
            discentre:= scalarp3d(nv,centre) - d;
            if ((discentre>=0) or (not oriented_faces)) then
               begin
               if oriented_faces then
                  begin      { Normalentest: Flaeche ist sichtbar }
                  vis:= true;
                  for k:= 1 to nef do edge[fe[k]].vis:= true;
                  end;
               with box do
                    begin
                    with pp[fp[1]] do   { Anfangswerte }
                         begin xmin:= x; xmax:= x;  ymin:= y;  ymax:= y; end;
                    zmax:= pdist[fp[1]];
                    for k:= 2 to npf do
                        begin
                        with pp[fp[k]] do     { Flaechenfenster }
                             begin
                             xmin:= min(xmin,x);  ymin:= min(ymin,y);
                             xmax:= max(xmax,x);  ymax:= max(ymax,y);
                             end;  { with }
                        zmax:= max(zmax,pdist[fp[k]]);
                        end;  { for k }
                    end;  { with box }
               end;  { if }
            end; { with  face }
       end; { for i }
{ Test und Zeichnen der Kanten(-teile):}
   for i:= 1 to ne do                           { Beginn der KANTENschleife }
       begin
       if not edge[i].vis then goto 10;
       par1[1]:= 0;  par2[1]:= 1;  nseg:= 1;    { : 1 sichtb. Anfangsintervall}
```

```
 { Punkte und Fenster der Kante : }
      i1:= edge[i].ep1;  i2:= edge[i].ep2;
      p1:= p[i1];  p2:= p[i2];  pp1:= pp[i1]; pp2:= pp[i2];
      xemax:= max(pp1.x,pp2.x);   yemax:= max(pp1.y,pp2.y);
      xemin:= min(pp1.x,pp2.x);   yemin:= min(pp1.y,pp2.y);
      zemin:= min(pdist[i1],pdist[i2]);
      for j:= 1 to nf do                      { Beginn der FLAECHENschleife }
          with face[j] do
          begin
          if not vis and oriented_faces then goto 5;
 { Fenstertest mit j-ter Flaeche: }
          if (xemax<=box.xmin) or (xemin>=box.xmax)
            or (yemax<=box.ymin) or (yemin>=box.ymax) or (box.zmax<=zemin)
              then   goto 5; { Kante wird nicht von j-ter Flaeche verdeckt }
 { Kante i Kante von j-ter Flaeche ?:}
          for k:= 1 to nef do  if i=fe[k] then goto 5; { naechste Flaeche}
 { Schnitt der Kante mit dem (konvexen) Flaechenpolygon in der Bildtafel: }
          for k:= 1 to npf do p_pol[k]:= pp[fp[k]];  p_pol[0]:= p_pol[npf];
          is_line_convex_polygon(pp1,pp2,p_pol,npf,t1,t2,ind);
          if ind=2 then  goto 5 ;   {Kante nicht verdeckt=> naechste Flaeche }
          if not is_permitted then  {Kante darf die Flaeche NICHT durchdr.}
 { Testpunkt und Test, ob Kante vor oder hinter der Flaeche : }
             begin
             tt:= ( max(0,t1) + min(1,t2) )/2;     diff3d(p2,p1, pd);
             lcomb2vt3d(1,p1, tt,pd, pt);
             dispt:= scalarp3d(nv,pt) - d;    {Distanz Testpunkt-Ebene}
             if (dispt*discentre>0)  then   goto 5;
               { Kante vor der Ebene => naechste Flaeche }
             end;  { if not is_permitted }
          if is_permitted then  { Kante darf die Flaeche durchdringen }
{ Bestimmung des Teils der Kante, der hinter der Flaeche liegt: }
             begin
             tt1:= max(0,t1);  tt2:= min(1,t2);   diff3d(p2,p1, pd);
             lcomb2vt3d(1,p1,tt1,pd, pt1);      lcomb2vt3d(1,p1,tt2,pd, pt2);
             dispt1:= scalarp3d(nv,pt1) - d;    {Distanz 1.Testpunkt-Ebene}
             dispt2:= scalarp3d(nv,pt2) - d;    {Distanz 2.Testpunkt-Ebene}
             test1:= dispt1*discentre;          test2:= dispt2*discentre;
             if ((test1>=0) and (test2>=0))  then   goto 5;
            { Kante vor der Ebene => naechste Flaeche }
             if dispt1*dispt2<0 then
                begin
                disp1:= scalarp3d(nv,p1) - d;   disp2:= scalarp3d(nv,p2) - d;
                ts:= disp1/(disp1-disp2);
                lcomb2vt3d(1,p1,ts,pd, ps);  cp_vt3d_vt2d(ps,pps);
                diff2d(pp2,pp1, ppd); {Zentralproj. ist keine lin. Abb. !!!}
                ts:= (scalarp2d(pps,ppd)-scalarp2d(pp1,ppd))/scalarp2d(ppd,ppd);
                if test1<0 then t2:= ts else t1:= ts;
                end;
             end; { if is_permitted}
             for l:= 1 to nseg do   {weiterhin sichtbare Intervalle:}
                 intmint(par1[l],par2[l],t1,t2,
                                 pa1[l],pa2[l],pa3[l],pa4[l],nt[l]);
             nseg0:= nseg; m:= 0;
             for l:= 1 to nseg0 do          { neue Intervallteilung }
```

```
                  begin
                  if nt[l]=0 then  nseg:= nseg-1   { 1 Segment weniger }
                    else
                     begin
                     m:= m+1;
                     par1[m]:= pa1[l];  par2[m]:= pa2[l];
                     if nt[l]=2 then      { 1 Segment mehr }
                        begin
                        m:= m+1;  nseg:= nseg+1;
                        par1[m]:= pa3[l];  par2[m]:= pa4[l];
                        end;
                     end;  { if }
                  end; { for l }
            if nseg<1 then goto 10;
5:          end; { with }
       for k:=1 to nseg do   { Zeichnen der sichtb. Segmente der i-ten Kante}
           begin
           diff2d(pp2,pp1, ppd);
           lcomb2vt2d(1,pp1, par1[k],ppd, qq1);
           lcomb2vt2d(1,pp1, par2[k],ppd, qq2);
           line2d(qq1,qq2,0);
           end;
10:    end; { for i (Kantenschleife) }
 end; { cp_lines_before_convex_faces }
{*************}
```

Beispiel 13.1 *Polyeder auf einem TORUS (orientierte Facetten)*

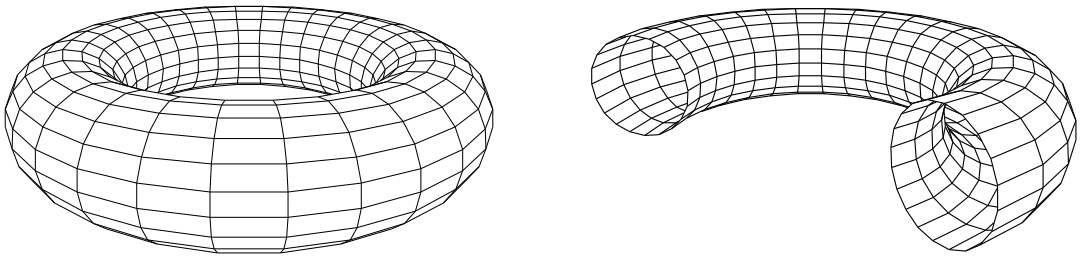


Abbildung 13.4: Torus (orient. Facetten) bzw. Torusteil (nicht orient. Facetten)

Beispiel 13.2 *Polyeder auf einem TORUS (nicht orientierte Facetten)*

13.2 Hilfsprogramme zum Hiddenline-Algorithmus

Um die Vorarbeit des Anwenders des Hiddeline-Algorithmus zu reduzieren, werden hier 4 Hilfsprogramme angegeben. Das erste Unterprogramm ist besonders für Polyeder, die drei restlichen zur Darstellung von parametrisierten Flächen geeignet.

13.2.1 Das Unterprogramm `aux_polyhedron`

Im Hauptprogramm müssen folgende Daten bereitstehen:

(1) `np,nf` (Anzahl der Punkte bzw. Facetten),

(2) die Punkte $P_i : \mathbf{p}_i,\ i = 1, \ldots,$ `np` der Facetten und Kanten,

(3) für jede Facette: `npf, nef` (Anzahl der Punkte bzw. Kanten dieser Facette),
`fp[1],...,fp[npf]` (Nummern der Punkte, die das Facettenpolygon bilden).

Das Unterprogramm `aux_polyhedron` numeriert die Kanten und berechnet

(1) `ne` (Anzahl der Kanten),

(2) für jede Kante: `ep1, ep2` (Nummern von Anfangs- und Endpunkt),

(3) für jede Facette: `fe[1],...,fe[npf]` (Nummern der Kanten des Facettenpolygons).

```
procedure aux_polyhedron;
{ne, nf : Anzahl der Punkte,Kanten,Flaechen
 face[i].npf   : Anzahl der Punkte (Kanten) der i-ten Flaeche
 edge.ep1, ep2 : Anfangs- bzw. Endpunkt der k-ten Kante
 face[i].fp[k] : k-ter Punkt der i-ten Flaeche (positiv orientiert!!!)
 face[i].fe[k] : k-te Kante der i-ten Flaeche
 ! Dieses UP berechnet aus nf,face[i].fp und face[i].npf:
 ne, face[i].fe[k], edge[i].ep1 und edge[i].ep2 !.}
```

Beispiel 13.3 *HÄUSER*
(Man beachte,

a) daß die Dachflächen nicht orientiert sind und

b) daß Kanten, die keine Randkanten einer Facette (Fenster,Tür) sind, nicht in dieser Facette liegen dürfen, sondern "etwas" davor. Die Menge der Kanten setzt sich hier aus der Menge der Kanten der Facetten und der zusätzlichen Menge der Kanten der Fenster und Türe zusammen.)

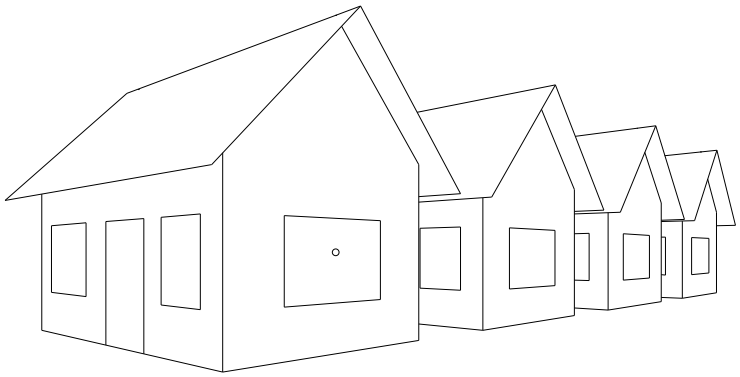


Abbildung 13.5: Häuser

Beispiel 13.4 *Darmstädter BAHNHOF*
(Man beachte, daß hier nicht konvexe Polygone auftreten. Durch Einführen von zusätzlichen Kanten lassen sich diese in konvexe Polygone zerlegen. Die zusätzlichen Kanten werden beim Zeichnen (z.B.) mit Hilfe einer geeigneten boolschen Variablen ignoriert.)

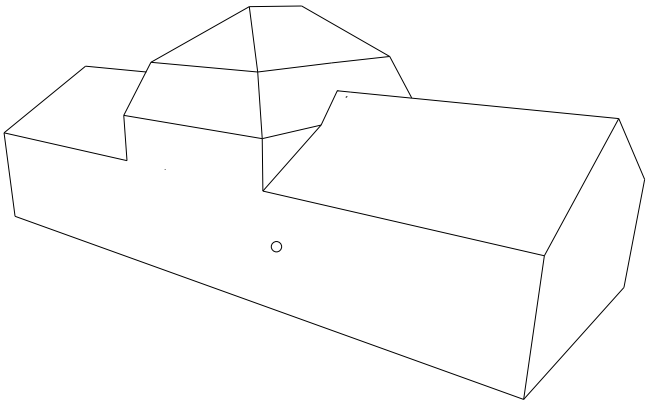


Abbildung 13.6: Bahnhof

Beispiel 13.5 *a) 3 Balken b) Kiosk*

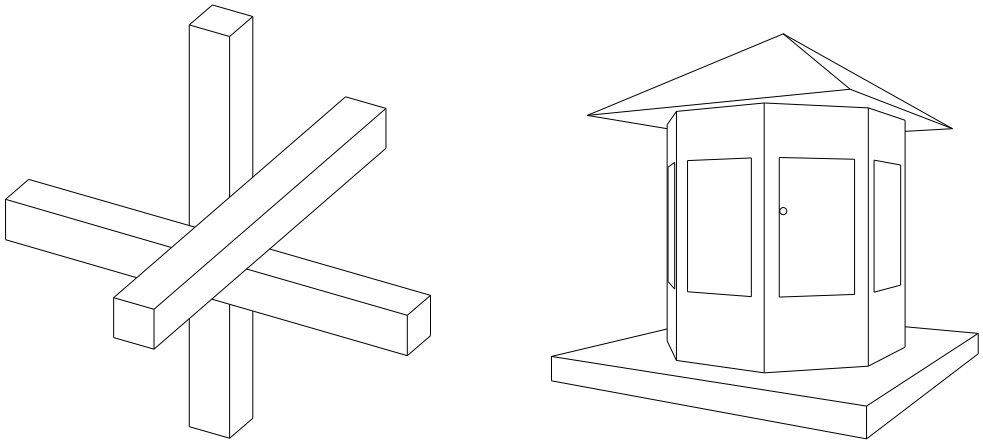
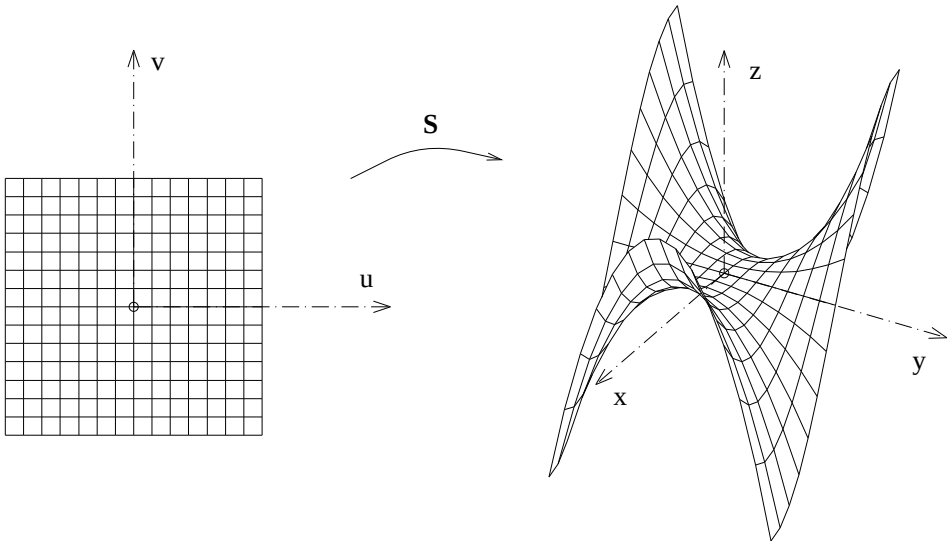


Abbildung 13.7: 3 Balken, Kiosk

13.2.2 Die Unterprogramme `aux_quadrangle, aux_cylinder, aux_torus` und Darstellung parametrisierter Flächen

Zur Darstellung einer **parametrisierten** Fläche $\Phi : \mathbf{x} = \mathbf{S}(u, v), u \in [a, b], v \in [c, d]$, wählt man meistens in der Parameterebene ($u - v-$Ebene) ein Rechteckgitter und dessen Bild im $\mathbb{R}^3$.
Sind die Bilder der Gitterrechtecke (im $\mathbb{R}^3$) fast eben, so kann man den obigen Hiddenline-Algorithmus zur Projektion dieser Vierecke verwenden. (Im Falle des TORUS sind die Vierecke sogar exakt eben.).
Der Algorithmus `aux_polyhedron` aus Kap. 13.2 kann zwar hier auch benutzt werden, ist aber für die besondere Struktur (nur viereckige Facetten) hier zu schwerfällig und langsam. Deshalb werden hier für die häufig auftretenden Fälle, daß dieFlächenstücke viereckig oder zylindrisch oder torusartig sind, Hilfsprogramme angegeben, die aus den **Vorgaben** n_1, n_2 (Anzahl der Teile des Rechteckgitters in u- bzw. v-Richtung), die Facetten und Kanten numeriert und die folgenden **Daten** berechnen:



Parameterebene Fläche im $\mathbb{R}^3$

Abbildung 13.8: Netz in der Parameterebene und sein Bild

(1) für jede Facette: `fp[1],..., fp[npf]` (Nummern der Punkte des Polygons),
`fe[1],..., fe[nef]` (Nummern der Kanten des Polygons),

(2) für jede Kante: `ep1, ep2` (Nummern von Anfangs- und Endpunkt),

(3) `np, ne, nf` : Anzahl der Punkte, Kanten bzw. Facetten,

(4) die Gleichungen $\mathbf{n}_i\mathbf{x} - d_i = 0$ der Ebenen, die die Facetten (`faces`) enthalten.

Die $n_1 \cdot n_2$ Punkte $P_1, P_2, \ldots$ müssen gemäß der folgenden Abbildung dem jeweiligen $u - v-$Netz zugeordnet werden (Für die ersten n_1 Punkte ist $v = a = const.$).

Im Fall "quadrangle" ist	$\Delta u = (b-a)/(n_1-1)$	und	$\Delta v = (c-d)/(n_2-1)$.
Im Fall "cylinder" ist	$\Delta u = (b-a)/n_1$	und	$\Delta v = (c-d)/(n_2-1)$.
Im Fall "torus" ist	$\Delta u = (b-a)/n_1$	und	$\Delta v = (c-d)/n_2$.

In jedem Fall ist zu beachten, daß der Punkt mit den Parametern $u = a+(i-1)\cdot\Delta u, v = c+(k-1)\cdot\Delta v$ die Nummer $(k-1)\cdot n_1 + i$ hat.
Um mehrere Flächen in einem Programm behandeln zu können, sind in den folgenden Unterprogrammen die Anzahlen der schon aufgenommenen Punkte `np0`, Kanten `ne0` und Facetten `nf0` nötig. Für die erste darzustellende Fläche ist `np0=ne0=nf0=0`.

```
procedure aux_quadrangle(n1,n2,np0,ne0,nf0: integer);
{****}
procedure aux_cylinder(n1,n2,np0,ne0,nf0: integer);
{****}
procedure aux_torus(n1,n2,np0,ne0,nf0: integer);
```

Im Hauptprogramm müssen für den Hiddenline-Algorithmus bereitstehen:

a) `n1,n2`,

b) Die Punkte $P_i : \mathbf{p}_i$, zeilenweise numeriert von unten nach oben (s. Zeichnung),

c) `npf,nef`, die Anzahl der Punkte bzw. Kanten in einer Facette. Normalerweise ist `npf=4, nef=4`. Bei Durchdringungen können beide allerdings auch größer sein.

Beispiel 13.6 *Zwei Tori (vgl. Aufg. 11.5)*

```
{*********************************************************************}
{***  Projektion zweier Tori mit UP cp_lines_before_convex_faces  ***}
{*********************************************************************}
program tori_h;
uses geograph;
 const {Achtung: es muss  array_size>=nfmax  sein  !!!}
       nfmax= 20000;  nemax=40000;    nsegmax=10;  npfmax=4;
 type  vts2d_pol  = array[0..npfmax] of vt2d;
       vts3d_pol  = array[0..npfmax] of vt3d;
       r_array_seg  = array[0..nsegmax] of real;
       i_array_seg  = array[0..nsegmax] of integer;
       box3d_dat = record
                    xmin,xmax,ymin,ymax,zmin,zmax : real;
                    end;
       face_dat = record
                  npf,nef : integer;
                  fp,fe : array[1..npfmax] of integer;
                  vis : boolean;
                  box : box3d_dat;
                  discentre,d : real;
                  nv : vt3d;
                  end;
       edge_dat = record
                  vis : boolean;
                  ep1,ep2 : integer;
                  end;
 var   face : array[1..nfmax] of face_dat;
       edge : array[1..nemax] of edge_dat;
       p : vts3d;     p0 : vt3d;   pdist : r_array;   error : boolean;
       n1,n2,np,nf,ne,i,j,k,ik,inz,ianf,i2tor : integer;
       r1,r2,xf,yf,x1,y1,x2,y2,dw,cdw,sdw : real;
     {$i include/proc_zpo.pas}
{***************}
begin {Hauptprogramm}
   graph_on(0);
   repeat
     writeln(' ***  2 Tori  *** ');    writeln;
     writeln(' n2 (>2 Unterteilungen des grossen Kreises) ? ');  readln(n2);
     writeln(' n1 (>2 Unterteilungen des kleinen Kreises) ? ');  readln(n1);
     writeln(' 2 Tori ? (ja=1) oder 1 Torus ? ');                readln(i2tor);
     r1:= 50;  r2:= 15;
{ Koordinaten der Punkte 1...n1 (kleiner Kreis): }
     dw:= pi2/n1;   cdw:= cos(dw);    sdw:= sin(dw);
     put3d(r1+r2,0,0, p[1]);   put3d(r1,0,0, p0);
     for i:= 2 to n1 do rotp0y(cdw,-sdw,p0, p[i-1], p[i]);
{ Koordinaten der restlichen Punkte des 1. Torus: }
     dw:= pi2/n2;   cdw:= cos(dw);    sdw:= sin(dw);
```

```
     for k:= 2 to n2 do
         for i:= 1 to n1 do
             begin
             ik:= i + (k-1)*n1;
             rotorz(cdw,sdw,p[ik-n1], p[ik]);
             end;
     np:= n1*n2;
     aux_torus(n1,n2,0,0,0);
{ 2. Torus:}
    if i2tor=1 then
       begin
       for i:= 1 to np do put3d(p[i].x-r1, -p[i].z, p[i].y,  p[np+i]);
       aux_torus(n1,n2,np,ne,nf);
       end;
     writeln(' np:',np);     writeln(' nf:',nf);     writeln(' ne:',ne);
     repeat
       init_central_projection;
{ Zeichnen : }
       draw_area(200,200,80,90,0.5);
       cp_lines_before_convex_faces(true,false);    {Hiddenline-Algorithmus}
       draw_end ;        writeln ;
       writeln(' Noch eine Projektion ? ( Ja = 1 )');     readln(inz);
     until inz=0;
   writeln('Noch einmal mit ANDEREN  n1, n2 ?  (ja: 1)'); readln(ianf);
   until ianf=0;
   graph_off;
 end.
```

Aufgabe 13.1 *ROHRKNOTEN*
Die Mittenkurve eines Rohrknotens liegt auf einem Torus und hat hier die Parameterdarstellung

$$\mathbf{x} = \mathbf{c}(u) := (h\cos(u), h\sin(u), r_2\sin(cu)),$$

mit $h = r_1 + r_2\cos(cu), c = 1.5$ *und* $0 \le u \le 4\pi$.
(r_1 ist der "große" Radius, r_2 der "kleine" Radius des Torus.) Jeder Punkt der Mittenkurve ist Mittelpunkt eines Kreises, der senkrecht zur Mittenkurve steht. Für die Ebene, die im Punkt $\mathbf{c}(u)$ *senkrecht zur Mittenkurve steht, wählen wir die folgenden Vektoren als Basiseinheitsvektoren:*

$$\mathbf{e}_1(u) := \dot{\mathbf{c}}(u) \times \mathbf{e}_z/\| \dots \|, \quad \mathbf{e}_2(u) := \mathbf{e}_1 \times \dot{\mathbf{c}}(u)/\|\dot{\mathbf{c}}(u)\|,$$

wobei $\mathbf{e}_z := (0, 0, 1)$ *ist.*
Also ist

$$\mathbf{x} = \mathbf{S}(u, v) := \mathbf{c}(u) + \mathbf{e}_1(u)r_3\cos v + \mathbf{e}_2(u)r_3\sin v, 0 \le r_3 \le r_2,$$

eine Parameterdarstellung eines Rohrknotens. r_3 ist die "Dicke" des Rohres.

Beispiel 13.7 *Möbiusband (***nicht** *vom Typ "quadrangle")*
(Die Berandungskurven der Bänder liegen auf einem Torus!)

13.3 Schnitt zweier Polygone im Raum, Schnitt zweier Polyeder

13.3.1 Schnitt zweier ebener von Polygonen begrenzte Flächen im Raum

Als Vorbereitung für die Bestimmung der Schnittkanten zweier sich durchdringender Polyeder überlegen wir uns zunächst ein Unterprogramm, das die Schnittstrecke zweier durch ebene konvexe Polygone berandete Flächenstücke im Raum berechnet.
Gegeben: Zwei ebene konvexe Polygone $P_{11}, P_{12}, \ldots, P_{1m}, P_{21}, P_{22}, \ldots, P_{2n}$ im $\mathbb{R}^3$. Die Polygone liegen nicht in einer gemeinsamen Ebene.
Gesucht: Der Schnitt (Strecke), der durch die Polygone berandeten ebenen Flächenstücke.

Idee:

(1) Man bestimmt zunächst die Schnittgerade der die Polygone enthaltenden Ebenen.

(2) Nun schneidet man diese Schnittgerade mit den Polygonen und enthält zwei Schnittstrecken.

(3) Der Durchschnitt der beiden Schnittstrecken ist die gesuchte Strecke.

Der **Algorithmus** (Prozedur `is_n1gon_n2gon3d`):

(0) Für jedes Polygon wird mit `box3d_of_pts` der kleinste achsenparallele Quader bestimmt, der das eine bzw. das andere Polygon enthält. Nur wenn sich beide Quader schneiden,beginnt der folgende Schnittalgorithmus.

(1) Es werden die Gleichungen $\mathbf{n}_1\mathbf{x} - d_1 = 0, \mathbf{n}_2\mathbf{x} - d_2 = 0$ mit Hilfe von `plane_equ` berechnet. `is_plane_plane` liefert einen Punkt und einen Richtungsvektor der Schnittgerade g_s beider Ebenen.

(2) Um die Gerade g_s mit dem ersten Polygon zu schneiden, führen wir in der Polygonebene das folgende Koordinatensystem ein: P_{11} ist der Nullpunkt und die Punkte P_{12}, P_{1m} sind die Achseneinheitspunkte.

Da die Polygonpunkte i.a. nicht exakt in einer Ebene liegen, berechnen wir mit Hilfe des Unterprogramms `ptco_plane3d` die Koordinaten der zugehörigen Lotfußpunkte Q_{1i} bezüglich des obigen Koordinatensystems. Speziell gilt dann $Q_{11} = (0,0), Q_{12} = (1,0)$ und $Q_{1m} = (0,1)$. Nachdem man noch zwei Punkte der Gerade g_s so in die Ebene projiziert hat, lassen sich mit `is_line_convex_polygon` die Parameter der Schnittkante t_1, t_2 bestimmen. Analog verfährt man mit dem zweiten Polygon und erhält Parameter s_1, s_2.

(3) `is_interv_interv` berechnet schließlich den Schnitt $[s,t] := [t_1,t_2] \cap [s_1,s_2]$ der Intervalle. Damit ist die Schnittstrecke der beiden Polygone bestimmt.

```
procedure is_interv_interv(var a,b,c,d,aa,bb : real; var inters: boolean);
{Berechnet den Schnitt der Intervalle [a,b], [c,d] .}
var a1,a2,b1,b2 : real;
begin
  a1:= min(a,b);    b1:= max(a,b);
  a2:= min(c,d);    b2:= max(c,d);
  aa:= max(a1,a2);  bb:= min(b1,b2);
  if bb<=aa then inters:= false else inters:= true;
end; {is_interv_interv}
```

```
{*****}

procedure box3d_of_pts(var p : vts3d_pol; np: integer; var box : box3d_dat);
var a,b : real;   i: integer;
{Bestimmt den zugehrigen (kleinsten) achsenparallelen Quader.}
begin
  with box do
       begin
       with p[1] do
            begin
            xmin:= x; xmax:= x;   ymin:= y; ymax:= y;  zmin:= z; zmax:= z;
            end;
       for i:= 2 to np do with p[i] do
           begin
           xmin:= min(xmin,x);  xmax:= max(xmax,x);
           ymin:= min(ymin,y);  ymax:= max(ymax,y);
           zmin:= min(zmin,z);  zmax:= max(zmax,z);
           end;  { for }
      end;  { with }
end;  { box3d_of_pts }
{***********}

function is_two_boxes3d(var box1,box2 : box3d_dat) : boolean;
{Schnitt zweier achsenparalleler Quader.}
begin
  is_two_boxes3d:= false;
  if (box1.xmax>box2.xmin) then if (box1.xmin<box2.xmax) then
     if (box1.ymax>box2.ymin) then if (box1.ymin<box2.ymax) then
        if (box1.zmax>box2.zmin) then if (box1.zmin<box2.zmax) then
           is_two_boxes3d:= true;
end;  {is_two_boxes3d }
{************}
procedure is_line_conv_pol_in_plane3d(var pl,rl: vt3d; var pp : vts3d_pol;
                                      npp : integer;
                                      var t1,t2 : real; var inters : boolean);
{Schnitt eines konv. Polygons mit einer in der Polygonebene liegenden Gerade.}
var pp1,v1,v2,pl2 : vt3d;      qql1,qql2 : vt2d;
    qq : vts2d_pol;    error : boolean;   i,ind : integer;
begin
  pp1:= pp[1];  diff3d(pp[npp],pp1, v2);  diff3d(pp[2],pp1, v1);
  for i:= 3 to npp-1 do
      ptco_plane3d(pp1,v1,v2,pp[i], qq[i].x,qq[i].y,error);
  put2d(0,0, qq[1]);    put2d(1,0, qq[2]);
  put2d(0,1, qq[npp]);  qq[0]:= qq[npp];
  ptco_plane3d(pp1,v1,v2,pl,qql1.x,qql1.y,error);
  sum3d(pl,rl, pl2);
  ptco_plane3d(pp1,v1,v2,pl2, qql2.x,qql2.y,error);
  is_line_convex_polygon(qql1,qql2,qq,npp, t1,t2,ind);
  if t1<>t2 then inters:= true else inters:= false;
end;  { is_line_conv_pol_in_plane3d }
{***********}
procedure is_n1gon_n2gon3d(var pp1,pp2: vts3d_pol; np1,np2: integer;
                           var  ps1,ps2 : vt3d; var intersection : boolean);
{Berechnet die Schnittstrecke zweier ebener konvexer Polygone im Raum, die
```

```
 NICHT in einer Ebene liegen.}
var   box1,box2 : box3d_dat;    t1,t2,s1,s2,s,t : real;
      nv1,nv2,ps,rs : vt3d;  d1,d2 : real;
      inters1,inters2,error,inters : boolean;
begin
  intersection:= false;
  box3d_of_pts(pp1,np1, box1);  box3d_of_pts(pp2,np2, box2);
  if is_two_boxes3d(box1,box2) then
     begin
     plane_equ(pp1[1],pp1[2],pp1[3], nv1,d1,error);
     plane_equ(pp2[1],pp2[2],pp2[3], nv2,d2,error);
     is_plane_plane(nv1,d1,nv2,d2, ps,rs, error);
     if not error then
        begin
        is_line_conv_pol_in_plane3d(ps,rs,pp1,np1, t1,t2,inters1);
        is_line_conv_pol_in_plane3d(ps,rs,pp2,np2, s1,s2,inters2);
        if inters1 and inters2 then
           begin
           is_interv_interv(t1,t2,s1,s2, s,t,inters);
           if inters then
              begin
              lcomb2vt3d(1,ps,s,rs, ps1);
              lcomb2vt3d(1,ps,t,rs, ps2);
              intersection:= true;
              end;   { if inters }
           end;  { if inters1,inters2 }
        end;  { if not error }
     end; { if is_boxes }
end;  { is_n1gon_n2gon }
{**************}
```

13.3.2 Schnitt zweier Polyeder

Gegeben: Zwei sich durchdringende Polyeder Π_1, Π_2.
Gesucht: Die Schnittkanten.

Idee:
Da man die Polyeder in der Regel auch darstellen möchte, ist es zweckmäßig die Datenstruktur des Hiddenlinealgorithmus `is_lines_before_convex_faces` zu verwenden. Dadurch ergeben sich einige Vereinfachungen.

Der **Algorithmus:**

(1) Man berechnet zunächst mit `boxes_of_faces` die für einen schnellen Vortest notwendigen achsenparallelen Quader.

(2) Da die Gleichungen der Flächen schon vorliegen (Sie sind auch für den Hiddenlinealgorithmus nötig.), verwendet man hier das für diesen Fall angepaßte Unterprogramm `is_face_face` um die Schnittkanten zu bestimmen.

(3) Die Schnittkante der i-ten Facette mit der k-ten Facette wird als zusätzliche Kante in das Array **edge** aufgenommen und in **face[i]** und **face[k]** vermerkt, daß diese Kante in der i-ten bzw. k-ten Facette liegt. (Letzteres ist notwendig, damit die Kante nicht mit der Facette,in der sie

liegt, verglichen wird. Durch Rechenungenauigkeiten könnte die Kante hinter der zugehörigen Ebene liegen und damit unsichtbar sein.)

Im Folgenden sind die Unterprogramme `boxes_of_faces, is_face_face` sowie die im Hauptprogramm notwendige Ergänzung für den Fall zweier sich durchdringender Tori angegeben.

