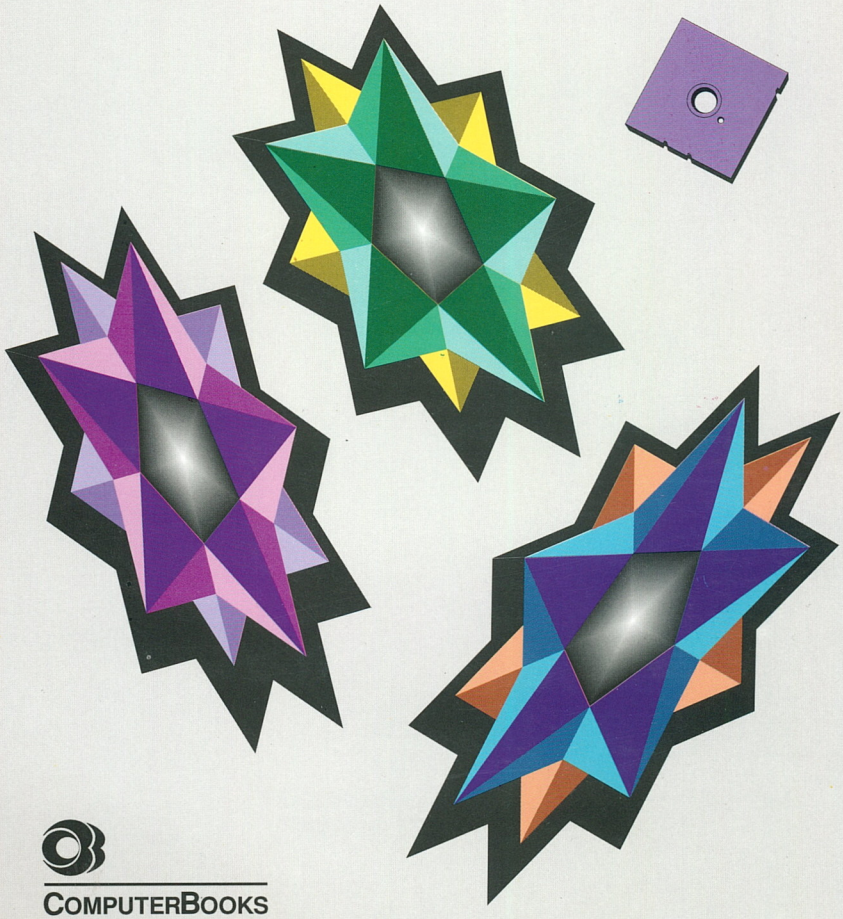
FÜZI JÁNOS

3D grafika és animáció IBM PC-n



ComputerBooks

FÜZI JÁNOS

3D grafika és animáció IBM PC-n

MÉRTANI ALAPOK
SÍKBELI—TÉRBELI ALAKZATOK
GÖRBÉK—FELÜLETEK—POLIÉDEREK
TERMÉSZETES PERSPEKTÍVA
MOZGATÁS A LÁTÓTÉRBEN

LEKTOR
DR. VERMES IMRE
BENKŐ LÁSZLÓ

COMPUTERBOOKS
BUDAPEST, 1995

Kiadó: ComputerBooks Kiadói Kft
1126 Bp., Tartsay Vilmos u. 12.
Tel.: 175-15-64; tel./fax: 175-35-91
Felelős kiadó: ComputerBooks Kft ügyvezetője
ISBN : 963 618 057 1
Borítóterv: Székely Edith

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani ***Dr. Vermes Imre*** egyetemi docensnek, a matematikai tudományok kandidátusának, a Budapesti Műszaki Egyetem Gépészmérnöki Kar Geometria Tanszék tanárának, értékes észrevételeiért, hasznos tanácsaiért valamint a kézirat többszöri és gondos lektorálásáért.

Köszönetet illeti ***Benkő Lászlót***, aki a Pascal programokat tesztelte, a programok ismertetését tartalmazó részeket lektorálta és a könyv szép megjelenése az ő kezét dicséri.

Aligha került volna sor e könyv megírására ***Benkő Tiborné*** nélkül, aki sok szervezési feladatot átvállalt és mindenképpen szívén viselte e vállalkozás sikerét.

Budapest, 1995. május 2.

A Szerző

TARTALOMJEGYZÉK

ELŐSZÓ

Napjainkban már természetes segédeszköznek tekintik az emberek a számítógépek alkalmazását az élet legkülönbözőbb területein. A felhasználók többsége a számítógépre telepített szoftverek használatát tanulja csupán meg; bonyolultabb programcsomagok esetén ez is komoly szellemi teljesítményt jelent.

Jelen könyv célja merőben más, tulajdonképpen olyan útmutatóként szolgálhat, amelyből az alapvető ismeretek birtokában meg lehet tanulni azt, hogy miként lehet geometriai, fizikai vagy egyéb síkbeli illetőleg térbeli problémát a számítógépes ábrázolással szemléltetni és ennek kapcsán megoldani.

Alapvető ismereteknek tekintjük a geometria, az ábrázoló geometriai vetítések, az egy- és többváltozós függvények differenciálszámításának alapösszefüggéseit, továbbá a Turbo Pascal programozási nyelv és a személyi számítógép használatának elemeit. Ezeket az ismereteket természetesen a könyv olvasása közben is megszerezheti az olvasó, erre szolgál a könyv végén megadott és esetenként idézett irodalom. Az elmondottak közvetlen következménye lehet az, hogy a könyvet nem bizonyos korosztálynak szántuk; jobb képességű középiskolások éppúgy használhatják, mint a felsőoktatási intézmények hallgatói. A legfontosabbnak azt tartjuk, hogy az olvasó őszintén érdeklődjék a témakör iránt, ugyanis csakis ilyen hozzáállással lehet megbirkózni az olvasás közben adódó problémákkal, a hiányoznak tűnő előismeretekben való felzárkózással. Ez a nemes küzdelem azonban minden új ismeret megszerzésének természetszerű velejárója.

A számítógépes ábrázolással való ismerkedés azonban sikerélményekhez is vezethet például olyan formában, ha a könyvben kinyomtatott programok gondos áttanulmányozása után az olvasó is megpróbál hasonló, netán egészen más problémákat megoldani, és e próbálkozások a képernyőn is szépen megjeleníthető sikerrel zárulnak.

A könyvben szereplő kidolgozott feladatok és illusztrált ábrák csak kis töredékét adják az előadódó problémáknak. A feladatok kiválogatásánál szem előtt tartottuk a módszerek sokféleségének bemutatását.

Az alkalmazott vetítési rendszerek a képsíkra - mely számunkra a számítógép képernyője - való merőleges illetve centrális vetítéseket alkalmazzák a szemléltetés igényével. Ezért e könyv a két (esetleg több) vetület alkalmazásának bemutatását nem tartja céljának. A szemléltetést azáltal is fokozni kívántuk, hogy bemutattuk a vetítési centrum, vetítési irány változtatásának illetve a szemléltetett tárgy elforgatásának lehetőségét.

Végezetül arra szeretnénk buzdítani az olvasót, hogy megszerzett tudását fokról-fokra haladva mindig próbálja ki a számítógépen, mert csak így szerezhetők szükség szerint aktivizálható ismeretek. Reményeink szerint a könyvet áttanulmányozók könnyebben fogják tudni megtanulni a bonyolult rajzoló szoftvereket, mert valamelyest bepillantást nyertek azok gondolatvilágába is.

1. Síkbeli alakzatok ábrázolása

A számítógépes ábrázolás a raszteres megjelenítés elvén alapul. Ez azt jelenti, hogy a rajzolási alapalakzat a pont (képernyőpont, pixel). A képernyő tulajdonképpen, bizonyos rendszerben elhelyezett, véges számú pont halmaza. A bonyolultabb alakzatokat, szakaszokat, sokszöglapokat pontokból építjük fel. Ez a megjelenítési mód magában hordozza az elérhető felbontás felső határát is, ugyanis a képernyőpont méreténél kisebb, finomabb részletek nem ábrázolhatók. A felbontás határai jól érzékelhetők ha majdnem vízszintes vagy majdnem függőleges szakaszt rajzolunk. A szakasz a képernyőn lépcsőzetesen elhelyezett, vízszintes (vagy függőleges), rövidebb szakaszokból tevődik össze. A VGA alapszabványban a legnagyobb felbontás 640×480 pixel (képpont), de léteznek ennél nagyobb felbontású grafikus meghajtók is.

1.1. Képernyő koordináták, lépték

A megjeleníthető képernyőpontok koordinátái természetes számok, ezt grafikus program írásakor szem előtt kell tartani, ami azt jelenti, hogy a kiszámított koordinátaértékeket kerekíteni kell a legközelebbi egész értékekre. A Turbo Pascal grafikus egységének utasításai csak egész paramétereket fogadnak el. Negatív egész számokat is, vagyis ha egy eljárás során a megjelenítendő pont valamely koordinátája negatívnak adódik, a fordító nem jelez hibát. A pont természetesen nem látszik, mert nem esik a látható képernyőtartományba. Hasonló a helyzet pozitív, az adott képernyő-korlátoknál nagyobb számok esetében is.

A képernyő koordinátarendszerének kezdőpontja a bal felső sarokban van, a vízszintes tengely (X) jobbra, a függőleges (Y) lefelé mutat. Ebből az következik, hogy a rendszer a hagyományos derékszögű koordinátarendszerrel ellentétes irányítású. Ezt úgy tudjuk megváltoztatni, hogy megadunk egy kezdőpontot (X_0, Y_0) és a kiszámított abszcisszákat hozzáadjuk a kezdőpont abszcisszájához, az ordinátákat pedig kivonjuk a kezdőpont ordinátájából (X tengely szerinti tükrözés).

A VGA grafikus meghajtó ***vgahi*** üzemmódjában egy sorban 640 pixel van, a teljes képernyő 480 sorból áll, ***vgamed*** üzemmódban pedig ugyancsak 640 pixel van egy sorban, de csak 350 sor egy képernyőben. Az utóbbi üzemmód előnye, hogy lehetővé teszi két lap használatát és ezáltal animációs programok készítését.

Az ábrázolás előkészítésének lényeges lépése a megfelelő lépték kiválasztása. Túl nagy rajz nem fér bele a képernyőbe, túl apró pedig nem látható jól, a felbontás korlátai miatt nem is tartalmazza az eredeti alakzat minden részletét. A léptéket legkönnyebben az ábrázolni kívánt tartomány köré írt téglalap és a felhasználandó képernyőtéglalap közötti megfeleltetés segítségével adhatjuk meg. Ezt a megfeleltetést a rajzolás során folyamatosan használjuk.

Legyen az ábrázolandó tartomány az $[a,b]\times[c,d]\subset\Re^2$ téglalap, amit a képernyő $[A,B]\times[C,D]\subset\{0,1,\ldots,639\}\times\{0,1,\ldots,479\}$ tartományában kívánunk megjeleníteni. A két tartomány hasonlóságát a

$$\frac{B-A}{b-a}=\frac{D-C}{d-c} \tag{1.1}$$

összefüggés biztosítja. Mivel a képernyő-koordináták egész számok, a fenti összefüggés általában nem teljesülhet pontosan.

Egy $(x,y)\in[a,b]\times[c,d]$ pontnak a képernyőn az (X,Y) pont felel meg:

$$\begin{cases} X=\left\lfloor\dfrac{Ab-Ba+(B-A)x}{b-a}+0.5\right\rfloor \\ Y=\left\lfloor\dfrac{Dd-Cc-(D-C)y}{d-c}+0.5\right\rfloor \end{cases} \tag{1.2}$$

ahol a szögletes zárójelek a bennük levő mennyiség egész részét jelentik. Program írásakor lehet a kerekítő utasítást (***round***) használni az (1.2) képletben szereplő egész rész függvény (***trunc***) helyett és akkor nem kell a 0.5-öt hozzáadni a zárójelben szereplő mennyiséghez.

A képernyő egy (X,Y) pontjának az értelmezési tartományban a

$$\left[\frac{(b-a)(X-0.5)-Ab+Ba}{B-A},\frac{(b-a)(X+0.5)-Ab+Ba}{B-A}\right]\times \times\left[\frac{-(d-c)(Y+0.5)+Dd-Cc}{D-C},\frac{-(d-c)(Y-0.5)+Dd-Cc}{D-C}\right] \tag{1.3}$$

téglalap felel meg.

Ha a léptéket explicit módon adjuk meg, vagyis azt mondjuk, hogy az ábrázolni kívánt alakzat egységnyi hosszúságú szakasza a képernyőn s pixel legyen, és a koordinátarendszer kezdőpontja a képernyő (X_0, Y_0) pontjába kerüljön, akkor az 1.2 megfeleltetési képlet helyett az

$$\begin{cases} X = X_0 + sx \\ Y = Y_0 - sy \end{cases} \tag{1.4}$$

összefüggéseket használjuk, *(1.3)* helyett pedig a

$$\left[\frac{X-X_0-0.5}{s},\frac{X-X_0+0.5}{s}\right]\times\left[\frac{-Y+Y_0-0.5}{s},\frac{-Y+Y_0+0.5}{s}\right] \tag{1.5}$$

téglalapról van szó.

1.2. Görbék a síkban

Amint már említettük, az ábrázolás alapeleme a pont. Ennek megjelenítésére szolgál a Turbo Pascal ***putpixel*** utasítása, amelynek a pont képernyő-koordinátáit és színét kell megadni. Szakaszok megjelenítésére használható a ***line*** (két megadott végpont közé rajzol egy szakaszt), a ***lineto*** (az aktuális pont és a megadott végpont közé rajzol egy szakaszt) és a ***linerel*** (az aktuális pont és az ehhez viszonyított relatív koordinátákkal megadott pont közé rajzol egy szakaszt) utasítás. A ***moveto*** vagy a ***moverel*** utasításokkal vihetjük az aktuális pontot a kívánt helyzetbe. A szakasz színét a rajzolási parancs előtt ***setcolor*** utasítással szabhatjuk meg.

Síkgörbéket kétféleképpen rajzolhatunk. Az egyik lehetőség a pontokból való előállítás. Ha elég sűrűn rajzoljuk a pontokat, a képernyőn folytonos görbét látunk. Ezt a módszert alkalmazzuk például implicit előállítású görbék vagy felületek szintvonalainak ábrázolására. A másik lehetőség a görbék egyenes szakaszokból álló törtvonalak segítségével való közelítése. Ebben az esetben is folytonos görbét látunk a képernyőn, ha elég számos, egyenként elég rövid szakaszt használunk a közelítéshez. Ezzel az eljárással rajzolhatjuk például az explicit, vagy a paraméteres előállítású görbéket, illetve a felületek esésvonalait.

1.2.1. Különböző előállítású görbék rajzolása

Többféle módon adhatunk meg síkgörbéket [2]. Az explicit előállítású görbék rajzolásának feladata azonos az egyismeretlenes valós függvények ábrázolásával, ugyanis egy görbe explicit egyenlete:

$$y = f(x)\,; \qquad f: I \subset \Re \rightarrow \Re \tag{1.6}$$

ahol I egy intervallum, vagy több intervallum egyesítése. Egy $[a,b]$ intervallumon folytonos görbét úgy is ábrázolhatunk, hogy az intervallumot felosztjuk megfelelően rövid részintervallumokra az

$$a = x_0 < x_1 < \ldots < x_{k-1} < x_k < \ldots < x_n = b \tag{1.7}$$

pontok segítségével, az osztópontokban kiszámítjuk az f függvény értékeit és lerajzoljuk a kapott pontokat összekötő

$$\overline{(x_{k-1}, y_{k-1}),(x_k, y_k)}\,, \qquad k \in \{1,2,\ldots,n\} \tag{1.8}$$

szakaszokat, amelyek tulajdonképpen az ábrázolandó görbe húrjai. Ha az osztópontok elég közel vannak egymáshoz (figyelembe véve a rajzolás léptékét is), a lerajzolt törtvonalat a képernyőn folytonos görbének látjuk. Meg kell jegyezni azonban, hogy egyenlő közű felosztás esetén az egyes szakaszok hossza nem lesz azonos, hanem függ a függvény növekedésétől az illető részintervallumon. Jó minőségű rajzot vagy úgy kaphatunk, hogy nagyon finom felosztással dolgozunk, amely biztosítja a leghosszabb szakasz megfelelő rövidségét is, vagy úgy, hogy a húrok hozzávetőleges egyenlőségét a felosztás

megfelelő megválasztásával biztosítjuk. Ez utóbbi módszert a következő alpontban vázoljuk.

A görbék megadásának másik elterjedt módja a paraméteres előállítás. Ebben az esetben a görbe pontjainak mindkét koordinátáját egy paraméter függvényében adjuk meg:

$$\begin{cases} x = x(t) \\ y = y(t) \end{cases}; \qquad t \in I \subset \Re \tag{1.9}$$

Az ábrázolást úgy végezzük, hogy az $I = [a,b]$ intervallumon osztópontokat veszünk fel:

$$a = t_0 < t_1 < \ldots < t_{k-1} < t_k < \ldots < t_n = b \tag{1.10}$$

és megrajzoljuk az

$$\overline{(x(t_{k-1}), y(t_{k-1})), (x(t_k), y(t_k))}\,, \qquad k \in \{1,2,\ldots,n\} \tag{1.11}$$

szakaszokat, vagyis a görbe húrjait. Itt is vigyázni kell arra, hogy a húrok elég rövidek legyenek, hogy a kapott ábra folytonos görbe illúzióját keltse.

Kevésbé egyszerű az implicit előállítású, vagyis az

$$F: D \subset \Re^2 \to \Re\,, \qquad F(x,y) = 0 \tag{1.12}$$

görbék ábrázolása. Egy ilyen görbe tulajdonképpen azon (x,y) pontok mértani helye, amelyekre teljesül a fenti egyenlet. Az ábrázolás úgy történik, hogy a D tartomány képernyő-megfelelőjét pixelenként végigpásztázzuk mind vízszintes, mind függőleges irányban és valahányszor az F függvény előjelet vált, berajzolunk egy pontot, vagyis megváltoztatjuk az illető pixel színét. Ez az eljárás megfelel az Euler-Monge előállítású felületek nulla névleges értékű szintvonala ábrázolásának, amint az a 2.4.2. alpontban látható.

1.2.2. Egyenlő ívhosszas rajzolás

A törtvonalas közelítés értelmében tulajdonképpen egyenlő húrszakaszos rajzolásról van szó. Rövid szakaszok esetén a szakaszok hossza a megfelelő ívhosszak közelítő értéke. Az eljárás előnye a görbék szaggatott vonalas rajzolásakor a legnyilvánvalóbb, hiszen egyforma hosszú szakaszokat és közöket csak így lehet rajzolni. Ezenkívül azonban főként az explicit, de a paraméteres előállítású görbék esetében is a képminőség lényeges javulását eredményezi anélkül, hogy túlságosan apró felbontásra lenne szükség.

Explicit előállítású görbék (1.6) esetén Δx abszcisszanövekedésnek megfelelő ívhez tartozó húr hossza:

$$\Delta s = c\sqrt{(\Delta x)^2 + \left[f(x+\Delta x) - f(x)\right]^2} \tag{1.13}$$

ahol c a lépték.

Ha van egy előre megszabott ívhosszunk, Δs_0 , akkor ha a fenti képlettel kapott érték ettől eltér, az új abszcisszalépést a

$$\Delta x' = \frac{\Delta s_0}{\Delta s}\Delta x \tag{1.14}$$

képlettel számíthatjuk ki. Ez az eljárás csak iterációs úton biztosítaná a szakaszok tökéletes egyenlőségét, ami azonban időigényes. Ha eleve nem túl nagy lépéssel kezdjük, akkor a célnak ez az első számítás is megfelel. Program írásakor az eltérés megengedhető határát is meg kell adni, hogy a rajzolás ne igényeljen túlságosan sok gépidőt.

Hasonló a helyzet paraméteres előállítású görbék esetén. A Δt paraméternövekedésnek megfelelő ívhosszhoz tartozó húr hossza:

$$\Delta s = c\sqrt{\left[x(t+\Delta t) - x(t)\right]^2 + \left[y(t+\Delta t) - y(t)\right]^2} \tag{1.15}$$

Ha $\Delta s \notin \{\Delta s_0 - \varepsilon\ ,\ \Delta s_0 + \varepsilon\}$, ahol Δs_0 az előre megszabott ívhossz, ε pedig az ettől való megengedett eltérés, akkor a paraméterlépést a

$$\Delta t' = \frac{\Delta s_0}{\Delta s} \Delta t \tag{1.16}$$

képlettel számíthatjuk újra.

1.2.3. Példák

a) Paraméteres előállítású görbék ábrázolását végzi a következő egyszerű eljárás (állandó paraméterlépéssel):

```
function x(t: real): integer;
begin
  x:=kx0+round(s*fx(t));
end;              { kiszámítja a képernyő-koordinátákat }
                  {        a paraméter függvényében     }
function y(t: real): integer;
begin
  y:=ky0-round(s*fy(t));
end;

procedure rzp;
var t: real;
begin
  t:=a; moveto(x(a), y(a));
  repeat begin
    t:=t+dt; lineto(x(t), y(t));
  end until t>=b;
end;
```

Meg kell adni a paraméter változásának korlátait (a, b), az elemi paraméterlépést (dt), a koordináták paraméteres függvényeit (fx, fy), a rajzolás léptékét (s) és a kezdőpont képernyő-koordinátáit (kx0, ky0).

Az *1.1. ábrán* Lissajoux-görbéket láthatunk, amelyeknek egyenlete:

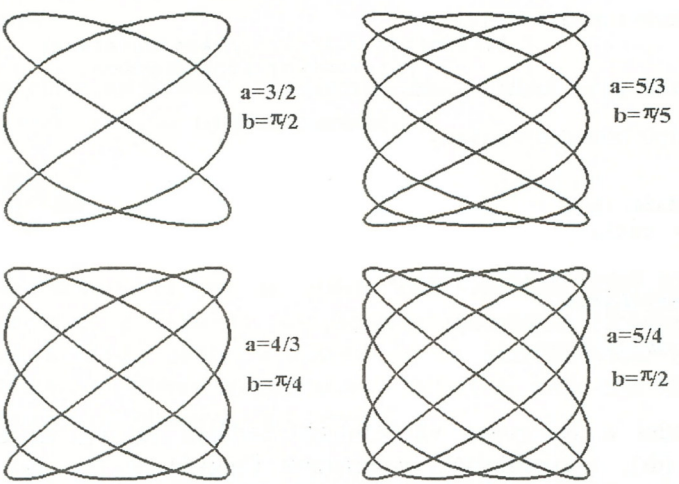
$$\begin{cases} x = \sin(at) \\ y = \cos(t+b) \end{cases}; \qquad t \in \Re$$

az *1.2. ábrán* pedig a következő egyenletű görbéket:

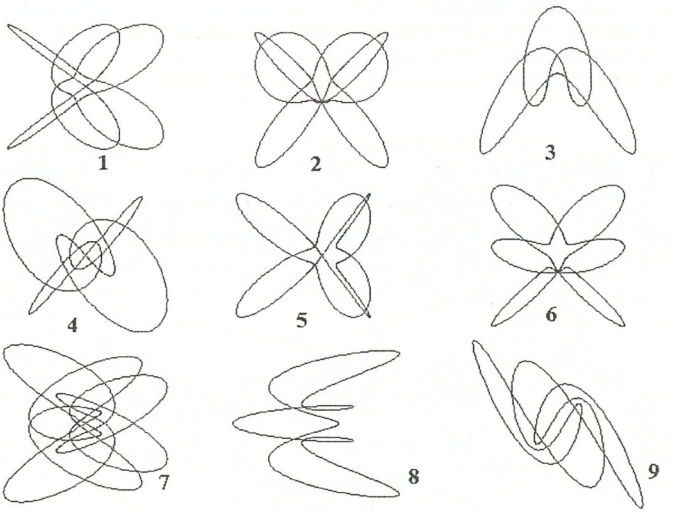
$$\begin{cases} x = [c + d \cdot \exp(\sin t)]\sin(at) \\ y = [c + d \cdot \exp(\sin t)]\cos(at) \end{cases}; \qquad t \in \Re$$

ahol az együtthatók a rajzok sorszámától függően:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
a	2/3	1/2	2/3	4/3	1/3	1/2	4/3	5/3	3/2
b	1/2	2/3	4/3	5/3	1/2	1/3	5/6	1/3	5/2
c					30				
d				3				6	10



1.1. ábra



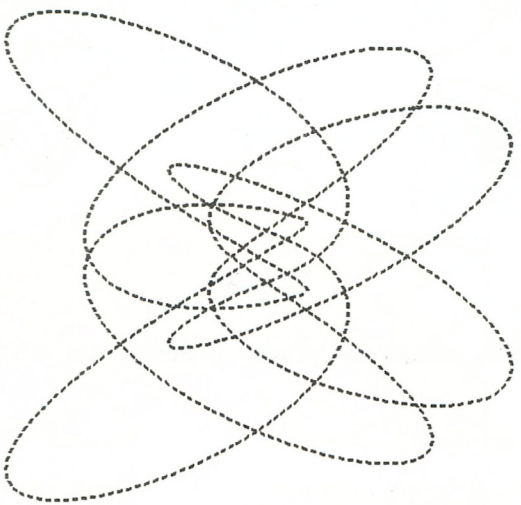
1.2. ábra

b) Egyenlő ívhosszú, szaggatott vonalas rajzolást tesz lehetővé a következő eljárás:

```
procedure rzp;
var
  t1, t2, dt, ds: real;
  f: byte;
  x1, x2, y1, y2: integer;

  procedure lepes;    { kiszámítja az egyenlő szakaszok          }
  begin               { rajzolásához szükséges paraméter-lépést }
    if (ds<ds0-eps) or (ds>ds0+eps) then begin
      dt:=ds0/ds*dt; t2:=t1+dt;
      x2:=kx0+round(lp*x(t2)); y2:=ky0-round(lp*y(t2));
      ds:=sqrt(sqr(x2-x1)+sqr(y2-y1));
    end
  end;

begin                 { minden második húrszakaszt meghúz }
  f:=0;
  t1:=a; dt:=pi/100;
  x1:=kx0+round(lp*x(t1)); y1:=ky0-round(lp*y(t1));
  repeat begin
    t2:=t1+dt;
    x2:=kx0+round(lp*x(t2)); y2:=ky0-round(lp*y(t2));
    ds:=sqrt(sqr(x2-x1)+sqr(y2-y1));
    lepes;
```

```
      if  f=0 then begin
        line(x1, y1, x2, y2); f:=1; end else f:=0;
      t1:=t2; x1:=x2; y1:=y2;
    end; until t1>=b;
  end;
```

Ezzel az eljárással készült az *1.3. ábra*, amelyen az *1.2. ábra* 7-es számú rajza látható. Az ábrán megfigyelhető a szaggatott vonal egyenletessége. A főprogramban a szakaszok-közök hosszát 4 pixelre szabtuk meg.



1.3. ábra

c) Explicit képlettel megadott görbék szaggatott vonalas (kevés változtatással folytonos vonalas), egyenlő ívhosszas rajzolására szolgál a következő eljárás:

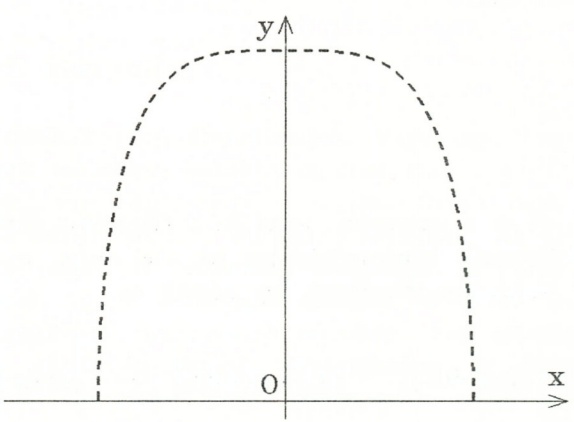
```
procedure rze;
var
  dx, ds: real;
  k, x1, y1, x2, y2: integer;
  f: byte;

procedure lepes;      { kiszámítja az egyenlő ívhosszas }
                      { rajzoláshoz                     }
begin                 { szükséges abszcisszalépést      }
 if x+dx>b then dx:=b-x;
 x2:=round(x0+lp*(x+dx)); y2:=round(y0-lp*y(x+dx));
 ds:=sqrt(sqr(dx*lp)+sqr(y2-y1));
```

```
  if (ds<ds0-eps) or (ds>ds0+eps) then dx:=ds0/ds*dx;
  if x+dx>b then dx:=b-x;{ez egy biztonsági  intézkedés arra az }
end;              { esetre ha a függvény b-nél nagyobb  }
                  { abszcisszákon nem értelmezett       }
begin
  x:=a;
  x1:=round(x0+lp*x); y1:=round(y0-lp*y(x));
  dx:=ds0/lp; f:=0;
  repeat begin
  for k:=1 to 4 do lepes;
  x:=x+dx; x2:=round(x0+lp*x); y2:=round(y0-lp*y(x));
  if f=0 then begin line(x1,y1,x2,y2); f:=1 end else f:=0;
  x1:=x2; y1:=y2;
  end until x=b;
end;
```

Az *1.4. ábrán* az

$$y = \sqrt{r^4 - x^4}\ ; \quad x \in [-r, r]$$

görbe látható. A lépésigazító eljárást többször (4-szer) meg kellett ismételni, a függvény meredeksége miatt az értelmezési tartomány szélei közelében.



1.4. ábra

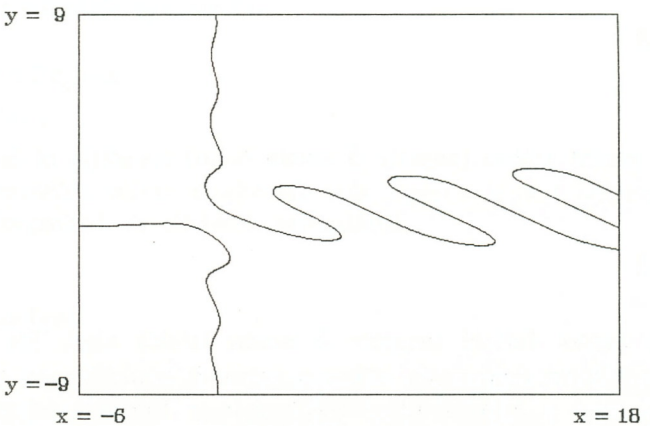
d) Implicit előállítású függvényt a következő eljárással rajzolhatunk:

```
procedure rajzx;
                                    { függőleges irányban végig-    }
begin                               { pásztázza a felhasznált kép-  }
  for k:=0 to m do begin            { ernyőtartományt               }
    x:=a+k*hx;
    for l:=0 to n do begin
      y:=c+l*hy;
      u:=f(x,y);
      if l>0 then
        if u*u0<0 then putpixel(kx0+k,ky0-l,15);
      u0:=u; end;
  end;
end;

procedure rajzy;                    { vízszintes irányban végig-    }
begin                               { pásztázza a felhasznált kép-  }
  for l:=0 to n do begin            { ernyőtartományt               }
    y:=c+l*hy;
    for k:=0 to m do begin
      x:=a+k*hx;
      u:=f(x,y);
      if k>0 then
        if u*u0<0 then putpixel(kx0+k,ky0-l,15);
      u0:=u; end;
  end;
end;

.....................................
hx:=(b-a)/m;
hy:=(d-c)/n;
rajzx; rajzy;
```

ahol $[a, b]\times[c, d]$ az a tartomány, amelyen a függvényt ábrázolni kívánjuk, kx0 és ky0 a felhasznált képernyőtéglalap bal alsó sarka, m és n a téglalap szélessége illetve magassága (pixelben). Így készült az

$$(x,y)\in[-6{,}18]\times[-9{,}9]\,,\qquad x(y+1)+\exp\left(\frac{x}{y^2+1}\right)\sin(x+2y)=0$$

egyenletű görbe *1.5. ábrán* látható képe.



1.5. ábra

1.3. Síkidomok ábrázolása

A síkidomokat kétféleképpen ábrázolhatjuk. Vagy úgy, hogy meghúzzuk a körvonalukat, ezek sokszögek esetében egyenes szakaszokból, általában pedig görbékből állanak, vagy úgy, hogy a síkidom teljes belső tartományának megváltoztatjuk a színét, kitöltött síkidomot rajzolunk. Az első esetben használhatók az előző pontokban bemutatott módszerek. A Turbo Pascal grafikus egységének vannak olyan parancsai, amelyek bizonyos esetekben tovább egyszerűsítik a síkidomok határvonalas rajzolását. Telt sokszögek és bizonyos szabályos görbe oldalú síkidomok ábrázolásához is léteznek célirányos utasítások, bonyolultabb görbe oldalú síkidomokat pedig beírt sokszögekkel közelítünk.

Interaktív program írásához, amely lehetővé teszi a síkidomok mozgatását a képernyőn, szükség van az egybevágósági (alaktartó) transzformációk képleteire. Ezek az eltolás, a forgatás és a tükrözés. A koordinátatengelyekkel

(a képernyő széleivel) párhuzamos, δ hosszúságú eltolás képletei

$$\begin{cases} x' = x + \delta \\ y' = y \end{cases} \tag{1.17}$$

amely az (x,y) pontot jobbra (negatív δ esetén balra) mozdítja el és

$$\begin{cases} x' = x \\ y' = y + \delta \end{cases} \tag{1.18}$$

amely az (x,y) pontot felfelé (negatív δ esetén lefelé) viszi. Ha egyenesen képernyőkoordinátákban dolgozunk, akkor a második képletben az előjelt meg kell változtatni, mert a képernyő-koordinátarendszer fordított irányítású. Különben nem tanácsos képernyőkoordinátákban végezni a transzformációkat, mert, a kerekítés miatt, sorozatos forgatások az alakzat deformálódásához vezetnek.

Adott (x_0,y_0) pont körüli, α szögű forgatás képlete:

$$\begin{cases} x' = x_0 + (x - x_0)\cos\alpha - (y - y_0)\sin\alpha \\ y' = y_0 + (x - x_0)\sin\alpha + (y - y_0)\cos\alpha \end{cases} \tag{1.19}$$

A fenti képlet pozitív α esetén balra (pozitív trigonometriai irányba), negatív α esetén pedig jobbra forgat.

Adott,

$$ax + by + c = 0 \tag{1.20}$$

egyenletű egyeneshez viszonyított tükrözés képlete:

$$\begin{cases} x' = \dfrac{(b^2 - a^2)x - 2aby - 2ac}{a^2 + b^2} \\ y' = \dfrac{(a^2 - b^2)y - 2abx - 2bc}{a^2 + b^2} \end{cases} \tag{1.21}$$

gyakorlati célokra azonban elegendő például az $x = x_0$ függőlegeshez viszonyított tükrözés egyszerűbb

$$\begin{cases} x' = 2x_0 - x \\ y' = y \end{cases} \tag{1.22}$$

képletét használni, mivel a tükrözés után úgyis a kívánt helyzetbe vihetjük az alakzatot forgatások és eltolások sorozatával.

1.3.1. Sokszögek

A Turbo Pascal igen célravezető utasításokat tartalmaz sokszögek ábrázolásának mindkét (határvonalas vagy kitöltött) formájára. Ezek a ***drawpoly*** illetve a ***fillpoly*** nevű parancsok. Mindkettő esetében meg kell adni a számbajövő csúcsok számát és ezek képernyőkoordinátáit pont típusú rekordokból álló sormátrixba szervezve. A ***drawpoly*** utasítás nyílt törtvonalak rajzolására is alkalmas. Zárt törtvonalat (sokszöget) akkor rajzol, ha a megadott első csúcs azonos az utolsóval. Ez azt jelenti, hogy sokszögek esetén a megadott csúcsok száma mindig eggyel nagyobb, mint az ábrázolt sokszög csúcsainak száma.

Érdemes a sokszögeket objektumként definiálni, ugyanis az objektum-orientált programozás [6] keretében a sokszögek egy egységként, igen kényelmesen kezelhetők. Ez főként interaktív rajzolóprogram írásakor nyilvánvaló, amint azt az 1.3.3. alpontban példával szemléltetjük.

1.3.2. Görbe oldalú síkidomok

A Turbo Pascal grafikus egysége tartalmaz görbe oldalú síkidomokat megjelenítő utasításokat is, mint például a ***fillellipse***, amely a képernyő széleivel párhuzamos tengelyű, telt ellipszist rajzol vagy a ***pieslice***, amely telt körcikk ábrázolására alkalmas. Általános görbe oldalú síkidomok rajzolásához azonban külön programot kell írni. Úgy járhatunk el, hogy a síkidom körvonalát képező görbét vagy görbedarabokat törtvonallal közelítjük. Ezáltal a görbe oldalú síkidomba egy sokszöget írunk, amelyet, tetszés szerint, körvo-

nalas vagy telt ábrázolással megjeleníthetünk. A következő alpontban az itt vázolt módszert példával szemléltetjük.

1.3.3. Példák

a) Az alábbi program sokszög rajzolására és a képernyőn való tetszőleges mozgatására alkalmas (a grafikus meghajtó elindítását végző utasításokkal kiegészítve).

```
const
  fel = #72;  le = #80;  bal = #75;  jobb = #77;
  a=40; delt=5; alf=pi/40;

type
  csucs = object              { a sokszög csúcsainak valós    }
    x, y: real;               { koordinátáit tároló objektum }
    procedure uj(ux, uy: real); end;

  pont = object               { pont típusú objektum, a sokszög }
    xp, yp: integer;          { csúcsai képernyő-koordinátáinak }
    procedure uj(up: csucs); end;    { tárolására szolgál    }

  soksz = object
    ncs: byte;          { a sokszög csúcsainak száma    }
    kp: csucs;          { kezdeti helyzet meghatározása }
    cs: array[1..20] of csucs;
    csp: array[1..20] of pont;
    procedure uj(ukp: csucs);   { a sokszög alakját       }
                                { meghatározó eljárás     }
    procedure eltxp;
    procedure eltxm;
    procedure eltyp;            { alaktartó transzformációk    }
    procedure eltym;            {  a sokszög mozgatására       }
    procedure forgj;
    procedure forgb;
    procedure tukr;
    procedure rz;               { a megjelenítő eljárás        }
    end;

var
  ca, sa: real;
  kr: char;
  kt: csucs;
  lp: soksz;
  l: integer;
```

```
procedure csucs.uj(ux, uy: real);
begin x:=ux; y:=uy; end;

procedure pont.uj(up: csucs);{ a csúcsok valós koordinátáiból }
                      { kiszámítja ezek képernyő-koordinátáit   }
begin xp:=round(up.x); yp:=round(up.y); end;

procedure soksz.eltxp;          { jobbra tol }
begin for l:=1 to ncs do cs[l].uj(cs[l].x+delt,cs[l].y); end;

procedure soksz.eltxm;          { balra tol }
begin for l:=1 to ncs do cs[l].uj(cs[l].x-delt,cs[l].y); end;

procedure soksz.eltyp;          { felfelé tol }
begin for l:=1 to ncs do cs[l].uj(cs[l].x,cs[l].y+delt); end;

procedure soksz.eltym;          { lefelé tol }
begin for l:=1 to ncs do cs[l].uj(cs[l].x,cs[l].y-delt); end;

procedure soksz.forgj;      { jobbra forgat a sokszög       }
                            { első csúcsa körül        }
begin
  for l:=2 to ncs do
    cs[l].uj(cs[1].x+(cs[l].x-cs[1].x)*ca-(cs[l].y-cs[1].y)*sa,
             cs[1].y+(cs[l].x-cs[1].x)*sa+(cs[l].y-cs[1].y)*ca);
end;


procedure soksz.forgb;      { balra forgat a sokszög }
                            { első csúcsa körül      }
begin
  for l:=2 to ncs do
    cs[l].uj(cs[1].x+(cs[l].x-cs[1].x)*ca+(cs[l].y-cs[1].y)*sa,
             cs[1].y-(cs[l].x-cs[1].x)*sa+(cs[l].y-cs[1].y)*ca);
end;

procedure soksz.tukr;       { a sokszög első csúcsán átmenő      }
  begin                     { függőlegeshez viszonyítva tükröz   }
  for l:=2 to ncs do
    cs[l].uj(2*cs[1].x-cs[l].x, cs[l].y);
end;

procedure soksz.rz;         { megjelenítő eljárás }
begin
  for l:=1 to ncs do csp[l].uj(cs[l]);
  setfillstyle(1,1); fillpoly(ncs,csp); end;
```

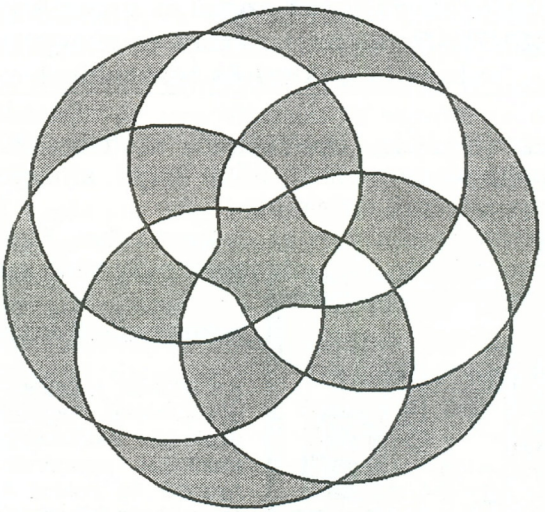
```
    procedure soksz.uj(ukp: csucs);
                          { adott kezdőponthoz viszonyítva     }
    begin                 { meghatározza a sokszög alakját     }
      kp:=ukp; ncs:=10;   { megadva csúcsainak koordinátáit    }
      cs[1]:=kp; cs[2].uj(kp.x,kp.y-a); cs[3].uj(kp.x+a,kp.y-a);
      cs[4].uj(kp.x+a,kp.y+a); cs[5].uj(kp.x+2*a,kp.y+a);
      cs[6].uj(kp.x+2*a,kp.y+2*a); cs[7].uj(kp.x,kp.y+2*a);
      cs[8].uj(kp.x,kp.y+a); cs[9].uj(kp.x-a,kp.y+a);
      cs[10].uj(kp.x-a,kp.y);
    end;              { ezt az eljárást módosítva tetszőleges alakú }
                      { sokszöget rajzolhatunk                      }
begin
  ca:=cos(alf); sa:=sin(alf);
  kt.uj(320, 240); lp.uj(kt); lp.rz;
  repeat begin              { mozgatási parancsok elfogadása }
   repeat kr:=upcase(readkey);
   until (kr='T') or (kr='J') or (kr='B') or
      (kr=fel) or (kr=le) or (kr=jobb) or (kr=bal) or (kr='Q');
    setfillstyle(1,0); bar(0,0,640,480);
    case kr of
    fel: lp.eltym;
    le: lp.eltyp;
    jobb: lp.eltxp;
    bal: lp.eltxm;
    'J': lp.forgj;
    'B': lp.forgb;
    'T': lp.tukr;
    end;
    if kr <> 'Q' then lp.rz;
  end; until kr='Q';       { kilépési feltétel }
end.
```

b) Az *1.6. ábrán* látható alakzat éleit képező folytonos, zárt görbe paraméteres egyenlete:

$$\begin{cases} x = (3+2\sin t)\sin\dfrac{5t}{6} \\ y = (3+2\sin t)\cos\dfrac{5t}{6} \end{cases}; \qquad t \in [0,12\pi]$$

A görbét apró ívekre osztva és az íveket a megfelelő húrokkal helyettesítve megkapjuk az alakzat sokszöges közelítését. Minden egyes tartományt külön síkidomnak tekintünk és kiválasztjuk a görbe osztópontjai közül azokat, amelyek az illető idom körvonalán találhatók. Ezeket megfelelő sorrendbe helyezve megkapjuk az idomot közelítő sokszöget, amelyet kívánt színűre rajzolhatunk.

Az ábra a lemezmellékleten található ROZSA.PAS program fehér-fekete, statikus változatával készült.

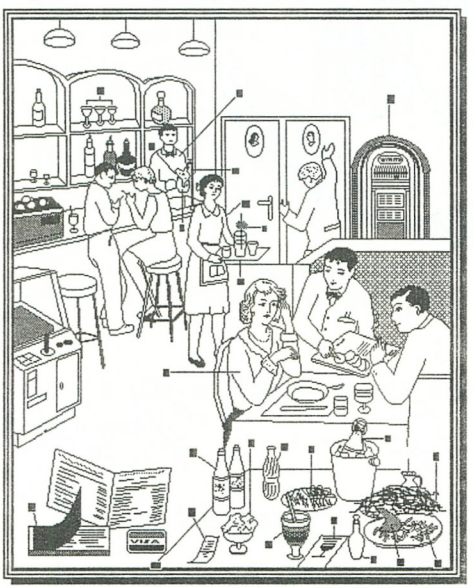


1.6. ábra

PICDIC MULTIMÉDIA nyelvoktató szoftverek

***ANGOL-MAGYAR* változat angol és amerikai hanggal,**
***NÉMET-MAGYAR* változat német és magyar hanggal,**
színes képekkel és tesztekkel.

A **PICDIC** számítógépes nyelvoktató szoftver 200 aprólékosan kidolgozott számítógépes grafikát, 5000 angol, amerikai illetve német, valamint 5000 magyar szót tartalmaz.
A képek témakörök szerint vannak csoportosítva. Ezekben találhatjuk meg a mindennapi életben leggyakrabban előforduló szókincset.



Kiknek ajánljuk szoftverünket ?

Kezdőknek, haladóknak, felnőtteknek és gyerekeknek egyaránt, vagyis mindazoknak, akik a tanulás izzadságos munkáját szeretnék sokkal hatékonyabbá és sokkal könnyebbé tenni a számítógép felhasználásával.

Ajánljuk azoknak, akik igazán szeretnék tudni, hogy mit is jelent egy szó.

És azoknak, akik szeretik a szavakat, és minél több szóval szeretnék bővíteni aktív szókincsüket.

De azoknak is, akik szeretik, ha a szavak megelevenednek, hozzájuk képek és hangok kapcsolódnak.

Érdeklődni és megrendelni az alábbi címeken lehet:

PROFI-SZOFT BT.
6500 Baja, Kölcsey u.112.
TEL/FAX: 79-325983
Borsódi Donát

Mándli János
6500 Baja, Szarka u.20.

COMPUTERBOOKS
1126 Budapest Tartsay V. u. 12.
TEL: 1751 564, 1753 591

A fenti szoftvereknek **DOS**-os és **Windows**-os verziója is megrendelhető. Az olasz és francia változatok Windows alatt futnak.

2. Térbeli alakzatok ábrázolása

A valóságos térbeli alakzatok, a tárgyak három kiterjedésűek. A mértan és számos természettudomány illetve műszaki tudományág nulla (pontok), egy (görbék) illetve két (felületek) kiterjedésű alakzatokkal is foglalkozik, amelyek matematikai absztrakciók, de igen alkalmas és szemléletes modelljei a valós tárgyaknak, jelenségeknek. A továbbiakban ezért mind a négy kategória ábrázolásáról lesz szó. A térbeli alakzatok képét egy sajátságos művelet, a vetítés segítségével kaphatjuk meg. Ebben a fejezetben a merőleges vetítéssel foglalkozunk. Ez ugyan eltér a természetes képformálási folyamattól, ami az emberi szemben megy végbe, azonban a kapott kép megőrzi az alakzat arányait. Ezenkívül kis méretű, vagy a szemlélőtől viszonylag távol levő alakzatok esetén (a fényképészet ilyen esetekben kis látószögről beszél) az említett eltérés elenyésző. Centrális, vagy középpontos vetítéssel kapott képek előnye, hogy a harmadik kiterjedésről (a szemlélőtől való távolság, "mélység") is hordoznak információt.

2.1. Merőleges vetítés

2.1.1. Tájékozódás a térben

Valamely tárgy helyzetét a térben úgy határozhatjuk meg, hogy egy koordinátarendszerhez viszonyítjuk, melynek tengelyei egymásra páronként merőlegesek. Nevezzük az xy síkot vízszintes síknak, a z tengelyt pedig függőleges tengelynek.

A térbeli (háromdimenziós) alakzatok, tárgyak kétdimenziós képét úgy kapjuk, hogy levetítjük őket egy síkra, amit képsíknak nevezünk. Ez elvileg úgy történik, hogy a tárgy felületének minden pontja esetén bizonyos törvényszerűség alapján meghatározzuk a megfelelő síkbeli pontot és eldöntjük, hogy ez látható vagy nem. Utóbbi eset úgy lehetséges, hogy a szóban forgó pontot a vetített alakzat valamely része eltakarja. Merőleges vetítés esetén a vetítési irány állandó és merőleges a képsíkra. A vetítő sugarak párhuzamos sugárnyalábot alkotnak. A kapott kép azonban elforgatható a kép középpontja (számítógépes ábrázolás esetén ez a képernyő középpontját jelenti) körül. Tájékozódás alatt a kép függőleges tengelyének a

meghatározását értjük. Úgy is mondhatjuk, hogy meghatározzuk a megfigyelő és a tárgy kölcsönös helyzetét a térben. Erre több lehetőség van, a továbbiakban ezek közül kettőről lesz szó.

Az elsőt nevezhetjük természetes orientációnak is, ugyanis megfelel a gravitációs térben élő ember ösztönös igyekezetének, hogy függőlegesen álljon és mindent ehhez az irányhoz viszonyítson. Ebben az esetben a függőleges irányt megtartjuk, azaz a kép függőleges tengelye a térbeli függőleges (z tengely) vetülete lesz. Ez a módszer alkalmazható felületek, domborzat, épületek ábrázolásakor. A forgatás úgy történik, hogy a megfigyelési (vetítési) irányt (és ennek megfelelően a képsíkot) módosítjuk, a tárgy helyzete a térben változatlan marad.

A második lehetőség a "kézbe fogható" tárgyak esete, amikor az orientáció közömbös. Ilyenkor célszerű felülnézetet készíteni, ugyanis ez a legegyszerűbb, és a tárgyat forgatni el a megfelelő irányba.

Ezenkívül még más, az illető alkalmazástól függő módozatok is megszabhatók a függőleges irány meghatározására, például műholdak esetén a keringési pálya érintője, vagy a keringési pálya síkjára emelt merőleges, bonyolult pályán mozgó repülőgép pilótája által látott kép szerkesztése esetén pedig a pilótaülés függőleges tengelye.

2.1.2. Merőleges vetítés a függőleges megtartásával

Ennél a vetítési eljárásnál a vetület függőleges tengelye a térbeli függőleges tengely (z) vetülete. Az eljárást a *2.1. ábra* szemlélteti. Legyen a vetítési irány a következő egyenletű egyenes:

$$\frac{x}{a}=\frac{y}{b}=\frac{z}{c} \qquad (d) \qquad (2.1)$$

ahol $a^2+b^2\neq 0$. Ellenkező esetben függőleges vetületet kapunk, vagyis a z tengely vetülete egy pont lesz, ezért nem alkalmazható a fentebb említett orientációs elv. Az $a^2 + b^2 \neq 0$, $c = 0$ eset pedig az xy síkkal párhuzamos vetítősugarat jelent.

A megfelelő képsík, amely a térbeli koordinátarendszer O kezdőpontját tartalmazza:

$$ax + by + cz = 0 \qquad (\pi) \qquad ; \qquad d \perp \pi \tag{2.2}$$

Ez azt jelenti, hogy a tárgyat a d egyenessel párhuzamos irányból nézzük. A vetítési irány egységvektora: $-e_a\vec{i} - e_b\vec{j} - e_c\vec{k}$
ahol:

$$e_a = \frac{a}{\sqrt{a^2+b^2+c^2}} \quad ; \quad e_b = \frac{b}{\sqrt{a^2+b^2+c^2}} \quad ; \quad e_c = \frac{c}{\sqrt{a^2+b^2+c^2}} \; . \tag{2.3}$$

A d egyenes és a z tengely által meghatározott sík egyenlete:

$$\begin{vmatrix} x & y & z \\ 0 & 0 & 1 \\ a & b & c \end{vmatrix} = 0 \qquad \text{azaz} \qquad -bx + ay = 0 \qquad (\sigma) \tag{2.4}$$

Ezt a síkot metszve a képsíkkal, a vetület függőleges tengelyét kapjuk:

$$\frac{x}{-ac} = \frac{y}{-bc} = \frac{z}{a^2+b^2} \qquad (O\eta) \tag{2.5}$$

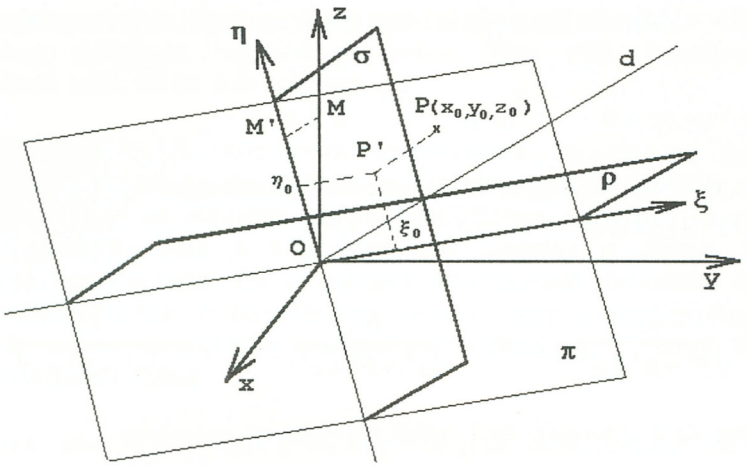
A kezdőponton átmenő, az η tengelyre merőleges sík:

$$-acx - bcy + (a^2+b^2)z = 0 \qquad (\rho) \tag{2.6}$$

Ezt metszve a képsíkkal megkapjuk a vetület vízszintes tengelyét:

$$\frac{x}{-b} = \frac{y}{a} \quad ; \quad z = 0 \qquad (O\xi) \tag{2.7}$$

Amint az a szerkesztésből kitűnik és az ábrán is látható, a π, σ és ρ síkok egymásra páronként merőlegesek, metszeteik pedig a vetítési irány és a vetület tengelyei.



2.1. ábra

Egy tetszőleges térbeli pont, $P(x_0, y_0, z_0)$ vetületének koordinátáit a $\xi\eta$ rendszerben úgy határozhatjuk meg a legkönnyebben, hogy kiszámítjuk a pont σ illetve ρ síkoktól való távolságát, minthogy a ξ és az η tengelyek a d vetítési irányra egyaránt merőlegesek:

$$\begin{cases} \xi_0 = d(P,\sigma) = \dfrac{-bx_0 + ay_0}{\sqrt{(a^2+b^2)}} \\ \eta_0 = d(P,\rho) = \dfrac{-c(ax_0+by_0)+(a^2+b^2)z_0}{\sqrt{(a^2+b^2)(a^2+b^2+c^2)}} \end{cases} \tag{2.8}$$

A pont távolsága a képsíktól a

$$d(P,\pi) = \frac{ax_0 + by_0 + cz_0}{\sqrt{a^2+b^2+c^2}} \tag{2.9}$$

képlettel adható meg. Ennek nincs ugyan jelentősége a vetület helyzetének meghatározásában, azonban annak megállapításához szükséges, hogy az illető pont az adott vetítési irány esetén látható-e vagy nem (azaz annak

megállapításához, hogy az ábrázolt tárgy vagy felület valamely részlete az illető pontot eltakarja-e vagy nem).

2.1.2.1. Felülnézet

Amint már szó volt róla, a térben szabadon elforgatható tárgyakról célszerű felülnézetet készíteni, vagyis az xy koordinátasíkra vetíteni, ugyanis ez a legegyszerűbb és ezért leggyorsabb vetítési mód. A tárgyat valamely szabadon megválasztható (lehetőleg - de nem feltétlenül - a kezdőponton átmenő) tengely körül elforgatva, azt bármely oldaláról megszemlélhetjük. A felülnézet vetítési egységvektora a függőleges tengely egységvektorának ellentettje, azaz $-\bar{k}$. A tárgyat a z tengely irányából nézzük. A kép koordináta tengelyei megegyeznek a térbeli xy sík tengelyeivel, azaz a $P(x_0, y_0, z_0)$ pont vetületének koordinátái a $\xi\eta$ rendszerben a következők lesznek:

$$\begin{cases} \xi_0 = x_0 \\ \eta_0 = y_0 \end{cases} \tag{2.10}$$

A pont távolsága a vetítési síktól megegyezik a pont z_0 koordinátájával. A poliéderek rajzolásáról szóló fejezetben látni fogjuk, hogy ez az adat hogyan használható a poliéder oldallapjai rajzolási sorrendjének megállapításához.

2.2. A megfigyelés tárgyának mozgatása

A mozgástan kimutatja, hogy bármely mozgás felbontható párhuzamos eltolások és forgatások sorozatára. Ezért a továbbiakban ezekről a sajátos mozgásokról lesz szó, természetesen az illető vetítési módok figyelembe vételével. Mozgatást megengedő program úgy írható, hogy minden elemi mozgástípushoz hozzárendelünk egy billentyűt és akkor a felhasználó megfelelő parancsok segítségével a kívánt módon mozgathatja a tárgyat látóterében illetve mozoghat ehhez viszonyítva.

2.2.1. Párhuzamos eltolás

Gyakorlatilag akkor van jelentősége, amikor az ábrázolt objektum kiterjedése nagyobb, mint ami az adott lépték mellett a képernyőbe befér és valamilyen okból kifolyólag nem akarjuk a léptéket csökkenteni. Ilyenkor legjobb a tárgyat tolni el mindkét orientáció esetén. Itt lépték alatt az egységnyi hosszúságúnak választott szakasz rajzi hosszát értjük (a képernyőn). A léptéket lehetőleg úgy kell megválasztani, hogy az ábrázolandó alakzat minden részlete beleférjen a képernyőbe. A képernyő felbontóképessége azonban véges, ezért előfordulhat, hogy nagyobb kiterjedésű alakzat esetén csak a kép minőségének a rovására tudnánk a léptéket kellő mértékben lecsökkenteni.

Ha felülnézettel dolgozunk, akkor a megfigyelési "ablak" $a_t\vec{i} + b_t\vec{j}$ irányba való δ hosszúságú eltolásának képlete:

$$\begin{cases} x = x_0 - a_t \dfrac{\delta}{\sqrt{a_t^2 + b_t^2}} \\ y = y_0 - b_t \dfrac{\delta}{\sqrt{a_t^2 + b_t^2}} \\ z = z_0 \end{cases} \tag{2.11}$$

vagyis a tárgyat az ellenkező irányba toljuk el az *xy* síkkal párhuzamosan.

Tetszőleges vetítési irány mellett célszerű a tárgyat ugyancsak az *xy* síkkal párhuzamosan mozgatni, ugyanis ezáltal nem módosítjuk a tárgyhoz viszonyított megfigyelési irányt. Ebben az esetben is megszabhatunk tetszőleges eltolási irányokat, ezek között azonban van kettő különleges, éspedig a vetítési irány merőleges vetülete az *xy* síkra, illetve az erre merőleges irány.

Az első esetben az eltolás képlete:

$$\begin{cases} x = x_0 + a\dfrac{\delta}{\sqrt{a^2+b^2}} \\ y = y_0 + b\dfrac{\delta}{\sqrt{a^2+b^2}} \\ z = z_0 \end{cases} \tag{2.12}$$

Ugyanazt az eredményt (az előbbivel egybevágó vetületet) kapjuk ha a tárgyat a függőleges tengely (z) mentén mozgatjuk a következő képlet szerint:

$$\begin{cases} x = x_0 \\ y = y_0 \\ z = z_0 - c\dfrac{\delta}{\sqrt{a^2+b^2}} \end{cases} \tag{2.13}$$

amint az a vetítési képletekkel ellenőrizhető. Pozitív δ esetén a megfigyelő a tárgy fölött "előre" fog elmozdulni, negatív δ esetén pedig "hátra".

A második, vagyis a d vetítési irány xy síkra való merőleges vetületére merőleges, eltolási irány képlete az eredetileg az O ponton áthaladó d vetítési irány jobb fele való elmozdításához (a tárgy képe balra fog elmozdulni) a következő képlet használható:

$$\begin{cases} x = x_0 + b\dfrac{\delta}{\sqrt{a^2+b^2}} \\ y = y_0 - a\dfrac{\delta}{\sqrt{a^2+b^2}} \\ z = z_0 \end{cases} \tag{2.14}$$

Ha δ negatív, az elmozdulás az ellenkező irányba történik. Interaktív program írásakor ez azt jelenti, hogy a nyíl billentyűk leütésekor a megfigyelési irány az O kezdőponthoz viszonyítva a nyíl irányában fog δ lépéssel elmozdulni, ezáltal a felhasználó a felület fölött bármely irányba haladhat.

2.2.2. *Forgatás a megfigyelési irány mozgatásával*

A tetszőleges irányú vetítés esetén, amikor a kép függőleges iránya jól meghatározott, a forgatást érdemes úgy végezni, hogy tulajdonképpen a megfigyelési irányt mozgatjuk ellenkező irányba és a tárgy az eredeti helyén marad, vagyis megkerüljük a tárgyat, hogy megnézzük a másik oldalát, nem pedig elforgatjuk. Ezt a műveletet a *2.2. ábra* szemlélteti. A szaggatott vonallal rajzolt körök sugarai:

$$r = \sqrt{a^2 + b^2} \quad ; \quad R = \sqrt{a^2 + b^2 + c^2} \tag{2.15}$$

az ábrán feltüntetett szögek szögfüggvényei pedig:

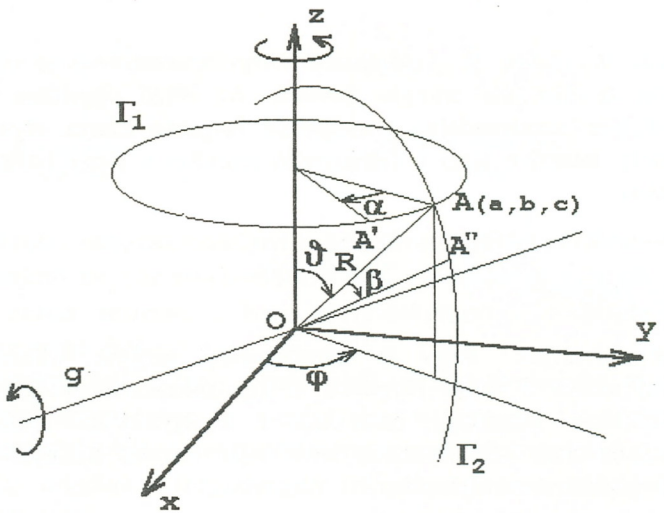
$$\begin{cases} \cos\vartheta = \dfrac{c}{R} \\ \sin\vartheta = \dfrac{r}{R} \end{cases} \quad ; \quad \begin{cases} \cos\varphi = \dfrac{a}{r} \\ \sin\varphi = \dfrac{b}{r} \end{cases} \tag{2.16}$$

A forgatás a z tengely körül a nyíl irányában elvégezhető a megfigyelési irányvektorral ellentétes $\vec{OA}$ vektor A végpontjának a Γ_1 körön az α szög növekedésének irányában való elmozdításával. A csillagászatban használatos fogalmazásban ezt azimutmódosításnak is nevezhetjük. A Γ_1 kör paraméteres egyenlete:

$$\begin{cases} x = r\cos(\varphi - \alpha) \\ y = r\sin(\varphi - \alpha) \\ z = c \end{cases} \tag{2.17}$$

Ennek segítségével felírhatók az új vetítési iránynak megfelelő A' pont koordinátái:

$$\begin{cases} a' = a\cos\alpha + b\sin\alpha \\ b' = -a\sin\alpha + b\cos\alpha \\ c' = c \end{cases} \tag{2.18}$$



2.2. ábra

A forgatás a σ síkra merőleges g (nutációs) tengely körül a nyíl irányában megegyezik a megfigyelési iránnyal ellentétes $\vec{OA}$ vektor A végpontjának elmozdításával a Γ_2 körön a β szög növekedésének irányában. Ugyancsak csillagászati kifejezéssel élve, azt mondhatjuk, hogy csökkentjük a megfigyelési irány "magasságát". A Γ_2 kör paraméteres egyenlete:

$$\begin{cases} x = R\sin(\vartheta + \beta)\cos\varphi \\ y = R\sin(\vartheta + \beta)\sin\varphi \\ z = R\cos(\vartheta + \beta) \end{cases} \tag{2.19}$$

Innen adódnak az új megfigyelési iránynak megfelelő A'' pont koordinátái:

$$\begin{cases} a'' = a\cos\beta + \dfrac{ac}{r}\sin\beta \\ b'' = b\cos\beta + \dfrac{bc}{r}\sin\beta \\ c'' = c\cos\beta - r\sin\beta \end{cases} \tag{2.20}$$

Természetesen, α illetve β előjelének megváltoztatásával az elmozdulás az illető körökön az ellenkező irányba történik. Az előző alponthoz hasonlóan, a nyíl billentyűkhöz hozzárendelve a megfelelő forgatási irányt, olyan programot írhatunk, amely lehetővé teszi a felhasználó számára a tárgy bármely irányból való szemlélését.

2.2.3. Tárgy forgatása

Amikor azt kívánjuk, hogy a megfigyelés tárgyát bármely oldalról szemlélhessük, akkor a tárgy forgatása a legalkalmasabb megoldás. Ilyenkor felülnézetet szokás készíteni, mert ez a legegyszerűbb vetítési eljárás. Mindazonáltal ez nem törvényszerű, lehet más vetítési iránnyal is dolgozni, csak ez több gépidőt vesz igénybe.

Először az általános esettel foglalkozunk, vagyis egy tetszőleges tengely körüli forgatás képletét határozzuk meg, azután pedig ebből a koordinátatengelyek körüli forgatások sajátos eseteit vezetjük le. A mozgástanban jól ismert jelölések mellett, a műveletetekben szereplő szögek és tengelyek értelmezését a *2.3. ábrán* láthatjuk. A g (nutációs) tengely az XY és $\xi\eta$ síkok metszés vonala.