```
procedure boxes_of_faces;
var i,j : integer;  pp : vts3d_pol;
begin
  for i:= 1 to nf do
      with face[i] do
           begin
           for j:= 1 to npf do pp[j]:= p[fp[j]];
           box3d_of_pts(pp,npf, box);
           end;
end;  { boxes_of_faces }
{************}
procedure is_face_face(i,k: integer;  var ps1,ps2 : vt3d;
                                      var intersection: boolean);
{Berechnet die Schnittstrecke zweier nicht in einer Ebene liegenden
 (konvexen) Flaechen eines Polyeders.}
var  t1,t2,s1,s2,s,t : real;      ps,rs : vt3d;  ppf: vts3d_pol;
     error,inters1,inters2,inters : boolean;
begin
  intersection:= false;
  if is_two_boxes3d(face[i].box,face[k].box) then
     begin
     is_plane_plane(face[i].nv,face[i].d,face[k].nv,face[k].d, ps,rs, error);
     if not error then
        begin
        with face[i] do for j:= 1 to npf do ppf[j]:= p[fp[j]];
        is_line_conv_pol_in_plane3d(ps,rs,ppf,face[i].npf, t1,t2,inters1);
        with face[k] do for j:= 1 to npf do ppf[j]:= p[fp[j]];
        is_line_conv_pol_in_plane3d(ps,rs,ppf,face[k].npf, s1,s2,inters2);
        if inters1 and inters2 then
           begin
           is_interv_interv(t1,t2,s1,s2, s,t,inters);
           if inters then
              begin
              lcomb2vt3d(1,ps,s,rs, ps1);
              lcomb2vt3d(1,ps,t,rs, ps2);
              intersection:= true;
              end;   { if inters }
           end;  { if inters1,inters2 }
        end;  { if not error }
     end; { if is_boxes }
end;  { is_face_face }
{**************}
.............................
{ Schnittkanten der Tori: }
    boxes_of_faces;
    for i:= 1 to nf1 do         { 1.Flaechenschleife }
        begin
        for k:= nf1+1 to nf do  { 2.Flaechenschleife }
            begin
```

```
            is_face_face(i,k, ps1,ps2,inters);
            if inters then
               begin
               p[np+1]:= ps1;  p[np+2]:= ps2;
               with edge[ne+1] do begin ep1:= np+1; ep2:= np+2;  end;
               np:= np+2;  ne:= ne+1;
               with face[i] do begin fe[nef+1]:= ne; nef:= nef+1;  end;
               with face[k] do begin fe[nef+1]:= ne; nef:= nef+1;  end;
               end;  { if }
            end;  { for k }
         end;  { for i }
.................................
```

Bei der Bestimmung von **Selbstdurchdringungen** muß man eine Fläche mit sich selbst schneiden. Dies führt bei benachbarten Facetten zu Problemen. Deshalb sollte man mit einem Unterprogramm Nachbarschaften aufspüren und nur nicht benachbarte Facetten auf eventuellen Schnitt untersuchen.

13.4 Schnitt einer Strecke mit einem polygonalen Flächenstück, Schnittpunkte eines Polygons mit einem Polyeder

13.4.1 Schnitt einer Strecke mit einem polygonalen Flächenstück

Gegeben: Eine Strecke P_1P_2 und ein ebenes Polygon $Q_1, Q_2, \ldots, Q_n$ im $\mathbb{R}^3$.
Die Polygonebene enthält nicht die Strecke.
Gesucht: Der Schnittpunkt der Strecke mit dem vom Polygon berandeten ebenen Flächenstück.

Der **Algorithmus:**

(1) Bestimmung der minimalen achsenparallelen Quader, die die Strecke bzw. das Polygon enthalten. Test, ob sich beide Quader schneiden.

(2) Falls sich beide Quader schneiden, wird mit `plane_equ` die Gleichung $\mathbf{nx} - d = 0$ der Polygonebene ε bestimmt. Anhand der Vorzeichen der Zahlen $\mathbf{n} \cdot \mathbf{p}_1 - d, \mathbf{n} \cdot \mathbf{p}_2 - d$, wobei $P_i : \mathbf{p}_i$ ist, kann man feststellen, ob die Strecke die Ebene ε durchdringt.

(3) Falls die Strecke die Ebene durchdringt, berechnet man den Schnittpunkt P_s und prüft mit Hilfe von `is_line_conv_pol_in_plane3d`, wie die Gerade durch P_s und Q_1 das Polygon schneidet. (Es gibt wenigstens den Schnittpunkt Q_1 !). Falls es zwei Schnittpunkte gibt und P_s diese trennt, liegt P_s in dem Polygon und die Strecke schneidet die polygonale Fläche im Punkt P_s.

Das Unterprogramm `is_line_ngon3d`:

```
procedure is_line_ngon3d(p1,p2: vt3d; var pp: vts3d_pol; npp: integer;
                                                  var ps: vt3d; var intersection: boolean);
 {Berechnet den Schnitt einer Strecke mit einem ebenen konv. n-Eck im Raum.
 Die Strecke darf NICHT in der Ebene des n-Ecks liegen.}
label 5;
var  d,t1,t2,xemin,xemax,yemin,yemax,zemin,zemax,dp1,dp2,ts : real;
     j : integer;    box : box3d_dat;   nv,rv,rs : vt3d;    inters : boolean;
begin
  intersection:= false;
  xemax:= max(p1.x,p2.x);   yemax:= max(p1.y,p2.y);  zemax:= max(p1.z,p2.z);
  xemin:= min(p1.x,p2.x);   yemin:= min(p1.y,p2.y);  zemin:= min(p1.z,p2.z);
  box3d_of_pts(pp,npp, box);
  if (xemax<=box.xmin) or (xemin>=box.xmax) or
     (yemax<=box.ymin) or (yemin>=box.ymax) or
     (zemax<=box.zmin) or (zemin>=box.zmax)
     then   goto 5;   {Kante wird nicht von j-ter Flaeche verdeckt}
  plane_equ(pp[1],pp[2],pp[3] , nv,d,error);
  dp1:= scalarp3d(p1,nv) - d;
  dp2:= scalarp3d(p2,nv) - d;
  if dp1*dp2<=0 then
     begin
     ts:= dp1/(dp1-dp2);             diff3d(p2,p1, rv);
     lcomb2vt3d(1,p1,ts,rv, ps);     diff3d(ps,pp[1], rs);
     is_line_conv_pol_in_plane3d(ps,rs,pp,npp, t1,t2,inters);
```

```
     if inters and (t1*t2<=0) then intersection:= true;
     end;    {if}
5: end;    { is_line_ngon3d }
```

13.4.2 Schnittpunkte eines Polygons mit einem Polyeder

Gegeben: Polygon mit Kanten $e_1, e_2, \ldots, e_n$, Polyeder mit konvexen Facetten $f_1, f_2, \ldots, f_m$.
Gesucht: Schnittpunkte des Polygons mit dem Polyeder.
Lösung : Wende das Unterprogramm `is_line_ngon3d` auf jedes Paar $e_i, f_j, 1 \leq i \leq n, 1 \leq j \leq m$ an.

Beispiel 13.8 *Schnitt eines Polygons auf einem Kreis mit einem Polyeder auf einem Torus*

Verwendet man die Datenstruktur des Hiddenline-Algorithmus `cp_lines_before_convex_faces`, so ist es besser die folgende Version `is_edge_face` zu verwenden. Die achsenparallelen Quader der Facetten und die Ebenengleichungen werden dann im Hauptprogramm mit `boxes_of_faces` bzw. `plane_equ` bereitgstellt.