A tárgyat a következő tengely körül kívánjuk forgatni:

$$\frac{x}{k} = \frac{y}{l} = \frac{z}{m} \qquad (f) \tag{2.21}$$

Az ábrán az xyz rendszer áll, a $\xi\eta\zeta$ pedig forog a ζ tengely körül, amely egybeesik az f forgástengellyel. Az Euler-féle szögek értelmezését ebben az

esetben a (2.22) egyenletek adják meg. Az illető szögek elnevezései és változási határai:

$0 \le \vartheta \le \pi$ - nutációsszög
$0 \le \psi \le 2\pi$ - precessziószög
ω - tiszta forgásszög

Az Euler-szögekről bővebben lásd [5], 185-188 old.

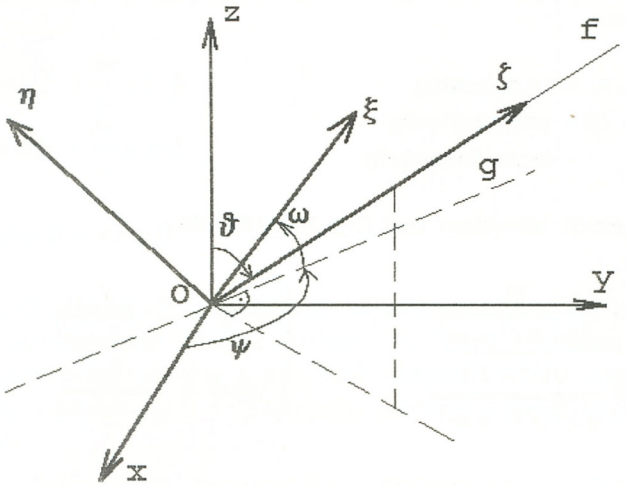
$$\begin{cases} \cos\vartheta = \dfrac{m}{\sqrt{k^2+l^2+m^2}} \\ \sin\vartheta = \dfrac{\sqrt{k^2+l^2}}{\sqrt{k^2+l^2+m^2}} \end{cases} ; \quad \begin{cases} \cos\psi = -\dfrac{l}{\sqrt{k^2+l^2}} \\ \sin\psi = \dfrac{k}{\sqrt{k^2+l^2}} \end{cases} \qquad (2.22)$$

A forgatási képlet meghatározásához a következő eljárást alkalmazzuk:
- transzformáció az *xyz* rendszerből a $\xi\eta\zeta$ rendszerbe $\omega = -\alpha$ mellett. Itt a $-$ jel onnan származik, hogy tulajdonképpen a megfigyelési irányból nézünk az origó felé.
- visszatranszformáció a $\xi\eta\zeta$ rendszerből az *xyz* rendszerbe $\omega = 0$ mellett. Ennek eredményeképpen a megfigyelés tárgya az f tengely körül a megfigyelési irányból nézve pozitív trigonometriai irányba fog elfordulni α szöggel (az origóból a forgástengely irányába nézve az elfordulás ellentétes irányba történik).
- az Euler-féle szögek értelmezését a kapott képletbe behelyettesítve megkapjuk a program írásához szükséges képletet.

Az *xyz* és $\xi\eta\zeta$ rendszerek közötti transzformáció a következő táblázat segítségével végezhető el:

	ξ	η	ζ
x	$\cos\omega\cos\psi - \sin\omega\sin\psi\cos\vartheta$	$-\sin\omega\cos\psi - \cos\omega\sin\psi\cos\vartheta$	$\sin\psi\sin\vartheta$
y	$\cos\omega\sin\psi + \sin\omega\cos\psi\cos\vartheta$	$-\sin\omega\sin\psi + \cos\omega\cos\psi\cos\vartheta$	$-\cos\psi\sin\vartheta$
z	$\sin\omega\sin\vartheta$	$\cos\omega\sin\vartheta$	$\cos\vartheta$

(2.23)



2.3. ábra

A transzformáció az *xyz* rendszerből a $\xi\eta\zeta$ rendszerbe $\omega = -\alpha$ mellett a következő képlet szerint történik:

$$\begin{cases} \xi = x(\cos\alpha\cos\psi + \sin\alpha\sin\psi\cos\vartheta) + y(\cos\alpha\sin\psi - \sin\alpha\cos\psi\cos\vartheta) - \\ \qquad - z\sin\alpha\sin\vartheta \\ \eta = x(\sin\alpha\cos\psi - \cos\alpha\sin\psi\cos\vartheta) + y(\sin\alpha\sin\psi + \cos\alpha\cos\psi\cos\vartheta) + \\ \qquad + z\cos\alpha\sin\vartheta \\ \zeta = x\sin\psi\sin\vartheta - y\cos\psi\sin\vartheta + z\cos\vartheta \end{cases} \tag{2.24}$$

A visszatranszformáció pedig a $\xi\eta\zeta$ rendszerből az *xyz* rendszerbe $\omega = 0$ mellett:

$$\begin{cases} x' = \xi\cos\psi - \eta\sin\psi\cos\vartheta + \zeta\sin\psi\sin\vartheta \\ y' = \xi\sin\psi + \eta\cos\psi\cos\vartheta - \zeta\cos\psi\sin\vartheta \\ z' = \eta\sin\vartheta + \zeta\cos\vartheta \end{cases} \tag{2.25}$$

A két transzformáció együttes felírása:

$$\begin{cases} x' = x[\cos\alpha + (1-\cos\alpha)\sin^2\psi\sin^2\vartheta] + y[-\sin\alpha\cos\vartheta - (1-\cos\alpha)\cdot \\ \quad \cdot \sin\psi\cos\psi\sin^2\vartheta] + z\sin\vartheta[-\sin\alpha\cos\psi + (1-\cos\alpha)\sin\psi\cos\vartheta] \\ y' = x[\sin\alpha\cos\vartheta - (1-\cos\alpha)\sin\psi\cos\psi\sin^2\vartheta] + y[\cos\alpha + (1-\cos\alpha)\cdot \\ \quad \cdot \cos^2\psi\sin^2\vartheta] + z\sin\vartheta[-\sin\alpha\sin\psi - (1-\cos\alpha)\cos\psi\cos\vartheta] \\ z' = x\sin\vartheta[\sin\alpha\cos\psi + (1-\cos\alpha)\sin\psi\cos\vartheta] + y\sin\vartheta[\sin\alpha\sin\psi - \\ \quad -(1-\cos\alpha)\cos\psi\cos\vartheta] + z[1-(1-\cos\alpha)\sin^2\vartheta] \end{cases} \tag{2.26}$$

Az Euler-féle szögek képleteinek figyelembe vételével a következő forgatási képletet kapjuk:

$$\begin{cases} x' = x\dfrac{k^2+(l^2+m^2)\cos\alpha}{k^2+l^2+m^2} + y\dfrac{-m\sqrt{k^2+l^2+m^2}\sin\alpha + kl(1-\cos\alpha)}{k^2+l^2+m^2} + \\ \quad + z\dfrac{l\sqrt{k^2+l^2+m^2}\sin\alpha + km(1-\cos\alpha)}{k^2+l^2+m^2} \\ y' = x\dfrac{m\sqrt{k^2+l^2+m^2}\sin\alpha + kl(1-\cos\alpha)}{k^2+l^2+m^2} + y\dfrac{l^2+(k^2+m^2)\cos\alpha}{k^2+l^2+m^2} + \\ \quad + z\dfrac{-k\sqrt{k^2+l^2+m^2}\sin\alpha + lm(1-\cos\alpha)}{k^2+l^2+m^2} \\ z' = x\dfrac{-l\sqrt{k^2+l^2+m^2}\sin\alpha + km(1-\cos\alpha)}{k^2+l^2+m^2} + \\ \quad + y\dfrac{k\sqrt{k^2+l^2+m^2}\sin\alpha + lm(1-\cos\alpha)}{k^2+l^2+m^2} + z\dfrac{m^2+(k^2+l^2)\cos\alpha}{k^2+l^2+m^2} \end{cases} \tag{2.27}$$

Ez a képlet használandó a program írásakor, természetesen annyi módosítással, hogy a forgástengely irányítényezőinek különböző kombinálásából származó állandó együtthatókat csak egyszer számítjuk ki, a program elején, és valamilyen néven tároljuk, temérdek gépidőt takarítva meg. Ha például α is állandó (azaz a forgatás állandó elemi szöggel történik), akkor a fenti képlet minden együtthatója állandó lesz, így ezeket érdemes valamilyen néven

tárolni. Felülnézet alkalmazása mellett igen gyors program írható, amelyben az egyes megjelenítések közötti időt jobbára csak a rajzolás gyorsasága határozza meg.

A továbbiakban a forgatás sajátos eseteit tárgyaljuk éspedig a koordinátatengelyek körüli forgatásokat. Az x tengely körüli forgatás azt jelenti, hogy a forgástengely megegyezik az x tengellyel, azaz

$$\begin{cases} k=1 \\ l=m=0 \end{cases}, \quad \text{más szóval} \quad \begin{cases} \vartheta=\pi/2 \\ \psi=\pi/2 \end{cases}.$$

Az általános forgatási képletből a fentiek figyelembe vételével következik:

$$\begin{cases} x'=x \\ y'=y\cos\alpha - z\sin\alpha \\ z'=y\sin\alpha + z\cos\alpha \end{cases} \tag{2.28}$$

Az y tengely körüli forgatáshoz a forgástengely ezzel a koordinátatengellyel kell egybeessen, vagyis:

$$\begin{cases} l=1 \\ k=m=0 \end{cases}, \quad \text{az Euler-féle szögekkel kifejezve:} \quad \begin{cases} \vartheta=\pi/2 \\ \psi=0 \end{cases}.$$

A keresett képlet:

$$\begin{cases} x'=x\cos\alpha + z\sin\alpha \\ y'=y \\ z'=-x\sin\alpha + z\cos\alpha \end{cases} \tag{2.29}$$

A z tengely körüli forgatás esetében nem érvényes a ψ szög felírása a forgástengely irányítényezőinek segítségével, ugyanis ebben az esetben

$\begin{cases} k = l = 0 \\ m = 1 \end{cases}$, a keresett képlet az Euler-szögek értékeinek figyelembe vételével kapható meg, most ugyanis: $\begin{cases} \vartheta = 0 \\ \psi = 0 \end{cases}$, így:

$$\begin{cases} x' = x\cos\alpha - y\sin\alpha \\ y' = x\sin\alpha + y\cos\alpha \\ z' = z \end{cases} \tag{2.30}$$

2.3. Sokszögek ábrázolása

2.3.1. Irányított sokszögek

Ha a térben tetszőleges helyzetű síkbeli sokszöget ábrázolunk, természetesen tudni szeretnénk, hogy éppen melyik oldalát látjuk. Ehhez azonban meg kell határozni a sokszög irányítását, vagyis el kell dönteni, hogy melyik a sokszög "felső", illetve "alsó" oldala. Ezt könnyen megtehetjük, ha a sokszöghöz hozzárendelünk egy körbejárási irányt, vagyis a sokszög csúcsainak sorrendjét. Azt mondjuk, hogy az a sokszög felső oldala (vagy "színe"), amelyik mindig a bal kezünk felé esik, ha a sokszög élein a megszabott irányban végigsétálunk. A másik oldal a sokszög alsó oldala (vagy "fonákja"). Zárt soklapok (poliéderek) esetén az irányítások összerendelése fordított sorrendben történik, itt ugyanis a felső (szemléletesebb kifejezéssel "külső") oldal meghatározott, és ehhez rendeljük hozzá a megfelelő körbejárási irányt.

2.3.2. Láthatóság

Ebben az alfejezetben egy módszert mutatunk be annak meghatározására, hogy adott irányítás és megfigyelési irány mellett egy sokszögnek melyik oldalát látjuk. A poliéderek ábrázolásáról szóló fejezetben látni fogjuk, hogy van amikor (konvex poliéderek esetében) ennek ismerete elegendő az illető sokszög láthatóságának megállapításához.

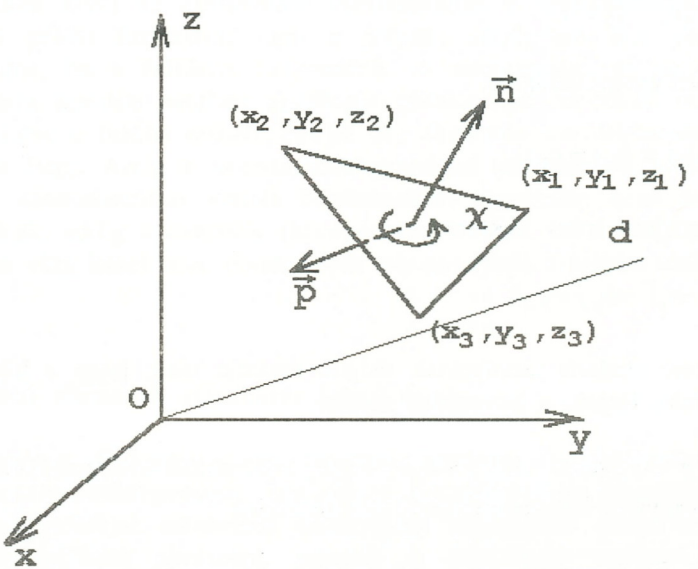
Minden sokszöghöz hozzárendelhetünk egy úgynevezett "normálisvektort", amely merőleges a sokszög síkjára és a sokszög felső oldala felől levő féltér irányába mutat (*2.4. ábra*). Könnyen belátható, hogy a normálisvektor maradéktalanul jellemzi a sokszög irányítását a térben. A normálisvektor és a körbejárási irány közötti összefüggés úgy fogalmazható meg, hogy ha egy jobbsodrású fúrót a körbejárási irányban hajtunk, akkor a fúró a normálisvektor irányába fog haladni.

Az ábrán látható háromszög csúcsai által meghatározott sík egyenlete:

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0 \qquad (\chi) \tag{2.31}$$

azaz:

$$a_h x + b_h y + c_h z + d_h = 0 \tag{2.32}$$



2.4. ábra

ahol:

$$a_h = \begin{vmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{vmatrix} = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$b_h = -\begin{vmatrix} x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \\ x_3 & z_3 & 1 \end{vmatrix} = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2) \tag{2.33}$$

$$c_h = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

A megfelelő normálisvektor:

$$\vec{n} = a_h\vec{i} + b_h\vec{j} + c_h\vec{k} \tag{2.34}$$

Legyen a vetítési irány vektora:

$$\vec{p} = -a_p\vec{i} - b_p\vec{j} - c_p\vec{k} \tag{2.35}$$

A két vektor skaláris szorzatának előjele mutatja meg, hogy a háromszögnek melyik oldalát látjuk, a következő módon:

- ha $\vec{n} \cdot \vec{p} < 0$ azaz $a_h a_p + b_h b_p + c_h c_p > 0$, akkor a normálisvektor felénk mutat, vagyis a háromszög felső oldalát látjuk
- ha $\vec{n} \cdot \vec{p} > 0$ azaz $a_h a_p + b_h b_p + c_h c_p < 0$, akkor a háromszögnek az alsó, a normálisvektorral ellentétes oldalát látjuk

A módszer akkor is alkalmazható, ha a sokszögnek több csúcsa van, egyszerűen ki kell választani ezek közül hármat, természetesen szem előtt tartva a körbejárási irányt.

2.4. Kétváltozós függvények (domborzat jellegű felületek) ábrázolása

Egy sík tartomány képe egy kétváltozós függvényen keresztül egy térbeli felület (domborzat jellegű felület, vagyis az értelmezési tartomány bármely pontjának a felület egy és csakis egy pontja felel meg). A felületek ilyen felírását Euler-Monge-féle előállításnak is szokás nevezni [4]. A felületeket többféle módon ábrázolhatjuk. Talán a legszemléletesebb a képies képet nyújtó (axonometrikus vagy pedig centrális) ábrázolás, főként a számítógép nyújtotta lehetőségek mellett, amikor is a vetítési irány változtatásával bármely szögből szemlélhetjük az ábrázolt felületet. Ezenkívül azonban más eljárások is ismertek, gondoljunk csak a földrajzi térképek szintvonalas, színkódos ábrázolási módjára, amely tulajdonképpen felülnézet, a domborzat jellegét és a magasságot a szintvonalak jelenléte illetve az egyes sávok színe érzékelteti. A szintvonalakon kívül az úgynevezett esésvonalakat is berajzolhatjuk, amelyek, mechanikai példát használva, azok a pályák, amelyeken egy pici, könnyű labda gurulna, ha a felületre helyeznénk. A szintvonalak és az esésvonalak két, egymásra minden pontban merőleges görbesereget alkotnak, meglehetősen jól érzékeltetve a felület térbeli alakját. Az ábrázolás szemléletessége a vetítés módjától is függ. Amíg a koordinátatengelyekkel párhuzamos háló használata mellett az axonometrikus vetítés felülnézetben egyáltalán nem érzékelteti a felület alakját, addig a centrális (középpontos) vetítés, ha a megfigyelési pont a felülethez elég közel van, olyan képet ad, amelyből a felület alakja elég jól kivehető.

2.4.1. Térbeli ábrázolás síkidomos közelítéssel

A számítógépes ábrázolás eleve magával hordozza a diszkretizálás, elemi részekre bontás szükségességét. Így eléggé kézenfekvő az az eljárás, hogy a felületet megpróbáljuk síkidomok (sokszögek) segítségével közelíteni, majd az így kapott felületet ábrázolni, ugyanis a sokszögek vetületei ugyancsak sokszögek lesznek, ezeket pedig legtöbb programozási nyelvezetben kényelmesen tudjuk rajzolni. Ezt az eljárást felületek poliéder-modellezésének nevezik. A Turbo Pascal 6.0 vagy újabb változatokban a "***Fillpoly***" utasítással kívánt színű telt sokszöget rajzolhatunk, csak a csúcsok képernyő-koordinátáit kell megadnunk. Az elemi sokszögek megfelelő módon kiválasztott rajzolási sorrendje esetén a felületről helyes képet kapunk, vagyis a felületnek azokat

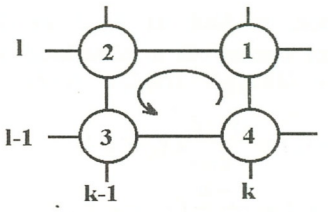
és csakis azokat a részeit fogjuk látni a képernyőn, amelyek az illető vetítési irány mellett valóban láthatók. A fedésben levő részeket az elejükbe rajzolt sokszögek el fogják takarni. A felületelemek irányításának megfelelő színt alkalmazva azt is érzékeltethetjük, hogy a felületdarab felső vagy alsó oldalát látjuk.

Legkényelmesebb az *Oxy* sík tengelyeivel párhuzamos, egyenlő felosztású négyszögű hálóval dolgozni, ami azt jelenti, hogy a felületet először az *Oxy* sík elemi négyzetei képeit alkotó görbe oldalú négyszögekre osztjuk, majd az ezekhez legközelebb álló síknégyszögek segítségével közelítjük. Ez a közelítés azonban csak elég finom háló esetén ad kielégítő eredményt, ugyanis az említett görbe oldalú négyszögek csúcsai általában nincsenek egy síkban. Minél apróbb szemű hálót használunk, annál kisebb lesz ez az eltérés és annál jobb képet kapunk. Pontosabb eredményt érünk el, ha a görbe oldalú négyszögeknek csak "kiegyenesítjük" az oldalait, vagyis az illető görbéket szakaszokkal helyettesítjük és az így kapott "torz" (nem egysíkú) négyszögeket vetítjük le. Ebben az esetben az egyetlen hibaforrás az marad, hogy nem lehet az illető felületelemek normálisvektorát egyértelműen meghatározni, és arra a kérdésre helyes választ adni, hogy a felületelemnek éppen melyik oldalát látjuk. Szerencsére ez a helyzet is csak a vetítési síkra majdnem merőleges felületelemek esetén áll fenn.

Háromszögű háló esetén ez a probléma nem áll fenn, ugyanis három pont mindig egy síkban van, a közelítés ebben az esetben csak annyiban áll, hogy a felületelemeket alkotó háromszögek görbe oldaléleit "kiegyenesítjük". Ezért van amikor háromszögű háló használata javallott, még ha kissé bonyolultabb is programozni.

2.4.1.1. Négyszögű háló

Legyen az ábrázolandó függvény: $f: D \rightarrow \Re$, egy kétváltozós valós függvény, ahol D az a tartomány, amelyen az ábrázolást kívánjuk végezni. Legyen ez egy téglalap alakú tartomány, éspedig $D = [a,b] \times [c,d]$. Ezt a tartományt kis téglalapokra osztjuk egy négyszögletű háló segítségével. A háló egy szeme a *2.5. ábrán* látható.



2.5. ábra

A háló csomópontjai (az ábrán látható négyszög csúcsai) a következő módon írhatók fel:

$$\left.\begin{aligned} x_1 &= x_4 = a+(b-a)\frac{k}{n} \\ x_2 &= x_3 = a+(b-a)\frac{k-1}{n} \end{aligned}\right\} \quad k \in \{1,2,\ldots,n\}$$

$$\left.\begin{aligned} y_1 &= y_2 = c+(d-c)\frac{l}{m} \\ y_3 &= y_4 = c+(d-c)\frac{l-1}{m} \end{aligned}\right\} \quad l \in \{1,2,\ldots,m\} \qquad (2.36)$$

A függvény a fenti pontokban a $z_i = f(x_i, y_i)$, $i \in \{1,2,3,4\}$értékeket veszi fel.

Ha a négy csúcs nincs egy síkban, akkor a négyszögről azt mondjuk, hogy "torz". Az ilyen négyszöghöz nem rendelhető egyértelműen hozzá egy normálisvektor. Elég finom felosztás esetén a csúcsok megközelítőleg egy síkban vannak. Ezenkívül vannak felületek, amelyek azzal a sajátossággal rendelkeznek, hogy egy görbe oldalú háló minden szemének a csomópontjai egy síkban vannak (például egyik koordináta tengellyel párhuzamos tengelyű félhenger).

Kiválasztunk három csúcsot, például az 1, 2 és 4 sorszámúakat. Akkor az $\vec{n}(a_h, b_h, c_h)$ normálisvektor összetevői a következő képletekkel számíthatók:

$$\begin{cases} a_h = (y_1 - y_4)(z_2 - z_1) = \dfrac{d-c}{m}(z_2 - z_1) \\ b_h = (z_1 - z_4)(x_2 - x_1) = \dfrac{b-a}{n}(z_4 - z_1) \\ c_h = (y_1 - y_4)(x_1 - x_2) = \dfrac{(b-a)(d-c)}{mn} \end{cases} \tag{2.37}$$

Megfigyelhető, hogy c_h állandó, valamint a másik két összetevő egy-egy tényezője is. Ezt a tulajdonságot érdemes kihasználni gyorsabb program írása érdekében. A normálisvektor és a vetítési irány vektora (2.34 és 2.35 képletek) közötti skaláris szorzat előjele mutatja meg, hogy az illető felületelemet felülről vagy alulról látjuk. Az első esetben

$$a_h a_p + b_h b_p + c_h c_p > 0 \tag{2.38}$$

a másodikban:

$$a_h a_p + b_h b_p + c_h c_p < 0 \tag{2.39}$$

A rajzolás sorrendjét úgy kell megválasztani, hogy a megfigyelőtől távolabb eső felületelemeket előbb, a közelebbieket pedig utóbb rajzoljuk, így nem jöhet létre téves fedés. Ezt egy rendezési algoritmussal automatikusan is el lehet végezni, amint azt a poliéderek rajzolásáról szóló fejezetben látni fogjuk, ez azonban elég sok gépidőt vesz igénybe, főként nagy számú felületelem esetén. Másik lehetőségként a sorrendet eleve megszabhatjuk. Az utóbbi esetben a program lényegesen gyorsabb, de adott sorrend nem minden vetítési szög esetén biztosít helyes eredményt. Ha interaktív programot akarunk írni, amely lehetővé teszi a felület bármely irányból való szemlélését, akkor meg kell határozni egy összefüggést a vetítési irány és a néhány előre megadott rajzolási sorrend valamelyike között. Ezt az eljárást egy példával szemléltetjük a 2.4.1.3. alpontban.

2.4.1.2. Háromszögű háló

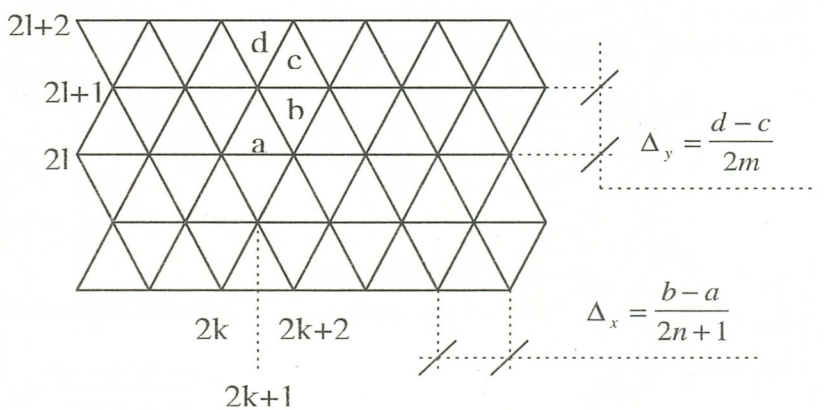
A háromszögű háló a *2.6. ábrán* látható. Egyenlő oldalú háromszögek esetén teljesülnie kell a

$$\frac{d-c}{2m}=\frac{b-a}{2n+1}\cdot\frac{\sqrt{3}}{2} \tag{2.40}$$

összefüggésnek, ami azt jelenti, hogy a D intervallum határai és n ismeretében m az

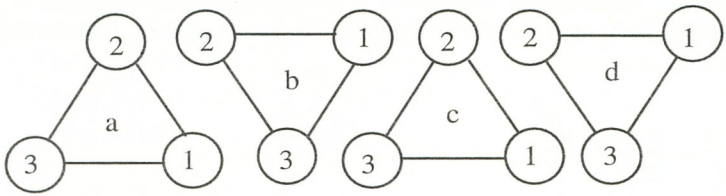
$$m=\left\lfloor\frac{d-c}{b-a}\cdot\frac{2n+1}{\sqrt{3}}\right\rfloor \tag{2.41}$$

képlettel határozható meg, ahol a szögletes zárójel a benne levő szám egész részét jelöli (Turbo Pascal-ban a ***round*** utasítás).



2.6. ábra

Az ábrán látható ***a***, ***b***, ***c*** és ***d*** típusú háromszögek adatait, illetve a megfelelő felületelem normálisvektor-összetevőit a továbbiakban részletesen meg is határozzuk. Az illető háromszögek sarkainak számozását a *2.7. ábrán* láthatjuk.



2.7. ábra

A felületelemek csúcsainak koordinátái:

$$\begin{cases} x_{1a} = a + \dfrac{2k+2}{2n+1}(b-a) \\ x_{2a} = a + \dfrac{2k+1}{2n+1}(b-a) \\ x_{3a} = a + \dfrac{2k}{2n+1}(b-a) \end{cases} \qquad \begin{cases} y_{1a} = y_{3a} = c + \dfrac{l}{m}(d-c) \\ y_{2a} = c + \dfrac{2l+1}{2m}(d-c) \end{cases}$$

$$k \in \{0,1,\ldots,n-1\} \qquad l \in \{0,1,\ldots,m-1\} \qquad (2.42)$$

$$z_{ia} = f(x_{ia}, y_{ia}) \quad ; i \in \{0,1,2,3\}$$

$$\begin{cases} x_{1b} = a + \dfrac{2k+3}{2n+1}(b-a) \quad ; k \in \{0,1,\ldots,n-1\} \\ x_{2b} = x_{2a} \\ x_{3b} = x_{3a} \end{cases} \qquad (2.43)$$

$$\begin{cases} y_{1b} = y_{2b} = y_{2a} \\ y_{3b} = y_{1a} \end{cases} \qquad \begin{cases} z_{1b} = f(x_{1b}, y_{1b}) \\ z_{2b} = z_{2a} \\ z_{3b} = z_{3a} \end{cases}$$

$$\begin{cases} x_{1c} = x_{1b} \\ x_{2c} = x_{3b} \\ x_{3c} = x_{2b} \end{cases} \qquad \begin{cases} y_{1c} = y_{3c} = y_{1b} \\ y_{2c} = c + \dfrac{l+1}{m}(d-c) \end{cases} ; l \in \{0,1,\ldots,m-1\}$$

$$\begin{cases} z_{1c} = z_{1b} \\ z_{2c} = f(x_{2c}, y_{2c}) \\ z_{3c} = z_{2b} \end{cases} \tag{2.44}$$

A normálisvektor első két koordinátája az a és c háromszögek esetén:

$$\begin{cases} a_h = \dfrac{d-c}{2m}(z_3 - z_1) \\ b_h = \dfrac{b-a}{2n+1}(z_1 - 2z_2 + z_3) \end{cases} \tag{2.46}$$

ahova természetesen az illető háromszögnek megfelelő z koordinátát kell behelyettesíteni. A b és d típusú háromszögek esetén a következő képletet kell használni:

$$\begin{cases} a_h = \dfrac{d-c}{2m}(z_2 - z_1) \\ b_h = \dfrac{b-a}{2n+1}(-z_1 - z_2 + 2z_3) \end{cases} \tag{2.47}$$

A normálisvektor harmadik összetevőjét mind a négy esetben a

$$c_h = \frac{(b-a)(d-c)}{m(2n+1)} \tag{2.48}$$

képlettel lehet kiszámítani.

A fenti képletekben szereplő ismétlődéseket jól kihasználja a "*hatpontos*" felírás, amely négy háromszöget egy egységnek tekint (*2.8. ábra*). Egy-egy ilyen egység objektum-orientált programozással kényelmesen kezelhető, amint azt a következő alpont ide vonatkozó példái is mutatják.

Az említett felületegység csúcspontjai a következők:

$$\begin{cases} x_1 = x_6 = a + \dfrac{2k}{2n+1}(b-a) \\ x_2 = x_5 = a + \dfrac{2k+2}{2n+1}(b-a) \\ x_3 = a + \dfrac{2k+1}{2n+1}(b-a) \\ x_4 = a + \dfrac{2k+3}{2n+1}(b-a) \end{cases} \quad ; \quad k \in \{0,1,\ldots,n-1\}$$

$$\begin{cases} y_1 = y_2 = c + \dfrac{l}{m}(d-c) \\ y_3 = y_4 = c + \dfrac{2l+1}{2m}(d-c) \\ y_5 = y_6 = c + \dfrac{l+1}{m}(d-c) \end{cases} \quad ; \quad l \in \{0,1,\ldots,m-1\} \qquad (2.49)$$

$$z_i = f(x_i, y_i) \quad ; i \in \{0,1,\ldots,6\}$$

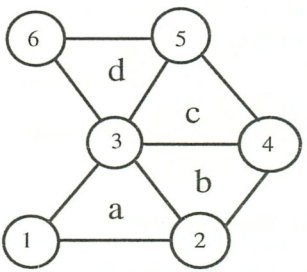
a következő táblázat pedig megadja az összetevő háromszögek normálisvektorainak a koordinátáit:

	a	b	c	d
a_h	$(z_1 - z_2)\Delta_y$	$(z_3 - z_4)\Delta_y$	$(z_3 - z_4)\Delta_y$	$(z_6 - z_5)\Delta_y$
b_h	$(z_2 - 2z_3 + z_1)\cdot \cdot \Delta_x$	$(-z_4 - z_3 + 2z_2)\cdot \cdot \Delta_x$	$(z_4 - 2z_5 + z_3)\cdot \cdot \Delta_x$	$(-z_5 - z_6 + 2z_3)\cdot \cdot \Delta_x$
c_h	$\dfrac{(b-a)(d-c)}{m(2n+1)} = 2\Delta_x\Delta_y$			

(2.50)

ahol a *2.7. ábrával* összhangban

$$\Delta_x = \frac{b-a}{2n+1} \quad ; \quad \Delta_y = \frac{d-c}{2m} \quad . \tag{2.51}$$



2.8. ábra

2.4.1.3. Példák

a) A négyszögű háló segítségével való ábrázolást az alábbi eljárással végezhetjük el:

```
type
  mtp=array[1..4] of pt;

  pt=object
    xp, yp: integer;
    procedure uj(ux, uy: integer); end;

  nsz=object
    c:mtp;
    procedure uj(u: mtp); end;

procedure pt.uj(ux, uy: integer);
begin xp:=ux; yp:=uy; end;

procedure nsz.uj(u: mtp);
begin c:=u; end;

.........................

procedure rajzol;

type
  mtv=array[1..4] of real;
var
   hl: nsz;
   x, y, z: mtv;
   k, l: integer;
   nv1, nv2, ex, ey: real;
   a1, b1, c1: real;
   kx, ky: array[1..4] of integer;

procedure negyszog;
var j: integer;
begin
   x[1]:=a+ex*k; x[4]:=x[1];     { a négyszögű hálószemek }
   x[2]:=a+ex*(k-1); x[3]:=x[2]; { sarkainak koordinátái  }
   y[1]:=c+ey*l; y[2]:=y[1];
   y[3]:=c+ey*(l-1); y[4]:=y[3];
   for  j:=1 to 4 do begin
      z[j]:=f(x[j],y[j]);  { függvényértékek a háló csúcsaiban}
           { a felületelem csúcsainak képsíkra való vetítése: }
      kx[j]:=round(s*(-bp*x[j]+ap*y[j])/nv1);
      ky[j]:=round((s*(cp*(ap*x[j]+bp*y[j])+
                    sqr(nv1)*z[j])/nv2));
      sk[j].uj(kx0+kx[j],ky0-ky[j]);
   end;
```

```
    hl.uj(sk);          { felületelem vetületét tárolja }
    { a normálisvektor első két koordinátájának kiszámítása: }
    al:=ey*(z[2]-z[1]);
    case nmv of
    1:bl:=(z[4]-z[1])*ex;      { az 1, 2, 4 csúcsok szerint }
    2:bl:=(z[3]-z[2])*ex;      { az 1, 2, 3 csúcsok szerint }
    end;                       { számítja a normálisvektort    }
                        { láthatósági feltételből megállapítja }
                                    { a felületelem színét: }
    if al*ap+bl*bp+cl*cp > 0 then setfillstyle(1,4)
      else setfillstyle(1,13);
     fillpoly(4,hl);          { felületelem vetületét rajzolja }
end;

procedure sor;          { egy felületelemekből álló sort rajzol }
begin
    if bp>0 then for l:=1 to m do negyszog
    else for l:=m downto 1 do negyszog;
end;

begin
    nv1:=sqrt(sqr(ap)+sqr(bp));       { vetítési képlet nevezői }
    nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
    ex:=(b-a)/n;                        { hálószem oldalhosszai }
    ey:=(d-c)/m;
    cl:=ex*ey;        { a normálisvektor harmadik - állandó -   }
                                                { koordinátája }
    setfillstyle(1,0); bar(0,0,getmaxx,getmaxy);
                  { felületelemsorkból lerajzolja a felületet }
    if ap>0 then for k:=1 to n do sor
                 else for k:=n downto 1 do sor
end;
```

Ahhoz, hogy a fenti eljárás működjön, meg kell még adni az ábrázolandó függvényt, az értelmezési tartomány határait (*a, b, c, d*), a vetítési irány irányítényezőit (*ap, bp, cp*), a képernyő középpontjának koordinátáit (*kx0, ky0*), a nagyítási tényezőt (*s*), illetve ki kell választani, hogy a hálót alkotó téglalapok négy csúcsa közül melyik három szerepeljen a normálisvektor meghatározásában (erre szolgál az nmv). Az eljárás megoldja a rajzolási sorrend kérdését is, bármely vetítési irány esetére, úgy, hogy a *k* és *l* indexek végigfutási sorrendjét (1-től a felső határig (*n* illetve *m*) vagy a felső határtól 1-ig) a vetítési irány *xy* síkra eső vetülete megfelelő irányítényezői előjelétől teszi függővé. Ezáltal mindig a távolabbi felületelemeket rajzolja be előbb és a közelebbieket utóbb.

Megtalálható a lemezmelléklet F3D4HF.PAS nevű forráskódjában, csekély változtatással, amely csupán az animációhoz (az újrarajzolási folyamat elrejtéséhez) szükséges , a program ugyanis a megfigyelési irány változtatását

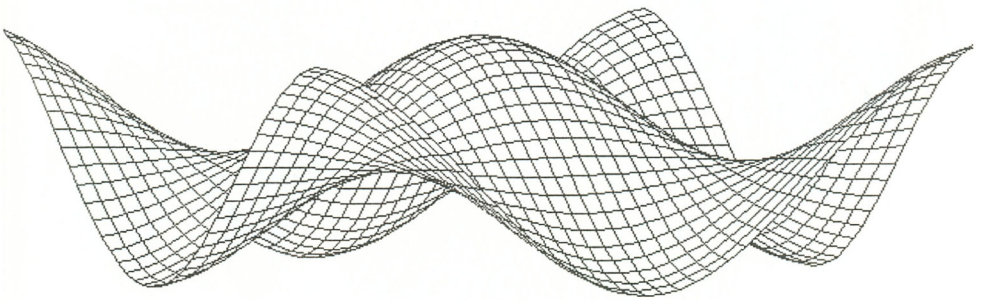
is lehetővé teszi. A program a felületek felső oldalát vörös, az alsót világos lila színnel rajzolja. Ha nem az animációt választjuk, akkor megfigyelhető a felületelemek rajzolási sorrendje is. A láthatósággal kapcsolatban jó példa a 6-os számú függvény ábrázolása. Már a beállított megfigyelési irány esetén is, nmv=1 -et választva a felület egyes elemei tévesen színeződnek, nmv=2 -vel pedig helyesen. Ez a jelenség az illető felületelemek torz (nem egysíkú) voltának tudható be.

A forgatási eljárás az új megfigyelési irány meghatározására szolgál. Az új iránytényezők meghatározására szolgáló összefüggések a (2.18) és (2.20) képletek alapján íródtak.

```
const
  alf=pi/30; bet=pi/30;
  fel = #72;  le = #80;  bal = #75;  jobb = #77;

.........................................

procedure forgat;
var au, bu, cu, r: real;
begin
    r:=sqrt(sqr(ap)+sqr(bp));
       { új megfigyelési irány megállapítása }
       {  a leütött billentyû függvényében:   }
    case kr of
    le:  begin
           au:=ap*cos(bet)+ap*cp/r*sin(bet);
           bu:=bp*cos(bet)+bp*cp/r*sin(bet);
           cu:=cp*cos(bet)-r*sin(bet);
         end;
    fel: begin
           au:=ap*cos(bet)-ap*cp/r*sin(bet);
           bu:=bp*cos(bet)-bp*cp/r*sin(bet);
           cu:=cp*cos(bet)+r*sin(bet);
         end;
    bal: begin
           au:=ap*cos(alf)+bp*sin(alf);
           bu:=-ap*sin(alf)+bp*cos(alf);
           cu:=cp;
         end;
    jobb:begin
           au:=ap*cos(alf)-bp*sin(alf);
           bu:=ap*sin(alf)+bp*cos(alf);
           cu:=cp;
         end;
    end;
    ap:=au; bp:=bu; cp:=cu;
end;
```

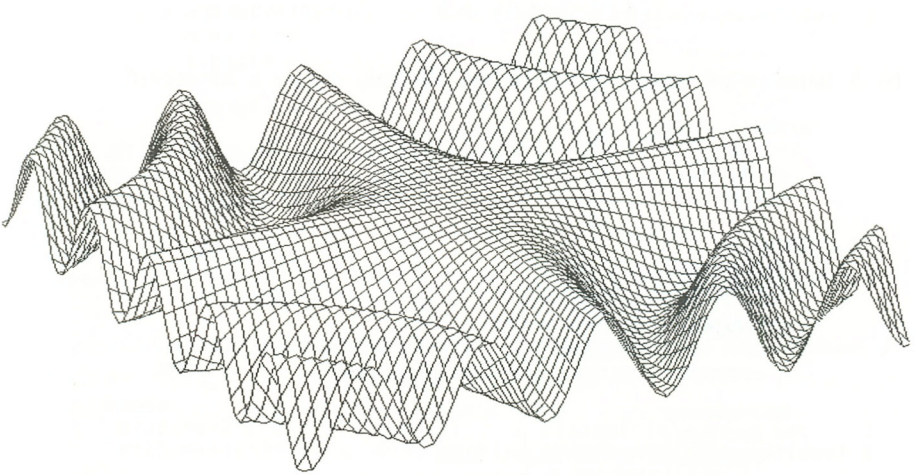
A nyíl billentyűk leütése a megfigyelési irány elfordulását eredményezi az adott irányba. A képernyőn a felület ellenkező irányba fordul el.

A program fehér-fekete változatával készültek az alábbi ábrák.



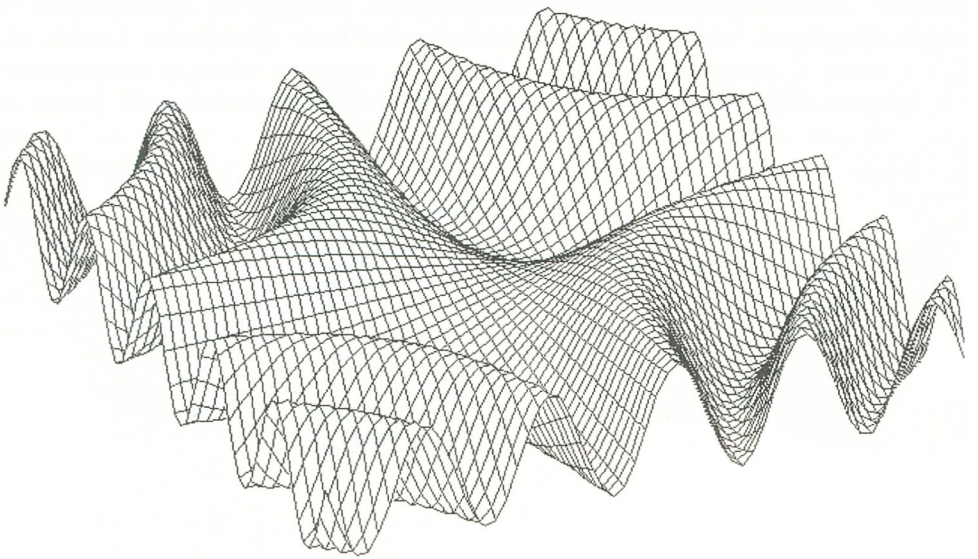
2.9.ábra

$$f:[-3,3]\times[-3,3]\to\Re \quad ; \quad f(x,y)=\frac{3}{2}\cdot\cos x\cdot\cos y$$



2.10.ábra

$$f:[-4,4]\times[-4,4]\to\Re \quad ; \quad f(x,y)=\cos xy$$



2.11.ábra

$$f : [-4,4]\times[-4,4] \to \Re \quad ; \quad f(x,y) = \sin xy$$

b) A háromszögű háló felhasználásával ábrázoló eljárás a következő:

```
const
 e:array[1..4,1..3] of byte=((1,2,3),(3,2,4),(4,5,3),(3,5,6));
{ ez a mátrix adja meg, hogy az egy egységként kezelt négy }
{ háromszög mindegyikének a hat pont közül melyek a sarkai }

type
  mxp=array[1..3] of pt;
  mtp=array[1..6] of pt;

  pt=object
    xp, yp: integer;
    procedure uj(ux, uy: integer); end;

  hsz=object
    hc: mxp;
    procedure uj(u: mxp); end;

  nhsz=object
 { négy háromszöget egy egységbe foglaló objektum }
    c: mtp;
    hmsz: array[1..4] of hsz;
    procedure uj(u: mtp); end;
```

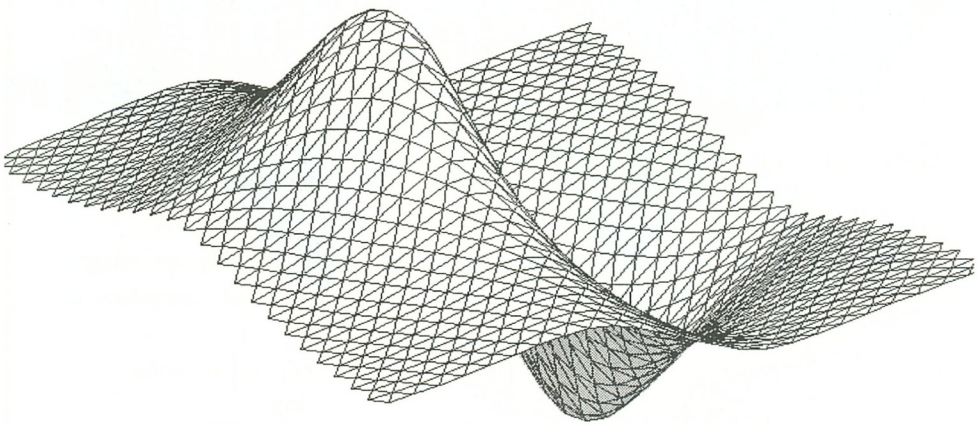
```
procedure pt.uj(ux, uy: integer);
begin xp:=ux; yp:=uy; end;

procedure hsz.uj(u: mxp);
begin hc:=u; end;

procedure nhsz.uj(u: mtp);
begin c:=u; end;

.....

procedure rajzol;
type
mtv=array[1..6] of real;
var
sk: mtp;
hsk: mxp;
hl: nhsz;
x, y, z: mtv;
k, l: integer;
nv1, nv2, ex, ey: real;
al, bl, cl: real;
kx, ky: array[1..6] of integer;

procedure negyharomszog;
                  { négy felületelemből álló egységet rajzol }
var i, j: integer;

    procedure rz;                    { egy háromszög rajzolása }
      begin
    { a normálisvektor első két koordinátájának kiszámítása: }
        case j of
        1:begin al:=(z[1]-z[2])*ey;
                bl:=(z[2]-2*z[3]+z[1])*ex; end;
        2:begin al:=(z[3]-z[4])*ey;
                bl:=(-z[4]-z[3]+2*z[2])*ex; end;
        3:begin al:=(z[3]-z[4])*ey;
                bl:=(z[4]-2*z[5]+z[3])*ex; end;
        4:begin al:=(z[6]-z[5])*ey;
                bl:=(-z[5]-z[6]+2*z[3])*ex; end;
        end;
   { láthatósági feltételből megállapítja a háromszög színét: }
        if al*ap+bl*bp+cl*cp > 0 then setfillstyle(1,4)
                                 else setfillstyle(1,13);
        fillpoly(3,hl.hmsz[j]);     { lerajzolja a háromszöget }
      end;

  begin
   x[1]:=a+2*k*ex; x[6]:=x[1];     { kiszámítja a háló hat     }
   x[2]:=a+(2*k+2)*ex; x[5]:=x[2];{ csomópontjának pontjának }
                                   { a koordinátáit }
    x[3]:=a+(2*k+1)*ex;
    x[4]:=a+(2*k+3)*ex;
    y[1]:=c+2*l*ey; y[2]:=y[1];
    y[3]:=c+(2*l+1)*ey; y[4]:=y[3];
    y[5]:=c+(2*l+2)*ey; y[6]:=y[5];
```

```
      for  j:=1 to 6 do begin
        z[j]:=f(x[j],y[j]);  { függvényértékek a csomópontokban }
           { a felületegység csúcsainak képsíkra való vetítése: }
        kx[j]:=round(s*(-bp*x[j]+ap*y[j])/nv1);
        ky[j]:=round((s*(-cp*(ap*x[j]+bp*y[j])+
                        sqr(nv1)*z[j])/nv2));
        sk[j].uj(kx0+kx[j],ky0-ky[j]);
      end;
       hl.uj(sk);                        { csúcsok vetületeit tárolja }
       for j:=1 to 4 do begin
         for i:=1 to 3 do                          { a négy háromszög }
           hsk[i].uj(sk[e[j,i]].xp,sk[e[j,i]].yp);
                                                    { meghatározása   }
         hl.hmsz[j].uj(hsk);                        { és tárolása     }
       end;
                                    { a négy háromszög lerajzolása }
                              { a vetítési iránytól függő sorrendben }
       if bp>0 then
                  for j:=1 to 4 do rz
              else for j:=4 downto 1 do rz
    end;

    procedure sor;          { felületegységekből álló sort rajzol }
    begin
      if ap>0 then for k:=0 to n-1 do negyharomszog
              else for k:=n-1 downto 0 do negyharomszog;
       {delay 100}
      { ezt aktiválva meg lehet figyelni a rajzolási sorrendet }
    end;

  begin
    nv1:=sqrt(sqr(ap)+sqr(bp));
    nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
    ex:=(b-a)/(2*n+1);
    ey:=(d-c)/2/m;
    cl:=2*ex*ey;
    setfillstyle(1,0); bar(0,0,getmaxx,getmaxy);
    if bp>0 then
         for l:=0 to m-1 do sor  { vetítési iránynak megfelelő }
         else for l:=m-1 downto 0 do sor
                                    { sorrendben rajzolt sorokból }
  end;                                 { előállítja a felületet  }
```

Ez az eljárás a 2.4.1.2. alpontban leírt hatpontos felírást alkalmazza, amely négy elemi felületháromszöget egy egységnek tekint. A rajzolási sorrendet az előző példához hasonlóan kezeli, azzal a különbséggel, hogy a háromszögű háló esetében már nem közömbös, hogy előbb *k* szerint, vagy előbb *l* szerint rajzoljuk a sorokat, ugyanis vigyázni kell arra, hogy a sorok szélei simák legyenek, különben nem valósul meg a helyes fedés. Ezenkívül az egy egységet alkotó háromszögek rajzolási sorrendjét is a vetítési iránytól függővé kellett tenni.

Ez az eljárás a lemezmellékleten az F3D3HF.PAS programban található, amely lehetővé teszi a megfigyelési irány változtatását és az újrarajzolási folyamat elrejtését is. Ha nem az animációt választjuk, megfigyelhetjük a felületelemek berajzolási sorrendjét. Ezt megkönnyíti a fenti eljárásban kapcsos zárójelben szereplő ***delay*** utasítás (ha aktiváljuk).

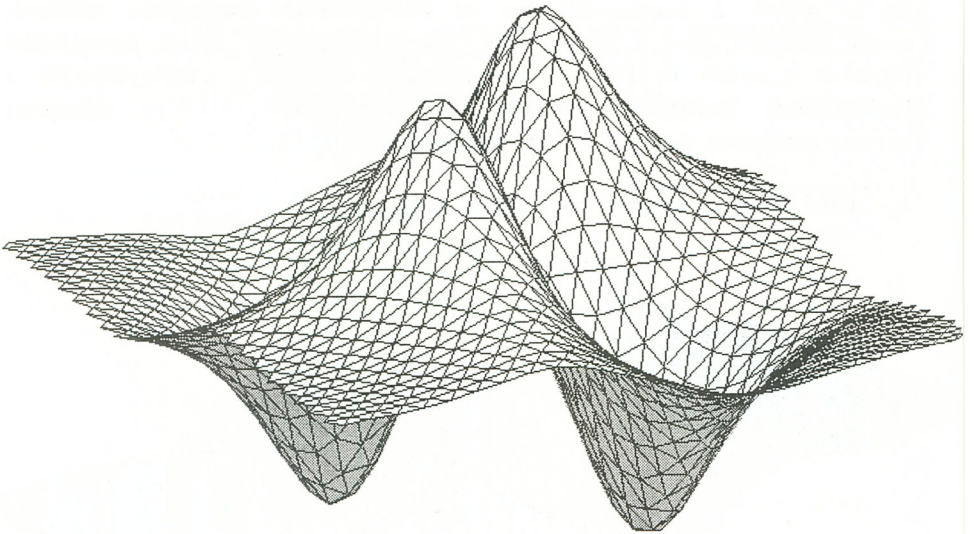
Az alábbi ábrák e program segítségével készültek.



2.12. ábra

$$f:[-2.4,2.4]\times[-2.4,2.4]\to\Re \quad ; \quad f(x,y)=\frac{4x}{\left(|x^3|+1\right)\left(y^2+1\right)}$$

Az ábrán megfigyelhető a felület alsó oldalának a felsőtől különböző színezése.



2.13.ábra

$$f:[-3,3]\times[-3,3]\to\Re \quad ; \quad f(x,y)=\frac{5xy}{\left(x^4+1\right)\left(y^4+1\right)}$$

2.4.2. Szintvonalak

A képies ábrázolásmód mellett az Euler-Monge-féle előállítású felületek szintvonalas ábrázolása is használatos. Előnyei főként a műszaki alkalmazások esetén nyilvánvalóak. Így készülnek például a földrajzi, meteorológiai térképek vagy a villamos erőtér potenciáleloszlását ábrázoló rajzok.

A szintvonalas ábrázolás kiterjeszthető nem sík tartományon, hanem görbe felületeken értelmezett függvények ábrázolására is, mint például a hőmérsékleteloszlás egy bonyolult alakú alkatrész felszínén. Ezzel a kérdéssel a nem geometriai tulajdonságok szemléltetéséről a 2.8 fejezetben foglalkozunk.

A függvényértékekről a szintvonalakra felírt számok, az egyes felületrészek meredekségéről pedig a szintvonalak sűrűsége hordoz információt.

Legyen az $f:\Re^2 \rightarrow \Re$ függvény folytonos és korlátos az $I = [a,b]\times[c,d]$ tartományon. Az α névleges értékű szintvonal képlete:

$$M(\alpha) = \left\{(x,y) \in I \;\middle|\; f(x,y) = \alpha\right\} \tag{2.51}$$

A szóban forgó szintvonal tehát azon pontok mértani helye, amelyekben a függvény behelyettesítési értéke éppen α.

Ha a szintvonalak névleges értékei nincsenek előre megszabva, akkor először kiszámítjuk a függvény I tartományra vonatkozó szélsőértékeit:

$$f_{\max} = \max_I\{f(x,y)\} \quad ; \quad f_{\min} = \min_I\{f(x,y)\} \tag{2.52}$$

Legcélravezetőbb ezt egy nemlineáris optimizálási algoritmus segítségével végezni, amely megadja a szélsőértékpontokat is (vagyis a tartomány azon pontjait, amelyekben a függvény felveszi az illető szélsőértékeket). Ezekre az esésvonalak ábrázolásánál amúgy is szükség lesz. Tekintettel a képernyő véges felbontására, erre a célra egészen egyszerű program is írható. Kiszámítjuk a függvény értékét az I tartomány minden képernyőpontnak megfelelő pontjában és a kapott értékek közül kiválasztjuk a legkisebbet és a legnagyobbat. Az így elért pontosság nem túl nagy, de a célnak megfelelő.

Ha p számú, egyenlő közű szintvonalat akarunk húzni, akkor az

$$\alpha_i = f_{\min} + i \cdot \frac{f_{\max} - f_{\min}}{p+1} \quad ; \quad i \in \{0,1,\ldots,p+1\} \tag{2.53}$$

névleges értékekkel kell dolgozzunk. Ha a legkisebb értéknél picivel nagyobb, illetve a legnagyobbnál picivel kisebb szélsőértékekkel dolgozunk, vagyis $f_{\max}$ helyett $f_{\max} - \varepsilon$, $f_{\min}$ helyett $f_{\min} + \varepsilon$ értékeket veszünk, ahol ε egy alkalmasan megválasztott kicsi pozitív szám, akkor a program a megfelelő szélsőértékek köré apró zárt görbét rajzol, vagyis bejelöli a szélsőértékpontokat.

Az I tartománynak megfelelő képernyőtéglalap legyen (X_0, Y_0, X_m, Y_n). Itt (X_0, Y_0) jelöli a szóban forgó téglalap bal alsó sarkát, illetve (X_m, Y_n) a jobb felső sarkot. Ha ezt a téglalapot pixelekre bontjuk, akkor minden sorban m illetve minden oszlopban n pixel lesz, ahol:

$$m = X_m - X_0 \quad ; \quad n = Y_0 - Y_n \tag{2.54}$$

Az I tartomány (x_k, y_l) pontjai illetve az (X_k, Y_l) képernyőpontok közötti megfeleltetés a következő módon történik:

$$\begin{cases} x_k = a + \dfrac{k}{m}(b-a) \\ y_l = c + \dfrac{l}{n}(d-c) \end{cases} \rightarrow \begin{cases} X_k = X_0 + k \\ Y_l = Y_0 + l \end{cases} ; \begin{cases} k \in \{0,1,\ldots,m\} \\ l \in \{0,1,\ldots,n\} \end{cases} \tag{2.55}$$

Ha azt akarjuk, hogy a képernyőtéglalap hasonló legyen az I tartományhoz, akkor természetesen teljesülnie kell az

$$\frac{m}{n} = \frac{b-a}{d-c} \tag{2.56}$$

feltételnek, illetve az eltérés lehető legkisebb kell legyen.

A szintvonalak egyszerre rajzolhatók, úgy hogy mindkét irányban pixelenként végigpásztázzuk az I tartománynak megfelelő képernyőtéglalapot és ahol a

$$g_i\ \{0,1,\ldots,m\}\times\{0,1,\ldots,n\}\to\Re\ ,\ g_i(k,l)=f(x_k,y_l)-\alpha_i\ ;\ i\in\{0,1,\ldots,p+1\} \tag{2.57}$$

függvények valamelyike előjelet vált, berajzolunk egy pontot, vagyis az illető pixel színét megváltoztatjuk. Ezt a műveletet a pásztázási sorrend felcserélésével megismételjük, hogy folytonos szintvonalakat kapjunk.

2.4.3. Szintsávok

A földrajzi térképekhez hasonlóan a magasságot színek segítségével is érzékeltethetjük. A szintvonalak rajzolásához hasonlóan végigpásztázzuk a kívánt képernyőtartományt, most azonban minden pontot berajzolunk, csak az illető pixelek színét változtatjuk aszerint, hogy a megfelelő pontban a függvény értéke melyik szintsávban helyezkedik el. A szintsávokat a következő módon értelmezhetjük:

$$S_i=\left\{(x,y)\in I\ \middle|\ \alpha_{i-1}\le f(x,y)\le\alpha_i\right\}\quad ;\quad i\in\{1,2,\ldots,p+1\} \tag{2.58}$$

Az ábrázoláshoz kiválóan alkalmas a Turbo Pascal "***putpixel***" utasítása, amelynél harmadik paraméterként a színkódot kell megadni. Mivel most minden pixel színét megváltoztatjuk, elegendő egyszer végigpásztázni a kívánt képernyőtartományt, nem szükséges kétszer, mint a szintvonalak ábrázolása esetén.

Ha azt akarjuk, hogy a program önműködően ossza be a szintsávokat, akkor elegendő a függvény szélsőértékeit megadni, és a szintvonalak rajzolásához hasonlóan kiszámítani a szintvonalak névleges értékeit, abból kifolyólag, hogy a szintvonalak tulajdonképpen a szintsávok határgörbéi.
Legyen

$$\Delta=\frac{f_{\max}-f_{\min}}{p+1} \tag{2.59}$$

Hogy éppen melyik szintsávban vagyunk azt a következő képlet segítségével határozhatjuk meg:

$$j = \left[\frac{f(x,y) - f_{\min}}{\Delta} \right] + 1 \quad , \tag{2.60}$$

ahol a szögletes zárójel a benne foglalt mennyiség egész részét jelöli. A kapott j érték a keresett színkód. A szintsávok száma kisebb kell legyen mint a megszabott színárnyalat szám (legfeljebb egyenlő, ha az alapszínt - általában fekete - is felhasználjuk). A VGA alapszabvány esetén ez a szám 16.

A szintvonalak rajzolásáról szóló alpontban megállapított, a függvény értelmezési tartománya és a képernyőtartomány közötti megfeleltetés (2.55) természetesen itt is érvényes.

Mind a szintvonalak, mind a szintsávok ábrázolásánál növelni lehet a sebességet szimmetrikus függvények esetében úgy, hogy az egyszer kiszámított függvényértéknek megfelelő színű pixelt mindenhová berajzoljuk, ahol tudjuk, hogy a szimmetria miatt a függvénynek ugyanaz az értéke. Így nem kell a tartomány minden pontjában kiszámítani a függvényértékeket. Hasonló módon lehet kihasználni bizonyos függvények periodikusságát is.

2.4.4. Esésvonalak

Az esésvonalak azoknak a görbéknek az értelmezési tartományra (xy síkra) való merőleges vetületei, amelyek mentén egy pici, könnyű labda gurulna, ha azt a felületre helyeznénk. Kétdimenziós potenciálterek esetén erővonalakról beszélünk. Háromdimenziós potenciálterek esetén (ilyen például minden elektrosztatikus tér) az erővonalak térbeli görbék.

Legyen az $f:\Re^2 \to \Re$ függvény differenciálható az $I = [a,b]\times[c,d]$ tartományon. Az f függvény gradiense az a vektor, amelynek koordinátái a függvény parciális deriváltjai, vagyis a

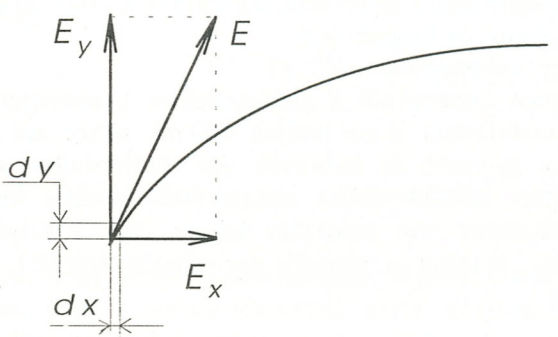
$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y) \right) \tag{2.61}$$

vektor. A gradiensvektor mindig a legnagyobb növekedés irányába mutat, vagyis amerre a felület a *"legmeredekebb"*.

Esésvonalnak azt a görbét nevezzük, amelynek a gradiensvektor minden pontjában érintője (2.14. ábra). Mivel az esésvonalon nem állapítottunk meg irányítást, ez az értelmezés nem mond ellent a jelen alpont első bekezdésének. A pici labda a gradiensvektorral éppen ellentétes irányba fog gurulni, tehát ugyancsak az esésvonalon.

Jelölje E_x illetve E_y a gradiensvektor koordinátáit:

$$\begin{cases} E_x = \dfrac{\partial f}{\partial x} \\ E_y = \dfrac{\partial f}{\partial y} \end{cases} \tag{2.62}$$



2.14. ábra

Az esésgörbe elemi ívhosszát egy, az érintővel kollineáris szakasszal közelítjük, amelynek koordinátatengelyekre eső vetületei *dx* illetve *dy*. Könnyen belátható, hogy e szakasz és az érintő vetületei között fennáll a következő arányosság:

$$\frac{dx}{dy} = \frac{E_x}{E_y} \tag{2.63}$$

A szóbanforgó szakasz hossza:

$$ds = \sqrt{dx^2 + dy^2} \tag{2.64}$$

Így a megfelelő vetületek képletei:

$$\begin{cases} dx = \dfrac{E_x}{E} ds \\ dy = \dfrac{E_y}{E} ds \end{cases} \tag{2.65}$$

ahol E a gradiensvektor hosszát jelöli:

$$E = \sqrt{E_x^2 + E_y^2} \tag{2.66}$$

Az esésvonalak meghúzása a következő algoritmus szerint történik:
- megadjuk az elemi ív hosszát (ds)
- megadunk egy kezdőpontot - (x^j, y^j)
- ebben a pontban kiszámítjuk a gradiensvektor koordinátáit. Ha a parciális deriváltakat analitikusan is elő tudjuk állítani, akkor ezt használjuk, mert így a program gyorsabb és pontosabb lesz. Előfordulhat azonban, hogy ez nem lehetséges, például abban az esetben, amikor még a függvény analitikus előállítását sem ismerjük, hanem csak bizonyos rácspontokban felvett értékeit. Ilyenkor a parciális deriváltakat közelítő képletekkel lehet kiszámítani:

$$\begin{cases} \dfrac{\partial f}{\partial x}(x,y) \approx \dfrac{f(x+h,y) - f(x-h,y)}{2h} \\ \dfrac{\partial f}{\partial y}(x,y) \approx \dfrac{f(x,y+h) - f(x,y-h)}{2h} \end{cases} \tag{2.67}$$

ahol minél kisebbnek választjuk h értékét annál pontosabb közelítő értékeket kapunk.
- a (2.65) képlettel kiszámítjuk a koordinátatengelyekkel párhuzamos elemi elmozdulásokat (dx , dy)

- a kapott értékek segítségével meghatározzuk az esésvonal új pontját:

$$\begin{cases} x^{j+1} = x^j + dx \\ y^{j+1} = y^j + dy \end{cases} \tag{2.68}$$

- "***lineto***" utasítással meghúzzuk az esésvonal elemi ívét megközelítő szakaszt, majd az új pontot kiindulási pontnak tekintve megismételjük az itt leírt műveleteket.

Természetesen minél kisebb a *ds* elemi ívhossz, annál pontosabb eredményt kapunk, viszont annál több gépidőt vesz igénybe a program.

A fent leírt algoritmus az első kiindulási pontból a növekvő függvényértékeknek megfelelő pontok felé vezet. Ellenkező irányba, vagyis csökkenő függvényértékek felé úgy haladhatunk, hogy (2.65) helyett a fenti algoritmusban a

$$\begin{cases} dx = -\dfrac{E_x}{E} ds \\ dy = -\dfrac{E_y}{E} ds \end{cases} \tag{2.69}$$

képletet használjuk. Amint látni fogjuk, tulajdonképpen mind a két képletre szükség lesz.