```
procedure is_edge_face(i,k: integer; var ps: vt3d; var intersection: boolean);
{Berechnet den Schnittpunkt der i-ten Kante mit der (konvexen) k-ten Flaeche
 eines Polyeders. Die Kante darf NICHT in der Ebene der k-ten Flaeche liegen.}
label 5;
var  t1,t2,xemin,xemax,yemin,yemax,zemin,zemax,dp1,dp2,ts : real;
     j : integer;   rs,p1,p2,rv : vt3d;   pp: vts3d_pol;   inters : boolean;
begin
  intersection:= false;
  with edge[i] do begin p1:= p[ep1];  p2:= p[ep2]; end;
  xemax:= max(p1.x,p2.x);   yemax:= max(p1.y,p2.y);  zemax:= max(p1.z,p2.z);
  xemin:= min(p1.x,p2.x);   yemin:= min(p1.y,p2.y);  zemin:= min(p1.z,p2.z);
  with face[k] do
       begin
       if (xemax<=box.xmin) or (xemin>=box.xmax) or
          (yemax<=box.ymin) or (yemin>=box.ymax) or
          (zemax<=box.zmin) or (zemin>=box.zmax)
          then   goto 5; { Kante wird nicht von j-ter Flaeche verdeckt }
       dp1:= scalarp3d(p1,nv) - d;
       dp2:= scalarp3d(p2,nv) - d;
       if dp1*dp2<=0 then
          begin
          ts:= dp1/(dp1-dp2);      diff3d(p2,p1, rv);
          lcomb2vt3d(1,p1,ts,rv, ps);
          for j:= 1 to npf do pp[j]:= p[fp[j]];
          diff3d(ps,pp[1], rs);
          is_line_conv_pol_in_plane3d(ps,rs,pp,npf, t1,t2,inters);
          if inters and (t1*t2<=0) then intersection:= true;
          end;  { if }
5:     end;  { with }
end;  { is_edge_face }
```

Bemerkungen:
Das Unterprogramm `is_line_ngon3d` kann man dazu verwenden, um näherungsweise die **Schnittpunkte** einer **Kurve** mit einer **parametrisierten Fläche** zu berechnen. (vgl. Kap. 12)

“quadrangle” “cylinder” “torus”

Abbildung 13.9: Netztypen

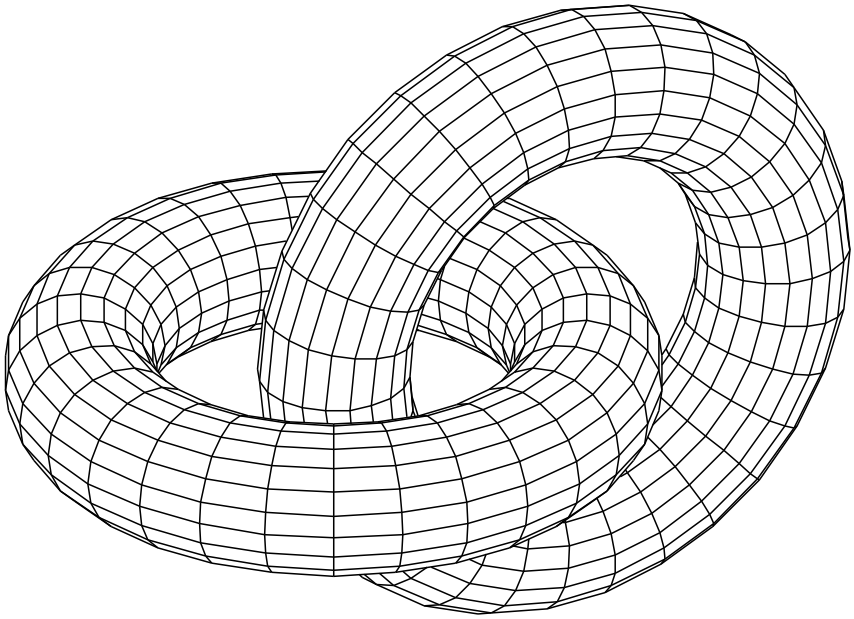


Abbildung 13.10: Zwei Tori

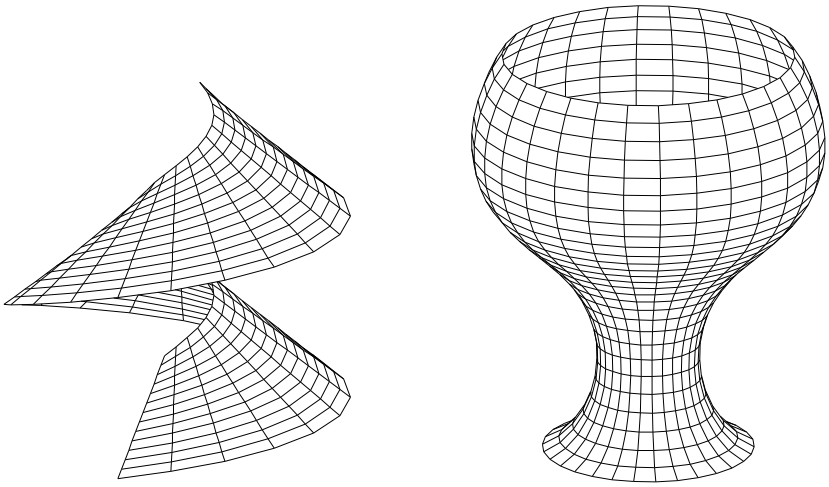


Abbildung 13.11: Verschraubung einer Strecke ("quadrangle"), Rotationsfläche ("cylinder")

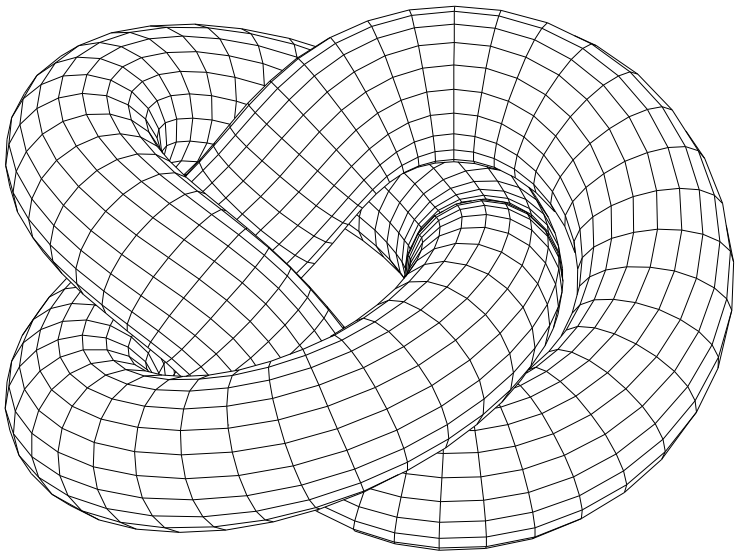


Abbildung 13.12: Rohrknoten (“torus”)

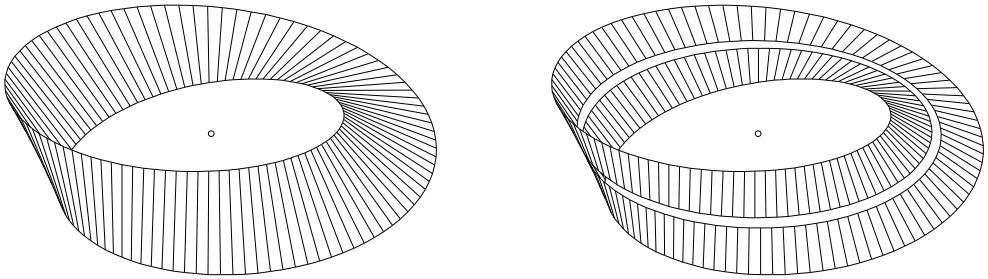


Abbildung 13.13: Möbiusband, geschlitztes Möbiusband

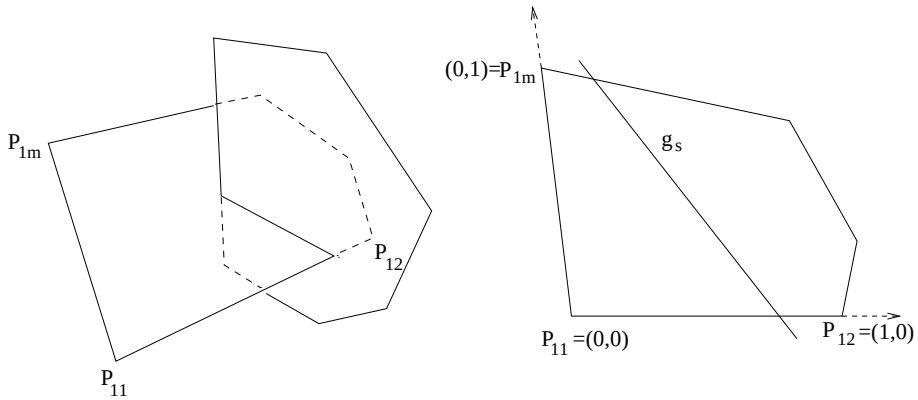


Abbildung 13.14: Zu (2) des Algorithmus

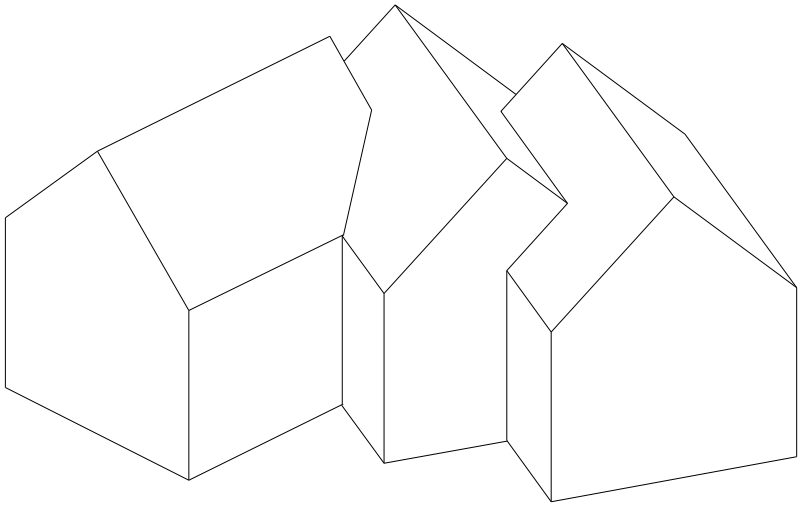
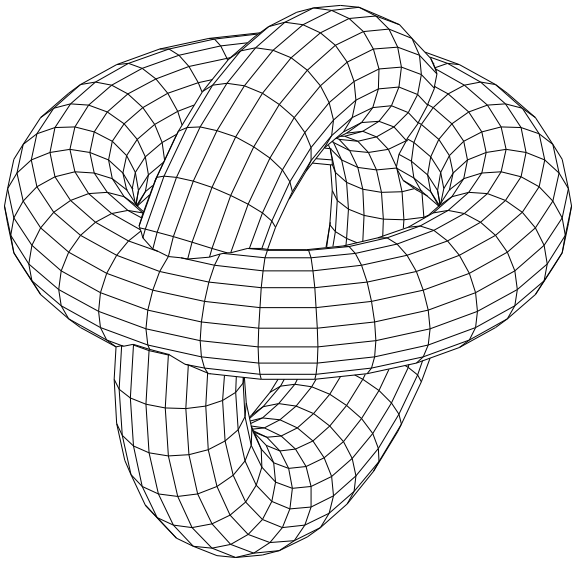
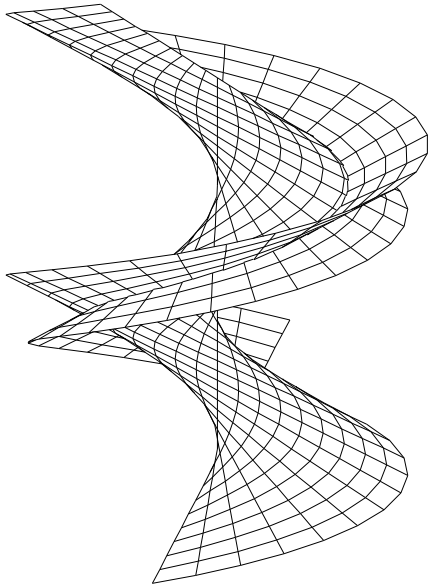


Abbildung 13.15: Durchdringungen



a) sich durchdringende Tori



b) selbstdurchdringende Strahlschraubfläche

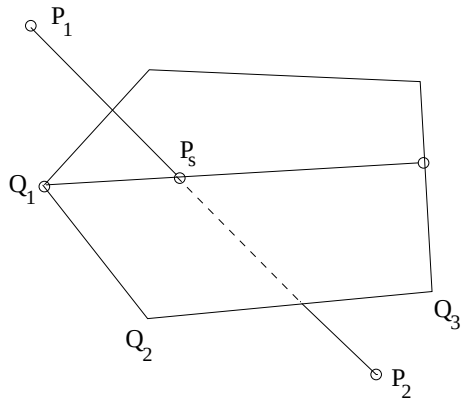


Abbildung 13.16: zu (3) des Algorithmus `is_line_ngon3d`

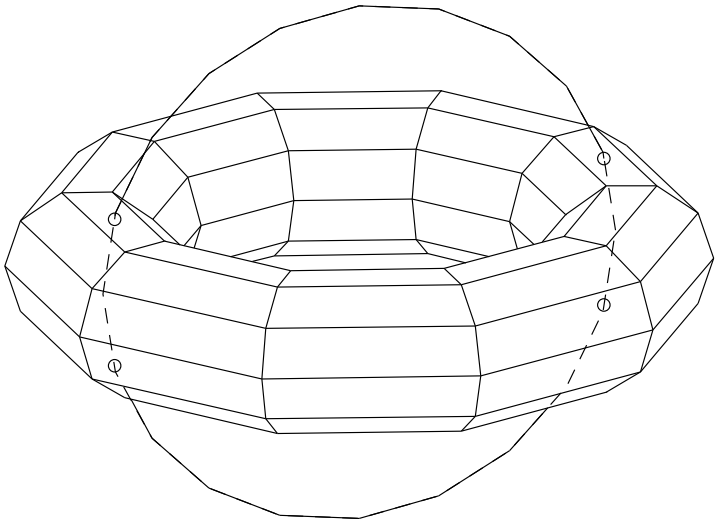


Abbildung 13.17: Schnitt Polygon-Polyeder

Kapitel 14

TRIANGULIERUNG IMPLIZITER FLÄCHEN

Die Triangulierung von Flächen ist ein wichtiges Hilfsmittel bei der Anwendung von Finite Element Methoden und zur Darstellung von Flächen durch Raytracing oder andere Hiddenlinealgorithmen (s. Kapitel 13). Die Triangulierung einer parametrisierten Fläche kann man durch eine Triangulierung ihres Definitionsbereichs erhalten. Aber die Bilder dieser Dreiecke können im Objektraum sehr verschieden in Gestalt und Größe ausfallen, was ihre Verwendung einschränken kann. Die Triangulierung einer impliziten Fläche ist wesentlich schwieriger. Die meisten Algorithmen unterteilen den zu betrachtenden Bereich in Quader und approximieren den Schnitt der Fläche mit diesen Quadern durch Polygone, die dann noch trianguliert werden (vgl. BL'88, SC'93). Da der Aufwand dieser Algorithmen zur Verwaltung der Daten erheblich ist, wird hier ein leichter zu programmierender **Verfolgungsalgorithmus** besprochen. Wir werden sehen, daß dieser Algorithmus mit Hilfe der Idee der *Normalform* (vgl. Abschn. 12.2.7) auch auf allgemeiner definierte Flächen anwendbar ist, insbesondere auch auf parametrisierte Flächen. Der Algorithmus liefert im wesentlichen Dreiecke, von ähnlicher Gestalt und Größe. Die Begrenzung der dargestellten Fläche muß nicht durch einen Quader bestimmt werden. Es ist möglich Begrenzungskurven vorzugeben (s. Beipiele).

14.1 Der Triangulierungs–Algorithmus

Der Algorithmus verwendet wesentlich die folgende Prozedur `surfacepoint`

14.1.1 Die Prozedur `surfacepoint`

Die Prozedur `surfacepoint` bestimmt zu einem Punkt $\mathbf{q}$ in der Nähe einer impliziten Fläche $\Phi: \ f(\mathbf{x}) = 0$ einen Flächenpunkt $\mathbf{p}$, der in der Nähe des zugehörigen Lotfußpunktes von $\mathbf{q}$ liegt. $\mathbf{p}$ ist im Falle einer Ebene exakt der Lotfußpunkt. Die Prozedur berechnet außerdem eine Flächeneinheitsnormale und zwei orthonormale Tangentenvektoren in $\mathbf{p}$.

Es seien nun die implizite Fläche $\Phi: \ f(\mathbf{x}) = 0$, für die stets der Gradient ∇f existiert und nicht der Nullvektor ist, und ein Punkt $\mathbf{q}$ in der Nähe der Fläche gegeben.

1. (a) $\mathbf{u}_0 = \mathbf{q}$

(b) repeat $\mathbf{u}_{k+1} := \mathbf{u}_k - \frac{f(\mathbf{u}_k)}{\nabla f(\mathbf{u}_k)^2}\nabla f(\mathbf{u}_k)$ (Newtonschritt für die Funktion $g(t) := f(\mathbf{u}_k + t\nabla f(\mathbf{u}_k))$)
until $\|\mathbf{u}_{k+1} - \mathbf{u}_k\|$ ist "genügend" klein.
Flächenpunkt $\mathbf{p} = \mathbf{u}_{k+1}$.

2. Die *Flächennormale* im Punkt $\mathbf{p}$ ist $\mathbf{n} := \nabla f(\mathbf{p})/\|...\|$.

3. Als *Tangentenvektoren* wählen wir
$\mathbf{t}_1 := (n_y, -n_x, 0)/\|...\|$ falls $n_x > 0.5$ or $n_y > 0.5$
oder $\mathbf{t}_1 := (-n_z, 0, n_x)/\|...\|$
und $\mathbf{t}_2 := \mathbf{n} \times \mathbf{t}_1$ wobei $(n_x, n_y, n_z) := \mathbf{n}$ ist.

14.1.2 Idee des Algorithmus

S0 Es sei δ_t die ungefähre Seitenlänge der Dreiecke der Triangulierung. Wähle eine Punkt $\mathbf{s}$ in der Nähe der Fläche. Bestimme mit der Prozedur `surfacepoint` einen Flächenpunkt $\mathbf{p}_1$. Umgebe $\mathbf{p}_1$ in der Tangentialebene mit einem regulären Sechseck $\mathbf{q}_2, ..., \mathbf{q}_7$. Bestimme mit `surfacepoint` zu $\mathbf{q}_2, ..., \mathbf{q}_7$ die Flächenpunkte $\mathbf{p}_2, ..., \mathbf{p}_7$. Die so entstandenen Dreiecke auf der Fläche sind die ersten 6 Dreiecke der Triangulierung (s. 14.3,14.6).
Wir nennen das Polygon $\mathbf{q}_2, ..., \mathbf{q}_7$ das (erste) *aktuelle Frontpolygon* Π_0. Falls die Triangulierung begrenzt werden soll (nicht nötig bei geschlossenen Flächen) durch Flächenkurven $\Gamma_1, \Gamma_2, ...$ (s. Beispiele), bestimmen wir weitere *Frontpolygone* $\Pi_1, \Pi_2, ...$ auf diesen Kurven.
Für spezielle Flächen (Zylinder Torus,...) kann es besser sein, mit einem vordefinierten ersten Frontpolygon zu beginnen. (s. Beispiele)

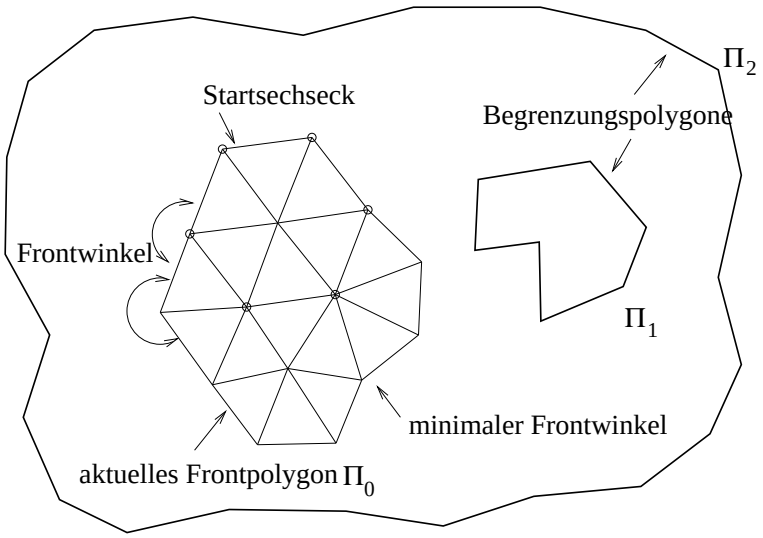


Abbildung 14.1: Bezeichnungen für den Algorithmus

S1 Für jeden Punkt des aktuellen Frontpolygons bestimmen wir den Außenwinkel. Wir nennen diese Winkel *Frontwinkel*.

S2 Prüfe, ob ein aktueller Frontpunkt $\mathbf{p}_i$ nahe einem

– Punkt von Π_0 ist, der von $\mathbf{p}_i$ und seinen Nachbarn verschieden oder
– Punkt von einem anderen Frontpolygon $\Pi_k, k > 0$ ist.

Im ersten Fall: Teile das aktuelle Frontpolygon Π_0 in ein kleineres und ein weiteres Frontpolygon. (s. Abb. 14.2, 14.9a,b)
Im zweiten Fall: Ist $\mathbf{p}_i$ nahe einem Punkt des Frontpolygons Π_m, so vereinige die Polygone Π_0, Π_m zu einem neuen (größeren) Frontpolygon. Lösche Π_m (s. Abb. 14.2, 14.9d,c).

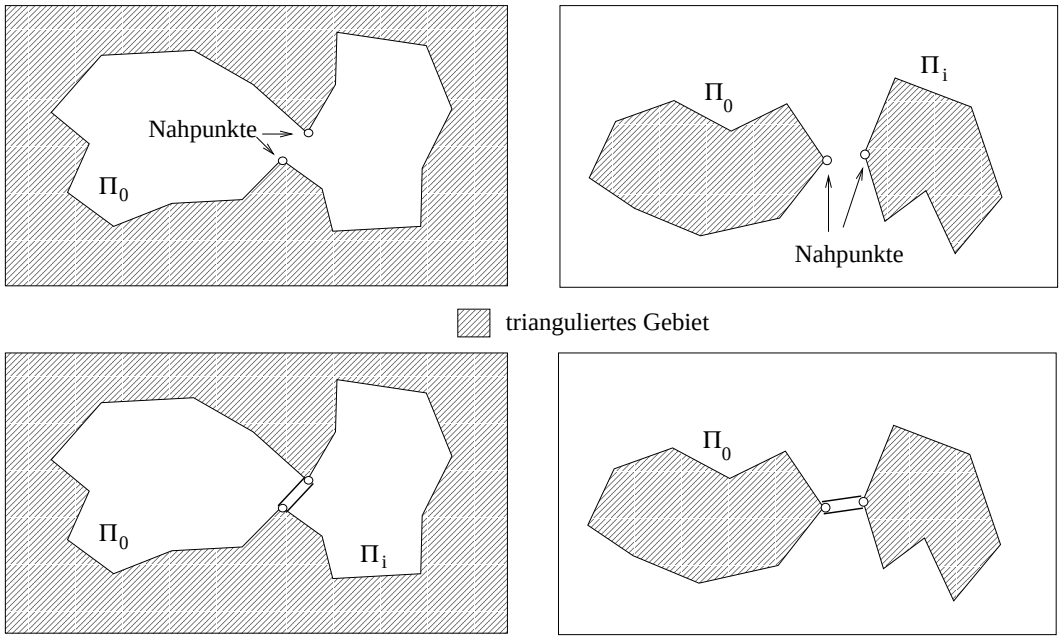


Abbildung 14.2: Teilen (links) und Vereinigen (rechts) des aktuellen Frontpolygons

S3 Bestimme einen Frontpunkt $\mathbf{p}_m$ des aktuellen Frontpolygons Π_0 mit minimalem Frontwinkel. Umgebe $\mathbf{p}_i$ mit $\approx$ reguläre Dreiecke mit Seitenlänge $\approx \delta_t$. Lösche $\mathbf{p}_m$ aus dem Polygon Π_0 und füge die neuen Punkte in das aktuelle Frontpolygon Π_0 ein.

S4 Wiederhole die Schritte S1,S2,S3 bis das aktuelle Frontpolygon Π_0 nur noch aus 3 Punkten besteht, die ein neues Dreieck liefern. Falls ein weiteres Frontpolygon existiert, so erkläre dies zum *aktuellen* Frontpolygon Π_0 und wiederhole die Schritte S1,S2,S3. Wenn kein Frontpolygon mehr existiert, so ist die Triangulierung beendet.
Falls die Fläche nicht beschränkt ist, sollte man sie durch Randpolygone oder durch eine "bounding box" begrenzen. (s. 14.8c,d).

14.1.3 Die Datenstruktur

δ_t ist die ungefähre *Kantenlänge* der Dreiecke.
Die *Punkte* der Triangulierung werden fortlaufend durchnumeriert. Für jeden Punkt speichern wir die folgenden Informationen:

- die Koordinaten,
- die Flächennormale $\mathbf{n}$ und Tangentenvektoren $\mathbf{t}_1, \mathbf{t}_2$ so, daß $\mathbf{n}, \mathbf{t}_1, \mathbf{t}_2$ orthonormal sind,

- der aktuelle Frontwinkel, falls $\mathbf{p}_i$ ein aktueller Frontpunkt ist,
- die boolsche Variable `angle_changed` mit `...= true`, falls sich der aktuelle Winkel durch Einfügen von Punkten geändert hat und deshalb neu berechnet werden muß,
  die boolsche Variable `boarder_point. ..= true` bedeutet, daß $\mathbf{p}_i$ auf dem Rand der Triangulierung liegt und bei weiteren Betrachtungen (Frontwinkel, Distanzcheck) ignoriert wird.

Die *Dreiecke* werden fortlaufend numeriert. Für jedes Dreieck werden die Nummern der Eckpunkte gespeichert.

Die *Frontpolygone* werden durch die Integer-Arrays ihrer Punkte repräsentiert.

14.1.4 Der Schritt S0

Es sei $\mathbf{s}$ ein Startpunkt in der Nähe der Fläche.
Die Prozedur `surfacepoint` bestimmt den ersten Punkt $\mathbf{p}_1$ der Triangulierung und das Orthonormalsystem $\mathbf{n}_1, \mathbf{t}_{11}, \mathbf{t}_{12}$. Die sechs Punkte $\mathbf{p}_2, ..., \mathbf{p}_7$ sind die zu

$$\mathbf{q}_{i+2} := \mathbf{p}_1 + \delta_t \cos(i\pi/3)\mathbf{t}_{11} + \delta_t \sin(i\pi/3)\mathbf{t}_{12},\ i := 0, ...5$$

gehörigen Flächenpunkte, ($\mathbf{p}_i$ = `surfacepoint`($\mathbf{q}_i$). Die Punkte $\mathbf{q}_2, ..., \mathbf{q}_7$ liegen in der Tangentialebene von $\mathbf{p}_1$ und bilden ein reguläres Sechseck.
Wir erhalten die ersten 6 Dreiecke:
$(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3), (\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4), (\mathbf{p}_1, \mathbf{p}_4, \mathbf{p}_5), (\mathbf{p}_1, \mathbf{p}_5, \mathbf{p}_6), (\mathbf{p}_1, \mathbf{p}_6, \mathbf{p}_7), (\mathbf{p}_1, \mathbf{p}_7, \mathbf{p}_2)$.

14.1.5 Der Schritt S1

Falls ein Punkt $\mathbf{p}_{0i}$ des aktuellen Frontpolygons $\Pi_0 = (\mathbf{p}_{01}, \mathbf{p}_{02}, ..., \mathbf{p}_{0N_0})$ gerade eingefügt worden ist oder falls ein Nachbar von $\mathbf{p}_{0i}$ ein neuer Punkt ist, ist es notwendig, den aktuellen Frontwinkel ω des Punktes $\mathbf{p}_{0i}$ neu zu berechnen. Es sei
$\mathbf{v}_1 := \mathbf{p}_{0,i-1}$ if $i > 1$ oder $\mathbf{v}_1 := \mathbf{p}_{0N_0}$ if $i = 1$,
$\mathbf{v}_2 := \mathbf{p}_{0,i+1}$ if $i < N_0$ oder $\mathbf{v}_2 := \mathbf{p}_{01}$ if $i = N_0$ und
(ξ_1, η_1, ζ_1) die Koordinaten von $\mathbf{v}_1$, (ξ_2, η_2, ζ_2) die Koordinaten von $\mathbf{v}_2$ bezgl. des orthonormalen Systems $\mathbf{n}, \mathbf{t}_1, \mathbf{t}_2$ am Punkt $\mathbf{p}_{0i}$,
ω_1 := Polarwinkel von (ξ_1, η_1), ω_2 := Polarwinkel von (ξ_2, η_2), dann ist der Winkel am Punkt $\mathbf{p}_{0i}$ $\omega = \omega_2 - \omega_1$, falls $\omega_2 \geq \omega_1$ andernfalls $\omega = \omega_2 - \omega_1 + 2\pi$.

14.1.6 Der Schritt S2

Um zu verhindern, daß neue Dreiecke alte Dreiecke überdecken, prüfen wir

- die Abstände von Punktepaaren des aktuellen Frontpolygons Π_0. Falls es Punkte $\mathbf{p}_{0i}, \mathbf{p}_{0j}, i < j$, (Nahpunkte) gibt, die nicht benachbart oder Nachbarn von Nachbarn sind und $\|\mathbf{p}_{0i} - \mathbf{p}_{0j}\| < \delta_t$ ist, wird das aktuelle Frontpolygon in das neue aktuelle Frontpolygon $(\mathbf{p}_{01}, ..., \mathbf{p}_{0i}, \mathbf{p}_{0j}, ..., \mathbf{p}_{0N_0})$ mit $N_0 - (j - i - 1)$ Punkte und ein weiteres Frontpolygon $(\mathbf{p}_{0i}, ..., \mathbf{p}_{0j})$ mit $j - i + 1$ Punkte zerlegt. (s. 14.2,14.9a,b) $\mathbf{p}_{0i}, \mathbf{p}_{0j}$ dürfen bei späteren Abstandsprüfungen nicht mehr verwendet werden.
- den Abstand der Punkte des aktuellen Frontpolygons Π_0 zu Punkten aller restlichen Frontpolygone $\Pi_k, k > 0$. Falls es Punkte $\mathbf{p}_{0i} \in \Pi_0$ und $\mathbf{p}_{mj} \in \Pi_m$ mit $\|\mathbf{p}_{0i} - \mathbf{p}_{mj}\| < \delta_t$ (Nahpunkte) gibt, werden die Polygone

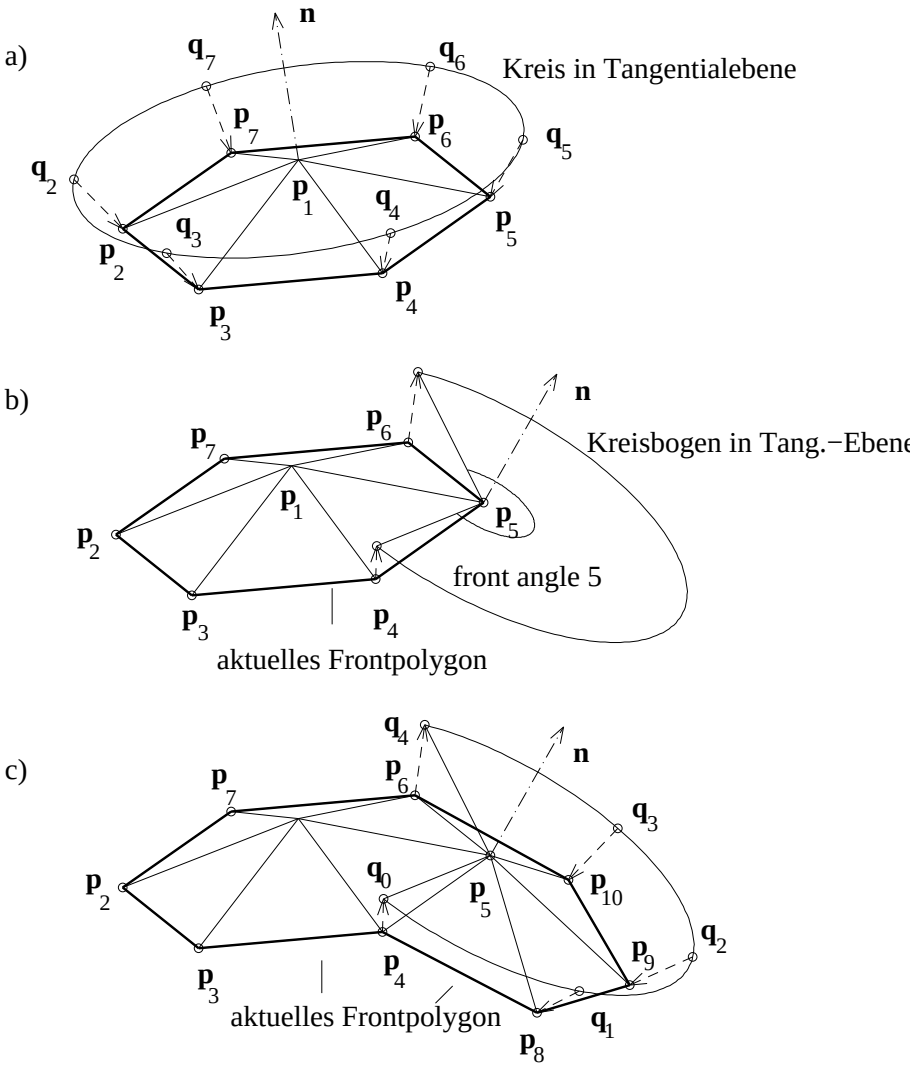


Abbildung 14.3: Die ersten Schritte des Algorithmus

$\Pi_0 = (\mathbf{p}_{01}, ..., \mathbf{p}_{0N_0})$ und $\Pi_m = (\mathbf{p}_{m1}, ..., \mathbf{p}_{mN_m})$
vereinigt zu einem neuen aktuellen Frontpolygon

$$\Pi_0 = (\mathbf{p}_{01}, ..., \mathbf{p}_{0i}, \mathbf{p}_{mj}, ...\mathbf{p}_{mN_m}, \mathbf{p}_{m1}, ..., \mathbf{p}_{mj}, \mathbf{p}_{0i}, ..., \mathbf{p}_{0N_0})$$

mit $N_0 + N_m + 2$ Punkte. Die Punkte $\mathbf{p}_{0i}$ und $\mathbf{p}_{mj}$ erscheinen zweimal ! Deshalb sollte man als nächtes die Frontwinkel dieser Punkte bei ihrem ersten Erscheinen im Polygon Π_0 bestimmen und zuerst den Punkt vervollständigen (mit Dreiecken umgeben, s. Schritt S3), der den kleinsten Winkel besitzt, und dann den zweiten (s. 14.9e). Danach wird das erste Erscheinen dieser beiden Punkte aus dem aktuellen Frontpolygon gestrichen. $\mathbf{p}_{0i}, \mathbf{p}_{mj}$ dürfen bei späteren Abstandsprüfungen nicht mehr verwendet werden.

Bemerkung:

a) Für "einfache" Flächen kann die Abstandsprüfung weggelassen werden. (s. Beispiele: Kugel, 6-peak-Fläche.)

b) Vor der Abstandsprüfung sollte man Punkte mit Frontwinkeln kleiner (ungefähr) 60° vervollständigen.

c) In einigen Fällen war eine Verzögerung der Abstandsprüfung nach einer Teilung des aktuellen Frontpolygons oder gar ein Weglassen, falls $|\Pi_0| < 10$, sinnvoll.

1. "Schlechte"Nahpunkte, die durch ein schon trianguliertes Gebiet verbunden sind (Abb. 14.4), können dadurch entdeckt werden, indem man den Winkel ω am Punkt $\mathbf{p}_{0i}$ gemäß Schritt 2 berechnet, wobei man anstelle von $\mathbf{v}_2$ den Punkt $\mathbf{p}_{mj}$ verwendet. Das Punktepaar $\mathbf{p}_{0i}, \mathbf{p}_{mj}$ ist ein "schlechtes", wenn der so berechnete Winkel ω größer ist als der Frontwinkel am Punkt $\mathbf{p}_{0i}$.

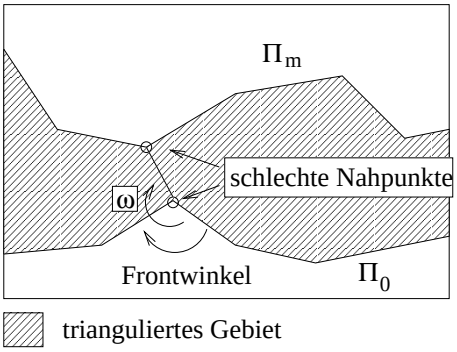


Abbildung 14.4: Schlechte Nahpunkte und ihre Entdeckung

2. Eine wesentliche Beschleunigung des Anstandstests erzielt man, durch Verwendung von "bounding boxes" für die Frontpolygone.

14.1.7 Der Schritt S3

Es sei $\mathbf{p}_{0m}$ ein Punkt des aktuellen Frontpolygons Π_0 mit minimalem Frontwinkel ω. *Vervollständige* die Triangulierung am Punkt $\mathbf{p}_{0m}$ auf die folgende Weise:

1. Bestimme die Nachbarn $\mathbf{v}_1, \mathbf{v}_2$ von $\mathbf{p}_{0m}$ (vgl. Schritt S1).

2. Bestimmung der Anzahl n_t von Dreiecke, die erzeugt werden sollen:
   Es sei $n_t := \texttt{trunc}(3\omega/\pi) + 1$, $\Delta\omega := \omega/n_t$
   Korrektur von $\Delta\omega$ in extremen Fällen:
   Falls $\Delta\omega < 0.8$ und $n_t > 1$ dann $n_t \to n_t - 1$ und $\Delta\omega = \omega/n_t$.
   Falls $n_t = 1$ und $\Delta\omega > 0.8$ und $\|\mathbf{v}_1 - \mathbf{v}_2\| > 1.25\delta_t$ dann $n_t = 2$ und $\Delta\omega \to \Delta\omega/2$.
   Falls ($\|\mathbf{v}_1 - \mathbf{p}_{0m}\|^2 < 0.2\ \delta_t^2$ oder $\|\mathbf{v}_2 - \mathbf{p}_{0m}\|^2 < 0.2\ \delta_t^2$), dann sei $n_t = 1$ (s. Abb. 14.5).

3. Erzeugung von Dreiecken:
   Falls $n_t = 1$, dann erhalten wir 1 neues Dreieck: $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{p}_{0m})$
   andernfalls seien $\mathbf{q}_0, \mathbf{q}_{n_t}$ die senkrechten Projektionen von $\mathbf{v}_1, \mathbf{v}_2$ in die Tangentialebene am Punkt $\mathbf{p}_{0m}$ und $\mathbf{q}_i$ sei der Punkt, der durch Drehung von $\mathbf{p}_{0m} + \delta_t(\mathbf{q}_0 - \mathbf{p}_{0m})/\|\mathbf{q}_0 - \mathbf{p}_{0m}\|$ um den Winkel $i\Delta\omega$ mit der Normale im Flächenpunkt $\mathbf{p}_{0m}$ als Drehachse entsteht. (Falls eine globale bounding box aktiv ist, wird die Kante $\mathbf{p}_{0m}\mathbf{q}_i$ gekürzt und die Variable `boarder_point` des entsprechenden neuen Flächenpunktes auf `true` gesetzt. Randpunkte werden bei weiteren Betrachtungen ignoriert.) Anwendung von `surfacepoint` auf $\mathbf{q}_i$ liefert die neuen Punkte $\mathbf{p}_{N+i}$, $i = 1, .., n_t - 1$, und die n_t neuen Dreiecke:
   $(\mathbf{v}_1, \mathbf{p}_{N+1}, \mathbf{p}_{0m}), (\mathbf{p}_{N+1}, \mathbf{p}_{N+2}, \mathbf{p}_{0m}), ..., (\mathbf{p}_{N+n_t-1}, \mathbf{v}_2, \mathbf{p}_{0m})$.

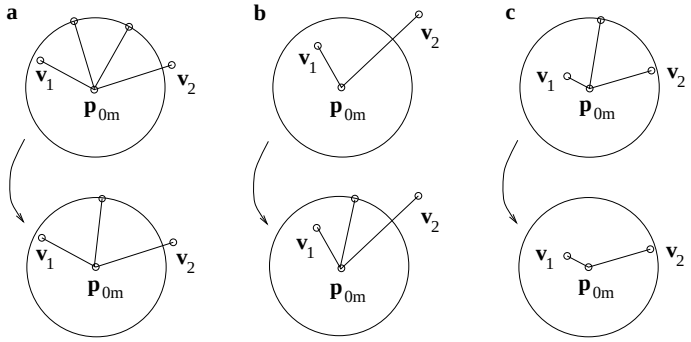


Abbildung 14.5: Korrekturen für extreme Fälle

4. Erneuere das aktuelle Frontpolygon:
   Streiche den Punkt $\mathbf{p}_{0m}$ und, falls $n_t > 1$, füge an seine Stelle die neuen Punkte $\mathbf{p}_{N+1}, ..., \mathbf{p}_{N+n_t-1}$ ein. Alle boolschen Variablen `angle_changed` der Punkte $\mathbf{v}_1, \mathbf{v}_2, \mathbf{r}_1, ..., \mathbf{r}_{n_t-1}$ werden auf `true` gesetzt.

14.2 Beispiele

Beispiel 14.1 *Kugel*
Triangulierung der Kugel $x^2 + y^2 + z^2 - 4 = 0$ *mit Startpunkt (1,1,1) und Kantenlänge* $\delta_t = 0.3$. *Die folgenden Abbildungen zeigen die ersten vier aktuelle Frontpolygone und die Situation nach der Erzeugung von 101 und 1531 Dreiecke. Die volle Triangulierung der Kugel umfaßt 1544 Dreiecke.*

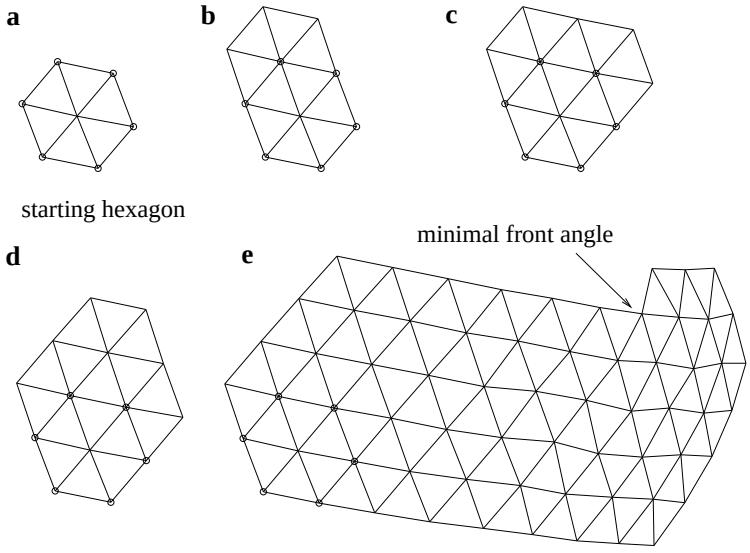


Abbildung 14.6: Triangulierung einer Kugel

Beispiel 14.2 *Zylinder*
Triangulierung des Zylinders $x^2 + y^2 - 1 = 0$,

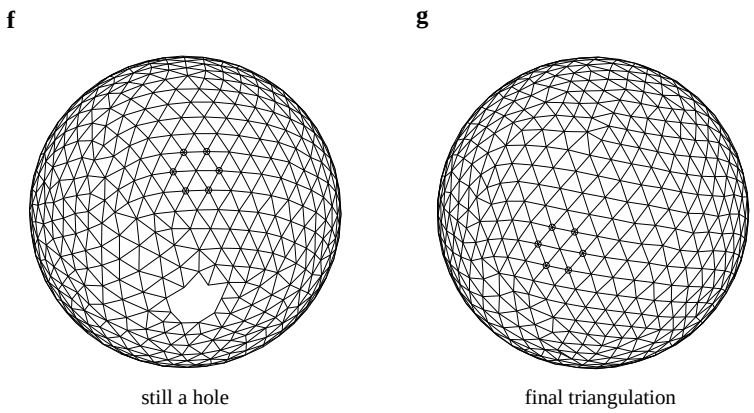


Abbildung 14.7: Triangulierung einer Kugel (Forts.)

1. *mit Starpukt* $(1,0,0)$ *und* $\delta_t = 0.2$. *Abb. 14.8a,b zeigt die Triangulation vor und nach der ersten Teilung des aktuellen Frontpolygons. Der Zylinder ist durch eine bounding box beschränkt.*
2. *mit Punkte auf dem Deckelkreis als erstes Frontpolygon und Punkte auf dem Bodenkreis als weiteres (begrenzendes) Frontpolygon (Abb. 14.8).*

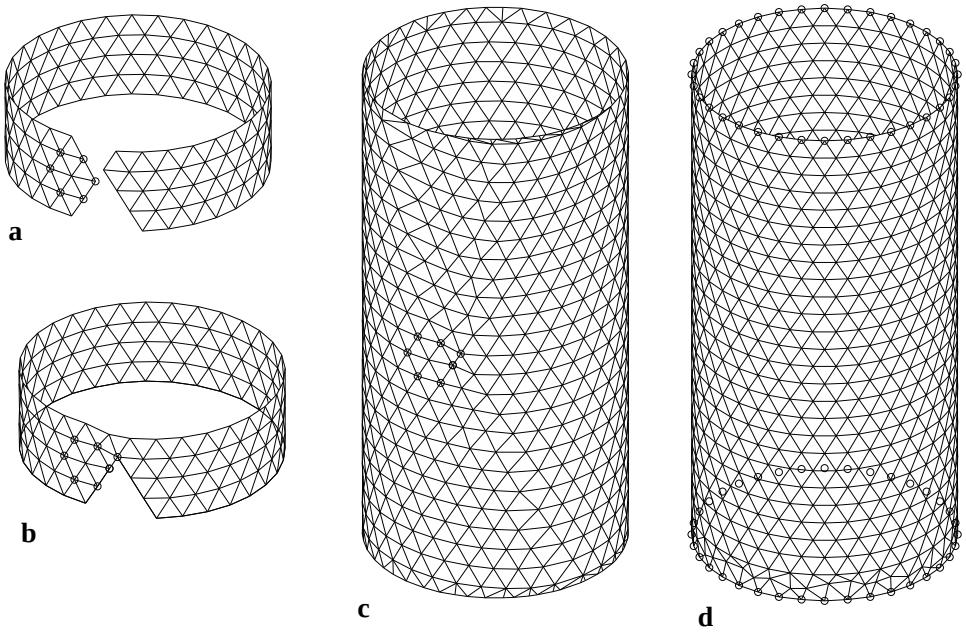


Abbildung 14.8: Triangulierung eines Zylinders

Beispiel 14.3 *Torus*
Triangulierung des Torus

$$(x^2 + y^2 + z^2 + r^2 - a^2)^2 - 4r^2(x^2 + y^2) = 0,\ r = 1, a = 0.35$$

mit $\delta_t = 0.1$*,*

1. *Startpunkt* $(1, 0, 0.5)$*. Abb. 14.9d,e zeigt die Triangulierung vor und nach der Vereinigung des aktuellen Frontpolygons mit dem weiteren Frontpolygon, das bei einer früheren Teilung des aktuellen Frontpolygons erzeugt wurde. Abb. 14.10f zeigt die vollständige Triangulation.*