Annak eldöntésére, hogy melyik esésvonalakat rajzoljuk meg általában célravezető módszer az, hogy a szélsőértékpontok körül, ezeknek a közelében levő megközelítőleg szintvonal alakú görbék megközelítőleg egyenlő távolságra levő pontjaiból kiindulunk mind növekvő, mind csökkenő irányba. Az esésvonalak a szélsőértékpontokban végződnek. Itt szükség van egy kilépési feltételre, a szélsőértékpontokban ugyanis mindkét parciális derivált nulla, ezért az algoritmus nem működhet. A kilépési feltétel: $E<\varepsilon$, ahol ε egy elég kicsi, pozitív szám. Ezenkívül az esésvonalak a tartomány szélén is véget érhetnek, ezért egy ebben az esetben működő kilépési feltételt is meg kell adni.

A szintvonalak és esésvonalak fontos tulajdonsága, hogy egymásra minden pontban merőlegesek. Két szintvonal sohasem metszi egymást, sem pedig két esésvonal, amint az említett görbecsaládok értelmezéséből belátható.

2.4.5. Példák

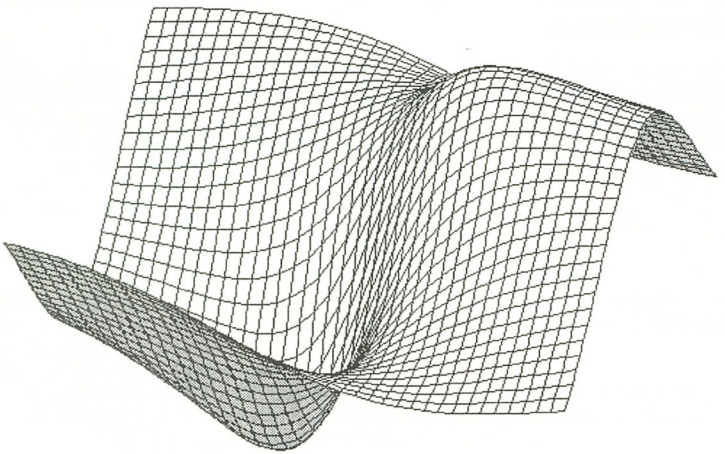
a) Szintvonalakat a következő eljárás segítségével ábrázolhatunk:

```
const
  m=640; n=480; kx0=0; ky0=480;      { képernyőtéglalap méretei }
var                                  { és kezdőpontja           }
  j, k, l, vsz: integer;
  a, b, c, d, hx, hy: real;
  u, x, y, z: real;
  dl, dt: array[0..50] of real;
  min, max: real;

.....................................................

procedure rajzx; { az x tengellyel 45°-nál kisebb szöget  }
 begin           { bezáró szintvonalrészeket jól kirajzolja  }
   for k:=0 to m do
     for l:=0 to n do begin
       x:=a+k*hx;     { a képernyőkoordináták és a függvény  }
       y:=c+l*hy ;    { értelmmezési tartományának pontjai   }
                      { közötti összefüggés }
       u:=f(x,y);
       for j:=0 to vsz do begin
         dl[j]:=min+z*j-u; { a függvényértéket összehasonlítja a}
                           { szintvonalak névleges értékeivel   }
         if l>0 then       { ha átment egy szintvonalon, rajzol }
                           { egy pontot }
           if dl[j]*dt[j]<0 then putpixel(kx0+k,ky0-l,15)
       end; dt:=dl; end;
 end;

 rocedure rajzy;  { az y tengellyel 45°-nál kisebb szöget    }
 begin            { bezáró szintvonalrészeket jól kirajzolja }
   for l:=0 to n do
     for k:=0 to m do begin
       x:=a+k*hx;
       y:=c+l*hy
       u:=f(x,y);
       for j:=0 to vsz do begin
         dl[j]:=min+z*j-u;
        if k>0 then
          if dl[j]*dt[j]<0 then putpixel(kx0+k,ky0-l,15);
       end; dt:=dl; end;
 end;
 .....................................................

 hx:=(b-a)/m;     { az egy pixelnek megfelelő távolságok a }
 hy:=(d-c)/n;     { függvény értelmezési tartományában     }
 z:=(max-min)/vsz; rajzx; rajzy;
```

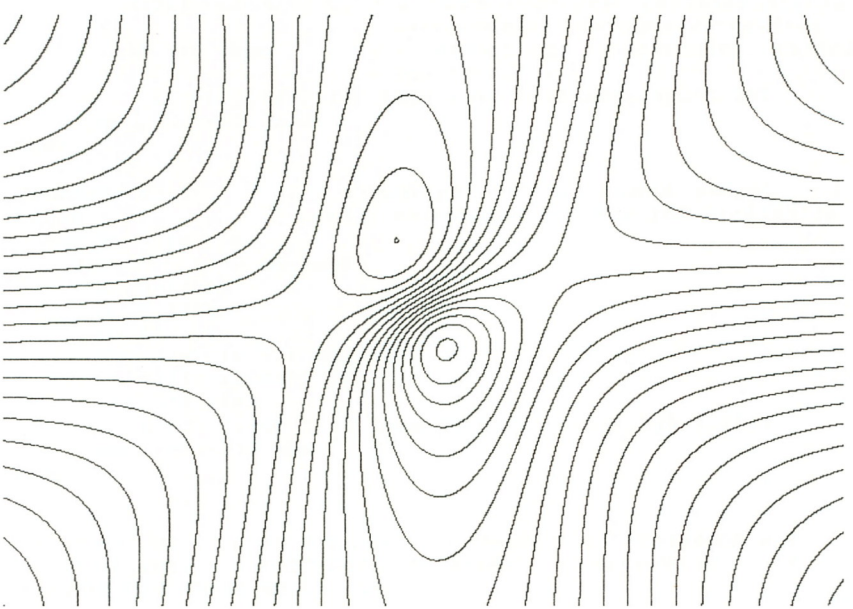
Meg kell adni az értelmezési tartomány határait (*a, b, c, d*), a függvénynek az értelmezési tartományra vonatkozó szélsőértékeit vagy az előre megszabott legnagyobb és legkisebb névleges értéket (*max, min*), valamint a rajzolni kívánt szintvonalszámot.

Az alábbiakban néhány függvény képies és szintvonalas ábrázolásának eredményét mutatjuk be.

$$f:[-6.4,6.4]\times[-4.8,4.8]\to\Re \quad ; \quad f(x,y)=\frac{3y-2x+x^2-yx^2+y^2}{x^2+y^2+1}$$



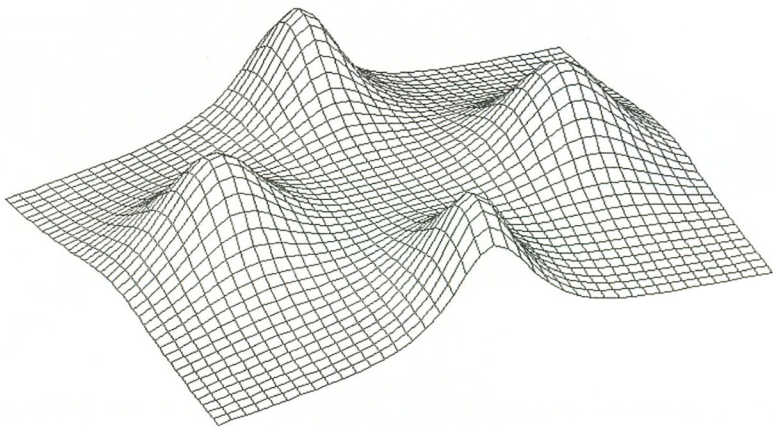
2.15. ábra



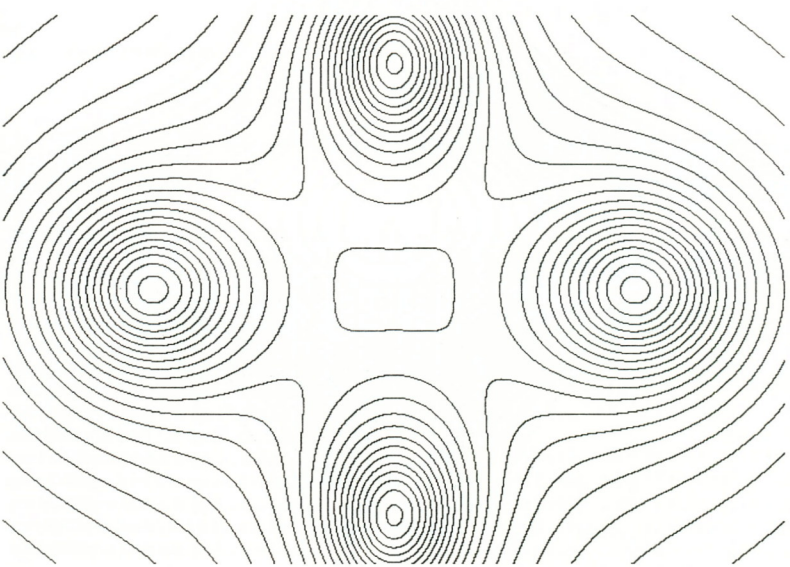
2.16. ábra

$$f:[-3.2,3.2]\times[-2.4,2.4]\to\Re \qquad ; \quad f(x,y)=\exp\left(\frac{1}{x^2-4x+y^2+5}\right)+$$

$$+\exp\left(\frac{1}{2x^2+y^2+4y+5}\right)+\exp\left(\frac{1}{x^2+4x+y^2+5}\right)+\exp\left(\frac{1}{2x^2+y^2-4y+5}\right)-4$$

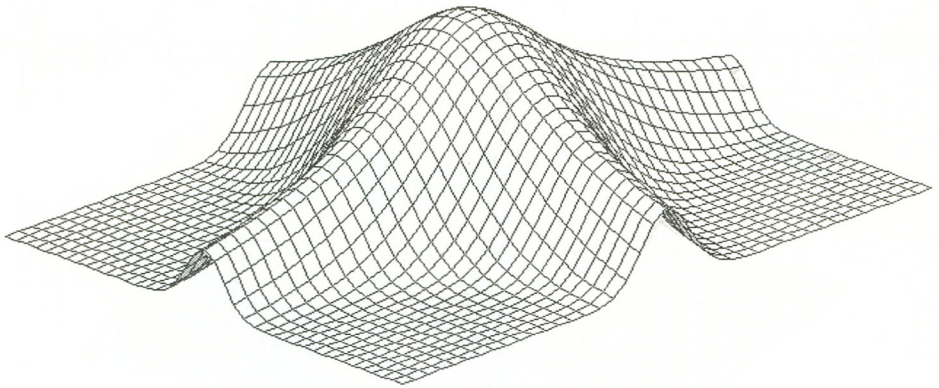


2.17. ábra

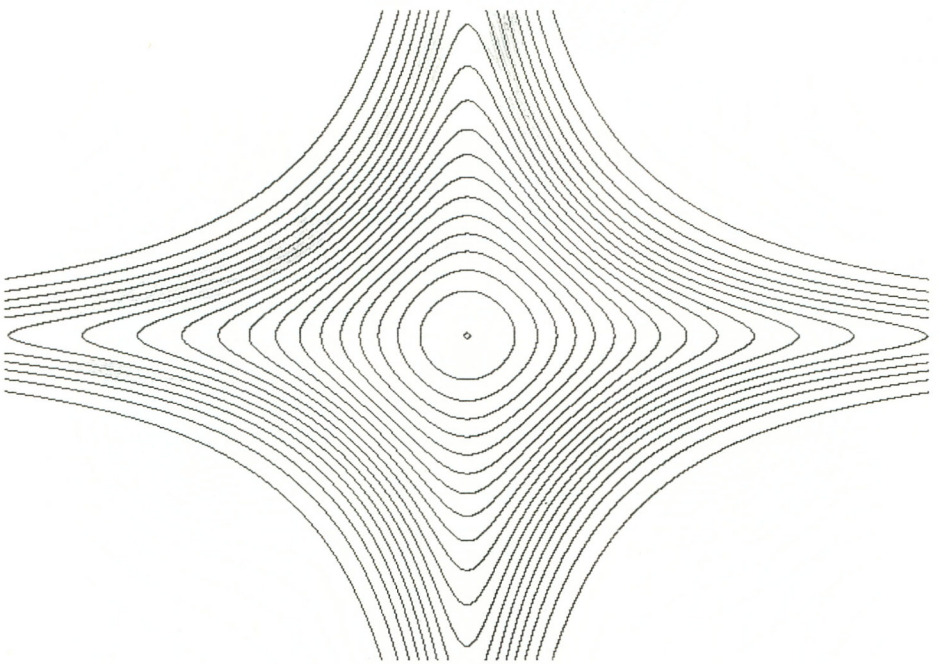


2.18. ábra

$$f:[-3.2,3.2]\times[-2.4,2.4]\to\Re \quad ; \quad f(x,y)=\frac{2\exp\left(-x^2y^2\right)}{\sqrt{x^2+y^2+1}}$$



2.19.ábra



2.20.ábra

A lemezmellékleten a szintvonalrajzoló program a SZINTVON.PAS néven található meg.

b) Szintsávok rajzolásához a fenti programban a két rajzoló eljárás helyett a következő használandó:

```
procedure rajzol;
begin
  for l:=0 to n do begin
    y:=c+l*hy;
    for k:=0 to m do begin
      x:=a+k*hx;
      { a függvényérték és a szintsávfelosztás }
      { összehasonlításából megállapítja a színkódot }
      { és a megfelelő helyre berajzolja  }
      { a megfelelő színű pixelt }
       q:=round(z*(f(x,y)-min)+1);
       putpixel(kx0+k,ky0-l,q);
     end;
  end;
end;
```

A lemezmellékleten a SZINTSAV.PAS nevű forráskód a következő színskálamódosító eljárást alkalmazza:

```
var
  pl: PaletteType;

.................................

{ vörös, zöld és kék alapszínekből keveri a kívánt árnyalatokat }
  procedure szin;
  begin
    getpalette(pl);
    setRGBpalette(pl.colors[1],63,0,0);
    setRGBpalette(pl.colors[2],63,21,0);
    setRGBpalette(pl.colors[3],63,42,0);
    setRGBpalette(pl.colors[4],63,63,0);
    setRGBpalette(pl.colors[5],32,63,0);
    setRGBpalette(pl.colors[6],0,61,0);
    setRGBpalette(pl.colors[7],0,63,32);
    setRGBpalette(pl.colors[8],0,63,63);
    setRGBpalette(pl.colors[9],0,42,63);
    setRGBpalette(pl.colors[10],0,21,63);
    setRGBpalette(pl.colors[11],0,0,63);
    setRGBpalette(pl.colors[12],21,0,63);
    setRGBpalette(pl.colors[13],42,0,63);
    setRGBpalette(pl.colors[14],63,0,63);
    setRGBpalette(pl.colors[15],56,0,56);
  end;
```

Így a legnagyobb függvényértékeknek megfelelő (legmagasabban fekvő) sáv vörös, a legmélyebb sáv pedig ibolya színű lesz.

c) Az esésvonalakat rajzoló eljárást a lemezmelléklet SZEV1.PAS nevű forráskódjában szereplő változatban mutatjuk be. A program az

$$f:[-12.8,12.8]\times[-9.6,9.6]\to\Re \quad ; \quad f(x.y)=4\cos\frac{x}{2}\cos\frac{y}{3}$$

függvény szintvonalait és esésvonalait rajzolja meg (*2.22.ábra*). A függvény képies képe a *2.21. ábrán* látható. A szintvonalak ábrázolásánál a program kiaknázza a függvénynek mind az *xz* mind az *yz* síkhoz viszonyított szimmetriáját negyedére csökkentve a szintvonalak ábrázolásához szükséges gépidőt. A képernyőn a szintvonalak fehér, az esésvonalak lila színűek.

```
const
  kx0=320; ky0=240; egr=0.0004;
  lp=25;  nm=8;       { a rajzolás léptéke és a számbajövő }
                      { szélsőértékhelyek száma            }
var
  k, q: integer;
  x, y: real;
  xsz, ysz, x0, y0: real;

.....................................

function f(x,y:real):real; { az ábrázolandó függvény }
begin
  f:=4*cos(x/2)*cos(y/3); end;

function fx(x,y:real):real;     { x szerinti parciális derivált }
begin
  fx:=-2*sin(x/2)*cos(y/3); end;

function fy(x,y:real):real;     { y szerinti parciális derivált }
begin
  fy:=-4*cos(x/2)*sin(y/3)/3; end;

procedure szelso;               { a szélsőértékhelyek megadása }
begin
  case q of
  1:begin xsz:=-2*pi; ysz:=0; end;
  2:begin xsz:=2*pi; ysz:=0; end;
  3:begin xsz:=4*pi; ysz:=-3*pi; end;
  4:begin xsz:=0; ysz:=-3*pi; end;
  5:begin xsz:=4*pi; ysz:=3*pi; end;
  6:begin xsz:=-4*pi; ysz:=-3*pi; end;
  7:begin xsz:=0; ysz:=3*pi; end;
```

```
  8:begin xsz:=-4*pi; ysz:=3*pi; end;
  end;
end;

procedure vonal;        { növekvő függvényértékek felé haladó }
                        { esésvonalat rajzol }
const ls=0.001;         { lépéshossz (a valós síkon) }
var ex, ey, e: real;
begin
  x:=x0; y:=y0;
  repeat begin
    ex:=fx(x,y); ey:=fy(x,y); e:=sqrt(ex*ex+ey*ey);
    x:=x+ex/e*ls; y:=y+ey/e*ls;
    lineto(round(kx0+x*lp), round(ky0+y*lp));
  end until (abs(x*lp)>320) or (abs(y*lp)>240) or (e<egr);
 { kilépési feltétel }
end;

procedure vonalv;       { csökkenő függvényértékek felé haladó }
                        { esésvonalat rajzol }
const ls=0.001;
var ex, ey, e: real;
begin
  x:=x0; y:=y0;
  repeat begin
    ex:=fx(x,y); ey:=fy(x,y); e:=sqrt(ex*ex+ey*ey);
    x:=x-ex/e*ls; y:=y-ey/e*ls;
    lineto(round(kx0+x*lp), round(ky0+y*lp));
  end until (abs(x*lp)>320) or (abs(y*lp)>240) or (e<egr);
end;

procedure esesvonal; { esésvonalak rajzolását előkészítő eljárás }
const ne=40; rk=0.5;
var alf: real;
begin
  for q:=1 to nm do begin
    szelso;
    for k:=1 to ne do begin
      alf:=2*k*pi/ne;
      x0:=xsz+2*rk*cos(alf);    { a szélsőértékek körüli ellipszis -}
                                { megközelítőleg   }
      y0:=ysz+3*rk*sin(alf);    { szintvonal - pontjaiból indítja az }
                                { esésvonalrajzoló }
      moveto(round(kx0+x0*lp), round(ky0+y0*lp)); { eljárásokat }
      vonal;
      moveto(round(kx0+x0*lp), round(ky0+y0*lp));
      vonalv;
    end;
  end;
end;
```

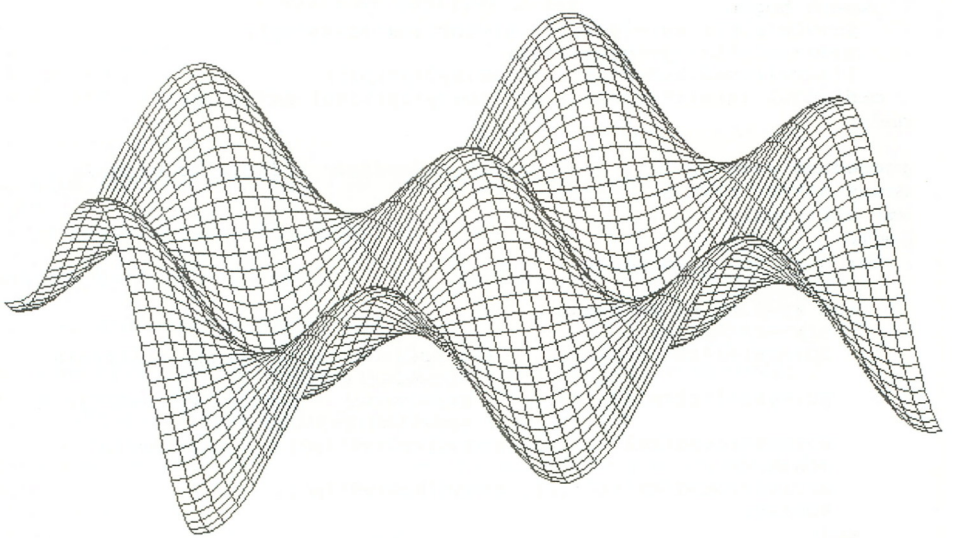
```
.............................
begin
.............................
  setcolor(13); esesvonal;
.............................
end.
```

A *2.23.ábrán* az

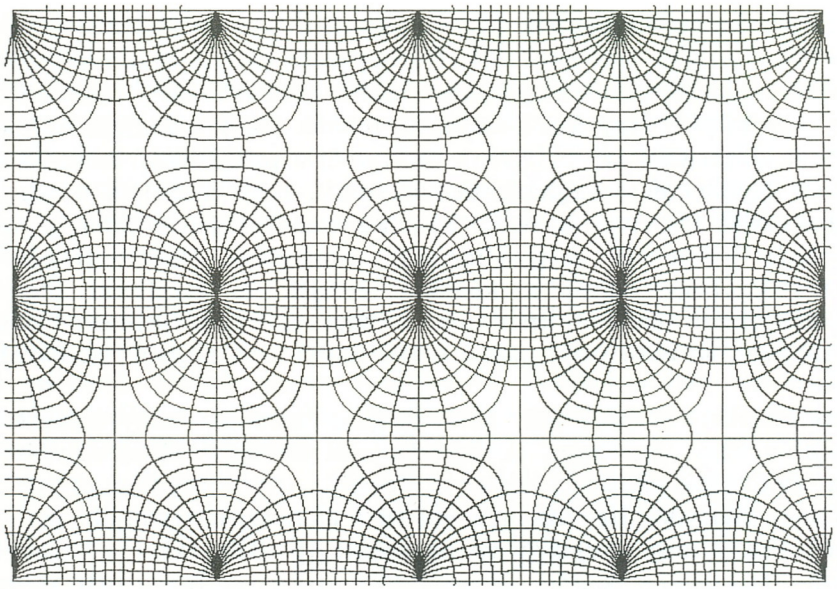
$$f:[-2.4,2.4]\times[-2.4,2.4]\to\Re \quad ; \quad f(x,y)=\frac{5xy}{(x^4+1)(y^4+1)}$$

függvény szintvonal-esésvonal-térképe látható, ahogyan azt a SZEV2.PAS program megrajzolta. A függvény axonometriás képe a 2.4.1.3. alpontban a *2.13. ábrán* látható.

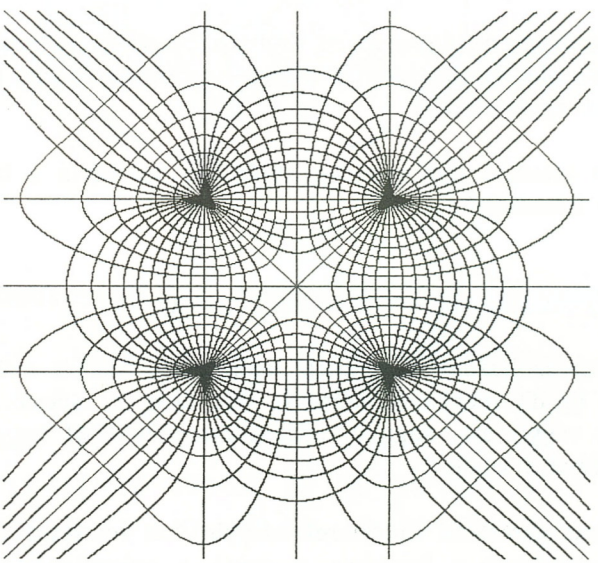
A *2.22. és 2.23. ábrákon* jól megfigyelhető a szintvonalak és az esésvonalak egymáshoz viszonyított merőlegessége.



2.21. ábra



2.22. ábra



2.23. ábra

2.5. Poliéderek rajzolása

A poliéder (soklap) síksokszögek által határolt test. Felszíne a teret két részre osztja, egy belső és egy külső részre. A felszínt alkotó sokszögeket lapoknak nevezzük. A poliéder lapjait egyenes szakaszok, a poliéder élei határolják. Egy poliédert egyértelműen meghatározhatunk azzal, hogy megadjuk a csúcsainak koordinátáit és azt, hogy az egyes lapokhoz melyik csúcsok tartoznak (ezáltal a poliéder éleit is meghatároztuk). Ábrázolását legkönnyebben úgy végezhetjük, hogy lapjait a képsíkra vetítjük és megfelelő sorrendben megjelenítjük. A sorrend azért fontos, hogy a képen létrejövő takarások megfeleljenek az illető megfigyelési irány mellett a valóságban létezőknek.

A poliédereket kétféleképen ábrázolhatjuk, aszerint, hogy átlátszóaknak tekintjük-e őket vagy nem. A második eset az egyszerűbb, ugyanis ha a takart élek nem látszanak, akkor a fentebb vázolt rajzolási eljárás teljes mértékben kielégítő. Ha viszont a poliédert átlátszónak tekintjük, és a hátsó (takart) éleit is meg akarjuk húzni (szaggatott vonallal), akkor a látható lapok megjelenítése után a poliéder összes élét megrajzoljuk (ha a már megrajzolt élekre rárajzolunk egy szaggatott vonalat, az nem fog látszani, ezért nem szükséges itt az éleket láthatóságuk szerint megkülönböztetni).

Ezenkívül használatos az úgynevezett drótvázas rajzolás is, amikor egyszerűen a poliéder összes élét lerajzoljuk. Ez a legkönnyebb ábrázolási mód, azonban a kapott kép nem egyértelmű.

A poliédereket célszerű felülnézetben ábrázolni, biztosítva a bármely irányba való elforgatás lehetőségét.

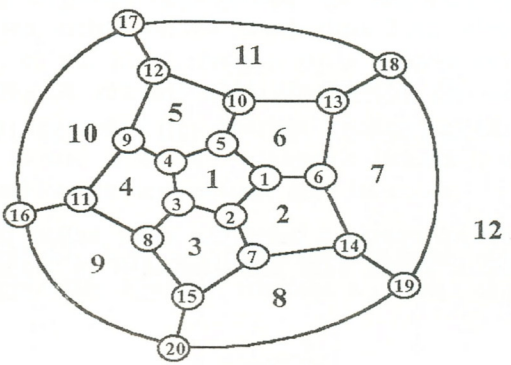
2.5.1. Csúcsok, élek, lapok - a poliéder topológiája

A poliéderek ábrázolásához a csúcsok koordinátái mellett szükség van ezek topológiájának (gráf) meghatározására is. Ez alatt a csúcsok összekötésének módját értjük, vagyis azt, hogy a csúcsok összekötéséből hogyan származnak az élek és ezekből a lapok.

A fentebb vázolt ábrázolási módszerek alapján, két mátrixra van szükségünk. Az egyik a csúcsoknak a lapokhoz, a másik a csúcsoknak az élekhez való

hozzátartozását tartalmazza. Átlátszatlan soklap ábrázolásánál elegendő az első, átlátszó soklap esetén pedig mind a kettő szükséges. Igaz, hogy a csúcsok-élek mátrix felírását elkerülhetjük azzal, hogy a lapok-élek mátrix és ***drawpoly*** utasítás segítségével húzzuk meg az éleket, így azonban minden él kétszer rajzolódik és megtörténhet (ritkán), hogy a kerekítések miatt a két szaggatott vonal nem esik egybe ezért a hátsó él is folytonos lesz. Ha viszont a látható lapok ábrázolásakor vastagabb vonalat használunk, akkor ez a hiányosság sem vezethet félreértéshez.

A csúcsok összekötési módját, valamint a csúcsok, élek és lapok számozását grafikusan is elvégezhetjük. A keletkezett ábrát gráfnak nevezzük és a szóban forgó mátrixok felírását nagyon megkönnyíti. Például a *2.25. ábrán* látható dodekaéder gráfja a *2.24. ábra.* A csúcsok helyén a körök állnak, az élek sorszámát pedig nem tüntettük fel.



2.24. ábra

A csúcsok és lapok közötti összerendelést az alábbi mátrix határozza meg:

$$\begin{bmatrix} 5 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 16 \\ 4 & 2 & 3 & 4 & 5 & 1 & 14 & 15 & 11 & 12 & 13 & 17 \\ 3 & 7 & 8 & 9 & 10 & 6 & 19 & 20 & 16 & 17 & 18 & 18 \\ 2 & 14 & 15 & 11 & 12 & 13 & 18 & 19 & 20 & 16 & 17 & 19 \\ 1 & 6 & 7 & 8 & 9 & 10 & 13 & 14 & 15 & 11 & 12 & 20 \end{bmatrix} \tag{2.52}$$

A mátrix *k*-adik oszlopa a *k* sorszámú lap csúcsait tartalmazza. A csúcsok sorrendjét az egyes oszlopokon belül úgy kell megadni, hogy a laphoz rendelt normálisvektor a poliéder külseje felé mutasson, vagyis a *2.9. ábrán* felénk. Ez azt jelenti, hogy a csúcsokat az óra járásával megegyező körben járási sorrendben kell megadni, kivéve a 12-es lapot, ezt ugyanis hátulról látjuk.

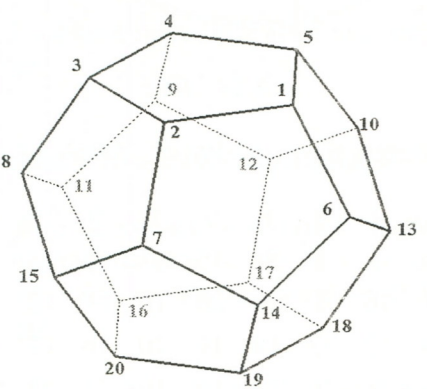
Általános poliéder esetén a fenti mátrix zérus (tulajdonképpen közömbös) elemeket is tartalmaz, ugyanis általában nem mindegyik lap ugyanannyi oldalú sokszög. Az ábrázoláshoz meg kell adni a lapok csúcsainak számát is, különben is a ***fillpoly*** utasítás ezt mint paramétert megköveteli.

A csúcsok-élek megfeleltetést a

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 3 & 4 & 5 & 1 & 6 & 7 & 8 & 9 & 10 & 14 & 15 & 11 & 12 & 13 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 13 & 14 & 15 & 11 & 12 & 16 & 17 & 18 & 19 & 20 & 17 & 18 & 19 & 20 & 16 \end{bmatrix} \tag{2.53}$$

mátrix adja meg, amelyet csupán a helyszűke miatt kellett kettéosztva felírni. Itt is a *k*-adik oszlop a *k*-adik élen levő csúcsokat (a *k*-adik él végpontjait) tartalmazza.



2.25. ábra

2.5.2. Konvex poliéderek

Konvexnek nevezzük azokat az alakzatokat, amelyeknek bármely két pontját összekötő szakasz az illető alakzathoz tartozik. Poliéderek esetében ezt a tulajdonságot úgy is megvizsgálhatjuk, hogy megnézzük, a poliéder minden átlója (az élektől különböző, a csúcsokat összekötő szakaszok) a poliéder belsejéhez (legfeljebb a felszínéhez - ezek a lapok átlói) tartozik-e. Amint látni fogjuk, a konvex poliédereket a legkönnyebb ábrázolni.

2.5.2.1. Lapok, élek láthatósága

Konvex poliéderek ábrázolásakor nem jöhet létre részleges takarás, egy-egy lap vagy teljes egészében látszik, vagy egyáltalán nem. A láthatóság megállapításához pedig elegendő meghatározni azt, hogy az illető lapnak az adott vetítési irány mellett melyik oldalát látjuk (lásd 2.3.2. alpont). Ha a lap belső oldala van felénk, az azt jelenti, hogy a lap a poliéder túloldalán van és nem látszik. Amelyik lapnak a külső oldala van felénk azt pedig teljes egészében látjuk, hiszen ha egy másik lap részben eltakarná, akkor a poliéder nem lenne konvex. A látható lapok élei ugyancsak láthatóak, azok az élek pedig, amelyek egy látható lapnak sem oldalai, nem láthatóak.

A fentiekből következik, hogy a rajzolás során a lapok megjelenítési sorrendje közömbös, a megjelenítés egyetlen feltétele a lapok irányításából következő láthatóság.