2. *Punkte auf einem Kreis als erstes aktuelles Frontpolygon und*
   (a) *dasselbe Polygon (mit umgekehrter Orientierung) als weiteres (begrenzendes) Frontpolygon (Abb. 14.10h).*
   (b) *einem zweiten Polygon als weiteres (begrenzendes) Polygon (Abb. 14.10g). (Torusteil)*

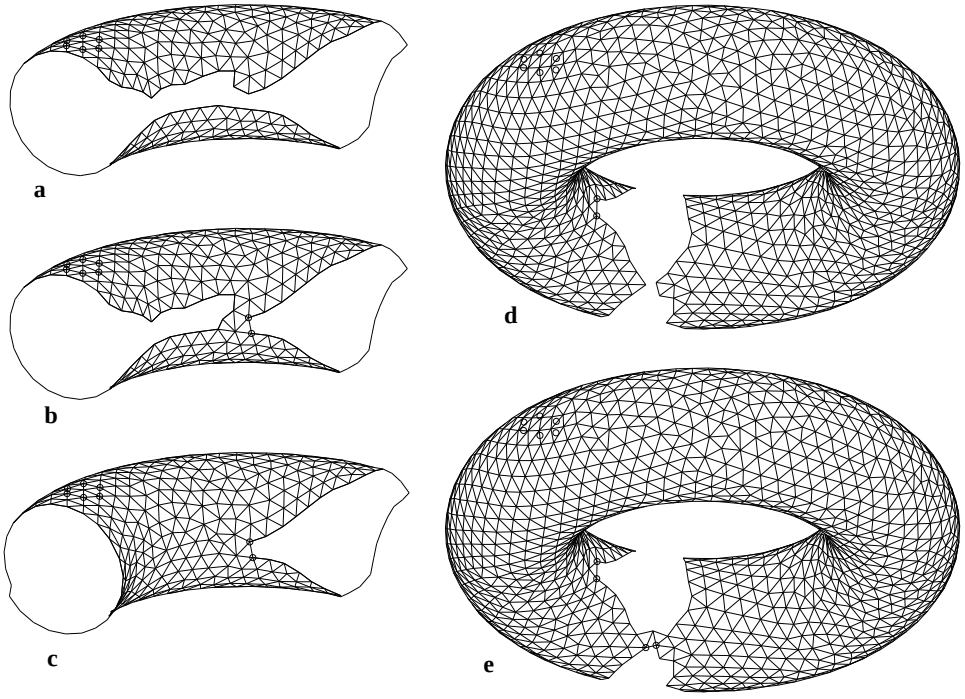


Abbildung 14.9: Triangulierung eines Torus

Beispiel 14.4 *Fläche vom Geschlecht 3*
Triangulierung einer impliziten Fläche vom Geschlecht 3 mit der Gleichung

$$r_z^4 z^2 - (1 - (x/r_x)^2 - (y/r_y)^2)((x - x_1)^2 + y^2 - r_1^2)(x^2 + y^2 - r_1^2)((x + x_1)^2 + y^2 - r_1^2) = 0,$$

$r_x = 6, r_y = 3.5, r_z = 4, r_1 = 1.2, x_1 = 3.9$ *und Startpunkt* $(0, 3, 0)$*, Kantenlänge* $\delta_t = 0.3$*. Die Trianagulierung besteht aus 7354 Dreiecke.*

Beispiel 14.5 *6-peak-Fläche*
Die Triangulierung der ziemlich komplizierten impliziten Fläche

$$(3x^2 - y^2)^2 y^2 - (x^2 + y^2)^4 - z^3 - 0.001z = 0$$

ist möglich ohne Teilung und Vereinigung.

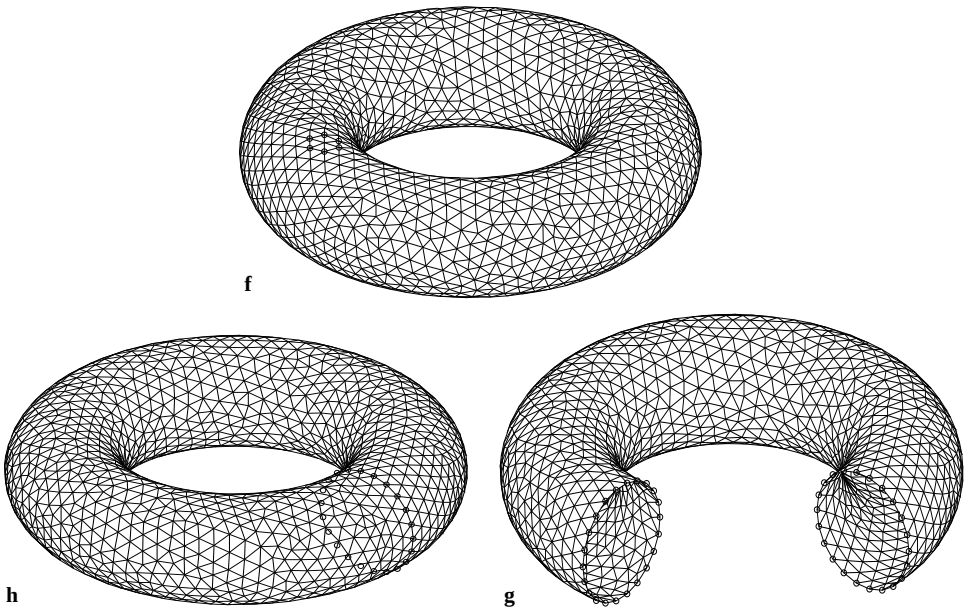


Abbildung 14.10: Triangulierung eines Torus (Forts.)

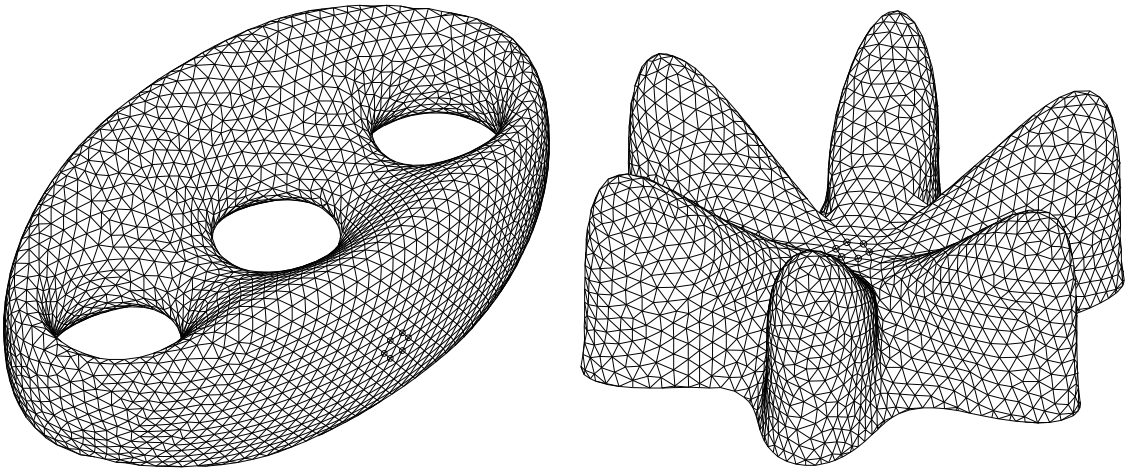


Abbildung 14.11: Fläche vom Geschlecht 3 bzw. Fläche mit 6 Peaks

Zum Schluß noch ein Beispiel, dessen implizite Darstellung Normalformen umfaßt (vgl. 12.2.7). Die Normalformen sind überhaupt der Schlüssel zur Triangulierung fast beliebiger Flächen. Mit Hilfe von Normalformen können sogar parametrisierte Flächen mit dem hier vorgestellten Algorithmus trianguliert werden.

Die in Aufgabe 12.7 beschriebene glatte Approximation von impliziten Flächen läßt sich mit Hilfe der Normalform auf fast beliebige Flächen anwenden. Da die Approximation wieder eine implizite Fläche ist, kann sie mit dem Triangulierungsalgorithmus für implizite Flächen dargestellt werden.

Beispiel 14.6 *Gegeben seien*

1. *die implizite Fläche* $\Phi_1 :\ (x-2)^4 + y^4 - r_1^4 = 0,\ r_1 = 2,$
2. *das parametrisierte Flächenstück*
   $\Phi_2 :\ \mathbf{x} = (10v - 5, 10u - 5, 6(u - u^2 + v - v^2)),\ 0 \le u \le 1, 0 \le v \le 0.8,$
3. *das parametrisierte Flächenstück*
   $\Phi_3 :\ \mathbf{x} = (6(u - u^2 + v - v^2) - 5, 10u - 5, 10v - 5),\ 0 \le u \le 1, 0.5 \le v \le 1.$

$h_1(\mathbf{x}) = 0, h_2(\mathbf{x}) = 0, h_3(\mathbf{x}) = 0$ *seien ihre Normalformen. Die implizite Fläche* $\Phi :\ f(\mathbf{x}) := h_1(\mathbf{x})h_2(\mathbf{x})h_3(\mathbf{x}) = c,\ c > 0$ *ist eine glatte Approximation des Flächenkomplexes* Φ_1, Φ_2, Φ_3. *Die Approximation ist unabhängig von ihren Darstellungen.*
Abb. 14.12 zeigt eine Triangulierung von $f(\mathbf{x}) = c$ *für* $c = 0.2$.

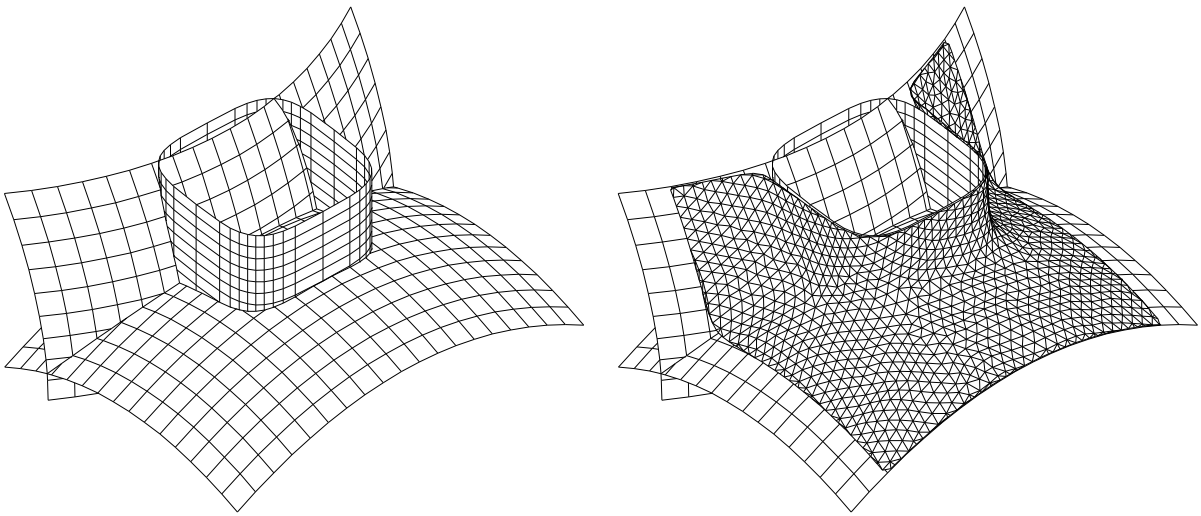


Abbildung 14.12: zu Beispiel 14.6

Kapitel 15

ÜBERGANGSFLÄCHEN

Problemstellung:
Bei der Produktion von Werkstücken verwendet man oft Formen, die aus Standard-Werkstücken wie Würfel, Zylinder, Kugel, Torus, ... zusammengesetzt werden. Dabei können Kanten in Form von Durchdringungskurven entstehen. Diese Kanten sollen vor der Verwendung der Formen durch geeignete Übergangsflächen "ausgerundet" werden.

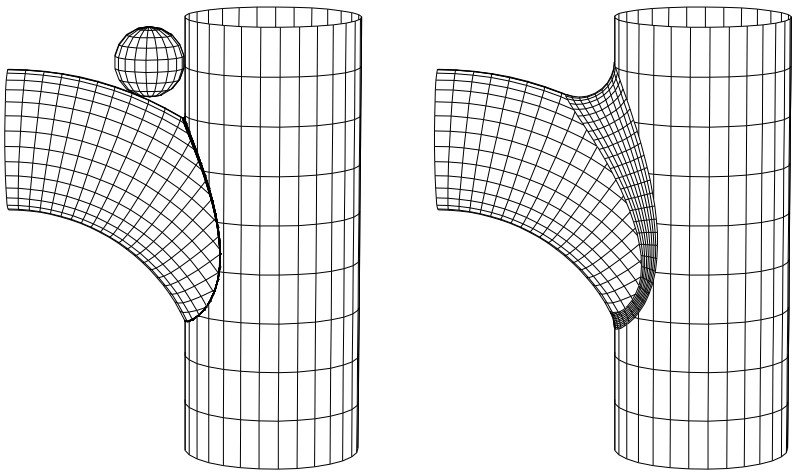


Abbildung 15.1: Übergangsfläche: mit rollender Kugel

Früher erzeugte man solche Übergangsflächen, indem man eine Kugel so durch den Raum bewegte, daß sie die beiden sich durchdringenden Flächen gleichzeitig berührte. Als Übergangsfläche benutzte man dann einen Teil der von der Kugel beim Rollen erzeugte Kanalfläche (s. Abb. 15.1). Fläche und Übergangsfläche hatten entlang der Übergangskurve (Berührkurve der Kugel) dieselben Tangentialebenen (G_1-Stetigkeit).

Diese klassische Methode dient bei einigen computerunterstützten Verfahren als Vorbild. Doch bietet der Computer mehr Möglichkeiten. Relativ einfache Konstruktionen von Übergangsflächen lassen sich für implizite Flächen angeben. Sie basieren auf der geschickten Verallgemeinerung von sehr einfachen Situationen im $\mathbb{R}^2$.

15.1 Funktionale Splines

15.1.1 Parabolische funktionale Splines (pFS)

Die beiden Geraden $\beta : y = 0$ und $\tau : x = 0$ schneiden sich im Punkt $\beta \cap \tau : (0,0)$. Die Parabel $\sigma : \mathbf{y - x^2 = 0}$ *berührt* die Gerade β im Schnittpunkt $\beta \cap \tau : (0,0)$. Ersetzt man die Geraden $\beta : y = 0$ und $\tau : x = 0$ durch geeignete sich schneidende Kurven $\beta : f(x,y) = 0$ und $\tau : g(x,y) = 0$, so ist $\sigma : F := f - g^2 = 0$ eine Kurve, die die Kurve $\beta : f = 0$ in den Schnittpunkten $\beta \cap \tau : f = 0, g = 0$ berührt. Diese Berühreigenschaft (G^1-Stetigkeit) bleibt auch dann noch erhalten, wenn man von einer Parabel der Schar $(1-\mu)y - \mu x^2 = 0, \quad 0 < \mu < 1$, ausgeht. Erhöht man den Exponenten von 2 auf mindestens 3 , so erhält man sogar *krümmungsstetige* Übergänge (G^2-Stetigkeit).

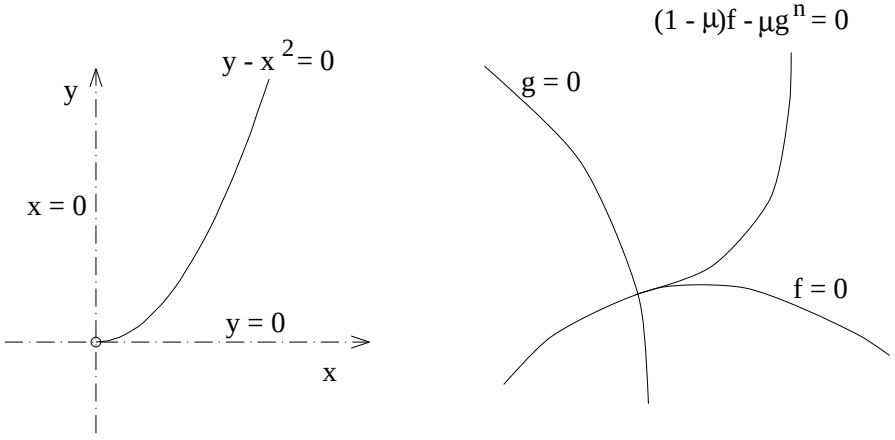


Abbildung 15.2: parabolische funktionale Splinekurven

Es seien $\beta : f(x,y) = 0$ und $\tau : g(x,y) = 0$ zwei sich schneidende **Kurven** im $\mathbb{R}^2$, f und g seien $(n-1)-$mal differenzierbar, $n \in \mathbb{N}$.
Jede Kurve σ der Schar

$$\sigma(\mu, n) : F := (1-\mu)f - \mu g^n = 0, \quad 0 < \mu < 1, \quad 2 \leq n \in \mathbb{N}$$

heißt *parabolische funktionale Spline* (pFS)Kurve, β ihre *Basis*, τ ihre *Transversale*, n ihr *Exponent* und μ ihr *Paramater*.

Als *Definitionsbereich* von F wählt man den Bereich von $\mathbb{R}^2$, in dem die Funktionen f und g positiv sind. (Um die gewünschten Übergangskurven zu erhalten, muß man u.U. f oder g mit -1 multiplizieren.)

Man rechnet leicht nach, daß in den Schnittpunkten $\beta \cap \tau : f = 0, g = 0$ die partiellen Ableitungen bis $(n-1)-$ter Ordnung von F mit denen von $(1-\mu)f$ identisch sind. Hieraus folgt:

$$\sigma(\mu, n) \text{ schließt in } \quad \beta \cap \tau \quad G^{n-1} - \text{stetig an die Basis } \beta \text{ an.}$$

(vgl. LI,HO,HA'90)

Beispiel 15.1 *a)* $\beta : f(x,y) = 1 - x^2 - y^2 = 0$ *(Kreis),* $\tau : g(x,y) = y = 0$ *(Gerade). Jede Kurve der Schar* $\sigma : F := (1-\mu)f - \mu g^n = 0, \quad 0 < \mu < 1, \quad n \geq 2,$ *berührt den Kreis in den Punkten* $(-1,0), (1,0)$.

b) $\beta : f = l_1 \cdot l_2 = 0, \quad \tau : g = l_0 = 0, \quad$ *wobei* $\quad l_i(x,y) := a_i x + b_i y - c_i = 0$ *Geraden sind.*

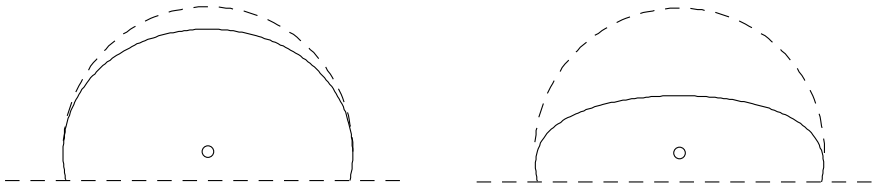


Abbildung 15.3: zu Beispiel 15.1 a)

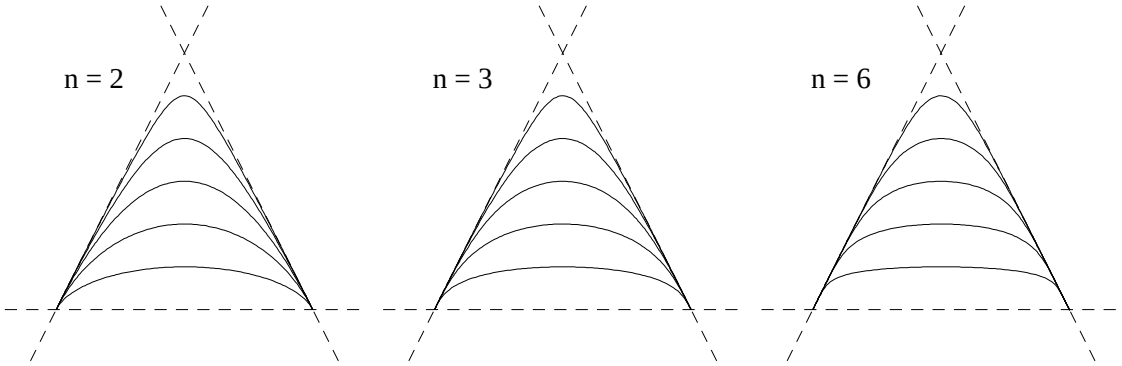


Abbildung 15.4: zu Beispiel 15.1 b)

Abbildung 15.5: Anwendungen von Kurven aus Beispiel 15.1 b)

Diese Methode läßt sich auf Flächen ausdehnen.

Beispiel 15.2 $\beta : f(x,y,z) = 1 - x^2 - y^2 - z^2 = 0$ *die Einheitskugel,*
$\tau : g(x,y,z) = z = 0$ *die Äquatorebene.*

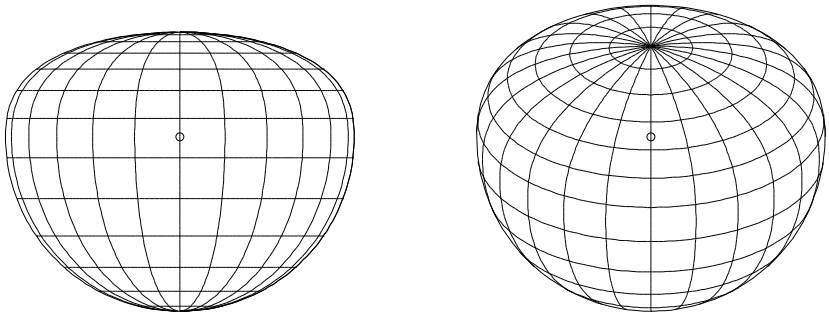


Abbildung 15.6: zu Beispiel 15.2

Jede Fläche σ der Schar $F := (1-\mu)f - \mu g^n = 0, \quad 0 < \mu < 1, \quad n \geq 2$, berührt die Kugel entlang dem Äquatorkreis.

Die Wirkung der beiden *Designparameter* μ und n läßt sich hier so beschreiben: μ legt fest, wie "nah" die neue Fläche der Kugel (μ klein) oder der Äquatorebene ($\mu \approx 1$) ist.
Mit wachsendem n schmiegt sich die Fläche $\sigma : F = 0$ entlang der Schnittkurve der Ausgangsflächen $\beta \cap \tau : f = 0,\ g = 0$ immer mehr der Basisfläche $\beta : f = 0$ an.

Geht man nun von zwei sich schneidenden **Flächen** $\beta_1 : f_1 = 0,\ \beta_2 : f_2 = 0$, den *Basisflächen*, aus, deren Schnittkurve $\beta_1 \cap \beta_2$ man ausrunden möchte, so muß man eine geeignete *Transversalfläche* $\tau : g = 0$ wählen, mit deren Hilfe man die Kurven $\Gamma_1 : f_1 = 0,\ g = 0$ und $\Gamma_2 : f_2 = 0,\ g = 0$ festlegt, entlang denen sich die Ausrundungsfläche an die Basisflächen β_1, β_2 anschmiegen soll. Die Vorzeichen der Funktionen f_1, f_2, g sind so zu wählen, daß die gewünschte Übergangsfläche im Positivbereich dieser Funktionen verläuft. Sind die Flächen $\beta_1 : f_1 = 0,\ \beta_2 : f_2 = 0$ und $\tau : g = 0$ hinreichend glatt, so ist auch die Übergangsfläche

$$\sigma : F := (1-\mu)f_1 f_2 - \mu g^n = 0, \quad 0 < \mu < 1, \quad n \geq 2,$$

im Positivbereich der Funktionen f_1, f_2 und g eine hinreichend glatte Fläche.(s. LI,HO,HA90, WA90).
Die Fläche $\sigma : F = 0$ heißt (analog zu den Kurven) eine zu den Basisflächen $\beta_1 : f_1 = 0,\ \beta_2 : f_2 = 0$ und der Transversal-Fläche $\tau : g = 0$ gehörige **p***arabolische* **f***unktionale* **S***pline-Fläche*, **pFS**.

Für parabolische funktionale Spline Flächen gilt wie für pFS-Kurven:
σ schließt in der Kurve $\Gamma = \beta \cap \tau$ G^{n-1}-stetig an die Bais β an.

Insbesondere gilt (LI,HO,HA'90):
Für $n \geq 2$ gehen die Tangentialebenen an den Flächenübergängen stetig ineinander über (G^1-stetig).
Für $n \geq 3$ sind außerdem die Normalkrümmungen an den Flächenübergängen stetig (G^2-stetig).

Beispiel 15.3

$$\begin{array}{rcccll} f_1(x,y,z) & = & r^2 - x^2 - y^2 - z^2 & = & 0 & \textit{(Kugel)} \\ f_2(x,y,z) & = & x & = & 0 & \textit{(Ebene)} \\ g(x,y,z) & = & z & = & 0 & \textit{(Ebene)} \end{array}$$

Beispiel 15.4

$$\begin{array}{rcccll} f_1(x,y,z) & = & r^2 - x^2 - z^2 & = & 0 & \textit{(Zylinder)} \\ f_2(x,y,z) & = & r^2 - y^2 - z^2 & = & 0 & \textit{(Zylinder)} \\ g(x,y,z) & = & z & = & 0 & \textit{(Ebene)} \end{array}$$

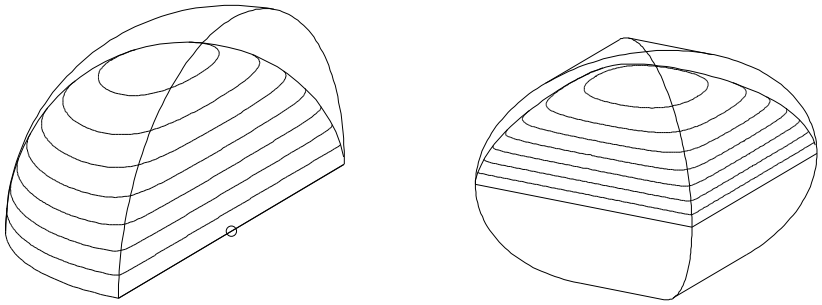


Abbildung 15.7: zu Beispiel 15.3 und Beispiel 15.4

Die obige Konstruktion läßt sich auf m Basisflächen $f_1 = 0,\ f_2 = 0, \ldots,\ f_m = 0$ ausdehnen.

Beispiel 15.5 $f_i = 0,\ i = 1, \ldots, 4,\ g = 0$ *Ebenen*

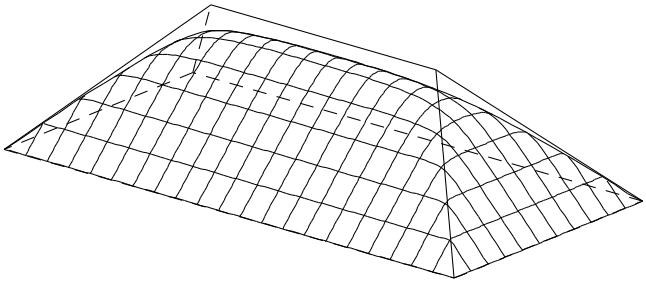


Abbildung 15.8: zu Beispiel 15.5

Beispiel 15.6 *Die Gleichung*

$$(1-\mu)(x^2+y^2-1)(x^2+z^2-1)(y^2+z^2-1) - \mu(9-x^2-y^2-z^2)^3 = 0$$

beschreibt eine G^2-stetige Übergangsfläche zwischen den drei Zylindern $x^2+y^2-1=0,\ x^2+z^2-1=0,\ y^2+z^2-1=0$. In Abb. 15.9 ist $\mu = 0.0003$.

15.1.2 Symmetrisierte parabolische funktionale Splines (spFS)

Parabolische funktionale Splines lassen sich noch auf die folgende Weise symmetrisieren:
Es seien zwei Flächen $\beta_1 : f_1 = 0,\ \beta_2 : f_2 = 0$ gegeben und jeweils eine Schnittkurve $\Gamma_i : f_i = 0,\ g_i = 0,\ i = 1, 2$, mit zwei verschiedenen transversalen Flächen $\tau_1 : g_1 = 0$ bzw. $\tau_2 : g_2 = 0$. Sind die Funktionen f_1, f_2, g_1, g_2 positiv, die Kurven Γ_1, Γ_2 disjunkt und die gegebenen Flächen hinreichend glatt, so ist die implizite Fläche

$$\sigma :\ F = (1-\mu) f_1 g_2^n - \mu f_2 g_1^n = 0 \quad \text{für} \quad \text{für} 0 < \mu < 1, \quad n \geq 2$$

eine Übergangsfläche zwischen den Basisflächen $\beta_1 : f_1 = 0$ und $\beta_2 : f_2 = 0$ durch die Kurven $\Gamma_1 : \beta_1 \cap \tau_1$ und $\Gamma_2 : \beta_2 \cap \tau_2$.
Die Fläche $\sigma : F = 0$ heißt eine **s***ymmetrisierte* **p***arabolische* **f***unktionale* **S***pline-Fläche,* **spFS**.

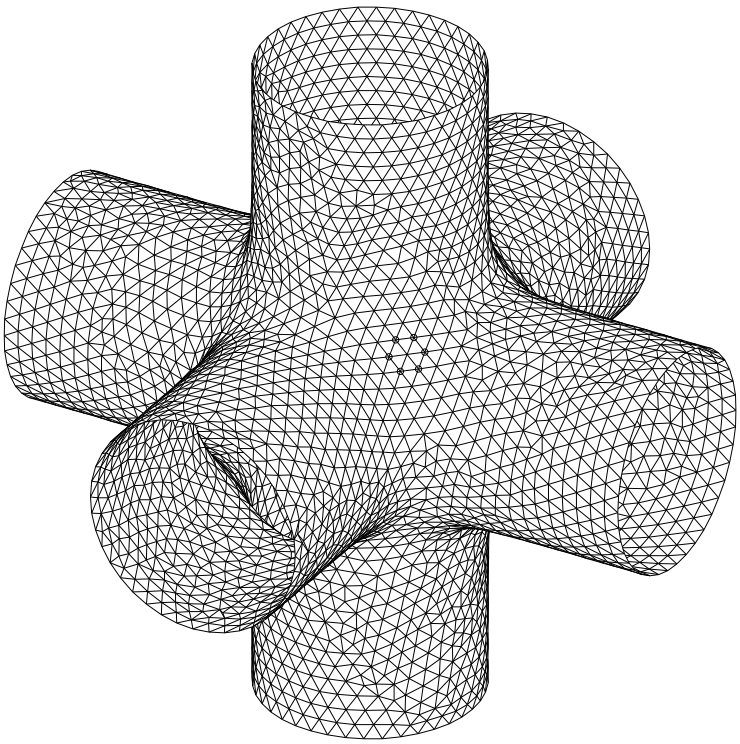


Abbildung 15.9: G^2-Übergangsfläche dreier Zylinder

Beispiel 15.7

$$\begin{array}{lcll} f_1(x,y,z) & = & 4-x^2-y^2 & (Zylinder), \\ f_2(x,y,z) & = & x^2-y^2-1 & (Zylinder), \end{array} \qquad \begin{array}{lcll} g_1(x,y,z) & = & z+1 & (Ebene) \\ g_2(x,y,z) & = & 1-z & (Ebene) \end{array}$$

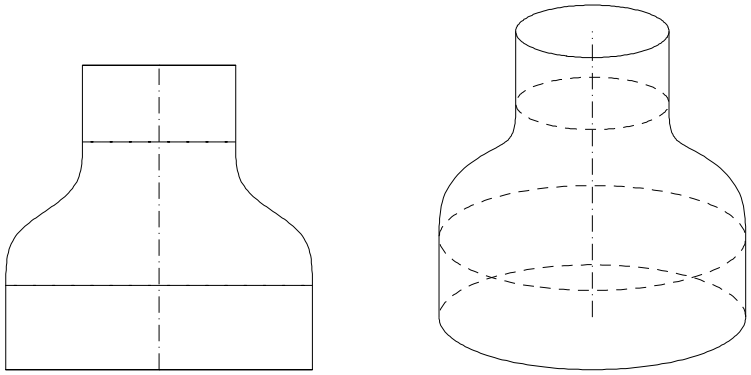


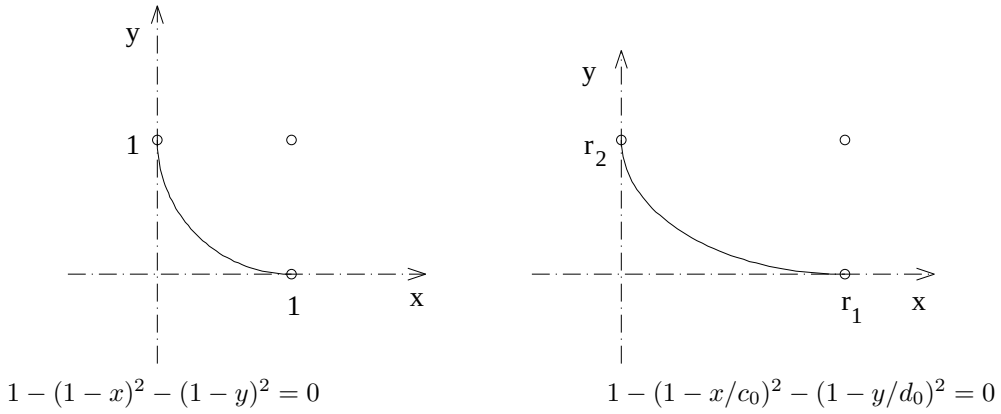
Abbildung 15.10: zu Beispiel 15.7

Die Methode der spFS läßt sich auch auf 3 und mehr Basisflächen ausdehnen (s. WA'90).

Bemerkung:
Für $n \geq 2$ gehen die Tangentialebenen an den Flächenübergängen stetig ineinander über (G^1-stetig).
Für $n \geq 3$ sind außerdem die Normalkrümmungen an den Flächenübergängen stetig (G^2-stetig).

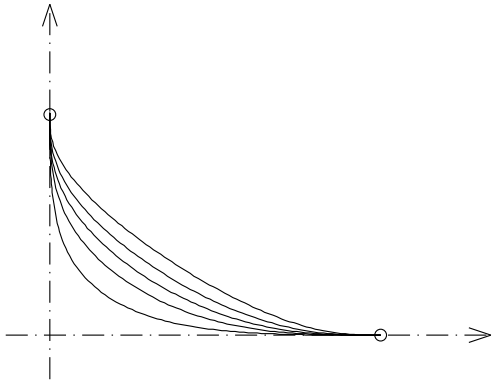
15.1.3 Elliptische funktionale Splines (eFS)

Der Kreis $1-(1-x)^2-(1-y)^2=0$ berührt die Gerade $\beta_1: y=0$ im Punkt $(1,0)$ und die Gerade $\beta_2: x=0$ im Punkt $(0,1)$. Ersetzt man auch hier die Koordinatenachsen durch die Gleichungen von zwei geeigneten sich schneidenden Flächen $\beta_1: f_1(x,y,z)=0$ und $\beta_2: f_2(x,y,z)=0$, so erhält man die Gleichung $F:=1-(1-f_1)^2-(1-f_2)^2=0$ einer Fläche, die die Fläche $\beta_1: f_1=0$ in der Schnittkurve $\Gamma_1: f_1=0, f_2=1$ und die Fläche $\beta_2: f_2=0$ in der Schnittkurve $\Gamma_2: f_1=1, f_2=0$ berührt.



$1-(1-x)^2-(1-y)^2=0$ $\qquad$ $1-(1-x/c_0)^2-(1-y/d_0)^2=0$

Abbildung 15.11: Korrelationskurven für elliptische funktionale Splines



$(1-\mu)\frac{xy}{c_0 d_0}-\mu(1-\frac{x}{c_0}-\frac{y}{d_0})^3=0,\ \mu=0.1,...,0.82$

Abbildung 15.12: Parabolische funktionale Splines als Korrelationskurven

Diese Methode läßt sich folgendermaßen verallgemeinern:

Es seien $c_0,d_0\in\mathbb{R}^+, \beta_1: f_1=0, \beta_2: f_2=0$ zwei sich schneidende **Flächen**, deren Funktionen f_1,f_2 die Wertebereiche $[0,c_0]$ und $[0,d_0]$ besitzen. Ferner sei $h(x,y)=0$ eine glatte Kurve, die die Koordinatenachsen in den Punkten $(c_0,0),(0,d_0)$ berührt. Sind die Funktionen f_1,f_2,h hinreichend differenzierbar, so ist die Fläche $\sigma: F:=h(f_1,f_2)=0$ eine Übergangsfläche, die die Fläche $\beta_1: f_1=0$ in der Kurve $\Gamma_1: f_1=0,\ f_2=d_0$ und die Fläche $\beta_2: f_2=0$ in der Kurve $\Gamma_2: f_1=c_0, f_2=0$ berührt (G^1-stetig). Wir nennen $\sigma: F=0$ eine zu den *Basisflächen* $\beta_1: f_1=0, \beta_2: f_2=0$ gehörige *elliptische Funktionale Spline-Fläche*, **eFS**. Die Parameter c_0,d_0 legen fest, wo die Übergangsfläche

ansetzt.
Beispiele für geeignete Korrelationskurven $h = 0$ sind:

- die "Superellipsen"

$$h(x,y) = 1 - (1 - x/c_0)^n - (1 - y/d_0)^n = 0 \quad \text{für } n \geq 2$$

und

- die parabolischen funktionale Spline Kurven

$$h(x,y) = (1-\mu)\frac{xy}{c_0 d_0} - \mu(1 - \frac{x}{c_0} - \frac{y}{d_0})^n = 0,\ 0 < \mu < 1,\ n \geq 2,$$

Für $n \geq 3$ sind auch die Normalkrümmungen stetig (G^2-Stetigkeit).
n heißt der *Exponent* der eFS–Fläche.

Beispiel 15.8

$$\begin{array}{rcccl} f_1(x,y,z) & = & x^2 + y^2 - 1.5^2 & = & 0 \qquad (Zylinder) \\ f_2(x,y,z) & = & y^2 + z^2 - 2.3^2 & = & 0 \qquad (Zylinder) \end{array}$$

$h(x,y) = (1-\mu)\frac{xy}{c_0 d_0} - \mu(1 - \frac{x}{c_0} - \frac{y}{d_0})^n = 0, \quad c_0 = 2.4,\ d_0 = 6,\ \mu = 0.2,\ n = 3,$

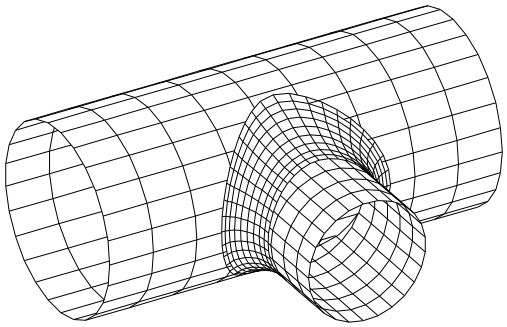


Abbildung 15.13: zu Beispiel 15.8

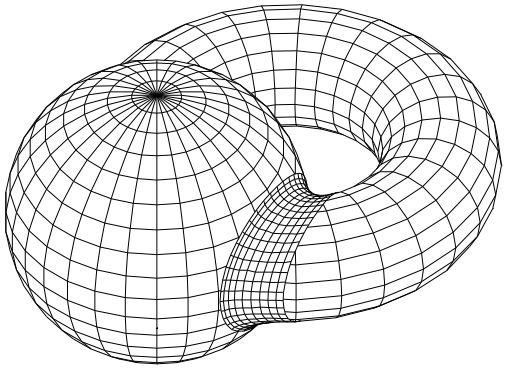


Abbildung 15.14: Übergangsfläche zwischen Torus und Kugel

Die elliptische Methode wird in der Literatur auch als *Potentialmethode* (HO,HO'85) bezeichnet.

15.2 Übergangsflächen für beliebige Flächen

Jede Fläche, deren Normalform (vgl. 12.2.7) numerisch zur Verfügung steht, kann als Basisfläche einer der obigen Methoden verwendet werden.
Abb. 15.2 zeigt eine Übergangsfläche zwischen zwei Bézier–Flächen, zwischen einer Bézierfläche und einer impliziten Fläche (Zylinder) sowie eine Übergangsfläche zwischen den beiden Übergangsflächen.

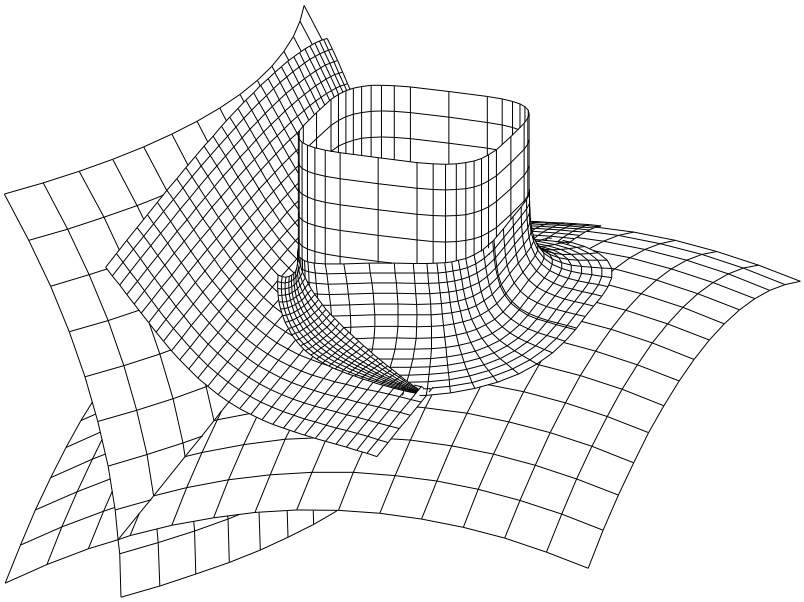


Abbildung 15.15: Übergangsflächen zwischen beliebigen Flächen

Kapitel 16

ABWICKELBARE FLÄCHEN

Eine Fläche heißt abwickelbar , wenn sie längen- und winkeltreu in den $\mathbb{R}^2$ abgebildet werden kann. In die Ebene abwickelbar sind genau die (allgemeinen) Zylinder, Kegel und die Tangentenflächen (Flächen die durch die Tangenten einer Raumkurve erzeugt werden). Abwickelbare Flächen heißen auch Torsen. Sie sind Regelflächen (oder Strahlflächen), d.h. sie werden von einer Geradenschar überdeckt. Doch ist nicht jede Regelfläche abwickelbar, z. B. das einschalige Hyperboloid. Eine Regelfläche ist abwickelbar, wenn sie längs jeder Erzeugenden (Gerade) immer dieselbe Tangentialebene besitzt.

Wir werden hier nur die in der Technik wichtigen senkrechten Kreis-Zylinder und Kegel, sowie die abwickelbaren Verbindungsflächen von Kurven behandeln.

16.1 Abwicklung eines senkrechten Kreiszylinders

Für die Abwicklung eines senkrechten Kreis-Zylinders denke man sich den Zylinder entlang einer Mantellinie aufgeschnitten und in die Ebene ausgebreitet. Bei der Abwicklung eines Punktes ist zu beachten, daß es darauf ankommt, an welcher Mantellinie aufgeschnitten wird . Wir wollen hier immer entlang der **Mantellinie** schneiden, die den Punkt $P_1 : \mathbf{p}_1$ des Basiskreises enthält.

Es sei jetzt

$$Z = \{\mathbf{q}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi + \mathbf{f}_3\alpha \mid 0 \le \varphi \le 2\pi, \alpha \in \mathbb{R}\}$$

ein senkrechter Kreiszylinder , d. h. es ist

$$\mathbf{f}_1 \cdot \mathbf{f}_2 = \mathbf{f}_1 \cdot \mathbf{f}_3 = \mathbf{f}_2 \cdot \mathbf{f}_3 = 0 \text{ und } \mathbf{f}_1^2 = \mathbf{f}_2^2 = r_z^2; \quad r_z \text{ ist der } \textit{Radius} \text{ von } Z.$$

Wir wickeln Z so ab, daß die Mantellinie $\{\mathbf{q}_0 + \mathbf{f}_1 + \mathbf{f}_3\alpha \mid \alpha \in \mathbb{R}\}$ auf die y-Achse und der Punkt $P_1 : \mathbf{p}_1 := \mathbf{q}_0 + \mathbf{f}_1$ auf den Nullpunkt abgebildet werden. Der Basiskreis $\{\mathbf{q}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi \mid \cdots\}$ wird dann auf eine Strecke der positiven $\overline{x}$-Achse der Länge $2\pi r_z$ abgewickelt.
Ein Punkt $Q : \mathbf{q}$ läßt sich durch

$$\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1 \cos\varphi_q + \mathbf{f}_2 \sin\varphi_q + \mathbf{f}_3\alpha_q, \text{ wobei (wegen } \mathbf{f}_i \cdot \mathbf{f}_j = 0, \text{ für } i \neq j)$$

$$\cos\varphi_q = \frac{(\mathbf{q} - \mathbf{p}_0) \cdot \mathbf{f}_1}{r_z^2}, \quad \sin\varphi_q = \frac{(\mathbf{q} - \mathbf{p}_0) \cdot \mathbf{f}_2}{r_z^2}, \quad \alpha_q = \frac{(\mathbf{q} - \mathbf{p}_0) \cdot \mathbf{f}_3}{\mathbf{f}_3^2}$$

ist, beschreiben . Mit Hilfe des Unterprogramms `polar_angle` berechnet man aus $\cos\varphi_q$ und $\sin\varphi_q$ den Winkel φ_q. Die Abwicklung von Q ist dann

$$\overline{Q} : \overline{\mathbf{q}} = (\varphi_q r_z, \alpha_q |\mathbf{f}_3|)$$

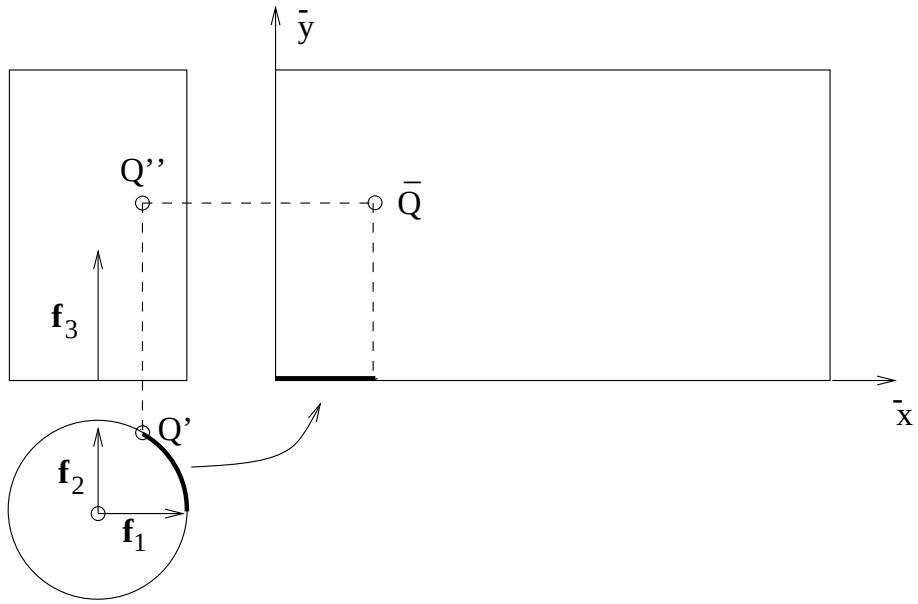


Abbildung 16.1: Abwicklung eines senkrechten Kreiszylinders

Das Unterprogramm `develop_cylpts` berechnet die Abwicklungen von Punkten des durch $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$ gegebenen senkrechten Kreiszylinders (Es ist $\mathbf{f}_i = \mathbf{q}_i - \mathbf{q}_0,\ i = 1, 2.$).