2.5.2.2. Példák

a) Szabályos ikozaédert rajzol (és forgat) az alábbi programrészlet:

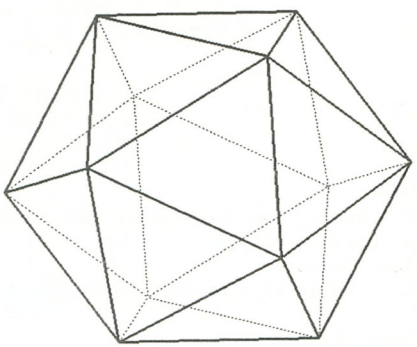
```
const
  e: array[1..20,1..3] of byte =      { lapok-csúcsok összerendelés }
    ((1, 3, 2), (1, 4, 3), (1, 5, 4), (1, 6, 5), (1, 2, 6),
     (2, 3, 10), (3, 4, 11), (4, 5, 7), (5, 6, 8), (6, 2, 9),
     (2, 10, 9), (3, 11, 10), (4, 7, 11), (5, 8, 7), (6, 9, 8),
     (7, 8, 12), (8, 9, 12), (9, 10, 12), (10, 11, 12), (11, 7, 12));
  f: array[1..30,1..2] of byte =        { lapok-élek összerendelés }
     ((1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (3, 4), (4, 5),
     (5, 6), (6, 2), (2, 10), (3, 11), (4, 7), (5, 8), (6, 9),
     (3, 10), (4, 11), (5, 7), (6, 8), (2, 9), (10, 9), (11, 10),
     (7, 11), (8, 7), (9, 8), (7, 12), (8, 12), (9, 12), (10, 12),
     (11, 12));
```

```
  kx0=320; ky0=240; alf=pi/240;
  fel = #72;  le = #80;  bal = #75;  jobb = #77; poz = #43;
  neg = #45;

type

  pont = object
    x, y: integer;
    procedure uj(ux, uy: integer); end;

  csucs = object
    xx, yy, zz: real; { csúcsok térbeli koordinátái }
    csp: pont;        { csúcsok vetületeinek képernyőkoordinátái }
    procedure uj(ux, uy, uz: real);
    procedure vt;
  end;

  hmsz = object
    cs: array[1..3] of csucs;
    s: array[1..3] of pont;
    lth: boolean;
    procedure uj(k: byte);
    procedure lt;
  end;

var
  gcs: array[1..12] of csucs;
  lp: array[1..20] of hmsz;
  ri, r1i, h1i: real;
  k, l: Integer;
  kr: char;
.....................................

procedure pont.uj(ux, uy: integer);
begin x:=ux; y:=uy; end;

procedure csucs.uj(ux, uy, uz: real);
begin xx:=ux; yy:=uy; zz:=uz; end;

procedure csucs.vt;             { vetítés (felülnézet) }
begin csp.uj(round(kx0+xx), round(ky0-yy)); end;

procedure hmsz.uj(k: byte);
begin for l:=1 to 3 do cs[l]:=gcs[e[k,l]]; end;

procedure hmsz.lt;         { láthatósági feltételt vizsgál }
begin
  if (cs[2].xx-cs[1].xx)*(cs[3].yy-cs[1].yy)-
     (cs[3].xx-cs[1].xx)*(cs[2].yy-cs[1].yy) > 0
  then lth:=true else lth:=false; end;

procedure rajz;
begin
  for k:=1 to 12 do gcs[k].vt;       { képsíkra vetíti a csúcsokat }
```

```
  for k:=1 to 20 do begin
     lp[k].uj(k); lp[k].lt; end;      { megszerkeszti a lapokat és  }
  bar(0,0,640,480);                   { megállapítja láthatóságukat }
  for k:=1 to 20 do
    if lp[k].lth=true then begin
      for l:=1 to 3 do                { a megjelenítéshez szükséges }
        lp[k].s[l]:=lp[k].cs[l].csp; { objektum aktualizálása      }
      fillpoly(3, lp[k].s); {látható lap vetületének megjelenítése }
    end;
  setlinestyle(1,0,1);
  for k:=1 to 30 do                   { minden élet meghúz szaggatott    }
                                      { vonallal - a látható élek esetén }
    line(gcs[f[k,1]].csp.x, gcs[f[k,1]].csp.y,
         gcs[f[k,2]].csp.x, gcs[f[k,2]].csp.y);
                                      { nincs változás, a hátsó élek     }
  setlinestyle(0,0,3);                { megjelennek }
  fg:=1;
end;

procedure forgat;
begin
  repeat begin
    repeat kr:=readkey;
    until (kr=fel) or (kr=le) or (kr=jobb) or (kr=bal) or (kr=poz) or
          (kr=neg) or (ord(kr)=13);
    if kr<>chr(13) then begin
    for k:=1 to 12 do { térbeli koordinátatengelyek körüli forgatás }
      case kr of
                { x tengely körül }
      le: gcs[k].uj(gcs[k].xx, gcs[k].yy*cos(alf)-gcs[k].zz*sin(alf),
          gcs[k].yy*sin(alf)+gcs[k].zz*cos(alf));
      fel:gcs[k].uj(gcs[k].xx, gcs[k].yy*cos(alf)+gcs[k].zz*sin(alf),
          -gcs[k].yy*sin(alf)+gcs[k].zz*cos(alf));
                { y tengely körül }
      bal:gcs[k].uj(gcs[k].xx*cos(alf)-gcs[k].zz*sin(alf), gcs[k].yy,
          gcs[k].xx*sin(alf)+gcs[k].zz*cos(alf));
      jobb:gcs[k].uj(gcs[k].xx*cos(alf)+gcs[k].zz*sin(alf),gcs[k].yy,
          -gcs[k].xx*sin(alf)+gcs[k].zz*cos(alf));
                { z tengely körül }
      neg:gcs[k].uj(gcs[k].xx*cos(alf)+gcs[k].yy*sin(alf),
          gcs[k].xx*sin(alf)+gcs[k].yy*cos(alf), gcs[k].zz);
      poz:gcs[k].uj(gcs[k].xx*cos(alf)-gcs[k].yy*sin(alf),
          gcs[k].xx*sin(alf)+gcs[k].yy*cos(alf), gcs[k].zz);
      end;
    rajz; end;
  end; until ord(kr)=13;
end;

begin
  ri:=220; r1i:=0.89443*ri; h1i:=0.44721*ri;
.........................................
setlinestyle(0,0,3); setfillstyle(1,0);
gcs[1].uj(0, 0, -ri); gcs[12].uj(0, 0, ri);
```

```
                { ikozaéder csúcsainak meghatározása }
  for k:=1 to 5 do begin
  gcs[k+1].uj(r1i*cos((2*k-3)*pi/5),r1i*sin((2*k-3)*pi/5),-h1i);
  gcs[k+6].uj(-r1i*cos((2*k-3)*pi/5),-r1i*sin((2*k-3)*pi/5),h1i); end;
  rajz; forgat;
.....................................
end.
```

A program lehetővé teszi az ábrázolt tárgy billentyűzetről vezérelt forgatását. Megtalálható a lemezmellékleten, FORG20FF.PAS néven. Az eredmény a *2.26. ábrán* látható.



2.26. ábra

2.5.3. Konkáv és csillagszerű poliéderek

Konkáv poliéderek ábrázolásakor már jöhet létre részleges vagy akár teljes takarás is, abban az értelemben, hogy a megfigyelő felé külső oldalát mutató lap nem egészen, vagy egyáltalán nem látszik, mert a poliéder másik lapja eltakarja. Az ábrázolási eljárás szempontjából megkülönböztetjük a csillagszerű poliédereket, amelyekre még írható viszonylag gyors, a megjelenítési sorrendet meghatározó algoritmus. Azokat a poliédereket nevezzük csillagszerűeknek, amelyeknek belsejében létezik legalább egy pont (általában egy belső tartomány), ahonnan a poliéder minden lapja belülről, takarás nélkül látható. Másképp fogalmazva, a szóban forgó pontot a felszín bármely pontjával össze lehet kötni egy kizárólag a poliéder belsejében fekvő szakasszal.

2.5.3.1. Rajzolási sorrend

Csillagszerű poliéderek esetében a rajzolási sorrendet úgy kell meghatározni, hogy a vetítési irányból nézve "*messzebb*" levő lapokat előbb, a "*közelebbieket*" pedig később jelenítsük meg. Az idézőjelbe tett helymeghatározás pontos matematikai megfelelője az illető lapok súlypontjának a képsíkhoz viszonyított magassága. A csúcsok képsíkhoz viszonyított magassága tetszőleges vetítési irány esetén a (2.9) egyenlettel számítható ki, felülnézetben pedig ezek z koordinátáival egyenlő. Ha a poliéder lapjai háromszögek, akkor a súlypont magassága a csúcsok magasságának középarányosa.

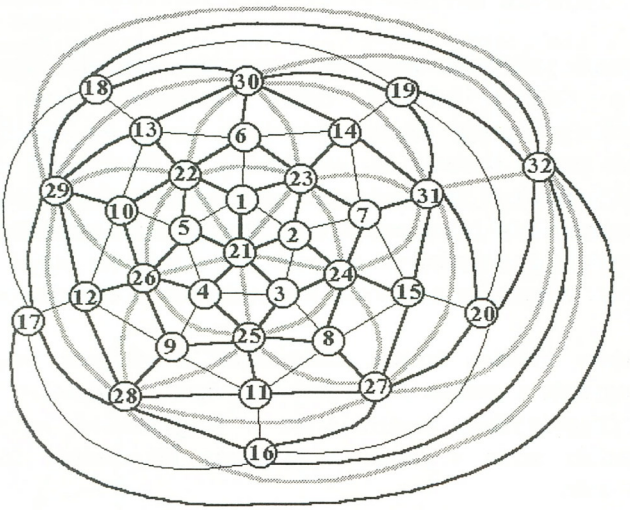
A poliéder minden lapjára kiszámítjuk a súlypont magasságát (negatív is lehet, ugyanis a szóban forgó képsík mögötti részleteket is ábrázoljuk), majd ennek függvényében megállapítjuk a megjelenítési sorrendet, úgy, hogy a megjelenítés során a lapok a magasság növekedése szerint következzenek egymás után.

2.5.3.2. Példák

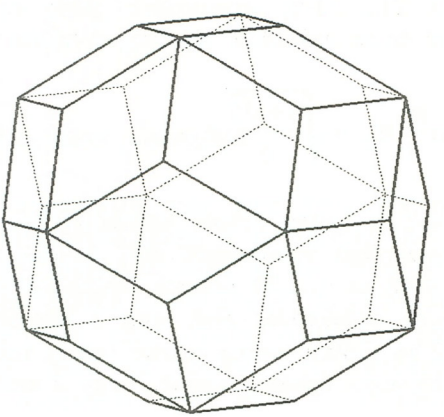
a) Tekintsük a *2.27. ábrán* látható gráfot. Az 1,...,20 jelölésű csúcsok egy dodekaéder csúcsai, a 21,...,32-ig számozottak pedig egy ikozaéderéi. Ha a dodekaéder R_{12} sugara és az ikozaéder R_{20} sugara között fennáll az

$$R_{20} = \frac{\sqrt{5}+1}{\sqrt{3}} \sin\frac{\pi}{5} R_{12} = \sqrt{\frac{5+\sqrt{5}}{6}} R_{12} \tag{2.54}$$

összefüggés, akkor csak a vastag fekete kötéseket használva a *2.28. ábrán* látható, 30 darab egybevágó rombuszból álló, konvex poliédert kapjuk. A rombuszok rövidebb átlói az 1,...,20 csúcsok összekötéséből (a gráfon vékony fekete vonal) származó dodekaéder élei, míg a hosszabb átlók a 21,...,32 csúcsok összekötéséből (a gráfon vastag szürke vonal) származó ikozaéder élei. A fenti összefüggés lényeges feltétele annak, hogy a szóban forgó négyszögek síkidomok legyenek.



2.27. ábra

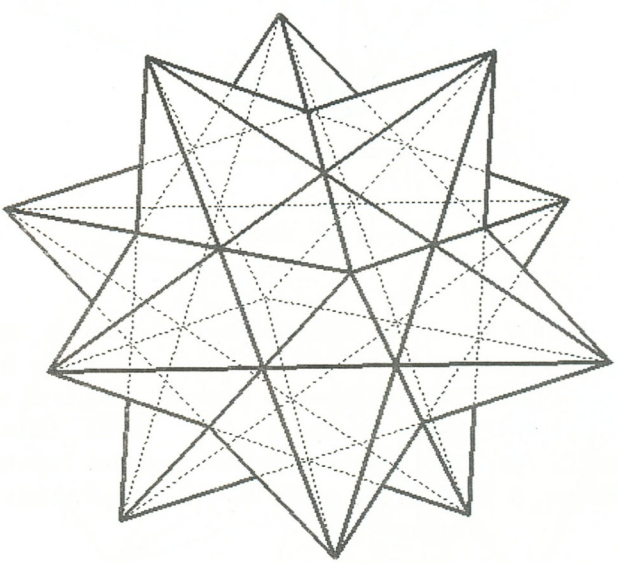


2.28. ábra

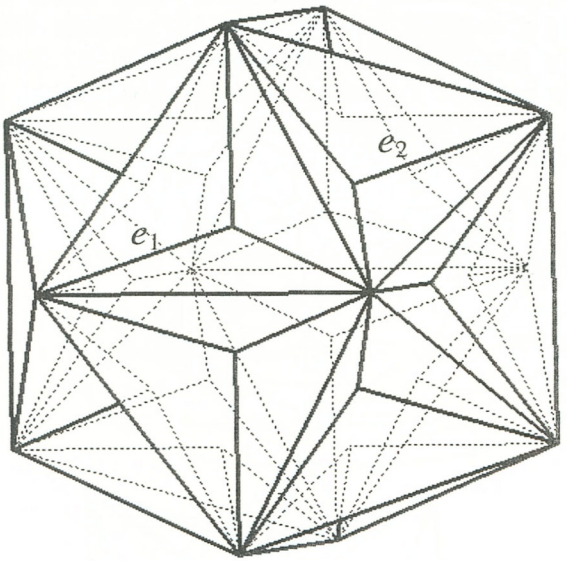
Ha az ikozaédernek a dodekaéderéhez viszonyított sugara nagyobb, mint ami a (2.54) összefüggésből adódik, akkor a gráf vastag fekete és vékony fekete vonallal meghúzott kötéseit használva a *2.29. ábrán* láthatóhoz hasonló csillagot, a vastag fekete és szürke kötésekkel pedig a *2.30. ábrán* láthatóhoz hasonló poliédert nyerünk. A szóban forgó két ábra sajátos esetet szemléltet, ugyanis a *2.29. ábrán* három-három, a *2.30. ábrán* pedig két-két él (például e_1 és e_2) kollineáris. Ezt az

$$R_{20} = \frac{\sqrt{5}+1}{2} \cdot \sqrt{\frac{5+\sqrt{5}}{6}} R_{12} \tag{2.55}$$

összefüggés biztosítja.



2.29.ábra

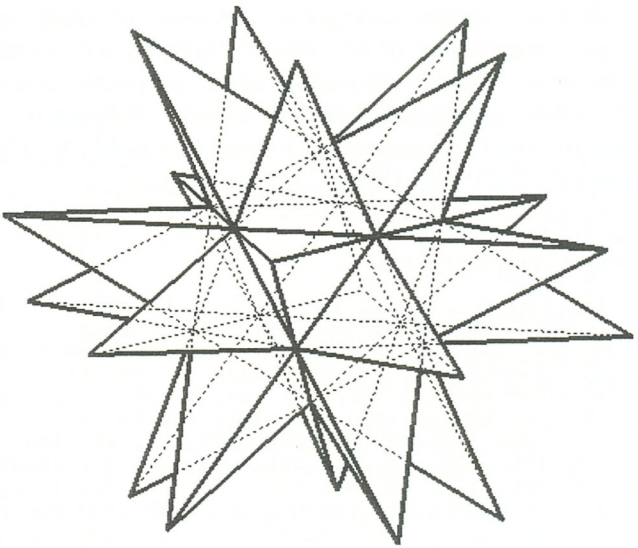


2.30. ábra

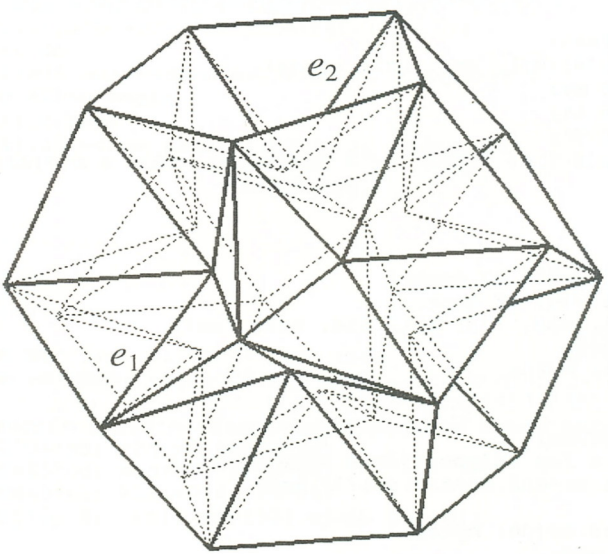
Ha viszont a dodekaédernek az ikozaéderéhez viszonyított sugara nagyobb, mint ami a (2.54) összefüggésből adódik, akkor a gráf vastag fekete és vastag szürke kötéseivel a *2.31. ábrán* láthatóhoz hasonló csillaghoz, a vastag fekete és vékony fekete kötésekkel pedig a *2.32. ábrán* láthatóhoz hasonló poliéderhez jutunk. Az ábrák ugyancsak sajátos eseteket képviselnek, a *2.29. ábrán* három-három, a *2.32. ábrán* pedig két-két él kollineáris (például e_1 és e_2), ami az

$$R_{20} = \frac{3-\sqrt{5}}{2} \cdot \sqrt{\frac{5+\sqrt{5}}{6}} R_{12} \tag{2.56}$$

összefüggés teljesülésével valósul meg.



2.31.ábra



2.32.ábra

A *2.29...32. ábrákon* látható csillagokat egyetlen program segítségével is ábrázolhatjuk, ha a romboéder (*2.28. ábra*) topológiájának megfelelő mátrixból mindig a keresett változat háromszög alakú lapjainak csúcsait választjuk ki, amint az az alábbi programrészletből is kiderül. Kihagytuk a programnak a 2.5.2.2. alpontban közölt programéval azonos részleteit. A teljes program a lemezmellékleten ATLCSG.PAS néven található meg.

```
const e: array[1..30,1..4] of byte =
      {a romboéder topológiája }

      ((21, 2, 23, 1), (21, 3, 24, 2), (21, 4, 25, 3),
      (21, 5, 26, 4), (21, 1, 22, 5), (22, 1, 23, 6),
      (23, 2, 24, 7), (24, 3, 25, 8), (25, 4, 26, 9),
      (26, 5, 22, 10), (22, 6, 30, 13), (23, 14, 30, 6),
      (23, 7, 31, 14), (24, 15, 31, 7), (24, 8, 27, 15),
      (25, 11, 27, 8), (25, 9, 28, 11), (26, 12, 28, 9),
      (26, 10, 29, 12), (22, 13, 29, 10), (27, 11, 28, 16),
      (28, 12, 29, 17), (29, 13, 30, 18), (30, 14, 31, 19),
      (31, 15, 27, 20), (27, 16, 32, 20), (28, 17, 32, 16),
      (29, 18, 32, 17), (30, 19, 32, 18), (31, 20, 32, 19));

hmsz = object
    cs1, cs2, cs3: csucs;
    s: array[1..3] of pont;
    z: real;    { háromszög súlypontjának magassága }
    n: byte;
    lth: boolean;
    procedure uj(us1, us2, us3: csucs);
    procedure ms;
    procedure lt;
    procedure vt;
    procedure sr(un: byte);  { a háromszög száma a megjelenítési }
                             { sorrendben }
end;

var
  gcs: array[1..32] of csucs;
  lp: array[1..60] of hmsz;
  ri, rd, r1i, r1d, r2d, h1i, h1d, h2d: real;
  k, l: Integer;
  fl, fg, vlt: byte;
.....................................

procedure hmsz.ms;
{ kiszámítja a lap súlypontjának magasságát }
begin z:=(cs1.zz+cs2.zz+cs3.zz)/3; end;

procedure hmsz.sr(un: byte);
begin n:=un; end;
```

```
procedure sorrend;
{ a lapokat súlypontjuk magassága szerinti növekvő }
var sf: byte;          { sorrendbe helyezi }
begin
  repeat begin
    sf:=0;
    for k:=1 to 60 do
      for l:=1 to 60 do
        if (lp[l].z-lp[k].z)*(lp[l].n-lp[k].n)<0 then begin
          fl:=lp[k].n; lp[k].sr(lp[l].n); lp[l].sr(fl); sf:=1; end;
  end; until sf=0;
end;

procedure rajz;
begin
  for k:=1 to 30 do begin
    case vlt of
    1,3:begin
      { változatok szerinti topológia kiválasztása }
      lp[k].uj(gcs[e[k,1]], gcs[e[k,2]], gcs[e[k,4]]);
      lp[30+k].uj(gcs[e[k,2]], gcs[e[k,3]], gcs[e[k,4]]); end;
    2,4:begin
      lp[k].uj(gcs[e[k,1]], gcs[e[k,2]], gcs[e[k,3]]);
      lp[30+k].uj(gcs[e[k,1]], gcs[e[k,3]], gcs[e[k,4]]); end;
    end;
    lp[k].ms; lp[k].lt; lp[k].vt;
    lp[30+k].ms; lp[30+k].lt; lp[30+k].vt; end;
  if fg=0 then for k:=1 to 60 do lp[k].sr(k);
  sorrend; bar(0,0,640,480); setlinestyle(0,0,3);
  for l:=1 to 60 do
  { a megállapított sorrendben megjeleníti a lapokat }
    for k:=1 to 60 do begin
      if l=lp[k].n then
        if lp[k].lth=true then fillPoly(3, lp[k].s);
    end;
  fg:=1;
end;
.....

begin
.....
   case vlt of
      1,4:begin rd:=125; ri:=1.777*rd; end;
      2,3:begin rd:=200; ri:=0.42*rd; end;
      end;
          { csúcsok koordinátáinak meghatározása kezdő helyzetben }
      r1d:=0.60706*rd; r2d:=0.98225*rd;
      h1d:=0.79465*rd; h2d:=0.18759*rd;
      r1i:=0.89443*ri; h1i:=0.44721*ri;
      gcs[21].uj(0, 0, -ri); gcs[32].uj(0, 0, ri);
```

```
for k:=1 to 5 do begin
  gcs[k+21].uj(r1i*cos((2*k-3)*pi/5),
               r1i*sin((2*k-3)*pi/5), -h1i);
  gcs[k+26].uj(-r1i*cos((2*k-3)*pi/5),
               -r1i*sin((2*k-3)*pi/5), h1i); end;
for k:=1 to 5 do begin
  gcs[k].uj(r1d*cos(2*(k-1)*pi/5),
            r1d*sin(2*(k-1)*pi/5), -h1d);
  gcs[k+5].uj(r2d*cos(2*(k-1)*pi/5),
              r2d*sin(2*(k-1)*pi/5),h2d);
  gcs[k+10].uj(-r2d*cos(2*(k-1)*pi/5),
               -r2d*sin(2*(k-1)*pi/5),h2d);
  gcs[k+15].uj(-r1d*cos(2*(k-1)*pi/5),
               -r1d*sin(2*(k-1)*pi/5), h1d); end;
```

A fenti programban a változatoknak (*vlt*), 1-től 4-ig, rendre a *2.29, 2.31, 2.32* illetve *2.30. ábra* felel meg.

b) A tárgy tetszőleges tengely körüli forgatását végzi az alábbi eljárás, amelyben az *f* egy byte típusú változó és az éppen használatban levő ablak számát tartalmazza. Megjegyzendő, hogy a grafikus meghajtó ***vgamed*** üzemmódban kell legyen és a képernyő-koordináták számításánál szem előtt kell tartani a ***vgahi*** üzemmódtól különböző felbontást. Az elfordítást követő újrarajzolás a hátsó, nem látható lapon történik, majd a kész rajz jelenik meg. A kérdéssel a 4. fejezetben bővebben foglalkozunk.

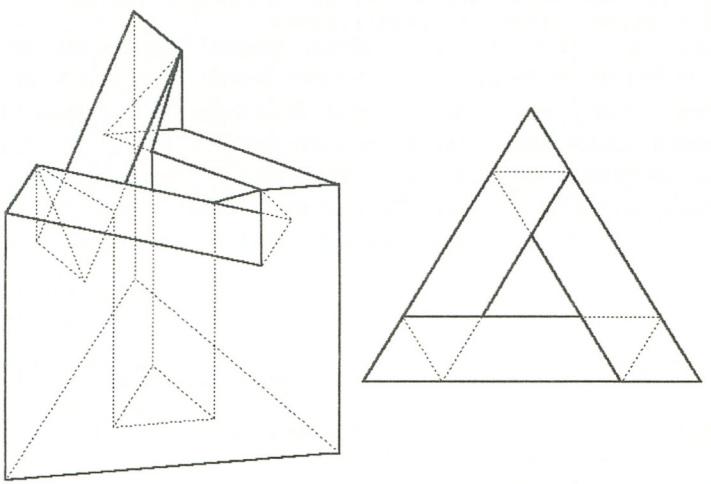
```
procedure forgat;
begin
  f:=1;
  repeat begin
    if f=1 then f:=0 else f:=1;
    setactivepage(f);
    for k:=1 to ncs do begin { ncs a poliéder csúcsainak száma }
                             { kiszámítja az elforgatott poliéder }
                                    { csúcsainak új koordinátáit }
      gcs[k].uj(a11*gcs[k].xx+a12*gcs[k].yy+a13*gcs[k].zz,
                a21*gcs[k].xx+a22*gcs[k].yy+a23*gcs[k].zz,
                a31*gcs[k].xx+a32*gcs[k].yy+a33*gcs[k].zz);
    end;
    rajz;                    { az aktív lapra rajzol }
    setvisualpage(f);        { a kész rajzot megjeleníti }
  end; until keypressed;     { billentyű leütéséig folyamatosan }
end;                         { forgatja a tárgyat }
```

A forgatási képlet (lásd 2.27. egyenletek) együtthatóit a program elején, a forgástengely irányítényezőinek (af, bf, cf) beolvasása után, a következő eljárás számítja ki:

```
procedure egyutthato;
var nvz: real;
begin
  nvz:=af*af+bf*bf+cf*cf;
  a11:=cos(dt)+af*af*(1-cos(dt))/nvz;
  a12:=(-cf*sqrt(nvz)*sin(dt)+af*bf*(1-cos(dt)))/nvz;
  a13:=(bf*sqrt(nvz)*sin(dt)+af*cf*(1-cos(dt)))/nvz;
  a21:=(cf*sqrt(nvz)*sin(dt)+af*bf*(1-cos(dt)))/nvz;
  a22:=cos(dt)+bf*bf*(1-cos(dt))/nvz;
  a23:=(-af*sqrt(nvz)*sin(dt)+bf*cf*(1-cos(dt)))/nvz;
  a31:=(-bf*sqrt(nvz)*sin(dt)+af*cf*(1-cos(dt)))/nvz;
  a32:=(af*sqrt(nvz)*sin(dt)+bf*cf*(1-cos(dt)))/nvz;
  a33:=cos(dt)+cf*cf*(1-cos(dt))/nvz;
end.
```

2.5.4. Általános poliéderek

Konkáv, nem csillagszerű poliéderek esetében minden olyan lappárra, amelynek tagjai külső oldalukat a megfigyelő felé mutatják (a többi biztos nem látszik), meg kell vizsgálni a takarási viszonyokat, és ennek függvényében megállapítani a rajzolási sorrendet, ami eléggé időigényes eljárás, főleg nagyobb számú lap esetén. Mi több, vannak olyan poliéderek is, amelyekre nem létezik olyan megjelenítési sorrend, amely helyes képet eredményezne, ilyen például a *2.33.a .ábrán* látható soklap *2.33.b* felülnézete. Ezek rajzolásakor már nem lehet kikerülni az élek vetületei közötti metszéspontok kiszámítását.



a *b*

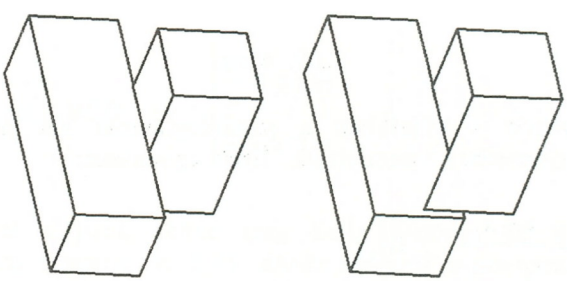
2.33. ábra

2.5.4.1. A takarás kérdése és a rajzolási sorrend

Nem csillagszerű például a *2.34.a ábrán* látható, két hasábból összeillesztett poliéder. Ha ezt olyan eljárással ábrázoljuk, amely a megjelenítési sorrendet a lapok súlypontjának (téglalapnál az átlók metszéspontja) magassága függvényében állapítja meg, akkor bizonyos vetítési irányok esetén előfordul a *2.34.b ábrán* látható téves takarás. Ezt elkerülni úgy lehet, hogy - miután a

látható lapokat a súlypontjuk magassága szerinti rajzolási sorrendbe rendeztük - minden látható lappár esetén megvizsgáljuk, hogy a vetületeik diszjunktake (ez azt jelenti, hogy a vetületeknek nincs közös tartományuk), és ha nem, akkor meghatározzuk, hogy melyik takarja a másikat. Ezt úgy végezzük el, hogy kiválasztjuk az egyik lap csúcsai közül azt, amelyiknek vetülete a másik lap vetületének belsejébe esik és kiszámítjuk, hogy a vetítősugár mentén milyen magasságon helyezkedik el a szóbanforgó csúcs és a másik lapnak az a pontja, amelyikben a csúcson áthaladó vetítősugár a lapot döfi.

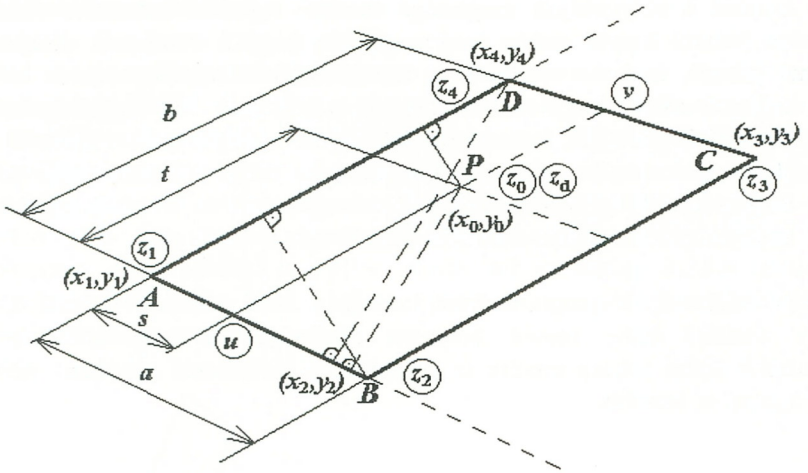
A feladat tehát olyan program írása, amely a *2.34.a ábrán* látható soklapot bármely vetítési irány esetén helyesen ábrázolja. A megoldási elv más, bonyolultabb alakú soklap esetére is érvényes, a kivitelezés azonban nehezebb és a program is lassúbb.



a) *b)*

2.34. ábra

A *2.35. ábrán* látható *ABCD* paralelogramma egy téglalap képsíkra eső vetülete. Ha felülnézetről van szó, akkor a vetítősugarak a z tengellyel párhuzamosak és a csúcsok vetületeinek koordinátái egyenlőek ezek x illetve y koordinátáival. A körökben szereplő koordináták azon pontok magasságai, amelyeknek a vetületei mellett állnak. Az $M(x_0,y_0,z_0)$ pont egy másik lap csúcsa, $N(x_0,y_0,z_d)$ pedig az a pont, amelyben az előbbin átmenő vetítősugár a téglalap síkját döfi. Mindkettő vetülete a $P(x_0,y_0)$ pont.



2.35. ábra

A P pont helyzetét a képsíkban a paralelogramma két nem párhuzamos oldalának tartóegyeneséhez viszonyítjuk. Ezek egyenletei:

$$\begin{cases} \dfrac{y-y_1}{y_2-y_1}=\dfrac{x-x_1}{x_2-x_1} & (d_1) \\ \dfrac{y-y_1}{y_4-y_1}=\dfrac{x-x_1}{x_4-x_1} & (d_2) \end{cases} \tag{2.57}$$

Adott $P(x_0, y_0, z_0)$ pont adott, $ax + by + c = 0$ (e) egyenletű egyenestől való távolságát a következő képlettel számíthatjuk:

$$\mathrm{d}_e P=\frac{ax_0+by_0+c}{\sqrt{a^2+b^2}} \tag{2.58}$$

Figyelembe véve a *2.35. ábrán* látható derékszögű háromszögek közötti hasonlóságokat, az *s/a* és *t/b* arányok számítására a következő képletek adódnak:

$$\begin{cases} \dfrac{s}{a} = \dfrac{(x_4 - x_1)(y_0 - y_1) - (x_0 - x_1)(y_4 - y_1)}{(x_4 - x_1)(y_2 - y_1) - (x_2 - x_1)(y_4 - y_1)} \\ \dfrac{t}{b} = \dfrac{(x_2 - x_1)(y_0 - y_1) - (x_0 - x_1)(y_2 - y_1)}{(x_2 - x_1)(y_4 - y_1) - (x_4 - x_1)(y_2 - y_1)} \end{cases} \tag{2.59}$$

ahol felhasználtuk azt a tényt, hogy az azonos arányban szereplő távolságokat azonos egyeneshez viszonyítjuk, ezért a (2.58) egyenlet nevezőjének megfelelő tényezők leegyszerűsödnek.