```
procedure develop_cylpts(q0,q1,q2: vt3d; rz: real; p: vts3d; n1,n2: integer;
                                                          var pd: vts2d);
{Berechnet die Abwicklung einer Punktreihe auf einem senkrechten Kreiszylinder}
 var f1,f2,av,vv : vt3d;     i : integer;
     rz2,a2,ab,cw,sw,al,wp : real;
begin
  diff3d(q1,q0, f1);       diff3d(q2,q0, f2);      vectorp(f1,f2,av);
  a2:= scalarp3d(av,av);   ab:= sqrt(a2);          rz2:= rz*rz;
  for i:= n1 to n2 do
      begin
      diff3d(p[i],q0, vv);
      cw:= scalarp3d(vv,f1) / rz2;      sw:= scalarp3d(vv,f2) / rz2;
      al:= scalarp3d(vv,av) / a2;       wp:= polar_angle(cw,sw);
      put2d(rz*wp,al*ab, pd[i]);
      end;
end; {develop_cylpts}
{*************}
```

Aufgabe 16.1 *Abwicklung eines ZYLINDERSTUMPFES:*

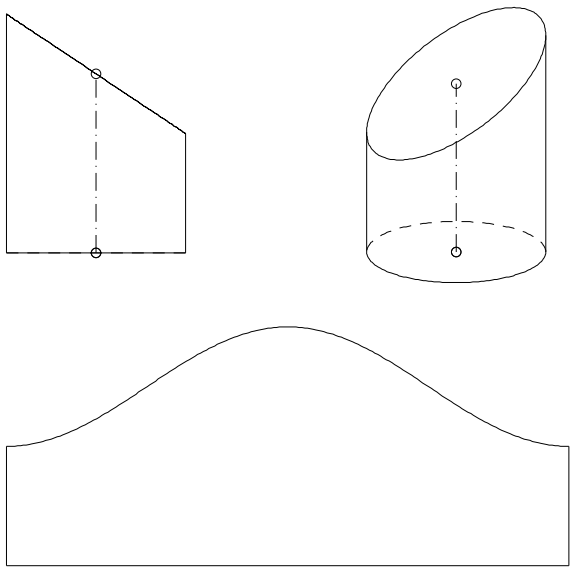


Abbildung 16.2: Abwicklung eines Zylinderstumpfes

16.2 Aufwicklung auf einen senkrechten Kreiszylinder

Unter der Aufwicklung eines Punktes auf einen senkrechten Kreiszylinder (Radius r_z) wollen wir die Umkehrung der in 16.1 beschriebenen Abwicklung verstehen. Ein Punkt $\overline{Q} : \overline{\mathbf{q}} = (\overline{x}, \overline{y})$ wird dabei auf den Punkt

$$Q : \mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1 \cos\varphi_q + \mathbf{f}_2 \sin\varphi_q + \mathbf{f}_3\alpha_q \quad \text{mit} \quad \varphi_q = \overline{x}/r_z, \quad \alpha_q = \overline{y}/|\mathbf{f}_3|$$

des Zylinders $Z = \{\mathbf{q}_0 + \mathbf{f}_1 \cos\varphi + \mathbf{f}_2 \sin\varphi + \mathbf{f}_3\alpha \mid \ldots\}$ abgebildet (aufgewickelt).

Das Unterprogramm `pts2d_on_cylinder` berechnet die Aufwicklung der Punktreihe **p2d**[i], $n1 \leq i \leq n2$, auf den durch $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$ bestimmten senkrechten Kreiszylinder mit Radius r_z. (Es ist $\mathbf{f}_i = \mathbf{q}_i - \mathbf{q}_0,\ i = 1, 2.$).

```
procedure  pts2d_on_cylinder(q0,q1,q2 : vt3d; rz : real; p2d : vts2d;
                                          n1,n2 : integer; var p: vts3d);
{Berechnet die Aufwicklung einer Punktreihe auf einen senkrechten Kreis=
 zylinder.}
var f1,f2,av : vt3d;    ab,wp,xi,eta,al : real;    i : integer;
begin
  diff3d(q1,q0, f1);     diff3d(q2,q0, f2);
  vectorp(f1,f2,av);     ab:= length3d(av);
  for i:= n1 to n2 do
      begin
      wp:= p2d[i].x/rz;    al := p2d[i].y/ab;
      xi:= cos(wp);        eta:= sin(wp);
      lcomb4vt3d(1,q0, xi,f1, eta,f2, al,av, p[i]);
      end;
end; {pts2d_on_cylinder}
{**************}
```

Aufgabe 16.2 *AUFWICKLUNG von KREISEN:*
(Als Hiddenline-Algorithmus kann man `curve_before_plane` *für die Umrißebene verwenden. Die Umrißebene erhält man mit Hilfe von* `cylinder_contpts`*.)*

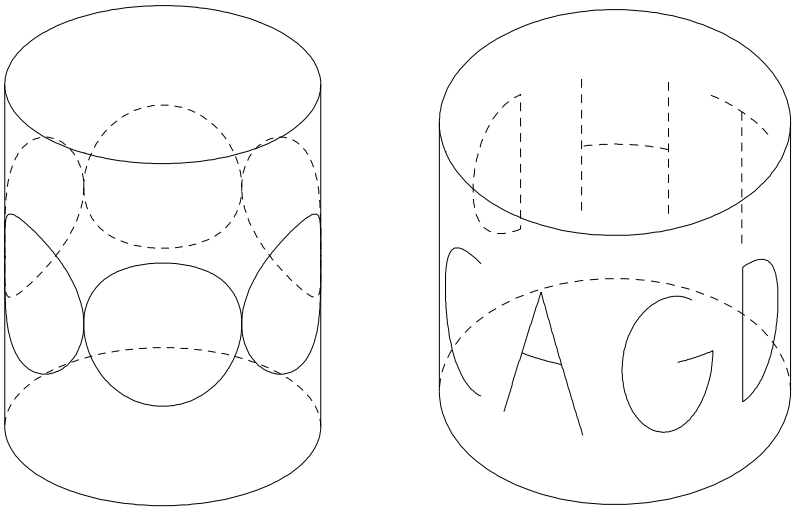


Abbildung 16.3: Aufwicklung von Kreisen bzw. Buchstaben auf einen Zylinder

16.3 Abwicklung eines senkrechten Kreiskegels

Es sei

$$K := \{\mathbf{q}_0 + \mathbf{f}_1\alpha\cos\varphi + \mathbf{f}_2\alpha\sin\varphi + \mathbf{f}_3(1-\alpha) \mid 0 \le \varphi \le 2\pi, \alpha \in \mathbb{R}\}$$

ein senkrechter Kreiskegel, d. h. $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ ist eine Orthogonalbasis. $r_k := |\mathbf{f}_1| = |\mathbf{f}_2|$ ist der Radius des Basiskreises, $S : \mathbf{s}_0 = \mathbf{q}_0 + \mathbf{f}_3$ die Spitze des Kegels. Wir wickeln K so ab, daß die Mantellinie durch den Punkt $P_1 : \mathbf{q}_1 = \mathbf{q}_0 + \mathbf{f}_1$ auf die x-Achse und die Spitze S auf den Nullpunkt abgebildet werden. Der Basiskreis $\{\mathbf{q}_0 + \mathbf{f}_1\cos\varphi + \mathbf{f}_2\sin\varphi \mid \ldots\}$ wird dann auf einen Keisbogen abgewickelt. Für einen Punkt $Q : \mathbf{q} \neq \mathbf{s}_0$ (Spitze) des Kegels gilt

$$\mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1\alpha_q\cos\varphi_q + \mathbf{f}_2\alpha_q\sin\varphi_q + \mathbf{f}_3(1-\alpha_q)$$

mit

$$\cos\varphi_q = \frac{(\mathbf{q}-\mathbf{p}_0)\cdot\mathbf{f}_1}{\alpha_q r_k^2}, \quad \sin\varphi_q = \frac{(\mathbf{q}-\mathbf{p}_0)\cdot\mathbf{f}_2}{\alpha_q r_k^2}, \quad \alpha_q = 1 - \frac{(\mathbf{q}-\mathbf{p}_0)\cdot\mathbf{f}_3}{\mathbf{f}_3^2}.$$

Mit Hilfe des Unterprogramms `polar_angle` berechnet man aus $\cos\varphi_q$ und $\sin\varphi_q$ den Winkel φ_q. Der φ_q entsprechende Winkel in der Abwicklung ist

$$\overline{\varphi}_q = \varphi_q r_k / \sqrt{r_k^2 + \mathbf{f}_3^2}.$$

Also wird der Punkt Q auf den Punkt

$$\overline{Q} : \overline{\mathbf{q}} = (\rho_q\cos\overline{\varphi}_q, \rho_q\sin\overline{\varphi}_q) \quad \text{mit} \quad \rho_q = \alpha_q\sqrt{r_k^2 + \mathbf{f}_3^2}$$

abgewickelt.

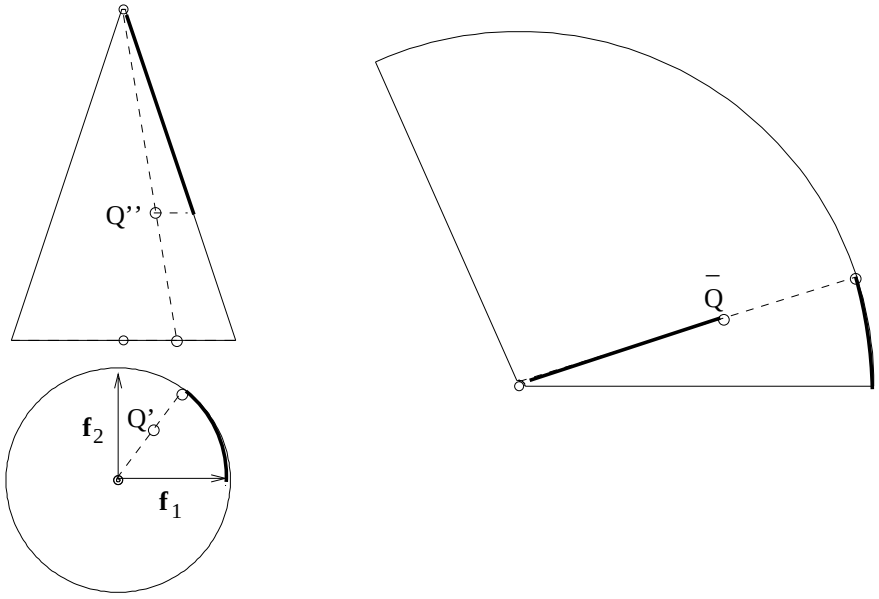


Abbildung 16.4: Abwicklung eines senkrechten Kreiskegels

Aufgabe 16.3 *Abwicklung eines KEGELSTUMPFs:*

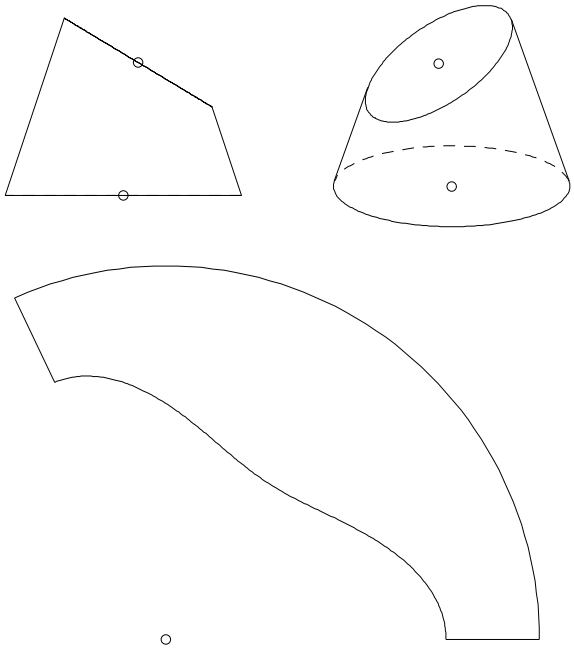


Abbildung 16.5: Abwicklung eines Kegelstumpfes

Das Unterprogramm `develop_conepts` berechnet die Abwicklung von Punkten $\mathbf{p}[i], n_1 \le i \le n_2$, des durch die Punkte $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$ und der Spitze **peak** bestimmten senkrechten Kreiskegels mit Radius `rk`. Der ganze Kegelmantel wird auf einen Kreissektor mit Radius `ml` und Winkel `psi` abgewickelt.

```
procedure develop_conepts(q0,q1,q2: vt3d; rk: real; peak: vt3d; p: vts3d; n1,n2: integer;
                                                          var pd: vts2d; var ml,psi: real);
{Berechnet die Abwicklung einer Punktreihe auf einem senkrechten Kreiskegel}
var f1,f2,av,vv : vt3d;    i : integer;
     rk2,a2,cw,sw,alq,w,wa,ra,q01,q02,q0a : real;
begin
  diff3d(q1,q0, f1);     diff3d(q2,q0, f2);
  vectorp(f1,f2,av);     diff3d(peak,q0, av);
  rk2:= rk*rk;             a2:= scalarp3d(av,av);
  ml:= sqrt(rk2 + a2);     psi:= pi2*rk/ml;
  for i := n1 to n2 do
      begin
      diff3d(p[i],q0,vv);
      q01:= scalarp3d(f1,vv);  q02:= scalarp3d(f2,vv);
      q0a:= scalarp3d(av,vv);
      alq:= 1 - q0a/a2;
      cw := q01/(alq*rk2);     sw:= q02/(alq*rk2);
      w:= polar_angle(cw,sw);
      wa := w*rk/ml;           ra:= alq*ml;
      pd[i].x:= ra*cos(wa);    pd[i].y:= ra*sin(wa);
     end;
 end; { develop_conepts}
{************}
```

16.4 Aufwicklung auf einen senkrechten Kreiskegel

Unter der Aufwicklung eines senkrechten Kreiskegels wollen wir die Umkehrung der in 16.3 beschriebenen Abwicklung verstehen. Dabei wird ein Punkt $\overline{Q} : \overline{\mathbf{q}} = (\overline{x}, \overline{y})$, der in der Abwicklung des Kegelmantels liegt, auf den Punkt

$$Q : \mathbf{q} = \mathbf{q}_0 + \mathbf{f}_1 \alpha_q \cos\varphi_q + \mathbf{f}_2 \alpha_q \sin\varphi_q + \mathbf{f}_3(1 - \alpha_q)$$

aufgewickelt. Es ist

$$\alpha_q = \sqrt{\overline{x}^2 + \overline{y}^2}/l_m, \text{ mit } \quad l_m = \sqrt{r_k^2 + \mathbf{f}_3^2},$$

und

$$\varphi_q = \overline{\varphi}_q l_m / r_k, \text{ wobei } \quad \overline{\varphi}_q \text{ mit } \quad \texttt{polar\_angle} \text{ aus } \quad (\overline{x}, \overline{y})$$

berechnet wird.

```
procedure pts2d_on_cone(q0,q1,q2: vt3d; rk: real; peak: vt3d; p2d: vts2d;
                                          n1,n2: integer; var p: vts3d);
{Berechnet die Aufwicklung einer Punktreihe auf einen senkrechten Kreiskegel.}
var f1,f2,av : vt3d;                  i : integer;
     rk2,a2,ml,al,waq,wq,axi,aeta : real;
begin
  diff3d(q1,q0, f1);    diff3d(q2,q0, f2);      diff3d(peak,q0, av);
  rk2:= rk*rk;          a2:=scalarp3d(av,av);   ml:= sqrt(rk2 + a2);
  for i:=n1 to n2 do
      begin
```

```
      waq:= polar_angle(p2d[i].x,p2d[i].y);       wq := ml*waq/rk;
      al := length2d(p2d[i]) / ml;
      axi:= al*cos(wq);        aeta:= al*sin(wq);
      lcomb4vt3d(1,q0, axi,f1, aeta,f2, 1-al,av, p[i]);
      end;
 end; {pts2d_on_cone }
{******************}
```

Aufgabe 16.4 *AUFWICKLUNG AUF EINEN KEGEL:*

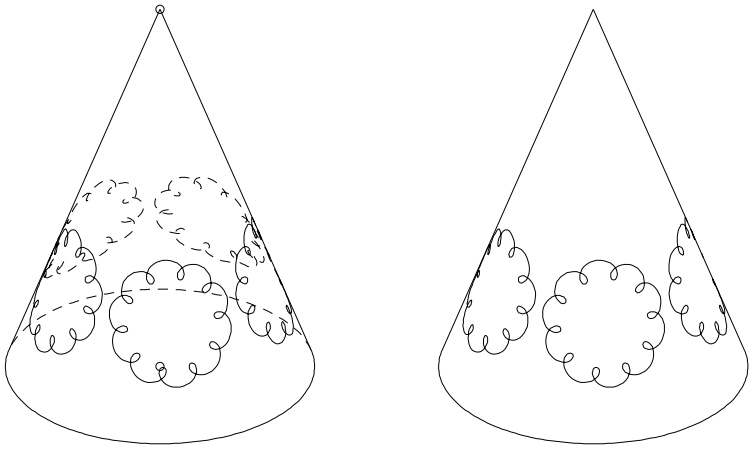


Abbildung 16.6: Aufwicklung von Zykloiden auf einen Kegel

16.5 Abwickelbare Verbindungsflächen von Kurven

Es seien $\Gamma_1 : \mathbf{p}(s)$ und $\Gamma_2 : \mathbf{q}(t)$ zwei nicht in einer Ebene liegende differenzierbare Kurven, für die stets $\mathbf{p}'(s) \neq 0, \mathbf{q}'(t) \neq 0$ ist. Um eine abwickelbare Verbindungsfläche der beiden Kurven zu erhalten, müssen wir jedem Punkt $P_i : \mathbf{p}(s_i)$ von Γ_1 einen Punkt $Q_i : \mathbf{q}(t_i)$ von Γ_2 so zuordnen, daß entlang der Verbindungsgerade P_iQ_i die Tangentialebene der zu erzeugenden Fläche dieselbe ist. Dies ist der Fall, wenn der Tangentenvektor $\mathbf{p}'(s_i)$ in P_i, der Tangentenvektor $\mathbf{q}'(t_i)$ in Q_i und der Richtungsvektor $\mathbf{q}(s_i) - \mathbf{q}(t_i)$ in einer Ebene liegen, d.h. wenn die Determinante

$$\det(\mathbf{p}'(s_i), \mathbf{q}'(t_i), \mathbf{p}(s_i) - \mathbf{q}(t_i)) = 0$$

ist.
Gibt man s_i vor, so kann man entweder
a) mit einem geeigneten Startwert für t_i und dem Newtonverfahren oder
b) mit der Regula falsi
ein t_i bestimmen.
Bei der Auswahl eines Startwertes für t_i ist darauf zu achten, daß er "realistisch" ist. Denn sind z.B. Γ_1 und Γ_2 zwei parallele Kreise mit gleichem Radius, so kann man als abwickelbare Übergangsfläche sowohl einen Zylinder als auch einen Kegel erhalten. Ein Kegel ist ungeeignet, da man normalerweise in der Technik eine "rohrähnliche" Fläche haben möchte.
Zur **Abwicklung** teilt man jedes von zwei Erzeugenden gebildete (räumliche!) Viereck in zwei Dreiecke und wickelt dieses Band von Dreiecken in die Ebene ab.

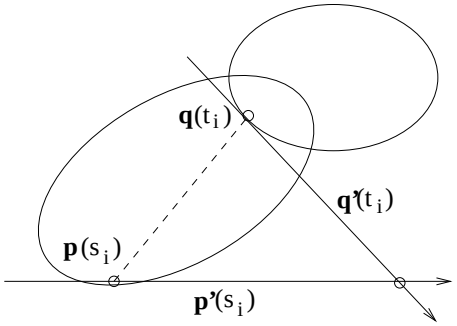


Abbildung 16.7: Bedingung für die Abwickelbarkeit

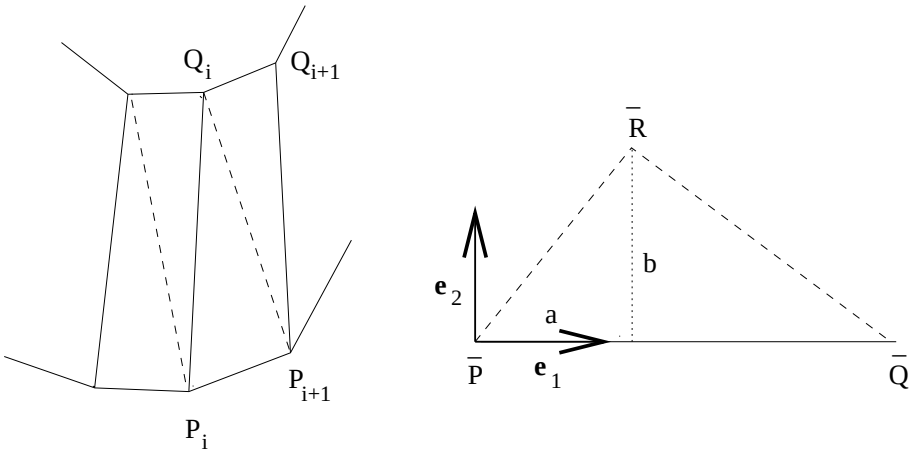


Abbildung 16.8: Abwicklung eines Dreiecks

Zur Abwicklung eines Dreiecks P, Q, R nötige Berechnungen, wobei die Abwicklung $\overline{P}, \overline{Q}$ der Punkte P, Q als bekannt vorausgesetzt wird:

(1) Längen der Dreicksseiten $PQ(= \overline{PQ}), PR, QR$.
(2) Einheitsvektor $\mathbf{e}_1$ in $\overline{PQ}-$Richtung, $\mathbf{e}_2 \perp \mathbf{e}_1$.
(3) $a =$ Projektion von RP auf $\mathbf{e}_1$, $\quad b =$ Projektion von RP auf $\mathbf{e}_2$
(4) $\overline{R} = \overline{P} + a\mathbf{e}_1 + b\mathbf{e}_2$ (Abwicklung von R).

Das Unterprogramm `development` wickelt die zugeordneten Punkte $P_i, Q_i,\ i = n_1, \ldots, n_2$, die eine Erzeugende bestimmen, ab und zeichnet die Abwicklung.

```
procedure development(p,q : vts3d; n1,n2 : integer; var pd,qd  : vts2d);
var  i : integer;    qp : vt3d;
{**}
procedure develop_triangle(p,q,r : vt3d; pd,qd : vt2d; var  rd : vt2d);
var  qpd,e1,e2 : vt2d;     lqp2,lqp,lrp2,lrq2,a,b : real;
begin
  diff2d(qd,pd, qpd);  lqp2:= scalarp2d(qpd,qpd);  lqp:= sqrt(lqp2);
  scale2d(1/lqp,qpd, e1);    rotor2d(0,1,e1, e2);   { e1,e2: Orthonormalbasis }
  lrp2:= distance3d_square(r,p);
```