Könnyen belátható, hogy a *P* pont akkor és csakis akkor van az *ABCD* paralelogramma belsejében, ha a fenti arányok értékei *0* és *1* között vannak, azaz:

$$(x_0, y_0) \in \operatorname{int} ABCD \quad \Leftrightarrow \quad \begin{cases} \dfrac{s}{a} \in (0,1) \\ \dfrac{t}{b} \in (0,1) \end{cases} \tag{2.60}$$

Ha ez a feltétel teljesül, akkor meg kell állapítani az *M* és *N* pontok magasságai közötti viszonyt. A *2.35. ábrán* szereplő *u* és *v* magasságok az

$$\begin{cases} u = \left(1 - \dfrac{s}{a}\right) z_1 + \dfrac{s}{a} z_2 \\ v = \left(1 - \dfrac{s}{a}\right) z_4 + \dfrac{s}{a} z_3 \end{cases} \tag{2.61}$$

képletekkel, az *N* pont magassága pedig a

$$z_d = \left(1 - \frac{t}{b}\right) u + \frac{t}{b} v \tag{2.62}$$

képlettel számítható ki.

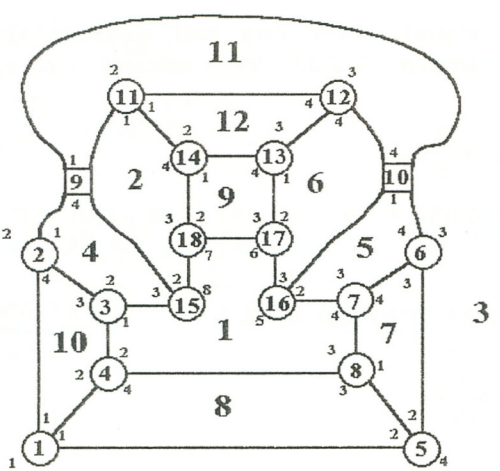
Végeredményben

$$z_d = \left(1-\frac{s}{a}\right)\left(1-\frac{t}{b}\right)z_1 + \frac{s}{a}\left(1-\frac{t}{b}\right)z_2 + \frac{s}{a}\frac{t}{b}z_3 + \left(1-\frac{s}{a}\right)\frac{t}{b}z_4 \qquad (2.63)$$

A kapott eredményt kell összehasonlítani az M csúcs z_0 magasságával annak megállapítására, hogy az a téglalap van-e fölül, amelynek a vetülete az $ABCD$ paralelogramma (ekkor $z_d > z_0$), vagy a poliédernek az a lapja, amelynek az M pont csúcsa (ekkor $z_0 > z_d$).

2.5.4.2. Példák

Az alábbiakban azt a programot ismertetjük, amelynek segítségével a *2.34.a. ábrán* látható test bármely helyzetbe elforgatható és a kapott kép bármely vetítési irány esetén a valóságnak megfelelő. A poliéder gráfja a *2.36. ábrán* látható. A körökbe a csúcsok sorszámát írtuk be, a lapokban a nagy számok a lapok sorszámát a csúcsok körüli apró számok pedig a csúcsok illető lapon belüli sorszámát jelentik. A 9-es és a 10-es csúcsok rajta vannak 3-as, 2-es illetve 6-os lapok egy-egy élén, de nem csúcsai az illető lapoknak.



2.36. ábra

A lemezmellékleten HASAB.PAS néven található program a megjelenítési sorrendet a következő algoritmus segítségével igazítja ki. A program előbb a lapokat súlypontjaik magassága szerinti sorrendbe helyezi. Erre azért van szükség, hogy kiszűrjük azokat az eseteket, amikor két lap között takarás áll fenn annak ellenére, hogy egyik lap csúcsának vetülete sem esik a másik vetületének belsejébe.

```
procedure hb.rajz;
var
  jel: byte;

  procedure sorrjav;
  var
    i: byte;
    kx41, ky41, kx21, ky21, kx01, ky01 : real;
    ar1, ar2, zl: real;

  begin
    for k:=2 to 12 do
      if lp[k].lth=true then
      begin
        kx41:=(lp[k].cs[4].xx-lp[k].cs[1].xx);
        ky41:=(lp[k].cs[4].yy-lp[k].cs[1].yy);
        kx21:=(lp[k].cs[2].xx-lp[k].cs[1].xx);
        ky21:=(lp[k].cs[2].yy-lp[k].cs[1].yy);
        for l:=1 to 12 do
          if l<>k then
          for i:=1 to lp[l].ncs do
          begin
            kx01:=(lp[l].cs[i].xx-lp[k].cs[1].xx);
            ky01:=(lp[l].cs[i].yy-lp[k].cs[1].yy);
            ar1:=(kx41*ky01-kx01*ky41)/(kx41*ky21-kx21*ky41);
            ar2:=(kx21*ky01-kx01*ky21)/(kx21*ky41-kx41*ky21);
                { ha teljesül az összehasonlítási feltétel, }
                { vagyis hogy az l-edik lap  }
                { egyik csúcsának vetülete a k-adik lap }
                { vetületének belsejébe esik: }
            if (ar1>0) and (ar1<1) and (ar2>0) and (ar2<1) then
          begin
          { akkor kiszámítja a döféspont magasságát }
              zl:=(1-ar1)*(1-ar2)*lp[k].cs[1].zz+ar1*
                (1-ar2)*lp[k].cs[2].zz+
                    ar1*ar2*lp[k].cs[3].zz+
                  (1-ar1)*ar2*lp[k].cs[4].zz;
          { összehasonlítja a magasságok sorrendjét }
          { a már fennálló sorrenddel }
```

```
            if (lp[l].cs[i].zz-zl)*(lp[l].n-lp[k].n) < -0.01 then
        begin
                  fl:=lp[k].n; lp[k].sr(lp[l].n); lp[l].sr(fl);
               jel:=1;
            end;
              i:=lp[l].ncs; { ha már megtörtént egy összehasonlítás,}
                            { akkor kilép }
             end;
           end;
        end;
    end;

begin
.....................................................

repeat begin jel:=0; sorrjav; end; until jel=0;
            { addig végzi az igazítást, ameddig azon belül }
            {      már nem történik sorrendváltoztatás      }
  for l:=1 to nlp do
    for k:=1 to nlp do
     begin
      if l=lp[k].n then
       begin
        setfillstyle(1,0);
        if lp[k].lth=true then fillPoly(lp[k].ncs, lp[k].s);
      { a poliéder lapjait a megállapított sorrendben megjeleníti }
       end;
    end;
end;
```

A sorrend összehasonlításnál azért kellett 0 helyett –0.01-et használni, mert különben bizonyos határhelyzetekben végtelen ciklus jött létre. Egy másik egyszerűsítő fogás az, hogy az 1-es számú lapot úgy viszonyítjuk a többihez (amelyek mind téglalapok), hogy ennek a csúcsain átmenő vetítősugarakkal dolgozunk, hogy ne kelljen az ő bonyolultabb alakját figyelembe venni.

2.6. Görbe felületű testek ábrázolása

Többféle módon ábrázolhatunk görbe felületű testeket. Megemlítjük a határvonalas rajzolást, amely valósághű képet ad, abban az értelemben, hogy azok a vonalak kerülnek megrajzolásra, amelyek a valóságban is látszanak (*2.40. ábra*). A felület azon pontjainak vetületeit jelenítjük meg, amelyekben a felület érintősíkja párhuzamos a vetítési iránnyal. Ezek a pontok a felületen az illető vetítési iránynak megfelelő kontúrnak nevezett görbét alkotják, vetületeik pedig a képsíkon a képhatárnak nevezett görbét [4]. A kontúrnak létezhet olyan darabja, amelyik takarás miatt nem látszik (például a *2.40. ábrán* szaggatott vonallal rajzolt rész). Sok esetben azonban a képhatárból nem lehet következtetni a felület alakjára. Ezért célszerű a felületre valamilyen mintát rajzolni, például két kölcsönösen ortogonális görbesereget. Még szemléletesebb képet kapunk, ha a megvilágítási viszonyokat is figyelembe vesszük. Ezt úgy lehet elérni, hogy a felületet apró foltokra osztjuk, amelyeket síkoknak tekintünk. Ha adott egy megvilágítási irány, akkor minden ilyen foltot az általa visszavert fénysugár és a vetítési (megfigyelési) irány közötti szög függvényében világosabbra vagy sötétebbre festünk. Ha a megvilágítást minden irányból egyenletesnek tekintjük, akkor a felületelem normálisa és a vetítési irány közötti szöggel dolgozhatunk. A kapott kép minőségét azonban erősen befolyásolja a rendelkezésre álló árnyalatok viszonylag kis száma.

Egy gyors és célravezető ábrázolási mód a görbe felületű testek poliéderes közelítésén alapszik. A felület képén egy törtvonalsereg jelenik meg, amely megfelelően finom felbontás esetén folytonos görbeseregnek tűnik és a felület alakját jól szemlélteti.

2.6.1. Paraméteres előállítás

A görbe felületű testeket egy zárt felület, vagy több felület egyesítéséből származó zárt felület határolja. A felületeket többféleképpen állíthatjuk elő. Legelterjedtebbek az implicit:

$$F: D \subset \mathfrak{R}^3 \rightarrow \mathfrak{R}, \quad F(x,y,z) = 0 \tag{2.64}$$

és a paraméteres:

$$\begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u,v) \end{cases} ; \qquad (u,v) \in D \subset \Re^2 \tag{2.65}$$

előállítások.

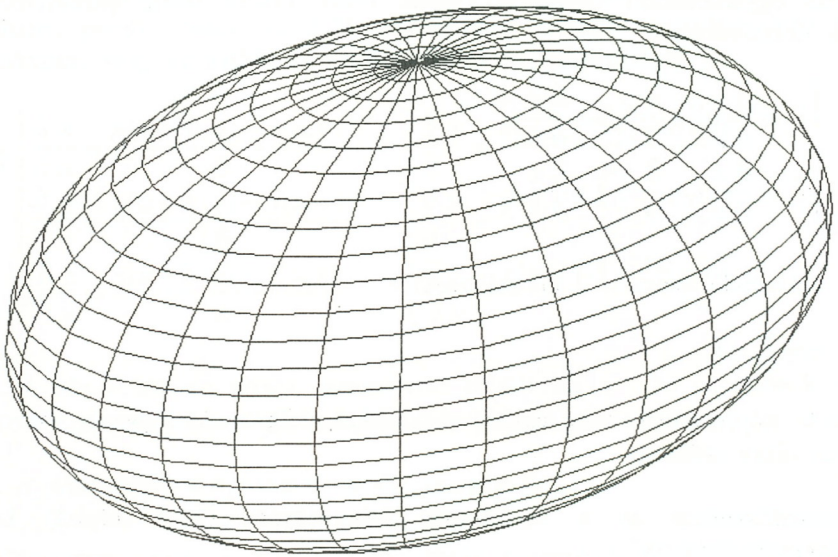
A *2.37. ábrán* látható ellipszoid implicit egyenlete:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1 = 0 \tag{2.66}$$

Ugyanaz a felület paraméteresen a gömbi koordinátarendszer szögei segítségével a következő egyenletekkel állítható elő:

$$\begin{cases} x = a \sin\vartheta \cos\psi \\ y = b \sin\vartheta \sin\psi \\ z = c \cos\vartheta \end{cases} , \qquad \begin{cases} \psi \in [0,2\pi] \\ \vartheta \in [0,\pi] \end{cases} \tag{2.67}$$

A paraméteres előállítás előnye, hogy lehetővé teszi a tájékozódást a felületen. Ha az egyik paramétert állandónak vesszük, akkor a fenti egyenletek egy, a felületen levő görbének az egyenletei. Az így kapott görbéket paramétergöbéknek (paramétervonalaknak) nevezzük. A földgömbön a délkörök és a szélességi körök paramétergörbék. A *2.37. ábrán* az ellipszoid meridián jellegű görbéit és szélességi görbéit közelítik a poliéderes ábrázolásból származó törtvonalak. Forgásellipszoid esetén (*a=b*) a szélességi görbék körök és a meridiángörbék egybevágó ellipszisek. Az ábra a lemezmellékleten található ZEPPELIN.PAS program segítségével készült.



2.37. ábra

2.6.2. Határvonalas ábrázolás

Paraméteres előállítású felületek képhatárát (másnéven képkontúr) viszonylag könnyen megrajzolhatjuk. A felületen felveszünk egy paramétervonalakból álló, eléggé sűrű hálót, amelynek csomópontjaiban kiszámítjuk a felület normálisvektorát. Azokat a pontokat, amelyekben a normálisvektor merőleges (vagy majdnem merőleges) a vetítősugárra kontúrpontoknak tekintjük, levetítjük a képsíkra és megjelenítjük. A zárójelben álló engedményre azért van szükség, mert különben a képkontúr pontjainak túlnyomó részét soha sem jelenítenénk meg. Ennek oka a használt háló véges felbontásában, illetve a számítások véges pontosságában rejlik.

A 2.65 egyenletekkel előállított felület v=áll. illetve u=áll. paramétervonalainak érintővektorai:

$$\begin{cases} \vec{t}_u = \dfrac{\partial x}{\partial u}\vec{i} + \dfrac{\partial y}{\partial u}\vec{j} + \dfrac{\partial z}{\partial u}\vec{k} \\ \vec{t}_v = \dfrac{\partial x}{\partial v}\vec{i} + \dfrac{\partial y}{\partial v}\vec{j} + \dfrac{\partial z}{\partial v}\vec{k} \end{cases} \tag{2.68}$$

Ezek vektoriális szorzata a normálisvektor:

$$\vec{n} = \vec{t}_u \times \vec{t}_v \tag{2.69}$$

A fenti képletekben nem egységvektorokkal dolgozunk, ugyanis erre jelen esetben nincs szükség.

A normálisvektor és a vetítősugár irányvektora (2.35 képlet) közötti merőlegesség feltétele:

$$\vec{p} \cdot \vec{n} = 0 \tag{2.70}$$

vagyis a kontúrpontokban a $\vec{p} \cdot (\vec{t}_u \times \vec{t}_v)$ vegyes szorzat nulla, azaz:

$$\begin{vmatrix} a_p & b_p & c_p \\ \dfrac{\partial x}{\partial u} & \dfrac{\partial y}{\partial u} & \dfrac{\partial z}{\partial u} \\ \dfrac{\partial x}{\partial v} & \dfrac{\partial y}{\partial v} & \dfrac{\partial z}{\partial v} \end{vmatrix} = 0 \tag{2.71}$$

Számítógépes ábrázolás esetén a

$$\left|\vec{p} \cdot (\vec{t}_u \times \vec{t}_v)\right| < \varepsilon \tag{2.72}$$

feltételnek eleget tevő pontok képsíkra eső vetületeit jelenítjük meg, ahol ε egy kicsi, pozitív szám. A 2.71 egyenletben szereplő parciális deriváltakat analitikusan, vagy az alábbi közelítő képletekkel számíthatjuk ki:

$$\begin{cases} \dfrac{\partial x}{\partial u} \approx \dfrac{x(u+\delta,v)-x(u,v)}{\delta} \\ \dfrac{\partial y}{\partial u} \approx \dfrac{y(u+\delta,v)-y(u,v)}{\delta} \\ \dfrac{\partial z}{\partial u} \approx \dfrac{z(u+\delta,v)-z(u,v)}{\delta} \end{cases} \qquad \begin{cases} \dfrac{\partial x}{\partial v} \approx \dfrac{x(u,v+\delta)-x(u,v)}{\delta} \\ \dfrac{\partial y}{\partial v} \approx \dfrac{y(u,v+\delta)-y(u,v)}{\delta} \\ \dfrac{\partial z}{\partial v} \approx \dfrac{z(u,v+\delta)-z(u,v)}{\delta} \end{cases} \tag{2.73}$$

A *2.39. ábrán* látható tórusz paraméteres előállítását a 2.76 egyenletek adják. A parciális deriválást analitikusan végezve, a (2.71) feltétel az

$$a_p \sin\varphi \cos\psi + b_p \sin\varphi \sin\psi + c_p \cos\varphi = 0 \tag{2.74}$$

összefüggéshez vezet. A tórusz *2.40. ábrán* megrajzolt határvonala az alábbi eljárás segítségével készült:

```
procedure rajzol;
const
  m=360; n=600; pim=2*pi/m; pin=2*pi/n;
  rr=11; r=4; s=20;
  kx0=320; ky0=240;
  eps=0.1;
  ap=-3; bp=0; cp=1.5;
var
  k, l: integer;
  nv1, nv2: real;
  x, y, z: real;
  kx, ky: integer;

function fx(k,l: integer): real;      { a paramétervonalak egyenletei }
begin fx:=(rr+r*sin(k*pim))*cos(l*pin); end;
function fy(k,l: integer): real;
begin fy:=(rr+r*sin(k*pim))*sin(l*pin); end;
function fz(k,l: integer): real;
begin fz:=r*cos(k*pim); end;

procedure rzpt;
begin
  x:=fx(k,l); y:=fy(k,l); z:=fz(k,l);      { kontúrpont koordinátái }
  kx:=round(s*(-bp*x+ap*y)/nv1);           { kontúrpont vetítése }
  ky:=round(s*(-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2);
  putpixel(kx0+kx,ky0-ky,15);          {képkontúrpont megjelenítése }
end;
```

```
begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  for l:= 1 to n do
    for k:= 1 to m do
          { megjelenítési feltétel }
      if abs((ap*cos(l*pin)+bp*sin(l*pin))*sin(k*pim)+cp*cos(k*pim))
           < eps then rzpt;
end;
```

A fenti eljárás a képkontúrt teljes egészében meghúzza. A nem látható részek szaggatott vonalas ábrázolása utólagos, WINDOWS/PAINTBRUSH alatti módosítás (törlés) eredménye.

2.6.3. Poliéderes közelítés

Amint már utaltunk rá, a görbe felületű testeket poliéderekkel közelíthetjük ábrázolás céljából. Ez azt jelenti, hogy a felület bizonyos rendszer szerint kiválasztott pontjait (a közelítő poliéder csúcsai) egyenes szakaszokkal összekötjük (a közelítő poliéder élei) és a kapott sokszögeket a már ismert módon, láthatóságuktól függően megjelenítjük. Legkönnyebb a paramétergörbék által képezett háló csomópontjait tekinteni a közelítő soklap csúcsainak. Így készült a *2.37. ábra.* A felületelemek általában görbe oldalú négyszögek, míg a közelítő poliéder lapjai pedig torz (nem egysíkú) négyszögek lesznek. Ez nagy görbület és durva felbontás esetén zavaró lehet. Az ábrázolt test szimmetriája biztosíthatja a négyszögek egysíkúságát. Például gömb esetében a szélességi körök azonos délkörök közötti elemi íveit bezáró húrok párhuzamosak, ebből következően a közelítő poliéder minden lapja síkidom. Ugyanez a helyzet fennáll a *2.39. ábrán* látható gyűrű vagy másnéven tóruszfelület esetében is.

Konvex testek ábrázolásakor a felületelemek megjelenítési sorrendje közömbös, a megjelenítés egyetlen feltétele az, hogy az illető lapok a külső oldalukkal legyenek a megfigyelő felé. Azért itt is vigyázni kell arra, hogy a felületelemek csúcsait a kifelé mutató normálisvektornak megfelelő sorrendbe helyezzük. Nem konvex testek ábrázolásakor a takarási viszonyokat is szem előtt kell tartani.

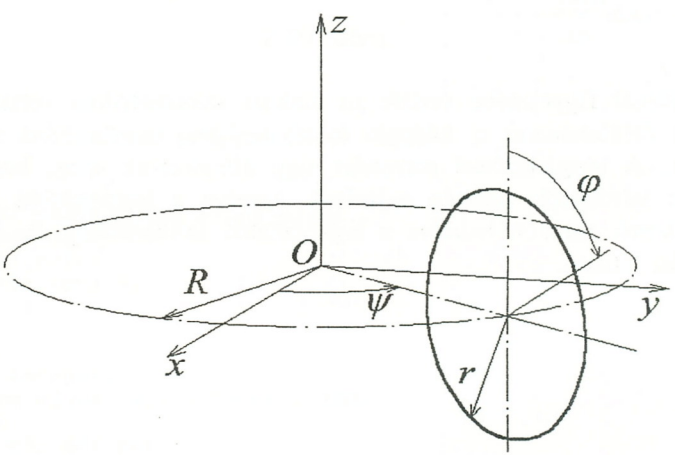
2.6.4. Négyszögű háló - gyűrű ábrázolása

Az alábbiakban egy példával szemléltetjük a paramétergörbék felhasználásával történő poliéderes ábrázolást és a megjelenítési sorrend meghatározását nem konvex test esetén. A *2.38. ábrán* látható kör z tengely körüli forgatásával képzett gyűrűt fogjuk ábrázolni. A leírókör középpontjának pályája az

$$\begin{cases} x_k = R\cos\psi \\ y_k = R\sin\psi \\ z_k = 0 \end{cases} \tag{2.75}$$

egyenletű kör, a keletkezett forgásfelület paraméteres előállítása pedig:

$$\begin{cases} x = (R + r\sin\varphi)\cos\psi \\ y = (R + r\sin\varphi)\sin\psi \\ z = r\cos\varphi \end{cases} ; \quad \begin{cases} \varphi \in [0, 2\pi] \\ \psi \in [0, 2\pi] \end{cases} \tag{2.76}$$



2.38. ábra

A felhasznált paramétergörbék egyenletei:

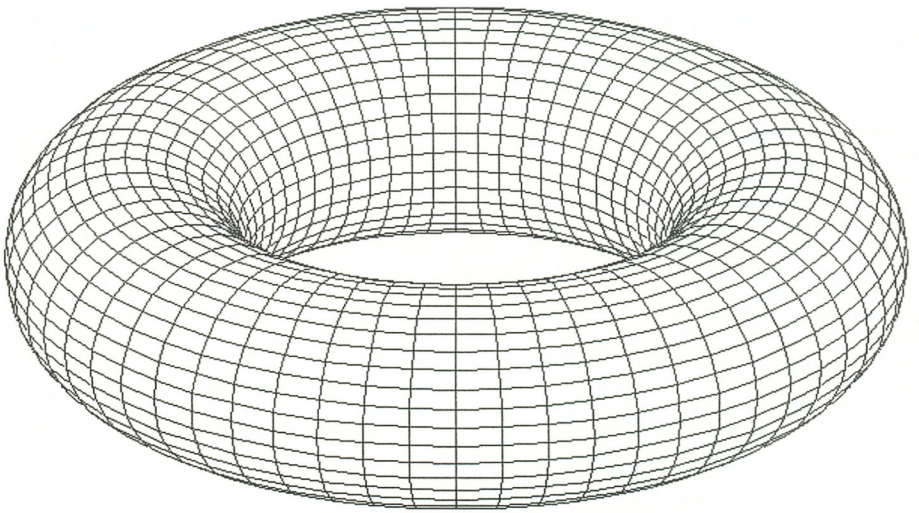
- a leírógörbék:

$$\begin{cases} x = (R + r\sin\varphi)\cos\dfrac{2l\pi}{n} \\ y = (R + r\sin\varphi)\sin\dfrac{2l\pi}{n} \\ z = r\cos\varphi \end{cases} \quad ; \quad \begin{cases} \varphi \in [0,2\pi] \\ l \in \{0,1,\ldots,n-1\} \end{cases} \tag{2.77}$$

- a pályagörbék:

$$\begin{cases} x = (R + r\sin\dfrac{2k\pi}{m})\cos\psi \\ y = (R + r\sin\dfrac{2k\pi}{m})\sin\psi \\ z = r\cos\dfrac{2k\pi}{m} \end{cases} \quad ; \quad \begin{cases} \psi \in [0,2\pi] \\ k \in \{0,1,\ldots,m-1\} \end{cases} \tag{2.78}$$

Az ábrázolásnál figyelembe vettük az alakzat szimmetrikus voltát, az egyszer kiszámított felületelemet a képnek mind a jobb, mind a bal oldalán megjelenítettük. A megjelenítési sorrendet úgy állapítottuk meg, hogy a felületelemeket a leírókörök mentén rajzoltuk, kezdve a legtávolabbi leírókörökkel és a pályakörök mentén haladva a legközelebbi leírókörökig. Az eredmény az alábbi ábrán látható.



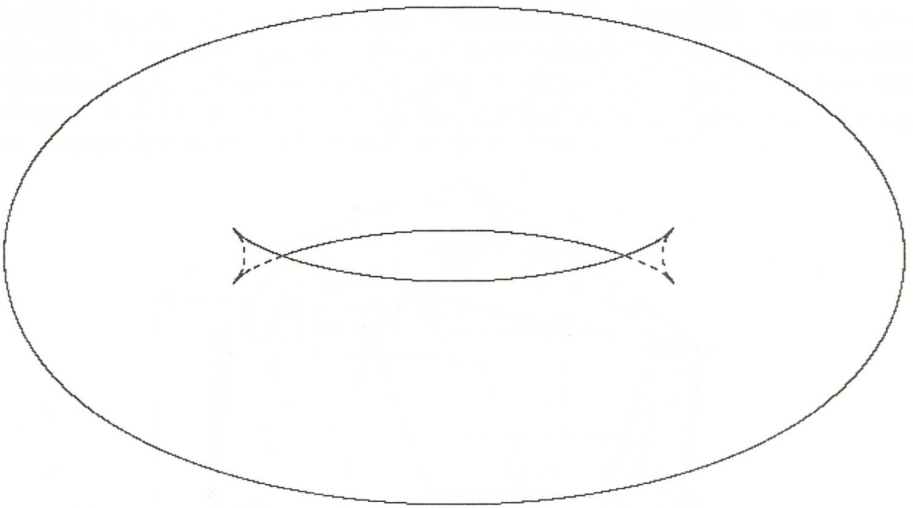
2.39. ábra

Az ábrázolást a következő programrészlet végzi.

```
const
  m=36; n=60; pim=2*pi/m; pin=2*pi/n;
  kx0=320; ky0=240;
  rr=11; r=4; s=20;
  ap=-3; bp=0; cp=1.5;

type
  pt=object
    xp, yp: integer;
    procedure uj(ux, uy: integer); end;
  nsz=object
    c1, c2, c3, c4: pt;
    procedure uj(u1, u2, u3, u4: pt); end;

procedure pt.uj(ux, uy: integer);
begin xp:=ux; yp:=uy; end;
procedure nsz.uj(u1, u2, u3, u4: pt);
begin c1:=u1; c2:=u2; c3:=u3; c4:=u4; end;

function fx(k,l: integer): real;      { a paramétergörbék egyenletei }
begin fx:=(rr+r*sin(k*pim))*cos(l*pin); end;
```

```
function fy(k,l: integer): real;
begin fy:=(rr+r*sin(k*pim))*sin(l*pin); end;
function fz(k,l: integer): real;
begin fz:=r*cos(k*pim); end;

procedure rajzol;
var
  k, l: integer;
  nv1, nv2, a1, b1, c1: real;
  x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4: real;
  kx1, ky1, kx2, ky2, kx3, ky3, kx4, ky4 : integer;
  s1, s2, s3, s4, t1, t2, t3, t4: pt;
  h1, hm: nsz;
procedure rzhl;
begin
  x1:=fx(k,l); x2:=fx(k-1,l);
  x3:=fx(k-1,l-1); x4:=fx(k,l-1);     { a felületelemek   }
  y1:=fy(k,l); y2:=fy(k-1,l);
  y3:=fy(k-1,l-1); y4:=fy(k,l-1);     { csúcsainak koordinátái }
  z1:=fz(k,l); z2:=fz(k-1,l);
  z3:=fz(k-1,l-1); z4:=fz(k,l-1);
  kx1:=round(s*(-bp*x1+ap*y1)/nv1);
  ky1:=round(s*(-cp*(ap*x1+bp*y1)+sqr(nv1)*z1)/nv2);
  kx2:=round(s*(-bp*x2+ap*y2)/nv1);            { vetítés }
  ky2:=round(s*(-cp*(ap*x2+bp*y2)+sqr(nv1)*z2)/nv2);
  kx3:=round(s*(-bp*x3+ap*y3)/nv1);
  ky3:=round(s*(-cp*(ap*x3+bp*y3)+sqr(nv1)*z3)/nv2);
  kx4:=round(s*(-bp*x4+ap*y4)/nv1);
  ky4:=round(s*(-cp*(ap*x4+bp*y4)+sqr(nv1)*z4)/nv2);
  s1.uj(kx0+kx1,ky0-ky1); s2.uj(kx0+kx2,ky0-ky2);
                                              { felületelemek       }
  s3.uj(kx0+kx3,ky0-ky3); s4.uj(kx0+kx4,ky0-ky4);
                                              { vetületeinek  be-   }
  t1.uj(kx0-kx1,ky0-ky1); t2.uj(kx0-kx2,ky0-ky2);
                                              { töltése a megfele-}
  t3.uj(kx0-kx3,ky0-ky3); t4.uj(kx0-kx4,ky0-ky4);
                                              { lő objektumokba }
  h1.uj(s1,s2,s3,s4); hm.uj(t1,t4,t3,t2);
  a1:=(y2-y1)*(z3-z1)-(y3-y1)*(z2-z1);
  b1:=(z2-z1)*(x3-x1)-(z3-z1)*(x2-x1); { normálisvektor koordinátái }
  c1:=(x2-x1)*(y3-y1)-(x3-x1)*(y2-y1);
end;

begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  for l:= 1 to round(n/2) do
    for k:= 1 to m do begin
      rzhl;
      if a1*ap+b1*bp+c1*cp > 0 then begin     { láthatóságtól függő }
        fillpoly(4,h1); fillpoly(4,hm); end; { megjelenítés }
    end;
end;
```

A tórusznak a 2.6.2 alpontban említett határvonalas képe az alábbi ábrán látható:



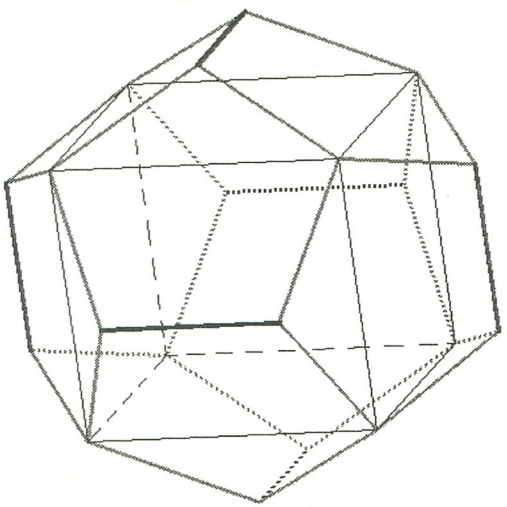
2.40. ábra

2.6.5. Háromszögű háló - gömb és gömbből származtatott csillag

Amint az a *2.37. ábrán* is látható, a görbe felületű testek paramétergörbék szerinti felosztása nem minden esetben egyenletes. Gömb, vagy gömbnek megfelelő topológiájú testek esetén a délkörök a pólusok közelében torlódnak. Ilyen testeket úgy lehet elképzelni, hogy a gömb valamilyen gumiszerű anyagból készült, amelyet egyes részeken megnyújtunk, másokon összezsugorítunk, azaz folytonos változtatásokat hajtunk végre, amelyeknél a felület megszakítása vagy összevarrása nem megengedett. A következőkben olyan eljárást mutatunk be, amellyel viszonylag egyenletes elosztású háromszögű háló szerkeszthető.

A legnagyobb lapszámú szabályos soklapból, az ikozaéderből indulunk ki. Az ikozaédernek 20 lapja (egybevágó, egyenlő oldalú háromszögek), 30 éle és 12 csúcsa van (*2.26. ábra*). Az ikozaéder és a *2.25. ábrán* látható dodekaéder között az az összefüggés, hogy ha az egyik szomszédos oldallapjainak közép-

pontjait összekötjük megkapjuk a másikat és fordítva. Egy másik érdekes összefüggés a szóbanforgó poliéderek kocka segítségével történő szerkesztésével kapcsolatos. Ha 12 ötszög egy-egy átlóját a *2.41. ábrán* látható módon egy kocka éleire illesztünk, dodekaédert nyerünk. Ha a kocka lapjaival párhuzamos dodekaéder-éleket (az ábrán vastag fekete szakaszok) szimmetrikusan meghosszabbítjuk, míg egyenlővé válnak a kocka élével, a kapott 12 csúcs egy ikozaéder csúcsai lesznek.