```
  lrq2:= distance3d_square(r,q);
  a:= (lqp2+lrp2-lrq2)/(lqp+lqp); { Proj. von lrp in e1-Richtg. (Kosinussatz) }
  b:= sqrt(lrp2-a*a);             {  "     "   "   " e2- "                    }
  lcomb3vt2d(1,pd, a,e1, b,e2, rd);
end; { develop_triangle }
{**}
begin
  pd[n1]:= null2d;   diff3d(q[n1],p[n1], qp);
  put2d(0,length3d(qp), qd[n1]);         line2d(pd[n1],qd[n1],0);
  for i:= n1+1 to n2 do
      begin
      develop_triangle(p[i-1],q[i-1], p[i], pd[i-1],qd[i-1],  pd[i]);
      develop_triangle(p[i],q[i-1], q[i], pd[i],qd[i-1], qd[i]);
      line2d(pd[i],qd[i],0);
      end;
 curve2d(pd,n1,n2,0);  curve2d(qd,n1,n2,0);
end;  {development }
{****************}
```

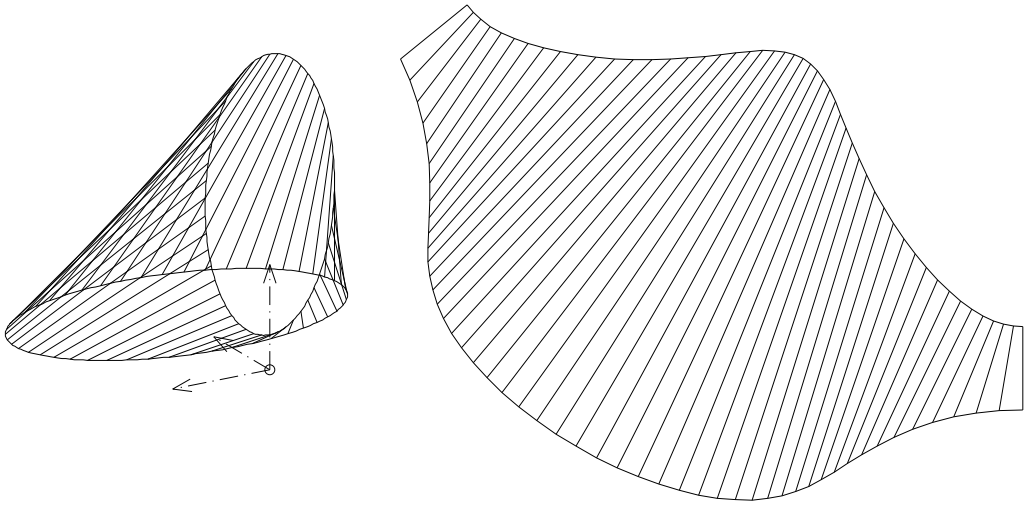


Abbildung 16.9: Abwicklung der Verbindungsfläche zweoer Ellipsen

Mehr über die Abwicklung von Verbindungsflächen findet man in WE,FU'88.

Kapitel 17

DACHAUSMITTELUNG, BÖSCHUNGSFLÄCHEN

17.1 Dachausmittelung

17.1.1 Problemstellung

Für ein Hausdach werden in der Regel die *Traufkanten* (ein Polygon) und die *Neigungen* (Winkel gegenüber der Horizontalen) der an den Traufkanten anliegenden Dachflächen vorgegeben. Durch das Schneiden von Dachflächen entstehen die sog. *Grat-* bzw. *Kehllinien* (s. Abb.). Horizontale Gratlinien nennt man auch *Firstlinien*. Die Aufgabe der Dachausmittelung besteht nun darin, Grat-, Kehl- und Firstlinien zu bestimmen. Diese Aufgabe ist in der Praxis normalerweise eindeutig lösbar. Doch gibt es auch Fälle, in denen mehere Lösungen existieren. In der klassischen Darstellenden Geometrie benutzt man bei der Dachausmittelung Höhenlinien als Hilfslinien (vgl. z.B. FU,KI,NI'87, S. 133-139).

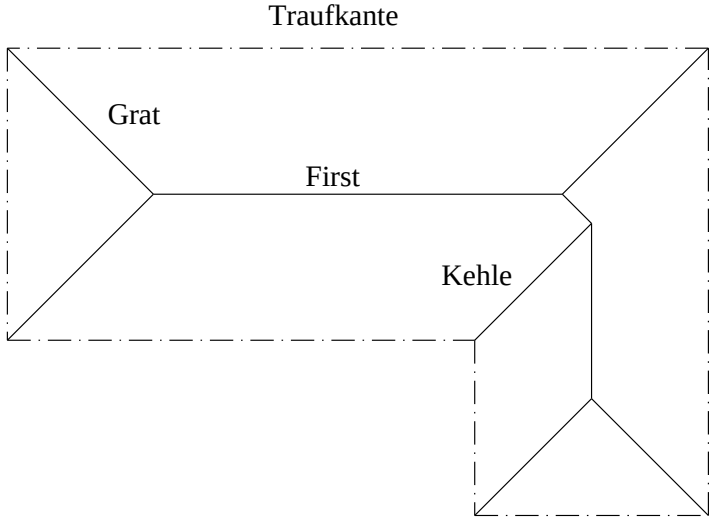


Abbildung 17.1: First-, Grat-, Kehllinien und Traufkanten eines Daches

17.1.2 Der Algorithmus zur Dachausmittelung

Vorgaben: 1) Traufkantenpolygon $P_1, \dots P_n$ mit nicht notwendig horizontalen Traufkanten,
2) Neigungswinkel der an den einzelnen Traufkanten anliegenden Dachflächen.

Durchführung:

(0) Schreibe die Punkte $P_1, \dots P_n$ in eine Liste Λ_p und die Ebenen $\varepsilon_1 \dots \varepsilon_n$ in eine Liste Λ_e. (Die Ebene ε_i enthält die Kante $P_i P_{i+1}$.)
Berechne die Gleichungen $\mathbf{n}_i \cdot \mathbf{x} = d_i, \quad i = 1, \dots, n$, der Ebenen.

(1) Schneide je drei benachbarte Ebenen $\varepsilon_{i-1}, \varepsilon_i, \varepsilon_{i+1}$. Der Schnittpunkt sei S_i. Die **minimale Höhe** der Punkte $S_1, \dots$ sei h_{min}.

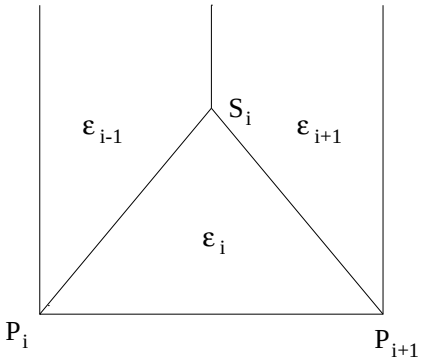


Abbildung 17.2: zu (1)

Λ_p :	P_1	P_2	P_3	P_4	P_5	.....	.....
Λ_e :		ε_1	ε_2	ε_3	ε_4	ε_5	.....
Λ_{mh} :		S_1	$\not{S}_2$	S_3	$\not{S}_4$	$\not{S}_5$	.....

Abbildung 17.3: Die Listen Λ_p, Λ_e und Λ_{mh}

(2) Bestimme unter den Schnittpunkten $S_1, \dots$ diejenigen mit minimaler Höhe h_{min} und schreibe sie in eine Liste Λ_{mh}. Falls zwei aufeinanderfolgende Punkte S_{i-1}, S_i auf der Höhe h_{min} liegen, wird derjenige, der weiter von P_i entfernt liegt, wieder aus der Liste Λ_{mh} gestrichen. (P_i, S_{i-1}, S_i liegen auf einer Gerade !)

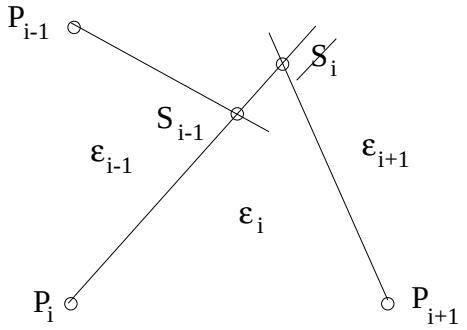


Abbildung 17.4: zu (2)

(3) Für jeden Schnittpunkt $S_i \in \Lambda_{mh}$ verfahre folgendermaßen:

(i) zeichne die Strecken $\overline{S_iP_i}, \overline{S_iP_{i+1}}$.

(ii) streiche die Ebene ε_i (mittlere der am Schnitt beteiligten Ebenen) aus der Ebenen-Liste Λ_e.

(iii) streiche die zu ε_i gehörigen Punkte P_i, P_{i+1} aus der Punkte-Liste Λ_p und füge in diese Lücke den Punkt S_i ein.

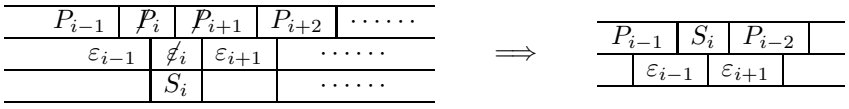


Abbildung 17.5: zu (3)

(4) Sind (nach dem Streichen in (3)) zwei benachbarte Ebenen gleich, so streiche die rechte Ebene aus der Liste Λ_e sowie den (zu den benachbarten Ebenen gehörigen) mittleren Punkt aus der Liste Λ_p.

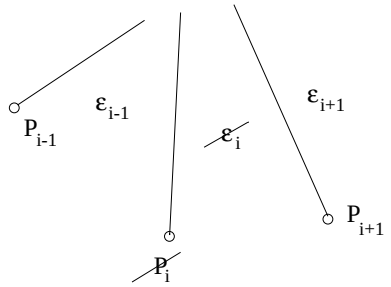


Abbildung 17.6: zu (4)

(5) Wiederhole (1)-(4) bis Λ_e nur noch zwei Ebenen enthält. Falls Λ_p noch zwei Punkte enthält, dann verbinde sie durch eine Kante.

Hinweis:
Dieser Algorithmus wurde im Rahmen des Seminars "Computerunterstützte Darstellende Geometrie" von *Jan Hadenfeld* und *Thomas Jäger* entwickelt.

Die folgenden **Beispiele** wurden mit diesem Algorithmus behandelt. Falls das Traufkantenpolygon nicht einfach zusammenhängend ist, füge man an einem Schnitt zwei Dachflächen mit einer 90 - Neigung ein (s. Beispiel).

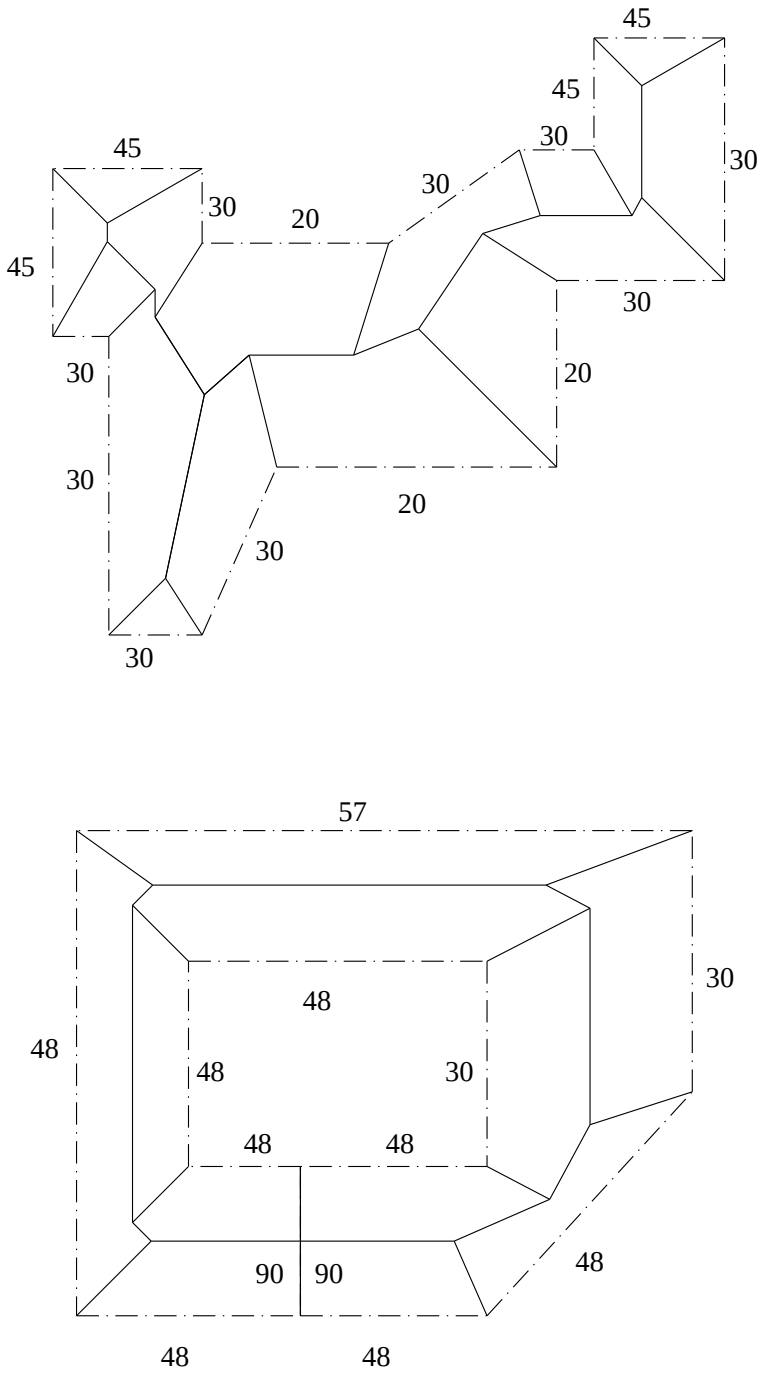


Abbildung 17.7: Beispiele mit Angabe der Dachneigungen in Grad

17.2 Böschungsflächen

17.2.1 Problemstellung

Beim Bau einer Straße, die entweder höher oder tiefer als das umliegende Gelände verläuft, werden entlang den Straßenrändern Erdaufschüttungen bzw. -abgrabungen vorgenommen.Im Fall von

Aufschüttungen kann man sich die Flächen, die dabei entstehen, als die Einhüllenden der Schüttkegeln vorstellen. Die Höhenlinien dieser Flächen werden in Bauplänen als die Einhüllenden der entsprechenden Höhenkreise der Schüttkegeln konstruiert (vgl. GR,BA '78). Um solche Flächen mit Hilfe des Computers darstellen zu können, muß man sich eine Parameterdarstellung verschaffen. Dies kann z. B. geschehen, indem man die Böschungsfläche als Lösung einer partiellen Differentialgleichung betrachtet (GÖ'92) oder die klassische Methode mit Zirkel und Lineal benutzt um eine Parameterdarstellung der Höhenlinien zu finden. Die letztere Methode sei hier kurz erläutert.

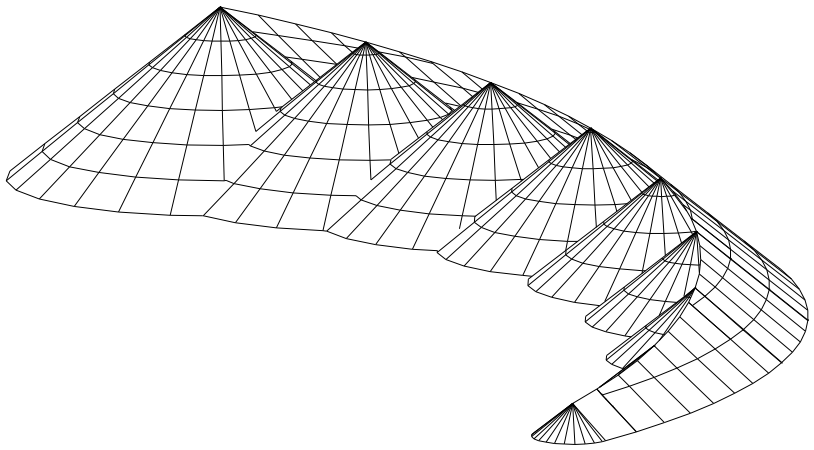


Abbildung 17.8: Kurve, Schüttkegel und ein Teil der Böschungsfläche

17.2.2 Die Parameterdarstellung einer Böschungsfläche

Gegeben: Differenzierbare Kurve (Straßenrand) $\Gamma : \mathbf{c}(s) = (x(s), y(s), z(s)), \quad s \in [a, b]$.
$m =$ Tangens des halben Öffnungswinkels der Schüttkegel.
Gesucht: Parameterdarstellung der zugehörigen Böschungsfläche.

Zur Bestimmung einer Parameterdarstellung der Böschungsfläche Φ verwenden wir die Idee, die man beim Zeichnen mit Zirkel und Lineal benutzt: Den Grundriß einer Höhenlinie der Höhe ζ erhält man als Einhüllende der Höhenkreise der Schüttkegel entlang der Kurve Γ.

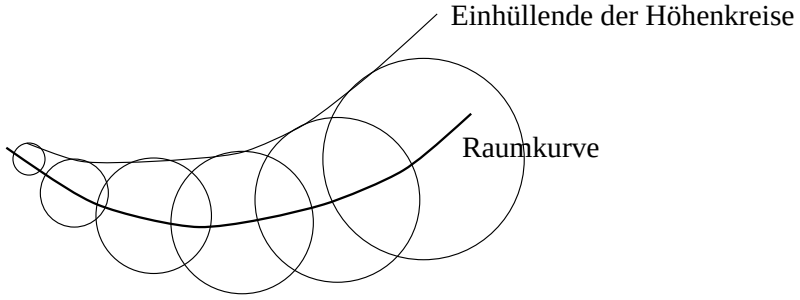


Abbildung 17.9: Höhenlinien als Einhüllende von Höhenkreisen der Schüttkegel

Der Höhenkreis $k(s, \zeta)$ der Höhe ζ des Schüttkegels an der Stelle $\mathbf{c}(s)$ hat den Radius $m \cdot (z(s) - \zeta)$.

Der Grundriß von $k(s,\zeta)$ erfüllt also die Gleichung

$$F(\xi,\eta,s) := (\xi - x(s))^2 + (\eta - y(s))^2 - m^2(z(s)-\zeta)^2 = 0.$$

Eine Einhüllende $\sum$ dieser Kreisschar genügt den beiden Gleichungen

$$F(\xi,\eta,s) = 0 \quad \text{und} \quad F_s(\xi,\eta,s) = 0$$

(vgl. [BR,SE'83], S. 647).
Löst man das System

$$(*) \qquad F(\xi,\eta,s) = (\xi - x(s))^2 + (\eta - y(s))^2 - m^2(z(s)-\zeta)^2 = 0$$

$$(**) \qquad F_s(\xi,\eta,s) = 2(\xi - x(s))\dot{x}(s) + 2(\eta - y(s))\dot{y}(s) - 2m^2(z(s)-\zeta)\dot{z}(s) = 0$$

nach ξ und η auf, so erhält man die Parameterdarstellung $\mathbf{e}(s) = (\xi(s), \eta(s), \zeta)$ von $\sum$. Faßt man die Höhe ζ als weiteren Parameter auf, so ergeben sich die Parameterdarstellungen

$$\mathbf{b}(s,\zeta) = (x, y, \zeta) + m(z-\zeta)\mathbf{r}_0(s)$$

mit

$$\mathbf{r}_0(s) = \left(\frac{m\dot{x}\dot{z} \pm \dot{y}\sqrt{\dot{x}^2+\dot{y}^2-(m\dot{z})^2}}{\dot{x}^2+\dot{y}^2}, \frac{m\dot{y}\dot{z} \mp \dot{x}\sqrt{\dot{x}^2+\dot{y}^2-(m\dot{z})^2}}{\dot{x}^2+\dot{y}^2}, \ 0 \right)$$

der Böschungsflächen. Dabei sind $x, \dot{x}, \ldots$ die Funktionen $x(s), \dot{x}(s) \ldots$. Die Vorzeichen vor den Wurzeln zeigen, daß es **zwei** Böschungsflächen zur Kurve Γ gibt.
Die **Tangente** an eine Höhenlinie ist $(\dot{\xi}, \dot{\eta}, 0)$. Sie ist senkrecht zu $(\xi - x, \eta - y, 0)$, wie man durch implizites Differenzieren der Gleichung $F(\xi,\eta,s) = 0$ nach s und unter Verwendung von $F_s = 0$ (s.o.) erkennt. Damit ist die Tangente auch senkrecht zu $\mathbf{r}_0(s)$ und ihre Richtung unabhängig von der Höhe ζ.
Ersetzt man den Parameter ζ durch den Parameter $t := m(z-\zeta)$,so erkennt man, daß die Böschungsflächen **Regelflächen** sind. Die Parameterdarstellungen sind dann

$$\bar{\mathbf{b}}(s,t) = \mathbf{c}(s) + t\mathbf{r}(s),$$

wobei

$$\mathbf{r}(s) = \left(\frac{m\dot{x}\dot{z} \pm \dot{y}\sqrt{\dot{x}^2+\dot{y}^2-(m\dot{z})^2}}{\dot{x}^2+\dot{y}^2}, \ \frac{m\dot{y}\dot{z} \mp \dot{x}\sqrt{\dot{x}^2+\dot{y}^2-(m\dot{z})^2}}{\dot{x}^2+\dot{y}^2}, \ \frac{-1}{m} \right)$$

Dabei sind (wie oben) $x, \dot{x}, \ldots$ die Funktionen $x(s), \dot{x}(s)\cdot$.

Eine Regelfläche, die entlang einer Erzeugenden immer dieselbe Tangentialebene hat, was hier der Fall ist, nennt man **torsal**. (Eine torsale Fläche ist abwickelbar). Man prüft ferner leicht nach, daß die **Tangentialebenen** der hier konstruierten Flächen alle die **gleiche Neigung** (Winkel mit der $z-$Achse) haben.

Beispiel 17.1 *Für die Gerade*

$\Gamma : \mathbf{c}(s) := (s, 0, \gamma s)$ *ist* $x = s, y = 0, z = \gamma s, \dot{x} = 1, \dot{y} = 0, \dot{z} = \gamma$ *und* $\mathbf{r}(s) = (m\gamma, \mp\sqrt{1-(m\gamma)^2}, -1/m)$

konstant.
D.h. die Böschungsflächen sind Ebenen.

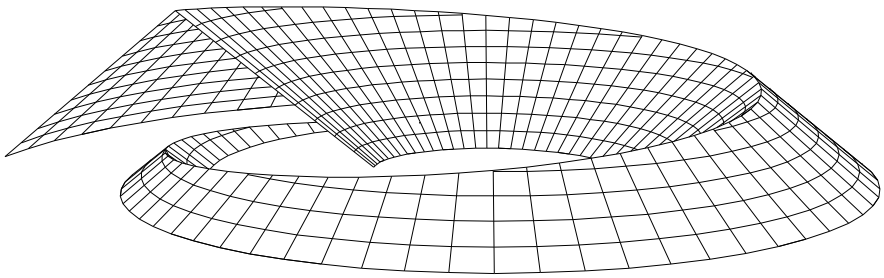


Abbildung 17.10: Böschungsflächen einer Schraublinie

Beispiel 17.2 *Falls* Γ *die Schraublinie* $\mathbf{c}(s) := (a\cos s, a\sin s, \gamma s)$ *ist, ist* $\dot{\mathbf{c}}(s) = (-a\sin s, a\cos s, \gamma)$ *und*

$$\begin{aligned}\mathbf{r}(s) &= (-ma\gamma\sin s \pm a\cos s\sqrt{a^2-(m\gamma)^2}, ma\gamma\cos s \pm a\sin s\sqrt{a^2-(m\gamma)^2}, \tfrac{-a^2}{m}) \\ &= ma\gamma(-\sin s, \cos s, 0) \pm a\sqrt{a^2-(m\gamma)^2}(\cos s, \sin s, 0) - (0,0,\tfrac{a^2}{m}).\end{aligned}$$

Hinweis:
Die Abbildungen in diesem Abschnitt über Böschungsflächen stammen aus der Diplomarbeit von Andreas Görg.

Kapitel 18

EIGENSCHAFTEN VON BEZIERKURVEN

18.1 Eigenschaften der Bernsteinpolynome

Die Definition der Bernsteinpolynome

$$B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}, \quad 0 \le i \le n,$$

ergänzen wir aus technischen Gründen durch $B_i^n(t) = 0$ für andere i. Auch für die Binomialkoeffizienten setzen wir $\binom{n}{i} = 0$, falls $i \notin \{0, 1, ..., n\}$.

Es gilt

(1) Beziehung zwischen der Bernstein– und der Monom–Basis:

$$t^i = \sum_{j=i}^{n} \frac{\binom{j}{i}}{\binom{n}{i}} B_j^n(t), \qquad B_i^n(t) = \sum_{j=i}^{n} (-1)^{j-i} \binom{n}{j} \binom{j}{i} t^j,$$

(2) Rekursion: $B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$,

(3) Skalierung: $B_i^n(ct) = \sum_{j=0}^{n} B_i^j(c) B_j^n(t)$,

(4) Ableitung: $\frac{d}{dt} B_i^n(t) = n \left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t) \right)$,
(Man beachte, daß $B_{-1}^{\cdots}(t) = 0, B_n^{n-1}(t) = 0$ ist.)

(5) Produkt: $B_i^n(t) B_j^n(t) = \frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}} B_{i+j}^{m+n}(t)$.

18.2 Der Casteljau-Algorithmus

Für das Polygon $\mathbf{b}_0, \mathbf{b}_1, ..., \mathbf{b}_n$ im $\mathbb{R}^2$ (oder $\mathbb{R}^3$) und einem $t \in \mathbb{R}$ definieren wir rekursiv für jedes $r = 1, ..., n$ das Polygon

$$\mathbf{b}_i^r(t) = (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t), \quad i = 0, ...n-r.$$

wobei $\mathbf{b}_i^0(t) = \mathbf{b}_i$ sei.
Das Polygon der Stufe $r = 0$ ist identisch mit dem Ausgangspolygon, das Polygon der Stufe $r = n$ ist ein Punkt.

Aus der Rekursionseigenschaft der Bernsteinpolynome (s. 18.1) folgt

$$\mathbf{b}_i^r(t) = \sum_{j=0}^{r} \mathbf{b}_{i+j} B_j^r(t), \quad r = 0, ..., n, \ i = 0, ..., n-r$$

(Beweis mit Hilfe vollständiger Induktion über r.)
Also ist

$$\mathbf{b}_0^n(t) = \sum_{j=0}^{n} \mathbf{b}_j B_j^n(t)$$

die Bézierkurve mit dem Kontrollpolygon $\mathbf{b}_0, \mathbf{b}_1, ..., \mathbf{b}_n$. Diese Methode, einen Punkt der Bézierkurve über lineare Interpolationen zu bestimmen, heißt der **Casteljau–Algorithmus**.

Wie für ein $t \in \mathbb{R}$ aus dem Kontrollpolygon die Zwischenpolygone und schließlich der Punkt der Bézierkurve entsteht, zeigt die Abb. 18.1 für $n = 3$.

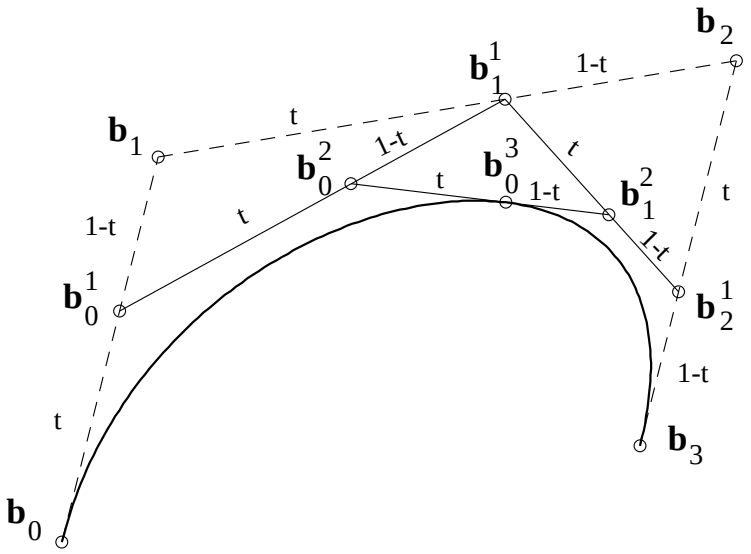


Abbildung 18.1: Casteljau–Algorithmus für n=3

Oder schematisch

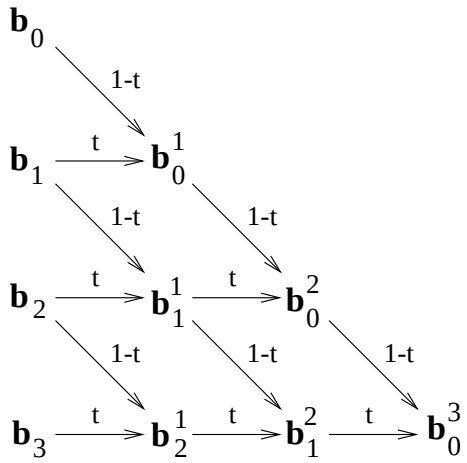


Abbildung 18.2: Schematische Darstellung des Casteljau–Algorithmus

18.3 Ableitungen einer Bézierkurve

Mit Hilfe der Ableitungen der Bernsteinpolynome (s.18.1) ergibt sich für die 1. Ableitung der Bézierkurve $\mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t)$:

$$\frac{d}{dt}\mathbf{b}(t) = n \sum_{i=0}^{n-1} (\mathbf{b}_{i+1} - \mathbf{b}_i) B_i^{n-1}(t) \ .$$

Läßt man die Tangentenvektoren alle im Nullpunkt des Koordinatensystems beginnen, so beschreiben sie eine weitere Bézierkurve mit den Kontrollpunkten $\Delta\mathbf{b}_i := \mathbf{b}_{i+1} - \mathbf{b}_i$.
Speziell gilt:

$$\frac{d}{dt}\mathbf{b}(0) = n(\mathbf{b}_1 - \mathbf{b}_0) \quad \text{und} \quad \frac{d}{dt}\mathbf{b}(1) = n(\mathbf{b}_n - \mathbf{b}_{n-1}).$$

Um höhere Ableitungen übersichtlich schreiben zu können, führen wir folgenden Differenzenoperator ein:

$$\Delta^r \mathbf{b}_i := \Delta^{r-1}\mathbf{b}_{i+1} - \Delta^{r-1}\mathbf{b}_i,$$

Es ist

$$\begin{aligned}
\Delta^0 \mathbf{b}_i &= \mathbf{b}_i \\
\Delta^1 \mathbf{b}_i &= \mathbf{b}_{i+1} - \mathbf{b}_i \\
\Delta^2 \mathbf{b}_i &= \mathbf{b}_{i+2} - 2\mathbf{b}_{i+1} + \mathbf{b}_i \\
\Delta^3 \mathbf{b}_i &= \mathbf{b}_{i+3} - 3\mathbf{b}_{i+2} + 3\mathbf{b}_{i+1} - \mathbf{b}_i \\
\cdots &= \cdots \\
\Delta^r \mathbf{b}_i &= \sum_{j=0}^{r} \binom{r}{j} (-1)^{r-j} \mathbf{b}_{i+j}
\end{aligned}$$

Die r-te Ableitung der Bézierkurve $\mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t)$ läßt sichjetzt wie folgt schreiben:

$$\frac{d^r}{dt^r}\mathbf{b}(t) = \frac{n!}{(n-r)!} \sum_{i=0}^{n-r} \Delta^r \mathbf{b}_i B_i^{n-r}(t)$$

Speziell für $t = 0$ und $t = 1$ erhält man

$$\frac{d^r}{dt^r}\mathbf{b}(0) = \frac{n!}{(n-r)!}\Delta^r\mathbf{b}_0 \quad \text{und} \quad \frac{d^r}{dt^r}\mathbf{b}(1) = \frac{n!}{(n-r)!}\Delta^r\mathbf{b}_{n-r}$$

18.4 Graderhöhung einer Bézierkurve

Eine wichtige Manipulation der Darstellung einer vorgegebenen Bézierkurve ist die sog. Graderhöhung. Sie ist vergleichbar mit dem Anfügen von Termen $0t^{n+1}, 0t^{n+2}, ...$ an ein Polynom $a_0 + a_1 t + \cdots a_n t^n$. Dabei ändert sich das Polynom nicht und der (scheinbare) Grad wird erhöht. Analog stellt man eine fest vorgegebene Bézierkurve $\mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t)$ in der Form $\mathbf{b}(t) = \sum_{i=0}^{n+1} \mathbf{b}_i^{(1)} B_i^{n+1}(t)$ mit geeigneten neuen Kontrollpunkten $\mathbf{b}_0^{(1)}, ..., \mathbf{b}_{n+1}^{(1)}$ dar. Um die neuen Kontrollpunkte zu bestimmen, multiplizieren wir die ursprüngliche Darstellung mit dem Faktor $t + (1 - t)$:

$$\begin{aligned}
\mathbf{b}(t) &= \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t)(t + (1-t)) \\
&= \sum_{i=0}^{n} \mathbf{b}_i \binom{n}{i} t^i (1-t)^{n-i}(t + (1-t)) \\
&= \sum_{i=0}^{n} \mathbf{b}_i \binom{n}{i} t^{i+1}(1-t)^{n-i} + \sum_{i=0}^{n} \mathbf{b}_i \binom{n}{i} t^i (1-t)^{n-i+1} \\
&= \sum_{i=1}^{n+1} \mathbf{b}_{i-1} \binom{n}{i-1} t^i (1-t)^{n+1-i} + \sum_{i=0}^{n} \mathbf{b}_i \binom{n}{i} t^i (1-t)^{n-i+1} \\
&= \sum_{i=0}^{n+1} \left(\mathbf{b}_{i-1} \binom{n}{i-1} + \mathbf{b}_i \binom{n}{i} \right) t^i (1-t)^{n+1-i} \\
&= \sum_{i=0}^{n+1} \mathbf{b}_i^{(1)} \binom{n+1}{i} t^i (1-t)^{n+1-i} = \sum_{i=0}^{n+1} \mathbf{b}_i^{(1)} B_i^{n+1}(t) \quad \text{, wobei} \\
\mathbf{b}_i^{(1)} : &= \frac{i}{n+1}\mathbf{b}_{i-1} + (1 - \frac{i}{n+1})\mathbf{b}_i, \ i = 0, ..., n+1 \\
&= \mathbf{b}_i^1(\frac{i}{n+1}) \quad \text{(s. Casteljau–Algor.)}
\end{aligned}$$

Die neuen Kontrollpunkte sind also: $\mathbf{b}_0, \mathbf{b}_1^1(\frac{n}{n+1}), \mathbf{b}_2^1(\frac{n-1}{n+1}), ..., \mathbf{b}_n^1(\frac{1}{n+1}), \mathbf{b}_n$.

Wesentliche Eigenschaften der Graderhöhung sind:

- Wiederholte Graderhöhung führt zu einer Approximation der Bézierkurve durch das Kontrollpolygon.
- Die größere Anzahl von Kontrollpunkten bietet mehr Freiheitsgrade, die Kurve zu verändern.
- Mehrere Bézierkurven lassen sich auf einen einheitlichen Grad bringen. Dies ist wichtig bei Tensorprodukt–Bézierflächen (s. ????).

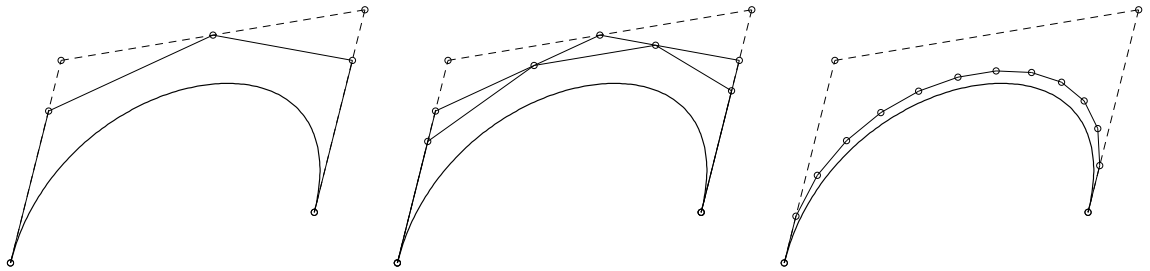


Abbildung 18.3: Kontrollpolygone bei ein-, zwei- und 10-maliger Graderhöhung

18.5 Bézier-Splinekurven

18.5.1 Zerlegung einer Bézierkurve

Eine Bézierkurve $\Gamma_0 : \mathbf{b}(t)$ ist normalerweise definiert für $0 \leq t \leq 1$. Sei nun $c \in (0,1)$. Dann ist $\Gamma_1 : \mathbf{b}(t)$ mit $0 \leq t \leq c$ ein Teil der gegebenen Bézierkurve. Wir wollen nun die Teilkurve Γ_1 als Bézierkurve $\mathbf{c}(t) = \sum_{i=0}^{n} \mathbf{c}_i B_i^n(s)$ mit $s \in [0,1]$ vom (selben) Grad n mit geeigneten Kontrollpunkten $\mathbf{c}_0, \mathbf{c}_1, ..., \mathbf{c}_n$ darstellen. Setzen wir $s := \frac{t}{c}$, so muß gelten:

$$\sum_{r=0}^{n} \mathbf{c}_r B_r^n(\frac{t}{c}) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t), \quad \text{für} \quad t \in [0,c]$$

Es gilt

$$\mathbf{c}_r = \mathbf{b}_0^r(c) \quad \text{(s. Casteljau–Alg.).}$$

(vgl. Abb. 18.1 und Abb. 18.2)
Denn

$$\begin{aligned}
\sum_{r=0}^{n} \mathbf{c}_r B_r^n(\frac{t}{c}) &= \sum_{r=0}^{n}\sum_{i=0}^{r} \mathbf{b}_i B_i^r(c) B_r^n(\frac{t}{c}) \\
&= \sum_{r=0}^{n}\sum_{i=0}^{n} \mathbf{b}_i B_i^r(c) B_r^n(\frac{t}{c}) \quad \text{wegen} \quad B_i^r(\cdots) = 0 \quad \text{für} \quad i > r \\
&= \sum_{i=0}^{n} \mathbf{b}_i \sum_{r=0}^{n} B_i^r(c) B_r^n(\frac{t}{c}) \\
&= \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) \quad \text{wegen Eigenschaft (3) der Bernsteinpolynome (s. Abschnitt 18.1)}
\end{aligned}$$

18.5.2 Glattheitsbedingungen

18.5.2.1 C^1– und C^2–Übergänge

Um zwei Bézierkurven zu einer einzigen wenigstens C^1-stetigen Kurve zusammensetzen zu können, muß erst erklärt werden, was eine Bézierkurve über einem beliebigen Parameterintervall sein soll. Die Bézierkurve

$$\Gamma_1 : \mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t), \ t \in [0,1]$$

läßt sich durch die Parametertransformation $t = \frac{s-u_0}{u_1-u_0}$ auf dem Intervall $[u_0, u_1]$ darstellen:

$$\mathbf{c}(s) := \mathbf{b}(\frac{s-u_0}{u_1-u_0}),\ s \in [u_0, u_1].$$

Für die Ableitung an der Stelle u_1 gilt:

$$\dot{\mathbf{c}}(u_1) = \frac{\dot{\mathbf{b}}(1)}{u_1-u_0} = n\frac{\mathbf{b}_n - \mathbf{b}_{n-1}}{u_1-u_0}.$$

Die weitere Bézierkurve

$$\Gamma_2 : \mathbf{b}'(t) = \sum_{i=0}^{n} \mathbf{b}_{n+i} B_i^n(t),\ t \in [0,1]$$

vom selben Grad wie Γ_1 soll nun C^1-stetig an Γ_1 angeschlossen werden. Hierzu führen wir die Parametertransformation $t = \frac{s-u_1}{u_2-u_1}$ durch:

$$\mathbf{c}(s) := \mathbf{b}'(\frac{s-u_1}{u_2-u_1}),\ s \in [u_1, u_2].$$

Da $\mathbf{b}_n$ der letzte Kontrollpunkt von Γ_1 und der erste von Γ_2 ist, beschreibt $\Gamma : \mathbf{c}(s), s \in [u_0, u_2]$ eine mindestens stetige Kurve. Damit Γ an der Stelle u_1 sogar **$\mathbf{C^1}$-stetig** ist, muß offensichtlich gelten:

$$\text{(C1)} \qquad \frac{\mathbf{b}_n - \mathbf{b}_{n-1}}{u_1-u_0} = \frac{\mathbf{b}_{n+1} - \mathbf{b}_n}{u_2-u_1}.$$

D.h. der Punkt $\mathbf{b}_n$ teilt die Strecke zwischen $\mathbf{b}_{n-1}$ und $\mathbf{b}_{n+1}$ im Verhältnis $(u_1-u_0) : (u_2-u_1)$ (s. Abb. 18.4).

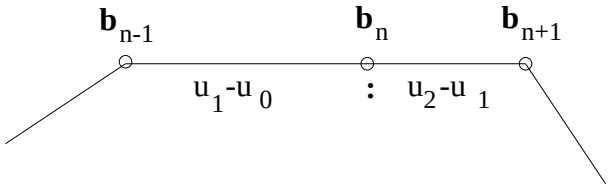


Abbildung 18.4: Zur C^1–Stetigkeit

Um eine zusätzliche Bedingung zu erhalten, die den C^2-Übergang an der Stelle $s = u_1$ garantiert, differenziert man $\mathbf{c}(s)$ zweimal:

$$\ddot{\mathbf{c}}(u_1) = \frac{\ddot{\mathbf{b}}(1)}{(u_1-u_0)^2} = \frac{\ddot{\mathbf{b}}(0)}{(u_2-u_1)^2}$$

Die 2. Ableitung einer Bézierkurve ist eine Bézierkurve, deren Kontrollpunkte aus Differenzen von Differenzen der gegebenen Kontrollpunkte bestehen (vgl. Abschnitt 18.3). Hieraus ergibt sich die zusätzliche Übergangsbedingung für **$\mathbf{C^2}$–Stetigkeit**:

$$\text{(C2)} \qquad \frac{\mathbf{b}_n - 2\mathbf{b}_{n-1} + \mathbf{b}_{n-2}}{(u_1-u_0)^2} = \frac{\mathbf{b}_{n+2} - 2\mathbf{b}_{n+1} + \mathbf{b}_n}{(u_2-u_1)^2}.$$

Um die geometrische Bedeutung dieser Beziehung zu erkennen, führen wir den Punkt $\mathbf{d}$ auf der Gerade durch $\mathbf{b}_{n-2}$ und $\mathbf{b}_{n-1}$ so ein, daß $\mathbf{b}_{n-1}$ die Strecke zwischen $\mathbf{b}_{n-2}$ und $\mathbf{d}$ im Verhältnis

$(u_1 - u_0) : (u_2 - u_1)$ teilt. Mit den Abkürzungen $\Delta_0 := u_1 - u_0,\ \Delta_1 := u_2 - u_1$ gilt also:

$$\begin{aligned} \Delta_1(\mathbf{b}_{n-1} - \mathbf{b}_{n-2}) &= \Delta_0(\mathbf{d} - \mathbf{b}_{n-1}) \\ \Leftrightarrow\ -\Delta_1(\mathbf{b}_{n-2} - 2\mathbf{b}_{n-1} + \mathbf{b}_n) + \Delta_1(\mathbf{b}_n - \mathbf{b}_{n-1}) &= \Delta_0(\mathbf{d} - \mathbf{b}_{n-1}) \\ \Leftrightarrow\ -\Delta_1(\mathbf{b}_{n-2} - 2\mathbf{b}_{n-1} + \mathbf{b}_n) &= \Delta_0(\mathbf{d} - \mathbf{b}_{n-1} - \mathbf{b}_{n+1} + \mathbf{b}_n) \quad \text{wegen (C1)} \end{aligned}$$

Sei nun $\mathbf{d}'$ der Punkt auf der Gerade durch $\mathbf{b}_{n+1}$ und $\mathbf{b}_{n+2}$ so gewählt, daß $\mathbf{b}_{n+1}$ die Strecke zwischen $\mathbf{d}'$ und $\mathbf{b}_{n+2}$ im Verhältnis $(u_1 - u_0) : (u_2 - u_1)$ teilt. Überlegungen wie oben für $\mathbf{d}$ liefern die Gleichung:

$$-\Delta_0(\mathbf{b}_{n+2} - 2\mathbf{b}_{n+1} + \mathbf{b}_n) = \Delta_0(\mathbf{d}' - \mathbf{b}_{n-1} - \mathbf{b}_{n+1} + \mathbf{b}_n)$$

Also gilt $C2$) genau dann, wenn $\mathbf{d} = \mathbf{d}'$ ist (s. Abb. 18.5).

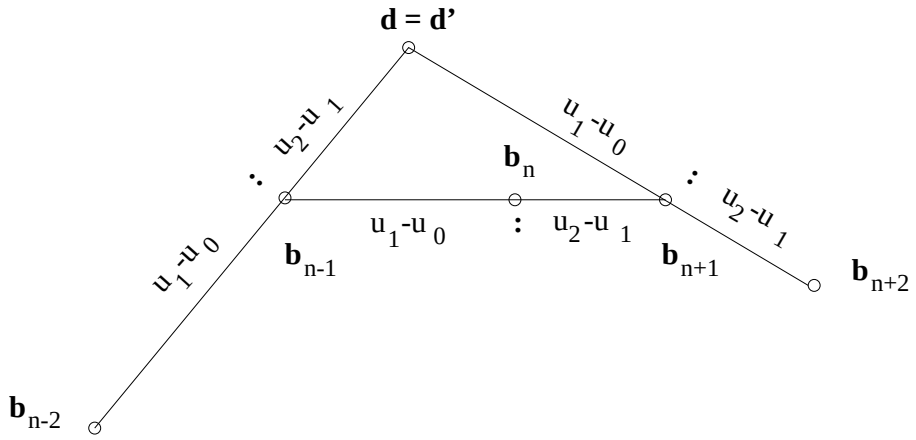


Abbildung 18.5: Zur C^2–Stetigkeit

18.5.2.2 G^1– und G^2–Übergänge

Sollen die beiden Bézierkurven

$$\Gamma_1 : \mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t),\ t \in [0,1]$$

$$\Gamma_2 : \mathbf{c}(t) = \sum_{i=0}^{m} \mathbf{c}_i B_i^m(t),\ t \in [0,1]$$

im Punkt $\mathbf{b}_n$ tangentenstetig, d.h. **G^1–stetig**, aneinander anschließen, so muß außer $\mathbf{b}_n = \mathbf{c}_0$ offensichtlich nur die Bedingung

$$\text{(G1)} \quad \mathbf{b}_n - \mathbf{b}_{n-1} \quad \text{parallel zu} \quad \mathbf{c}_1 - \mathbf{c}_0$$

erfüllt sein.

Soll zusätzlich die Krümmung im Punkt $\mathbf{b}_n$ stetig sein, d.h. **G^2–Stetigkeit**, so muß

$$\frac{\|\dot{\mathbf{b}}(1) \times \ddot{\mathbf{b}}(1)\|}{\|\dot{\mathbf{b}}(1)\|^3} = \frac{\|\dot{\mathbf{c}}(0) \times \ddot{\mathbf{c}}(0)\|}{\|\dot{\mathbf{c}}(0)\|^3}$$

gelten. Mit

$$\dot{\mathbf{b}}(1) = n(\mathbf{b}_n - \mathbf{b}_{n-1}) \quad \text{und} \quad \dot{\mathbf{c}}(0) = m(\mathbf{c}_1 - \mathbf{c}_0),$$

$$\ddot{\mathbf{b}}(1) = n(n-1)((\mathbf{b}_n - \mathbf{b}_{n-1}) - (\mathbf{b}_{n-1} - \mathbf{b}_{n-2})) \quad \text{und} \quad \ddot{\mathbf{c}}(0) = m(m-1)((\mathbf{c}_2 - \mathbf{c}_1) - (\mathbf{c}_1 - \mathbf{c}_0))$$

erhält man die Bedingung

$$\text{(G2)} \quad \frac{(n-1)\|(\mathbf{b}_n - \mathbf{b}_{n-1}) \times (\mathbf{b}_{n-1} - \mathbf{b}_{n-2})\|}{n\|(\mathbf{b}_n - \mathbf{b}_{n-1})\|^3} = \frac{(m-1)\|(\mathbf{c}_1 - \mathbf{c}_0) \times (\mathbf{c}_2 - \mathbf{c}_1)\|}{m\|(\mathbf{c}_1 - \mathbf{c}_0)\|^3}.$$

(Im ebenen Fall ersetzt man das Vektorprodukt durch die Determinante.) Eine wesentliche Konsequenz aus der letzten Darstellung der Krümmung ist: Man darf den Kontrollpunkt $\mathbf{b}_{n-2}$ bzw. $\mathbf{c}_2$ parallel zur Tangente im Punkt $\mathbf{b}_n$ bzw. $\mathbf{c}_0$ verschieben ohne, daß sich die Krümmung von Γ_1 bzw. Γ_2 in diesem Punkt verändert. Hat man also eine krümmungsstetige Lösung gefunden, darf man noch die Kontrollpunkte $\mathbf{b}_{n-2}$ und $\mathbf{c}_2$ unabhängig von einander parallel zur Tangente im gemeinsamen Punkt $\mathbf{b}_n$ verschieben.

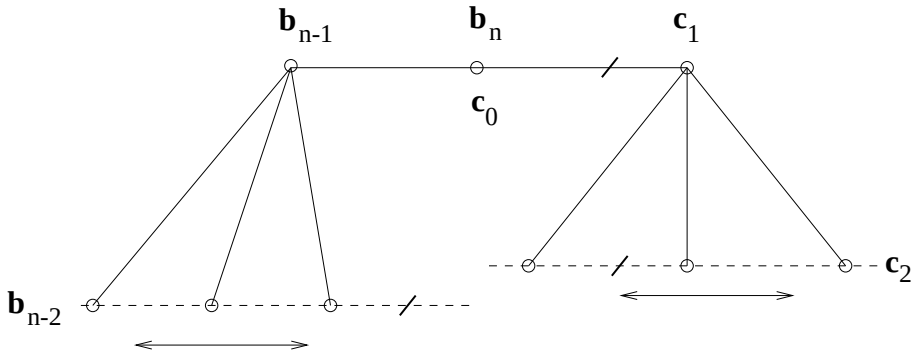


Abbildung 18.6: Zur G^2–Stetigkeit

Die Bedingung $G2$ kann durch als eine Beziehung zwischen den Flächeninhalten der Dreiecke $\mathbf{b}_{n-2}, \mathbf{b}_{n-1}, \mathbf{b}_n$ und $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$ interpretiert werden.

Kapitel 19

RATIONALE BEZIERKURVEN

19.1 Rationale Kurven und projektive Kurven

Bézierkurven sind parametrisierte Kurven, deren Parameterdarstellungen nur Polynome verwenden. Leider lassen sich so wichtige und geometrisch einfache Kurven wie Kreise nicht durch polynomiale Parameterdarstellungen beschreiben. Dieser Nachteil ist u.a. das Motiv für die Erweiterung der als Parameterfunktionen zulässigen Funktionen auf rationale Funktionen. Denn jeder Kegelschnitt hat eine rationale Darstellung (s.u.). Da eine Kurve Γ mit einer rationalen Darstellung

$$(\frac{p_1(t)}{q_1(t)}, \frac{p_2(t)}{q_2(t)})$$

wobei die Funktionen p_i und q_i Polynome sind, in homogenen Koordinaten die polynomiale Darstellung

$$< (p_1(t)q_2(t), p_2(t)q_1(t), q_1(t)q_2(t)) >,$$

besitzt, lassen sich ebene Kurven mit rationalen Koeffizientenfunktionen als Zentralprojektion einer Bézierkurve im $\mathbb{R}^3$ auf die Einbettungsebene $x_3 = 1$ (s. Anhang2) auffassen.
Die analoge Aussage gilt für Kurven im $\mathbb{R}^3$. Sie lassen sich als Zentralprojektion einer Bézierkurve im $\mathbb{R}^4$ auf den Einbettungsraum $x_4 = 1$ auffassen. Damit lassen sich die Vorteile des Bézier–Darstellung einer polynomialen Kurve auch für rationale Kurven nutzen.

19.2 Rationale Bézierkurven

Es sei nun $n > 0$ festgewählt und die Vektoren $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n,\ \mathbf{b}_i \neq \mathbf{0}$, beschreiben ein Polygon im $\mathbb{R}^3$. Dann ist $\mathbf{x}(t) := \mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \cdots + \mathbf{b}_n B_n^n(t), \quad 0 \leq t \leq 1$, eine *Bézier–Kurve* vom (maximalen) Grad n. Die Punkte $\mathbf{b}_0, \cdots, \mathbf{b}_n$ heißen **Kontrollpunkte** der Bézierkurve. (vgl. 6.4) Faßt man die 1-dimensionalen Unterräume

$$< \mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \cdots + \mathbf{b}_n B_n^n(t) >, 0,$$

als Punkte der reellen projektiven Ebene mit der Ferngerade $x_3 = 0$ auf, so bezeichnet man den affinen Anteil dieser *projektiven Bézierkurve* als **rationale Bézierkurve**.
Die Kontrollpunkte der Bézierkurve im $\mathbb{R}^3$ (von der wir ausgegangen sind), lassen sich folgendermaßen beschreiben:
$\mathbf{b}_i = w_i(x_i, y_i, 1),\ w_i > 0$ falls $\mathbf{b}_i$ nicht auf der Ferngerade $x_3 = 0$ und

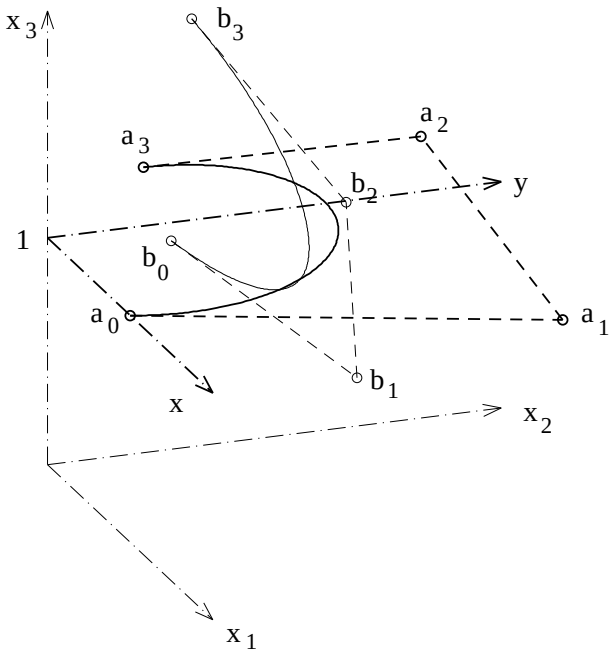


Abbildung 19.1: Zentralprojektion einer Bézierkurve in die Ebene $x_3 = 1$

$\mathbf{b}_i = w_i(x_i, y_i, 0),\ w_i > 0$ falls $\mathbf{b}_i$ auf der Ferngerade
liegt. Beim Übergang zu inhomogenen Koordinaten wird ein Kontrollpunkt entweder auf den affinen Punkt $\mathbf{a}_i := (x_i, y_i)$ oder auf den Fernpunkt in Richtung $\mathbf{a}_i := (x_i, y_i)$ abgebildet. Der Punkt $\mathbf{a}_i$ heißt *eigentlicher* bzw. *uneigentlicher* Kontrollpunkt der rationalen Bézierkurve und die Zahl w_i heißt das *Gewicht* des Kontrollpunktes $\mathbf{a}_i$.
Eine rationale Bézierkurve hat also folgende affine Beschreibung:

$$\frac{\sum_{i=0}^{n} w_i \mathbf{a}_i B_i^n(t)}{\sum_{i=0}^{n} \delta_i w_i B_i^n(t)},$$

wobei $\delta_i = 1$ für eigentliche und $\delta_i = 0$ für uneigentliche Kontrollpunkte zu setzen ist.
Die rationalen Bézierkurven haben (u.a.) die folgenden Eigenschaften:

Sind $\mathbf{a}_i, w_i, i = 0, ..., n$, eigentliche Kontrollpunkte bzw. die Gewichte einer rationalen Bézierkurve Γ, so gilt

1. Die Kurve Γ enthält die Kontrollpunkte $\mathbf{a}_0, \mathbf{a}_n$ (erster bzw. letzter Punkt des Kontrollpolygons).

2. Die Tangente im Punkt $\mathbf{a}_0$ bzw. $\mathbf{a}_n$ hat die Richtung $\mathbf{a}_1 - \mathbf{a}_0$ bzw. $\mathbf{a}_n - \mathbf{a}_{n-1}$.

3. Eine Erhöhung des Gewichts w_i bewirkt eine Veränderung der Kurve auf den Kontrollpunkt $\mathbf{a}_i$ zu. (s. Abbildung) Die Abhängigkeit der Kurve von den Gewichten ist redundant, d. h. dieselbe Kurve kann durch dieselben Kontrollpunkte, aber mit verschiedenen Gewichten dargestellt werden.

19.3 Kegelschnitte als rationale Bézierkurven

Eine Bézierkurve vom Grad zwei mit nicht kollinearen Kontrollpunkten $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ im $\mathbb{R}^3$ ist immer eine Parabel. Denn jede Komponente ist ein Polynom vom Grad 2, so da diese Kurve eine Parameterdarstellung der Form

$$\mathbf{x}(t) = \mathbf{p}_0 + t\mathbf{f}_1 + t^2\mathbf{f}_2$$

hat (vgl. Kapitel 5). Erzeugt man mit einer Parabel im $\mathbb{R}^3$, deren Ebene den Nullpunkt nicht enthält, eine projektive (ebene) Kurve (s.o.) und betrachtet ihre Realisierung in der Ebene $x_3 = 1$, so ergibt sich eine Ellipse oder eine Hyperbel oder eine Parabel. Z.B. sind

$$< (t^2(1,0,0) + 2t(1-t)(0,0,1/2) + (1-t)^2(0,1,0) >,\ t \in \mathbb{R},$$

Punkte des projektiven Kegelschnitts mit der Gleichung $x_1x_2 = x_3^2$ (vgl. Anhang2). Dieser Kegelschnitt ist in der affinen Einbettungsebene $x_3 = 1$ eine Hyperbel (s. Abbildung). Der Übergang zu inhomogenen Koordinaten liefert die rationale Parameterdarstellung $x(t) := t/(1-t), y(t) = (1-t)/t$ der Hyperbel $y = 1/x$.

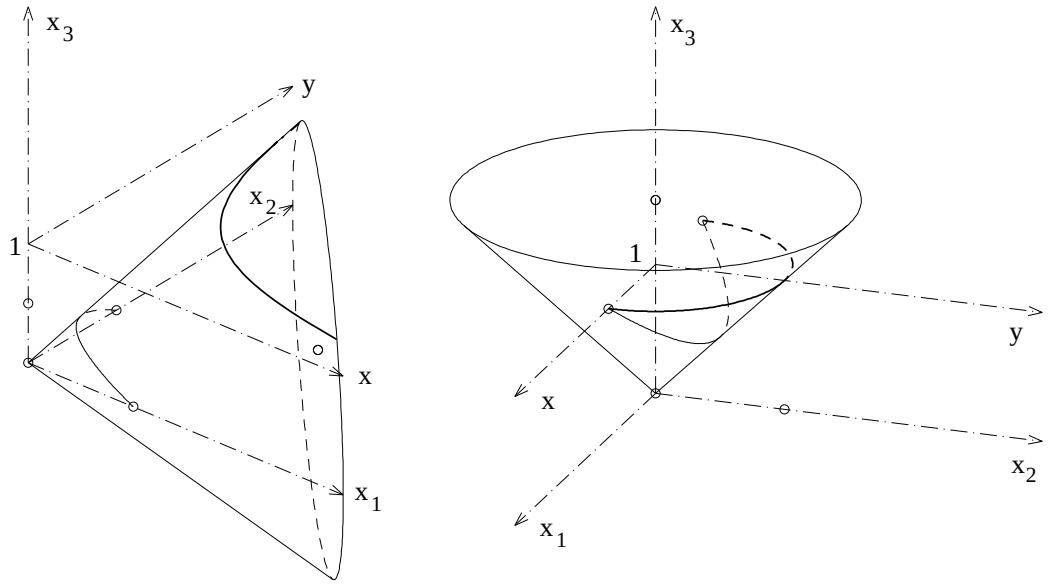


Abbildung 19.2: a) Hyperbel als rationale Bézierkurve mit zwei uneigentlichen Kontrollpunkten, b) Halbkreis als rationale Bézierkurve mit einem uneigentlichem Kontrollpunkt

In dem folgenden Beispiel ist der Kontrollpunkt b_1 uneigentlich:

$$< (t^2(1,0,1) + 2t(1-t)(0,1,0) + (1-t)^2(-1,0,1) >,\ 0 \le t \le 1,$$

ist die homogene Darstellung eines Halbkreises. Die projektiven Punkte liegen auf dem projektiven Kegelschnitt mit der Gleichung $x_1^2 + x_2^2 = x_3^2$ (s. Abbildung).

Kapitel 20

BEZIER-FLÄCHEN

Es gibt zwei naheliegende Möglichkeiten, die Béziertechnik für Kurven auf Flächen zu übertragen: Tensorprodukt–Bézierflächen und Dreiecks–Bézierflächen. Die letzteren stehen der Idee der Bézierkurven wohl am nächsten.

20.1 Tensorprodukt-Bézierfläche

Es sei $\mathbf{b}^m(u) = \sum_{i=0}^m \mathbf{b}_i B_i^m(u)$ eine Bézierkurve im $\mathbb{R}^3$. Nun nehmen wir an, daß die Kontrollpunkte von einem weiteren Parameter v abhängen, und zwar sollen sie selbst auf Bézierkurven liegen: $\mathbf{b}_i(v) = \sum_{j=0}^n \mathbf{b}_{ij} B_i^n(v)$. Damit beschreibt

$$\mathbf{b}^{m,n}(u,v) = \sum_{i=0}^m \sum_{j=0}^n \mathbf{b}_{ij} B_i^m(u) B_i^n(v), u, v \in [0,1]$$

eine Fläche, die zu den *Kontrollpunkten* oder *Kontrollnetz* $\mathbf{b}_{ij}$ gehörige **(m,n)–Tensorprodukt–Bézierfläche**.
Die Fläche enthält die Punkte $\mathbf{b}_{00}, \mathbf{b}_{m0}, \mathbf{b}_{0n}, \mathbf{b}_{mn}$ und die Randkurven sind Bézierkurven.
Man beachte, daß eine $(1,1)$–Tensorprodukt–Bézierfläche zwar Geraden enthält, aber i.a. nicht eben ist. Z.B. erhält man für $\mathbf{b}_{00} = (0,0,0), \mathbf{b}_{10} = (1,0,0), \mathbf{b}_{01} = (0,1,0), \mathbf{b}_{11} = (1,1,1)$ einen Teil der Quadrik $z = xy$.

20.2 Der Casteljau-Algorithmus

Die Grundidee des Casteljau–Algorithmus für Kurven ist die lineare Interpolation von Punktepaaren. Überträgt man diese Idee auf Tensorprodukt–Bézierflächen, so muß man eine **bi**lineare Interpolation für vier Punkte definieren. Sie ist, wie bei Kurven, am einfachsten Fall ablesbar: Eine (1,1)-Tensorprodukt–Bézierfläche auf den vier Punkten $\mathbf{b}_{00}, \mathbf{b}_{10}, \mathbf{b}_{01}, \mathbf{b}_{11}$ hat die folgende Darstellung:

$$\mathbf{b}^{1,1}(u,v) = (1-u)(1-v)\mathbf{b}_{00} + u(1-v)\mathbf{b}_{10} + (1-u)v\mathbf{b}_{01} + uv\mathbf{b}_{11}$$

Oder in Matrixform:

$$\mathbf{b}^{1,1}(u,v) = (1-u, u) \begin{pmatrix} \mathbf{b}_{00} & \mathbf{b}_{01} \\ \mathbf{b}_{10} & \mathbf{b}_{11} \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

Wir gehen zunächst von einem $(n \times n)$–Kontrollnetz aus und bestimmen (wie bei Kurven) für $r = 1, 2, .., n$ und (einem Parameterpaar (u,v) Zwischenvektoren, die durch bilineare Interpolation

entstehen:

$$\mathbf{b}_{i,j}^r = (1-u,u)\begin{pmatrix} \mathbf{b}_{i,j}^{r-1} & \mathbf{b}_{i,j+1}^{r-1} \\ \mathbf{b}_{i+1,j}^{r-1} & \mathbf{b}_{i+1,j+1}^{r-1} \end{pmatrix}\begin{pmatrix} 1-v \\ v \end{pmatrix},$$

wobei $\mathbf{b}_{i,j}^0 := \mathbf{b}_{i,j}$ ist. Dann sei $\mathbf{b}_{0,0}^n$ der Punkt, der dem Parameterpaar (u,v) zugeordnet wird.

Falls $m > n$ ist, ist ab $r = n$ der zweite Index konstant $j = 0$ und es wird nur noch linear interpoliert (wie bei Bézierkurven). Der Punkt $\mathbf{b}_{0,0}^m$ ist dann der Flächenpunkt.
Analog verfährt man, falls $m < n$ ist.

20.3 Graderhöhung

Es ist oft von Vorteil, wenn für eine (m,n)–Tensorprodukt–Bézierfläche $m = n$ ist. Falls dies nicht der Fall ist, läßt sich dies mit Hilfe geeigneter Graderhöhungen erreichen.
Die Graderhöhung von (m,n) auf $(m+1,n)$ der Tensorprodukt–Bézierfläche

$$\mathbf{b}^{m,n}(u,v) = \sum_{j=0}^{n}\left[\sum_{i=0}^{m}\mathbf{b}_{ij}B_i^m(u)\right]B_i^n(v)$$

führt auf die $n+1$ Graderhöhungen für die Bézierkurven in der eckigen Klammer:

$$\sum_{i=0}^{m}\mathbf{b}_{ij}B_i^m(u) = \sum_{i=0}^{m+1}\mathbf{b}_{ij}^{(1,0)}B_i^{m+1}(u), \quad j = 0,...n$$

mit

$$\mathbf{b}_{ij}^{(1,0)} := (1-\frac{i}{m+1})\mathbf{b}_{i,j} + \frac{i}{m+1}\mathbf{b}_{i-1,j}, \quad i = 0,...,m+1.$$

20.4 Ableitungen einer Bézier-Fläche

Die partielle Ableitung der Tensorprodukt–Bézierfläche

$$\mathbf{b}^{m,n}(u,v) = \sum_{j=0}^{n}\sum_{i=0}^{m}\mathbf{b}_{ij}B_i^m(u)B_i^n(v)$$

nach u ist

$$\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v) = \sum_{j=0}^{n}\left[\frac{\partial}{\partial u}\sum_{i=0}^{m}\mathbf{b}_{ij}B_i^m(u)\right]B_i^n(v).$$

Mit dem Resultat für die Ableitung einer Bézierkurve (18.3) ergibt sich:

$$\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v) = m\sum_{j=0}^{n}\left[\sum_{i=0}^{m-1}\Delta^{1,0}\mathbf{b}_{ij}B_i^{m-1}(u)\right]B_i^n(v),$$

wobei $\Delta^{1,0}\mathbf{b}_{i,j} := \mathbf{b}_{i+1,j} - \mathbf{b}_{i,j}$. Analog erhält man die partielle Ableitung nach v und alle höheren Ableitungen.

Da die Vektoren $\Delta^{1,0}\mathbf{b}_{0,0}, \Delta^{0,1}\mathbf{b}_{0,0}$ Tangentenvektoren der im Punkt $\mathbf{b}_{0,0}$ beginnenden Randkurven sind, ist $\Delta^{1,0}\mathbf{b}_{0,0} \times \Delta^{0,1}\mathbf{b}_{0,0}$ ein Normalenvektor der Fläche in diesem Punkt, falls beide linear unabhängig sind. D.h. die Tangentialebene in den Eckpunkten einer Tensorprodukt–Bézierfläche wird i.a. jeweils von dem Eckpunkt und seinen Nachbarpunkten im Kontrollnetz aufgespannt.

20.5 Dreiecks–Bézierflächen

Eine formale Verallgemeinerung der Bernstein–Polynome auf Funktionen mit zwei Variablen, würde von der Beziehung $1 = (u + v + (1 - u - v))^n = \cdots$ ausgehen. Damit die auftretenden Terme alle positiv sind, muß (u, v) in dem Dreieck $(0, 0), (1, 0), (0, 1)$ liegen. Zwei der drei Dreiecksseiten spielen als Intervalle auf den Koordinatenachsen eine besondere Rolle. Um diese Bevorzugung zu vermeiden, führt man "homogene" Koordinaten u, v, w mit der Bedingung $u + v + w = 1, u, v, w \geq 0$ ein. u, v, w nennt man *Baryzentrische Koordinaten.* Die verallgemeinerten Bernsteinpolynome ergeben sich aus der Entwicklung von $(u + v + w)^n$ zu:

$$B^n_{ijk}(u, v, w) := \frac{n!}{i!j!k!} u^i v^j w^k$$

mit $i + j + k = n,\ i, j, k \geq 0$ und $u + v + w = 1,\ u, v, w \geq 0$.
Mit den Abkürzungen $\mathbf{I} := (i, j, k),\ \mathbf{u} := (u, v, w)$ und $|\mathbf{I}| := i + j + k,\ |\mathbf{u}| := u + v + w$ ist

$$B^n_{\mathbf{I}}(\mathbf{u}) := \frac{n!}{i!j!k!} u^i v^j w^k, \quad |\mathbf{u}| = 1 \quad \text{und} \quad \sum_{|\mathbf{I}|=n} B^n_{\mathbf{I}} = 1.$$

Ist nun $\mathbf{b}_{00n}, \mathbf{b}_{10,n-1}, ...\mathbf{b}_{n00}, \mathbf{b}_{01,n-1}, \mathbf{b}_{11,n-2}, ..., \mathbf{b}_{n-1,10}, ..., \mathbf{b}_{0n0}$ ein dreieckiges Netz von Punkten des $\mathbb{R}^3$, den **Kontrollpunkten**, so ist

$$\mathbf{b}^n(\mathbf{u}) := \sum_{|\mathbf{I}|=n} \mathbf{b}_{\mathbf{I}} B^n_{\mathbf{I}}(\mathbf{u})$$

die zugehörige **Dreiecks–Bézierfläche**.

Die Abb. 20.1 zeigt die Anordnung der Punkte für den Fall $n = 4$.

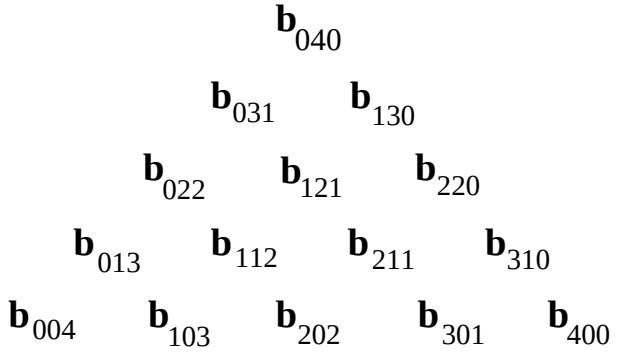


Abbildung 20.1: Kontrollpunkte einer Dreiecks–Bézierfläche für den Fall $n = 4$

Um den **Casteljau–Algorithmus** für Dreiecks–Bézierflächen übersichtlich formulieren zu können, führen wir noch die folgenden Abkürzungen ein: $\mathbf{e}_1 := (1, 0, 0), \mathbf{e}_2 := (0, 1, 0), \mathbf{e}_3 := (0, 0, 1)$ und $\mathbf{o} := (0, 0, 0)$.
Es sei nun $\{\mathbf{b}_{\mathbf{I}} || \mathbf{I}| = n\}$ ein dreieckiges Netz von Punkten im $\mathbb{R}^3$ und $\mathbf{u}$ ein Parametervektor in baryzentrischen Koordinaten. Wir setzen für $r = 1, ..., n$ und $\mathbf{I} = n - r$

$$\mathbf{b}^r_{\mathbf{I}} := u\mathbf{b}^{r-1}_{\mathbf{I}+\mathbf{e}_1}(\mathbf{u}) + v\mathbf{b}^{r-1}_{\mathbf{I}+\mathbf{e}_2}(\mathbf{u}) + w\mathbf{b}^{r-1}_{\mathbf{I}+\mathbf{e}_3}(\mathbf{u})$$

mit $\mathbf{b}_{\mathbf{I}}^0(\mathbf{u}) := \mathbf{b}_{\mathbf{I}}$. Dann ist $\mathbf{b}_{\mathbf{o}}^n(\mathbf{u})$ ein Punkt der Dreiecks–Bézierfläche. Der Nachweis, daß der Casteljau–Algorithmus wirklich einen Punkt der Dreiecks–Bézierfläche liefert, verwendet (analog zum Kurvenfall) die Rekursionsformeln für Bernsteinpolynome:

$$B_{\mathbf{I}}^n(\mathbf{u}) = uB_{\mathbf{I}-\mathbf{e}_1}^{n-1}(\mathbf{u}) + vB_{\mathbf{I}-\mathbf{e}_2}^{n-1}(\mathbf{u}) + wB_{\mathbf{I}-\mathbf{e}_3}^{n-1}(\mathbf{u}), \quad |\mathbf{I}| = n.$$

Kapitel 21

B-SPLINEKURVEN

Bézierkurven haben zwei entscheidende Nachteile:

- Soll die Kurve durch viele Kontrollpunkte beschrieben werden, ist der Grad der Kurve hoch.
- Ändert man einen Kontrollpunkt, so ändert sich die ganze Kurve.

Diese Nachteile lassen sich durch Verwendung von Basisfunktionen mit *lokaler* Wirkung beseitigen.

21.1 Die B–Spline–Basisfunktionen

Die B-Spline-Basisfunktionen sind stückweise polynomial mit lokalem Träger, d.h. sie sind nur auf vorgegebenen Intervallen ungleich Null. Sie werden rekursiv definiert:

Ein Vektor $(t_0, t_1, ..., t_m) \in \mathbb{R}^m$ mit der Eigenschaft $t_0 < t_1 \cdots < t_m$ heißt *Knotenvektor*.
Es sei nun $n, k \in \mathbb{N}, n, k \geq 1$ und der Knotenvektor $T := (t_0, t_1, ..., t_{n+k})$ fest vorgegeben. Mit den Funktionen

$$N_{i1} := \begin{cases} 1 & , \quad t_i \leq t \leq t_{i+1} \\ 0 & , \quad \text{sonst} \end{cases}$$

für $i = 1, 2, ..., n + k$ definieren wir rekursiv für $k > 1$:

$$N_{ik} := \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t).$$

Die Funktionen N_{ik} heißen **B–Spline–Basisfunktionen** der **Ordnung k**. Sie sind *stückweise aus Polynomen (k-1)-ten Grades* zusammengesetzt und haben die folgenden Eigenschaften

(B1) $N_{ik}(t) > 0 \quad$ für $\quad t_i < t < t_{i+k}, \quad$ das Intervall $[t_i, t_{i+k}]$ heißt *Träger* von N_{ik},

(B2) $N_{ik}(t) = 0 \quad$ für $\quad t_0 \leq t \leq t_i, \quad t_{i+k} \leq t \leq t_{n+k}$,

(B3) $\sum_{i=0}^{n} N_{ik}(t) = 1 \quad$ für $\quad t \in [t_{k-1}, t_{n+1}]$,

(B4) für $t_i \leq t_l \leq t_{i+k}$ sind die Funktionen N_{ik} an den Knotenwerten t_l C^{k-2}–stetig.

Wählt man für den Trägervektor speziell $(0, 1, 2, ..., n + k)$ so heißen die Basisfunktionen *uniform*.
Die folgenden Abbildungen zeigen Graphen von einigen B–Spline–Basisfunktionen der Ordnung 1,2 und 3. Es ergeben sich dann stückweise Polynome vom Grad 0,1 bzw. 2.

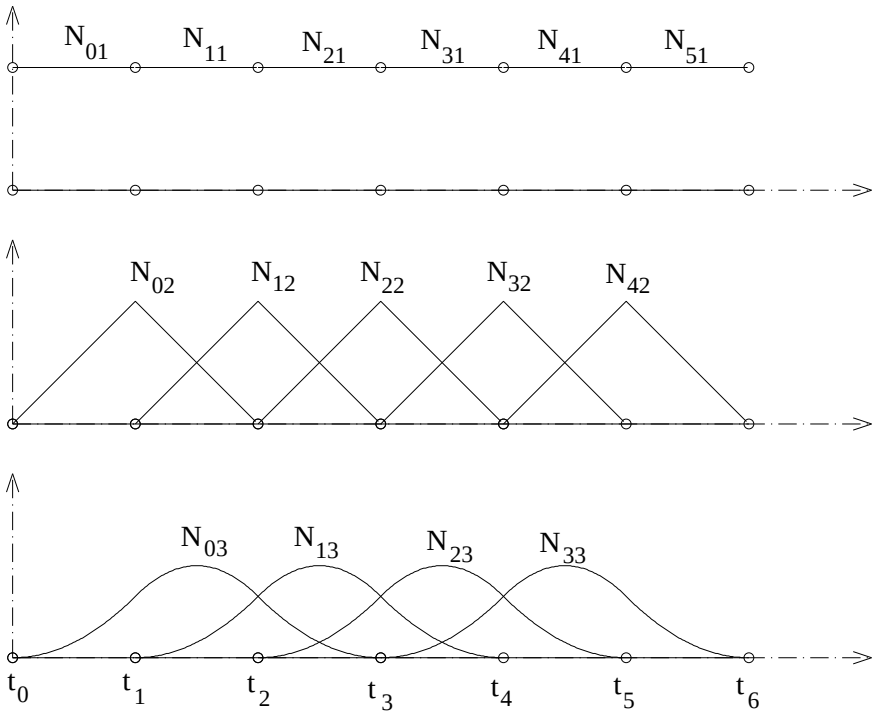


Abbildung 21.1: B–Spline–Basisfunktionen mit ausschließlich einfachen Knoten

Eine wichtige Erweiterung der Definition der Basisfunktionen ist die Möglichkeit *Knoten zusammenfallen* zu lassen.Ist z. B. $t_1 = t_0$, so ist $N_{01} = 0$ und man läßt bei der Berechnung von N_{02} den Term $\frac{t-t_0}{t_1-t_0}N_{01}(t)$ weg. Speziell für $t_0 = t_1 = 0, t_2 = 1$ ergibt sich also $N_{02} = 1 - t$, das Bernsteinpolynom $B_1^1(t)$. Dies ist kein Zufall. Es gilt allgemein:

$$T = (\underbrace{0, 0, ..., 0}_{k-mal}, \underbrace{1, 1, ...1}_{k-mal}) \quad \Longrightarrow \quad N_{ik}(t) = B_i^{k-1}(t).$$

Fallen im Knotenvektor l Knoten zusammen, so ist N_{ik} nur noch C^{k-l-1}–stetig.

Abb. 21.2 zeigt Basisfunktionen für $k = 3$ mit zusammenfallenden Knoten. Man beachte die Unstetigkeit von N_{23} und N_{33} im mittleren Bild am 3-fachen Knoten $t_3 = t_4 = t_5$.

Analog zur Bézierkurve definiert man jetzt mit Hilfe der B–Spline–Basisfunktionen die **B–Spline–Kurve**:
Es seien $n, k \in \mathbb{N},\ T = (t_0, t_1, ..., t_{n+k})$ ein Knotenvektor und $\mathbf{d}_i \in \mathbb{R}^2, i = 0, 1, ..., n$, dann beschreibt

$$\mathbf{x}(t) := \sum_{i=0}^{n} \mathbf{d}_i N_{ik}(t), \quad t_{k-1} \le t \le t_{n+1}$$

eine Kurve, die **B–Spline–Kurve** mit den **Kontrollpunkten** oder **de–Boor–Punkten** $\mathbf{d}_0, ..., \mathbf{d}_n$. Soll die Kurve durch die Punkte $\mathbf{d}_0, \mathbf{d}_n$ gehen, so müssen die ersten k Knoten und die letzten k Knoten zusammen fallen.

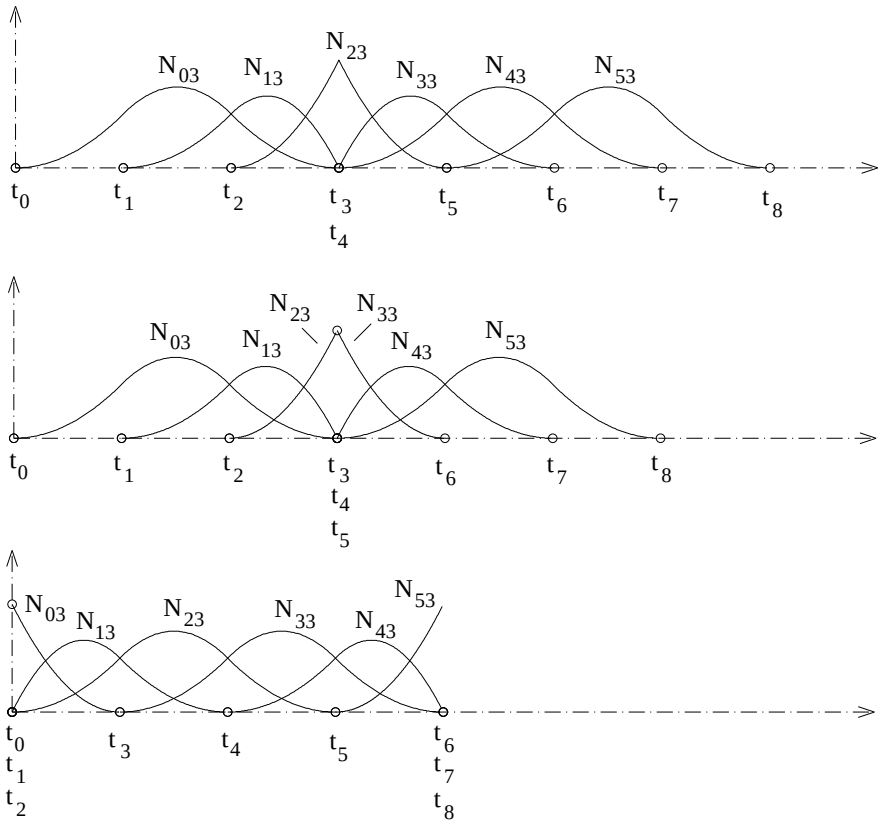


Abbildung 21.2: B–Spline–Basisfunktionen mit Mehrfachknoten

21.2 Der de-Boor-Algorithmus

Anhang A

DIE REELLE PROJEKTIVE EBENE

A.1 Die reelle affine Ebene

P: Punkte der reellen Anschauungsebene.
G: Geraden der reellen Anschauungsebene als Teilmengen von **P**.
$(\mathbf{P}, \mathbf{G}, \in)$ heißt *reelle affine Ebene*.

Algebraische Beschreibung der reellen affinen Ebene:

$$\begin{array}{rcl} \mathbf{P} & = & \mathbb{R}^2 \\ \mathbf{G} & = & \{\{(x,y) \in \mathbb{R}^2 |\ ax+by+c=0\} |\ (0,0) \neq (a,b) \in \mathbb{R}^2\} \\ & = & \{\{(x,y) \in \mathbb{R}^2 |\ y=mx+d\} |\ m,d \in \mathbb{R}\} \cup \{\{(x,y) \in \mathbb{R}^2 \mid x=c\} | c \in \mathbb{R}\} \\ \mathbf{A}(\mathbb{R}) & := & (\mathbf{P}, \mathbf{G}, \in) \end{array}$$

Eine Permutation von **P**, die eine Permutation von **G** induziert, heißt *Kollineation* (von $\mathbf{A}(\mathbb{R})$).

Resultat:
Ist κ eine Kollineation von $\mathbf{A}(\mathbb{R})$, so gibt es $a, b, c, d, s, t \in \mathbb{R}$ so, daß

$$\kappa : (x,y) \to (ax+by+s, cx+dy+t), \quad ad-bc \neq 0,$$

ist. κ heißt auch *Affinität*.

Beispiele:

a)	(x,y)	$\to$	$(x+s, y+t)$,	$s,t \in \mathbb{R}$	Translation.
b)	(x,y)	$\to$	(x, dy),	$0 \neq d \in \mathbb{R}$	Streckung an der x-Achse in y-Richtung.
c)	(x,y)	$\to$	(ax, y),	$0 \neq a \in \mathbb{R}$	Streckung an der y-Achse in x-Richtung.
d)	(x,y)	$\to$	(ax, ay),	$0 \neq a \in \mathbb{R}$	Streckung am Punkt$(0,0)$.
e)	(x,y)	$\to$	$(x+by, y)$,	$b \in \mathbb{R}$	Scherung an der x-Achse.
f)	(x,y)	$\to$	$(x, y+cx)$,	$c \in \mathbb{R}$	Scherung an der y-Achse.

Sind A, B, P drei verschiedene kollineare Punkte und ist $\vec{AP} = t\vec{PB}$, so heißt t das *Teilverhältnis* $[AP : PB]$

Resultat:
Eine Affinität läßt Teilverhältnisse invariant.

A.2 Die reelle projektive Ebene

$\overline{\mathbf{P}}$: Punkte der reellen Anschauungsebene $\cup$ Parallelklassen der (affinen) Geraden
$\overline{\mathbf{G}}$: $\{g \cup \|(g) | g \quad \text{affine Gerade}\} \cup g_\infty$,
wobei $\|(g)$ die Menge der zu g parallelen Geraden, g_∞ die Menge der Parallelklassen ist.
$(\overline{\mathbf{P}}, \overline{\mathbf{G}}, \in)$ heißt *reelle projektive Ebene*.

Algebraische Beschreibung:

$$\begin{aligned}
\overline{\mathbf{P}}_1 &= \mathbb{R}^2 \cup \mathbb{R} \cup \{\infty\}, \quad \infty \notin \mathbb{R} \\
\overline{\mathbf{G}}_1 &= \{\{(x,y) \in \mathbb{R}^2 |\ y = mx + d\} \cup \{(m)\} |\ m, d \in \mathbb{R}\} \\
&\quad \cup \{\{(x,y) \in \mathbb{R}^2 |\ x = c\} \cup \{(\infty)\} |\ c \in \mathbb{R}\} \\
&\quad \cup \underbrace{\{\{(m) |\ m \in \mathbb{R}\} \cup \{(\infty)\}\}}_{g_\infty}
\end{aligned}$$

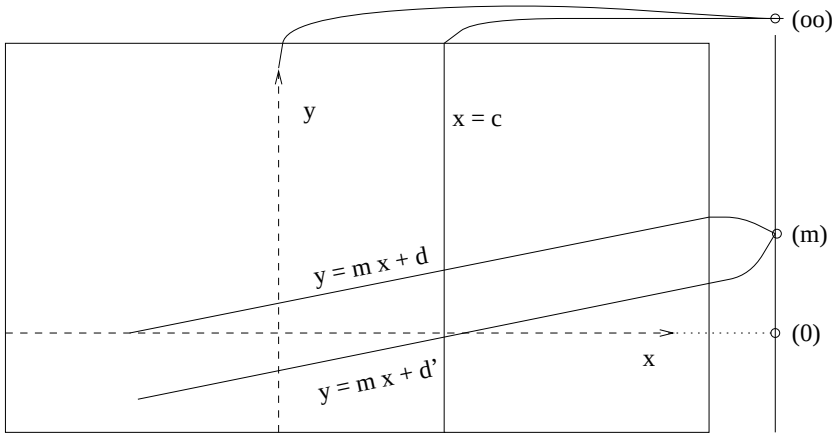


Abbildung A.1: inhomogenes Modell der reellen projektiven Ebene

$\mathrm{P}_1(\mathbb{R}) := (\overline{\mathbf{P}}_1, \overline{\mathbf{G}}_1, \in)$ heißt *inhomogenes Modell* der reellen projektiven Ebene.

Eine andere Möglichkeit, die reelle projektive Ebene zu beschreiben, ist die folgende:

Es sei $O := (0,0,0)$ und $< x >$ sei der von dem Vektor $\mathbf{x} \neq \mathbf{o}$ aufgespannte Unterraum.

$$\begin{aligned}
\overline{\mathbf{P}}_2 &= \{\text{Geraden im } \mathbb{R}^3 \text{ durch Punkt } O\} \\
&= \{< (x_1, x_2, x_3) > | (0,0,0) \neq (x_1, x_2, x_3) \in \mathbb{R}^3\} \\
\overline{\mathbf{G}}_2 &= \{\text{Ebenen im } \mathbb{R}^3 \text{ durch Punkt } O\} \\
&= \{\{< (x_1, x_2, x_3) > \in \overline{\mathbf{P}}_2 | ax_1 + bx_2 + cx_3 = 0\} | (0,0,0) \neq (a,b,c) \in \mathbb{R}^3\}
\end{aligned}$$

$\mathrm{P}_2(\mathbb{R}) = (\overline{\mathbf{P}}_2, \overline{\mathbf{G}}_2, \in)$ heißt *homogenes Modell* der reellen projektiven Ebene.

Isomorphismus zwischen beiden Modellen:

Idee:
Einbettung des inhomogenen Modells in den $\mathbb{R}^3$ als Ebene $x_3 = 1$.
Identifizierung eines Punktes P mit der Gerade $P \vee O, \quad O := (0,0,0)$.

$$\begin{array}{rcccll} (x,y) & \to & <(x,y,1)> & = & <(xx_3, yx_3, x_3)>, & x_3 \neq 0. \\ (m) & \to & <(1,m,0)> & = & <(x_1, mx_1, 0)>, & x_1 \neq 0. \\ (\infty) & \to & <(0,1,0)> & = & <(0,x_2,0)>, & x_2 \neq 0. \end{array}$$

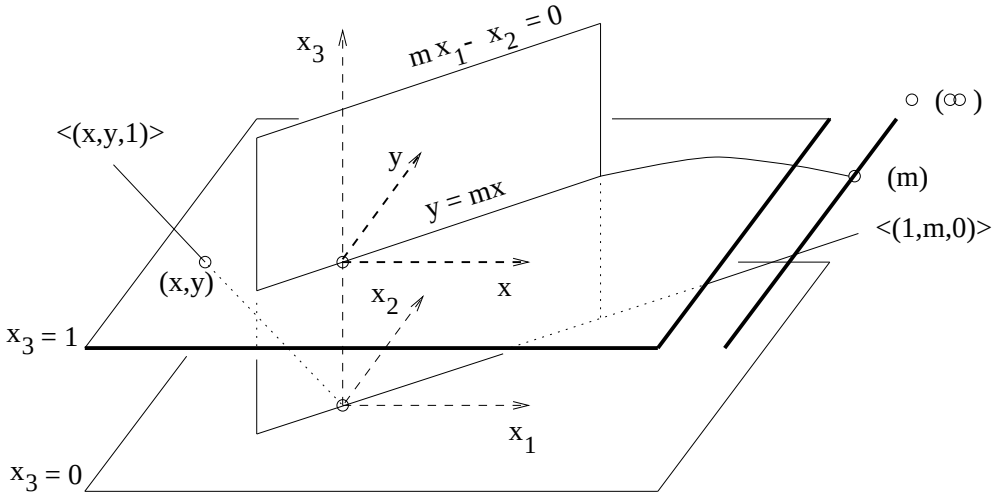


Abbildung A.2: Isomorphismus zwischen dem homogenen und dem inhomogenen Modell

Umkehrung:

$$\begin{array}{lcccl} <(x_1,x_2,x_3)> & \to & \left(\frac{x_1}{x_3}, \frac{x_2}{x_3}\right) & , & \text{falls} \quad x_3 \neq 0 \\ <(x_1,x_2,0)> & \to & \left(\frac{x_2}{x_1}\right) & , & \text{falls} \quad x_1 \neq 0 \\ <(0,x_2,0)> & \to & (\infty) & , & \text{falls} \quad x_2 \neq 0. \end{array}$$

Speziell:

$$\begin{array}{rclrcl} (0,0) & \longleftrightarrow & <(0,0,1)>, & (0) & \longleftrightarrow & <(1,0,0)> \\ (\infty) & \longleftrightarrow & <(0,1,0)>, & (1,1) & \longleftrightarrow & <(1,1,1)> \end{array}$$

Zuordnung "Gerade in $\mathrm{P}(\mathbb{R})$" $\longleftrightarrow$ "Gerade in $\mathrm{P}(\mathbb{R})$":

$$\begin{array}{lcll} y = mx + d & \longleftrightarrow & \frac{x_2}{x_3} = m\frac{x_2}{x_3} + d & \longleftrightarrow \quad mx_1 - x_2 + dx_3 = 0 \\ x = c & \longleftrightarrow & \frac{x_1}{x_3} = c & \longleftrightarrow \quad x_1 - cx_3 = 0 \\ g_\infty & \longleftrightarrow \text{ Gerade durch } <(1,0,0)>, & <(0,1,0)> & \longleftrightarrow \quad x_3 = 0 \end{array}$$

Bemerkung:
Natürlich sind auch andere Einbettungen möglich.
Z.B.: $(x,y) \longrightarrow <(1,x,y)>$, d.h. g_∞ ist die Gerade $x_1 = 0$ (Ebene im $\mathbb{R}^3$).

A.3 Kollineationen der reellen projektiven Ebene

Eine Permutation der Punkte, die eine Permutation der Geraden induziert, heißt *Kollineation*.

Fortsetzung der Affinitäten

	in $\mathrm{P}_1(\mathbb{R})$			in $\mathrm{P}_2(\mathbb{R})$
a)	(x,y)	$\to$	$(x+s, y+t)$	$<(x_1,x_2,x_3)> \to <(x_1+sx_3, x_2+tx_3, x_3)>$
	(m)	$\to$	(m)	(im $\mathbb{R}^3 R:$ Scherung an x_1, x_2- Ebene)
	(∞)	$\to$	(∞)	$\begin{pmatrix} 1 & 0 & s \\ 0 & 1 & t \\ 0 & 0 & 1 \end{pmatrix}$
b)	(x,y)	$\to$	(x, dy)	$<(x_1,x_2,x_3)> \to <(x_1, dx_2, x_3)>$
	(m)	$\to$	(md)	(im $\mathbb{R}^3:$ Streckung an x_1, x_3- Ebene in x_2- Richtung.)
	(∞)	$\to$	(∞)	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{pmatrix}$
c)	(x,y)	$\to$	(ax, ay)	$<(x_1,x_2,x_3)> \to <(ax_1, ax_2, x_3)>$
	(m)	$\to$	(m)	(im $\mathbb{R}^3:$ Streckung an x_3- Achse)
	(∞)	$\to$	(∞)	$\begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{pmatrix}$
d)	(x,y)	$\to$	$(x, y+cx)$	$<(x_1,x_2,x_3)> \to <(x_1, x_2+cx_1, x_3)>$
	(m)	$\to$	$(m+d)$	(im $\mathbb{R}^3$: Scherung an x_2, x_3- Ebene)
	(∞)	$\to$	(∞)	$\begin{pmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

e) allgemein:

$$\begin{array}{rcl} (x,y) & \to & (ax+by+s, cx+dy+t) \\ (m) & \to & (\frac{c+dm}{a+bm}) \end{array}$$

(Man "rechne" mit ∞ in "üblicher" Weise:

$$m \to \begin{cases} \frac{c+dm}{a+bm}, & \text{falls} \quad a+bm \neq 0 \\ \infty, & \text{falls} \quad a+bm = 0 \end{cases}, \qquad \infty \to \begin{cases} \frac{d}{b}, & \text{falls} \quad b \neq 0 \\ \infty, & \text{falls} \quad b = 0 \end{cases}.)$$

$$<(x_1,x_2,x_3)> \quad \to \quad <(ax_1+bx_2+sx_3, cx_1+dx_2+tx_3, x_3)>$$

D.h. die Fortsezung einer affinen Abbildung läßt sich im homogenen Modell durch eine lineare Abbildung mit einer Matrix der Form

$$\begin{pmatrix} a & b & s \\ c & d & t \\ 0 & 0 & 1 \end{pmatrix}$$

beschreiben.

A.4 Projektive Äquivalenz der Kegelschnitte

In diesem Abschnitt werden wir sehen, daß alle *nichtausgeartete* Kegelschnitte (Ellipse, Hyperbel, Parabel) projektiv äquivalent sind, d.h. sie lassen sich durch eine projektive Kollineation in einander überführen.

A.4.1 Die Hyperbel

Beschreibt man die Hyperbel $y = \frac{1}{x}$ über die Ersetzungen $x = x_1/x_2, y = x_2/x_3$ (s.o.) in homogenen Koordinaten, so genügen die Punkte der Hyperbel der Gleichung

$$x_1 x_2 = x_3^2 \qquad (KS1)$$

Die Menge aller (projektiven) Punkte, die dieser Gleichung genügen nennt man *nichtausgearteter Kegelschnitt*. Man beachte, daß auch die (projektiven) Punkte $< (1,0,0) >, < (0,1,0) >$ der Ferngeraden g_∞ dieser Gleichung genügen.

A.4.2 Die Parabel

Für die Parabel $y = x^2$ erhält man beim übergang zu homogenen Koordinaten die Gleichung

$$x_2 x_3 = x_1^2 \qquad (KS2).$$

Man beachte, daß auch der (projektive) Punkt $< (0,1,0) >$ der Ferngerade g_∞ dieser Gleichung genügt.
Die Kegelschnitte mit den Gleichungen (KS1) bzw. (KS2) lassen sich offensichtlich durch eine einfache Koordinatentransformation, d.h. durch eine projektive Kollineation in einander überführen.

A.4.3 Der Kreis

Geht man von dem Einheitskreis $x^2 + y^2 = 1$ aus, so erhält man in homogenen Koordinaten die Gleichung

$$x_1^2 + x_2^2 = x_3^2 \qquad (KS3).$$

Dieser Gleichung genügt kein Punkt der Ferngerade g_∞.
Die Gleichung (KS3) beschreibt im $\mathbb{R}^3$ einen Kegel mit Spitze $(0,0,0)$ und der x_3-Achse als Achse. Um zu erkennen, daß diese Gleichung mit Hilfe einer linearen Abbildung des $\mathbb{R}^3$, d.h. mit einer projektiven Abbildung der projektiven Ebene, in die Gleichung (KS2) bzw. (KS1) transformiert werden kann, schreiben wir (KS3) um in

$$x_1^2 = (x_3 - x_2)(x_3 - x_2).$$

Mit Hilfe der Koordinatentransformation

$$x_1' = x_1, \quad x_2' = x_3 - x_2, \quad x_3' = x_3 + x_2$$

erkennt man die projektive Äquivalenz dieses Kegelschnitts zu den anderen.

Zusammenfassung:
Im homogenen Modell der reellen projektiven Ebene ist
ein *Punkt* ein 1-dimensionaler Unterraum
eine *Gerade* ein 2-dimensionaler Unterraum und
ein *Kegelschnitt* die Menge der 1-dimensionale Unterräume auf einem (elliptischen) Kegel mit Spitze im Nullpunkt.

Die folgenden Bilder zeigen die Kegelschnitte Hyperbel, Parabel und Kreis mit ihren Einbettungen im homogenen Modell der reellen projektiven Ebene.

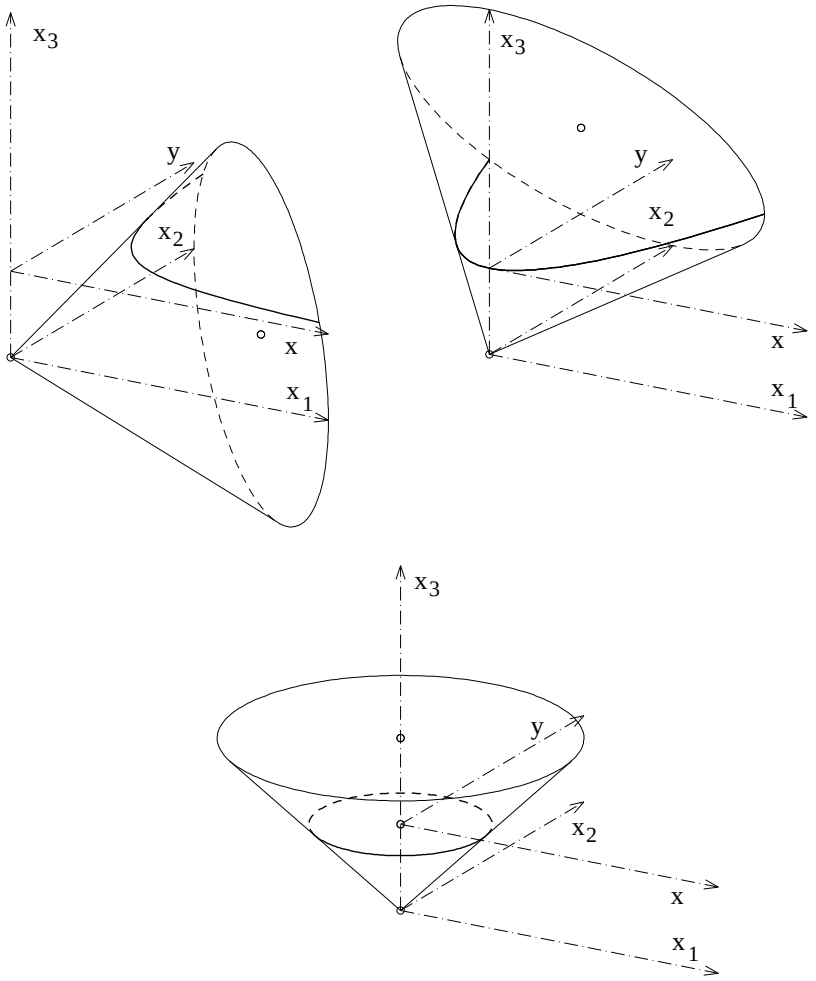


Abbildung A.3: Die projektiven Kegelschnitte und ihre Kegel mit den Gleichungen (KS1), (KS2) bzw. (KS3)