2.41. ábra

A kocka és a dodekaéder élhosszai közötti arány pontosan az úgynevezett aranymetszet, azaz

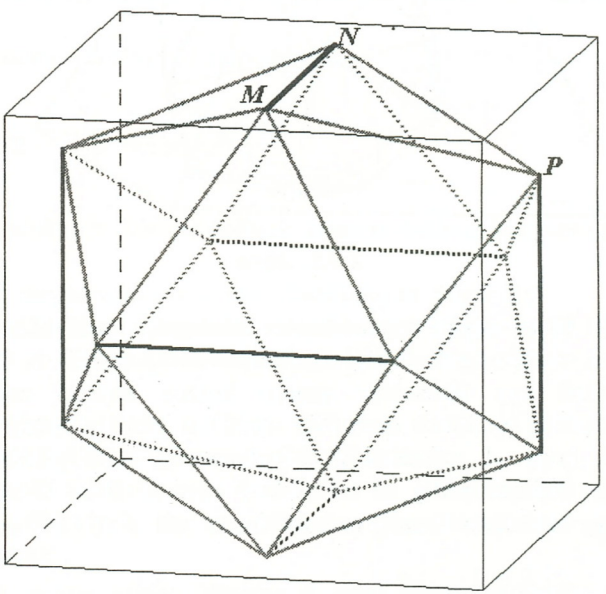
$$\frac{a}{l_{12}} = 2\cos\frac{\pi}{5} = \frac{\sqrt{5}+1}{2} \tag{2.79}$$

A kocka lapjaival (és éleivel) párhuzamos dodekaéder-élek távolsága az illető lapoktól egyenlő a hosszuk felével ($l_{12}/2$).

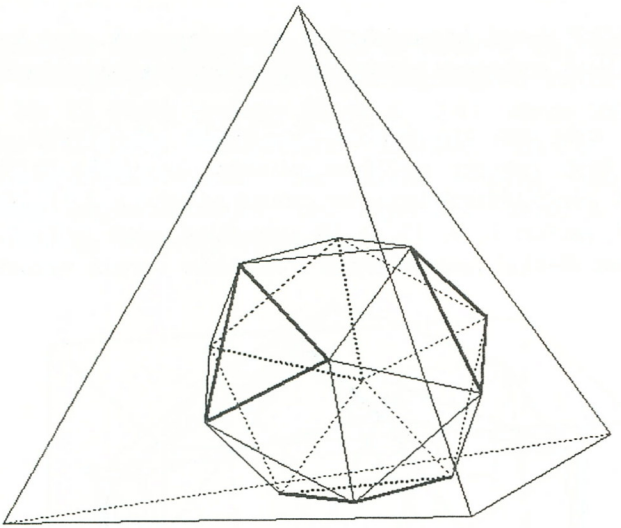
A már említett ikozaéder-szerkesztést úgy is elvégezhetjük, hogy egy kocka lapjainak a *2.42. ábrán* látható módon kiválasztott szimmetriatengelyein felvesszük azokat a szakaszokat (az ábrán vastag fekete szakaszok), amelyek

egyenlő oldalú *MNP* típusú háromszögeket eredményeznek. Azt kapjuk, hogy kocka éle és az illető szakaszok közötti arány ugyancsak az aranymetszet.

Az ikozaédernek még van egy érdekes illeszkedési tulajdonsága, éspedig az, hogy négy-négy lapja egy-egy szabályos tetraéder lapjainak síkjában helyezkedik el. A *2.43. ábrán* látható tetraéder csúcsai például a *2.31. ábrán* látható csillagnak a 2.27. gráfon 1, 9, 15 és 18 számokkal jelölt csúcsai. Az ikozaéder megvastagított élekkel határolt lapjai a tetraéder lapjain vannak.



2.42.ábra



2.43. ábra

Egy ikozaéder húsz olyan tetraéderre bontható, amelyeknek alapjuk az ikozaéder lapjai, magasságuk pedig az ikozaéder apotémái. A *2.44. ábrán* az *A*, *B*, *C* pontok egy ikozaéder azonos laphoz tartozó csúcsai, az őket összekötő ívek pedig az illető ikozaéder köré írt gömb főkörei. Az ikozaéder olyan helyzetű, hogy *A* csúcsa a *z* tengelyen, az *ABC* gömbháromszög *AD* felező merőleges főköre pedig az *xz* síkban van. A *B* és *C* csúcsok *xy* síkra eső vetületei egy szabályos ötszög csúcsai, ezért $\pi/5$ a *K'ON'* szög mértéke.

A szóbanforgó gömbháromszög íveit *n* egyenlő részre osztva és a megfelelő pontokat főkörívekkel összekötve olyan rajzolatot kapunk, amely *h* gömbi háromszögből áll és *p* csomópontja van, ahol *h* és *p* a következő képletekkel számíthatók:

$$p = \frac{(n+1)(n+2)}{2} \quad ; \quad h = n^2 \tag{2.80}$$

A fenti képletek teljes indukcióval könnyen igazolhatók.

Ezt az eljárást az ikozaéder minden lapjára elvégezve és a kapott gömbi háromszögek csúcsait egyenes szakaszokkal összekötve egy olyan poliédert nyerünk, amelynek összesen P csúcsa és H lapja van, amelyeket a

$$P = 10n^2 + 2 \quad ; \quad H = 20n^2 \tag{2.81}$$

képletekkel számíthatunk ki. P meghatározásánál az egy ikozaéder-laphoz tartozó belső csomópontokat 20-szal, az egy élen levőket 30-cal szoroztuk majd az eredményhez hozzáadtuk az ikozaéder csúcsainak számát (12).

Az ikozaéder geometriájából kiszámítható:

$$\alpha = \operatorname{arctg} 2 \quad ; \quad \beta = \operatorname{arctg} \frac{1+\sqrt{5}}{2} \tag{2.82}$$

szögekre az osztópontok koordinátáinak kiszámításához lesz szükség.

Az osztópontok meghatározása a következőképpen történik:
- az AB és AC íveket n egyenlő részre osztjuk. Legyen M illetve N az A-tól számított k-adik osztópont a megfelelő íveken. Főkörívvel összekötjük őket.
- megismételjük az eljárást a CA és CB ívek esetén. A C csúcstól számított l-edik osztópontokat köti össze a GH főkörív.
- most még csak feltételezzük, de később be is bizonyítjuk, hogy l változtatásával a GH ívek és az MN ív Q metszéspontjai MN-t egyenlő részekre osztják.

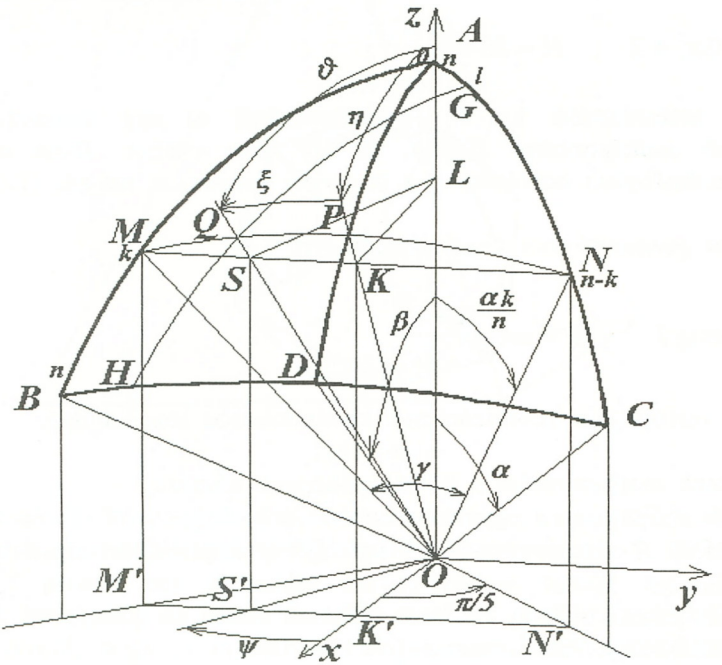
A Q pont koordinátái a ξ és η szögek függvényében:

$$\begin{cases} x = r\cos\xi \sin\eta \\ y = r\sin\xi \\ z = r\cos\xi \cos\eta \end{cases} \quad ; \quad \begin{cases} \eta \in [0, \beta\,] \\ \xi \in \left[-\gamma/2, \gamma/2\right] \end{cases} \tag{2.83}$$

ahol r a gömb sugara, γ pedig az MN ív mértéke:

$$\gamma = 2\arcsin\left(\sin\frac{\alpha\, k}{n}\sin\frac{\pi}{5}\right) \quad ; \qquad k \in \{0,1,\ldots,n\} \tag{2.84}$$

A fenti képletet megkaphatjuk a gömbi trigonometriát [1] használva, vagy pedig az *OKL*, *OKN* és *OK'N'* derékszögű háromszögekben felírható trigonometriai összefüggésekből.



2.44. ábra

A (2.83) egyenletekben szereplő szögek a következő összefüggésekkel számíthatók:

$$\begin{cases} \eta = \arcsin \dfrac{\sin\dfrac{\alpha k}{n}\cos\dfrac{\pi}{5}}{\cos\dfrac{\gamma}{2}} \\ \xi = \left(\dfrac{n-l}{k} - \dfrac{1}{2}\right)\gamma \end{cases} ; \quad \begin{cases} k \in \{0,1,\ldots,n\} \\ l \in \{n-k, n-k+1,\ldots,n\} \end{cases} \tag{2.85}$$

ahol szem előtt kell tartani azt, hogy γ is függ k-tól, a (2.84) összefüggés szerint.

A fenti képletek megállapításánál az *AB*, *AC*, *BC* és *MN* íveket egyenlő részekre osztottuk és feltételeztük, hogy a *GQH* ív a gömb főköre. Ennek bizonyításához elegendő megmutatni, hogy a *G*, *Q*, *H* és *O* pontok egy síkban vannak, ami akkor teljesül, ha

$$\begin{vmatrix} 0 & 0 & 0 & 1 \\ x_h & y_h & z_h & 1 \\ x_q & y_q & z_q & 1 \\ x_g & y_g & z_g & 1 \end{vmatrix} = 0 \tag{2.86}$$

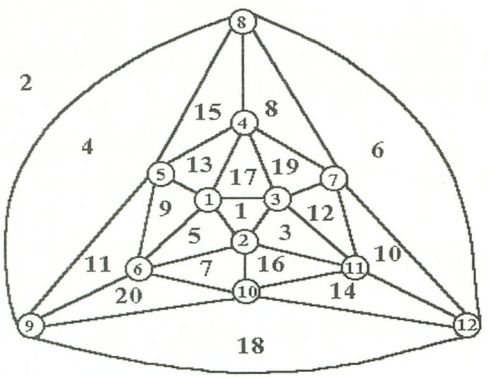
ahol a szóbanforgó pontok koordinátái a (2.83) képletekkel számítandók. A pontok helyzeteinek figyelembe vételével:

$$\begin{cases} \xi_h = (n-2l)\arcsin\left(\sin\alpha \sin\dfrac{\pi}{5}\right) \\ \xi_q = (2n-k-2l)\arcsin\left(\sin\dfrac{\alpha k}{n}\sin\dfrac{\pi}{5}\right) \\ \xi_g = (n-l)\arcsin\left(\sin\dfrac{\alpha (n-l)k}{n}\sin\dfrac{\pi}{5}\right) \end{cases} \quad \begin{cases} \eta_h = \beta \\ \eta_q = \arcsin\dfrac{\sin\dfrac{\alpha k}{n}\cos\dfrac{\pi}{5}}{\sqrt{1-\sin^2\dfrac{\alpha k}{n}\sin^2\dfrac{\pi}{5}}} \\ \eta_g = \arcsin\dfrac{\sin\dfrac{\alpha (n-l)k}{n}\cos\dfrac{\pi}{5}}{\sqrt{1-\sin^2\dfrac{\alpha (n-l)}{n}\sin^2\dfrac{\pi}{5}}} \end{cases} \tag{2.87}$$

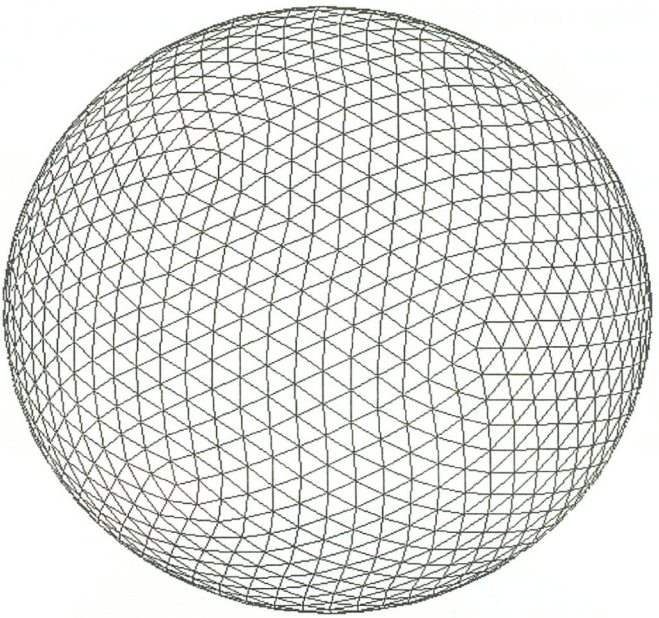
Behelyettesítve a (2.83) egyenletekbe, (2.86) azonosságnak adódik bármely megengedett *k* és *l* értékre.

Az ikozaéder gráfja a *2.45. ábrán* látható. Ha egy lapjára elvégezzük a fent leírt felosztást, egybevágósági transzformációkkal (forgatást és tükrözést használunk) az egész gömb lefedhető. Így készült a *2.46. ábrán* látható gömbmozaik n=12 esetén.

A felosztást az ikozaéder 1-es számú lapján végezzük. A 2-es számú lap az 1-es lapnak az ikozaéder középpontjához viszonyított tükörképe, a 3-as számú pedig az 1-esnek a 2 és 3 csúcsokat összekötő élhez viszonyított tükörképe. A 4-es számú lap a 3-as tükörképe az ikozaéder középpontjához viszonyítva. A kapott négy lapot $2\pi/5$ egész számú többszörösével elforgatva az ikozaéder 1 és 12 csúcsokon átmenő tengelye körül, az összes többi lap megkapható. A láthatóság megállapításánál figyelemmel kell lenni arra, hogy a tükrözések megváltoztatják az egyes lapokhoz tartozó csúcsok sorrendjét és ezáltal a lapok irányítását.



2.41. ábra



2.42. ábra

Ha a sugarat nem állandónak vesszük, hanem a ξ és η szögek értékeitől függőnek, akkor más, topológiailag az ikozaéderrel ekvivalens görbe felületű testeket kapunk. Néhány példa a *2.47 ... 2.49 ábrákon* látható.

A *2.47. ábrán* látható "tarajos gömb" sugarának képlete:

$$r = 230 + \frac{2000(n-k)(n-l)(n-k-l)}{n^3} \tag{2.88}$$

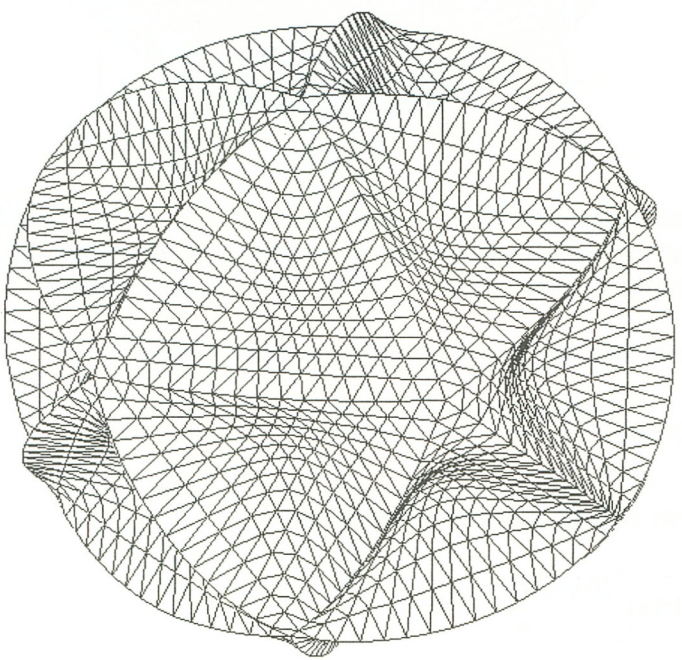
a *2.48. ábrán* látható "dudoros gömb"-ének:

$$r = 115 + \left(\frac{36\,k\,l(2n-k-l)}{n^3} \right)^2 \tag{2.89}$$

míg a *2.49. ábrán* látható csillagot az

$$r = 230 - 400\frac{k(n-k)-(n-l)(n-k-l)}{n^2} \tag{2.90}$$

képlettel nyerjük. A felosztás n=16-tal készült, de a fenti képletek biztosítják a rajzok arányainak megmaradását elméletileg bármely n-re. Gyakorlatilag azonban nem érdemes 32-nél nagyobb értékekkel dolgozni, mert a képernyő korlátozott felbontása miatt a részletek összemosódnak. Ezenkívűl a nagy számú változó tárolása is gondot okoz. Dinamikus helyfoglalás mellett is például n=32 esetén a program nem futtatható a TurboPascal fejlesztési környezetből, mert túlcsordul a *heap*. Ilyenkor végrehajtható (.EXE) file-t kell készíteni és a fejlesztési környezeten kívülről futtatni.
A *2.50. ábrán* látható csillagot ugyanez a program rajzolta, a (2.90) képlettel, n=2 esetén.

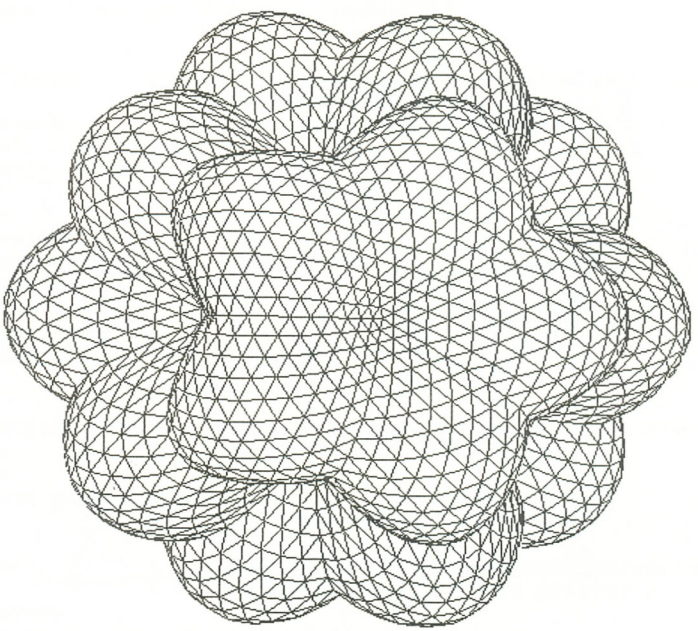


2.47. ábra

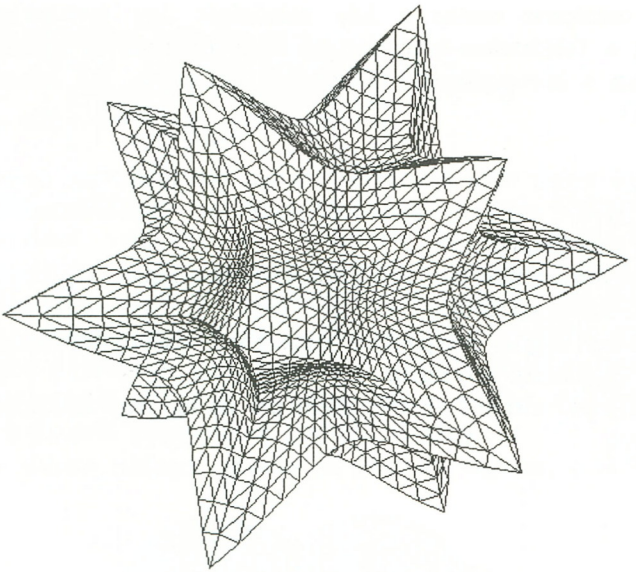
Nagyszámú osztópont esetén a kép minőségét úgy javíthatjuk, hogy nem húzzuk meg a felületelem-háromszögek éleit (*setcolor(0)* utasítással lehet ezt elérni), hanem a háromszögeket váltakozva befestjük. Így készült a 2.51.ábra (n=32 esetén).

A szóbanforgó testek nem konvexek, viszont csillagszerűek, ezért a következő, viszonylag gyors eljárást alkalmazhatjuk a felületelemek megjelenítési sorrendjének megállapítására:

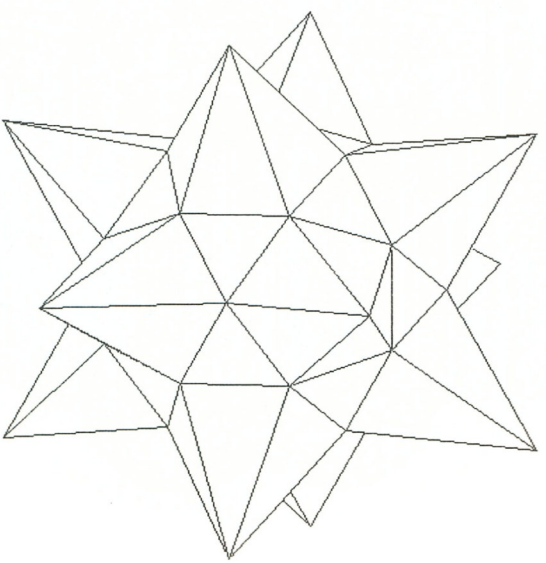
- a testhez hozzárendeljük az ikozaédert, amelynek segítségével a felület-elemhálót szerkesztettük.
- az ikozaéder lapjait súlypontjuk képsíkhoz viszonyított magassága szerinti növekvő sorrendbe helyezzük.
- a megállapított sorrendben megjelenítjük az egy-egy laphoz tartozó felület-tartományt.



2.48. ábra



2.49. ábra



2.50. ábra

A fenti ábrák a lemezmellékleten is megtalálható GOMBFF.PAS program segítségével készültek. Az alábbi programrészletből hiányzik a grafikus meghajtó elindítását és az ábrázolt alakzat forgatását végző rész.

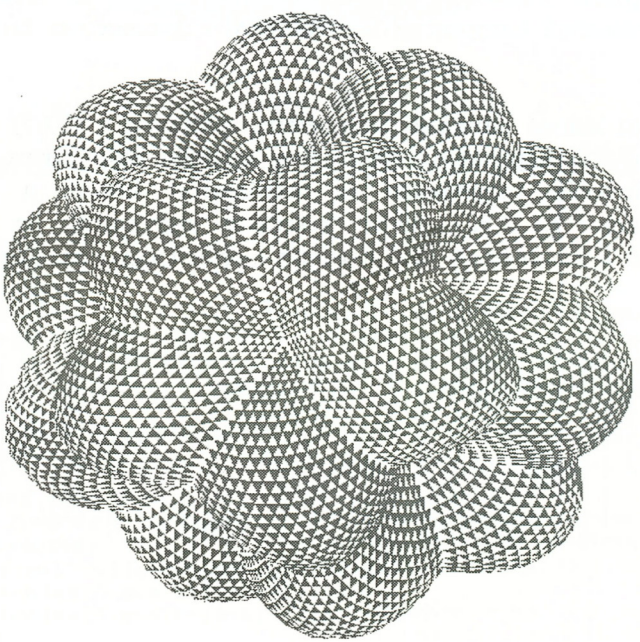
```
const
  n=16; n2=n*n; n3=n2*n;
     { az egy ikozaéderlapnak megfelelő felülettartományhoz tartozó }
     { háromszögek illetve csúcsok száma: }
  lsz=n2; csz=round((n+1)*(n+2)/2);
  r0=230; dt=pi/30;

type
  mtr = array[0..n] of integer;

  csucs = object
    xx, yy, zz: real;
    procedure uj(ux, uy, uz: real); end;

  pont = object
    x, y: integer;
    procedure uj(ux, uy: integer); end;


  hmszptr=^hmsz;        { a felületelem-háromszögek dinamikus tárolását}
                        { lehetővé tevő pointer }
  hmsz = object         { felületelem-háromszög objektum }
    cs1, cs2, cs3: csucs;
    s: array[1..3] of pont;
    z: real;
    n: byte;
    lth: boolean;
    procedure uj(us1, us2, us3: csucs);
    procedure ms;
    procedure lt;
    procedure vt;
    procedure sr(un: byte);
  end;

  ihmsz = object        { az ikozaéder lapjainak súlypontjuk szerinti }
    z: real;            { sorszámát tároló objektum }
    n: byte;
    procedure sr(un: byte);
  end;

  csptr=^gbcs;          { a közelítő poliéder csúcsainak koordinátáit }
                        { tároló objektumhoz rendelt pointer }
  gbcs = object
    gcs: array[1..csz] of csucs;
  end;
```

```
var
  hv: mtr;
  ilp: array[1..20] of ihmsz;
  ics: array[1..10,1..3] of csucs;
  lp: array[1..lsz] of hmszptr;
  stcs: array[1..20] of csptr;
  kp: array[1..20] of real;
  r, alf, bet, bt1, bt2, gm1, gm2, gn1, gn2, skn, psi, ksi, khi: real;
  vlt: byte;
  i, j, k, l, h: Integer;
  ct, sz: integer;
  kr: char;
procedure csucs.uj(ux, uy, uz: real);
begin
    xx:=ux; yy:=uy; zz:=uz; end;
procedure pont.uj(ux, uy: integer);
begin
    x:=ux; y:=uy; end;
procedure hmsz.uj(us1, us2, us3: csucs);
begin cs1:=us1; cs2:=us2; cs3:=us3; end;
procedure hmsz.ms;
begin z:=(cs1.zz+cs2.zz+cs3.zz)/3; end;
procedure hmsz.lt;          { láthatóság }
begin
  if (cs2.xx-cs1.xx)*(cs3.yy-cs1.yy)-(cs3.xx-cs1.xx)*(cs2.yy-cs1.yy)
    > 0 then lth:=true else lth:=false; end;
procedure hmsz.vt;          { vetítés }
begin
  s[1].uj(round(320+cs1.xx), round(240-cs1.yy));
  s[2].uj(round(320+cs2.xx), round(240-cs2.yy));
  s[3].uj(round(320+cs3.xx), round(240-cs3.yy)); end;
procedure hmsz.sr(un: byte);
begin n:=un; end;

procedure ihmsz.sr(un: byte);
begin n:=un; end;

function arcsin(a: real): real;
begin
  if 1-a*a<1e-6 then arcsin:=pi/2 else arcsin:=arctan(a/sqrt(1-a*a));
end;

procedure haladv; { az osztópontok sorszámát meghatározó haladvány }
begin
  hv[0]:=1;
  for k:=1 to n do
    hv[k]:=hv[k-1]+k;
end;
procedure ikozacs; { az ikozaéder 10 lapja csúcsainak meghatározása }
begin              { a fennmaradó 10 lap ezek középponthoz viszo- }
  skn:=sin(alf);   { nyított szimmetrikusa }
  khi:=arcsin(skn*gn1);
```

```
  psi:=skn*gn2/cos(khi);
  ics[1,1].uj(0,0,1);
  ics[1,2].uj(cos(khi)*psi, -sin(khi), cos(khi)*sqrt(1-sqr(psi)));
  ics[1,3].uj(cos(khi)*psi, sin(khi), cos(khi)*sqrt(1-sqr(psi)));
  for l:=1 to 3 do
    ics[6,l].uj(-ics[1,l].xx*bt2+ics[1,l].zz*bt1, ics[1,l].yy,
    ics[1,l].xx*bt1+ics[1,l].zz*bt2);
  for k:=2 to 5 do for l:=1 to 3 do begin
    ics[k,l].uj(ics[k-1,l].xx*gm2+ics[k-1,l].yy*gm1,
    -ics[k-1,l].xx*gm1+ics[k-1,l].yy*gm2, ics[k-1,l].zz);
    ics[k+5,l].uj(ics[k+4,l].xx*gm2+ics[k+4,l].yy*gm1,
    -ics[k+4,l].xx*gm1+ics[k+4,l].yy*gm2, ics[k+4,l].zz);
  end;
end;
procedure ikoza;      { az ikozaéderlapok súlypontjainak }
begin                 { kiszámítása }
  for k:=0 to 4 do begin
    kp[1+4*k]:=(ics[1+k,1].zz+ics[1+k,2].zz+ics[1+k,3].zz)/3;
    kp[2+4*k]:=-kp[1+4*k];
    kp[3+4*k]:=(ics[6+k,1].zz+ics[6+k,2].zz+ics[6+k,3].zz)/3;
    kp[4+4*k]:=-kp[3+4*k];
  end;
  for k:=1 to 20 do ilp[k].z:=kp[k];
  for k:=1 to 20 do ilp[k].sr(k);
end;
procedure sorrend;    { az ikozaéder lapjait súlypontjuk magassága }
var sf, fl: byte;     { szerinti sorrendbe helyező eljárás }
begin
  repeat begin
    sf:=0;
    for k:=1 to 20 do
      for l:=1 to 20 do
        if (ilp[l].z-ilp[k].z)*(ilp[l].n-ilp[k].n)<0 then begin
          fl:=ilp[k].n;  ilp[k].sr(ilp[l].n);  ilp[l].sr(fl);  sf:=1;
end;
  end; until sf=0;
end;
procedure lerajz;     { megjelenítő eljárás }
var u, v, w: integer;
begin
  setfillstyle(1,0); bar(0,0,640,480);
  for v:=1 to 20 do
    for u:=1 to 4 do
      for w:=0 to 4 do begin
        sz:=u+4*w;
        if ilp[sz].n=v then begin
          ct:=0;
          for k:=0 to n-1 do
            for l:=0 to k do begin
              ct:=ct+1;
                { felületelem-háromszögek generálása }
              lp[ct]^.uj(stcs[sz]^.gcs[hv[k]+l],
               stcs[sz]^.gcs[hv[k+1]+l+1],
```

```
                stcs[sz]^.gcs[hv[k+1]+l]);
              if l<k then begin
                ct:=ct+1;
                lp[ct]^.uj(stcs[sz]^.gcs[hv[k]+l],
                  stcs[sz]^.gcs[hv[k]+l+1],
                  stcs[sz]^.gcs[hv[k+1]+l+1]);
              end;
            end;
          for k:=1 to ct do begin
            lp[k]^.lt; lp[k]^.vt;
            if (u=1) or (u=4) then begin
              if lp[k]^.lth=true then begin
                setfillstyle(1,0); fillPoly(3, lp[k]^.s); end;
            end else
              if lp[k]^.lth=false then begin
                setfillstyle(1,0); fillPoly(3, lp[k]^.s); end;
          end;
        end;
      end;
  end;
procedure rajz;
begin
  ikozacs; ikoza; sorrend;
  ct:=0;
  for k:=0 to n do begin
    skn:=sin(alf*k/n);
    khi:=arcsin(skn*gn1);
    psi:=skn*gn2/cos(khi);
    for l:=n-k to n do begin
      if k=0 then ksi:=0 else ksi:=(-1+2*(n-l)/k)*khi;
      case vlt of
      1:r:=r0;
      2:r:=r0+2000.0*(n-k)*(n-l)*(n-k-l)/n3;
      3:r:=0.5*r0+sqr(36.0*k*l*(2*n-k-l)/n3);
      4:r:=r0-400.0*(k*(n-k)-(n-l)*(n-k-l))/n2;
      end;
      ct:=ct+1;
     { a közelítő poliéder csúcsainak koordinátái: }
      stcs[1]^.gcs[ct].uj(r*cos(ksi)*psi, r*sin(ksi),
                          r*cos(ksi)*sqrt(1-sqr(psi)));
      stcs[3]^.gcs[ct].uj(-
      stcs[1]^.gcs[ct].xx*bt2+stcs[1]^.gcs[ct].zz*bt1,
      stcs[1]^.gcs[ct].yy,
      stcs[1]^.gcs[ct].xx*bt1+stcs[1]^.gcs[ct].zz*bt2);
      for i:=1 to 4 do begin
          stcs[1+4*i]^.gcs[ct].uj(stcs[4*i-3]^.gcs[ct].xx*gm2
          +stcs[4*i-3]^.gcs[ct].yy*gm1, -stcs[4*i-3]^.gcs[ct].xx*gm1
          +stcs[4*i-3]^.gcs[ct].yy*gm2, stcs[4*i-3]^.gcs[ct].zz);
          stcs[3+4*i]^.gcs[ct].uj(stcs[4*i-1]^.gcs[ct].xx*gm2
          +stcs[4*i-1]^.gcs[ct].yy*gm1, -stcs[4*i-1]^.gcs[ct].xx*gm1
          +stcs[4*i-1]^.gcs[ct].yy*gm2,stcs[4*i-1]^.gcs[ct].zz); end;
      for i:=0 to 4 do begin
        stcs[2+4*i]^.gcs[ct].uj(-stcs[1+4*i]^.gcs[ct].xx,
          -stcs[1+4*i]^.gcs[ct].yy, -stcs[1+4*i]^.gcs[ct].zz);
```

```
        stcs[4+4*i]^.gcs[ct].uj(-stcs[3+4*i]^.gcs[ct].xx,
          -stcs[3+4*i]^.gcs[ct].yy, -stcs[3+4*i]^.gcs[ct].zz); end;
    end;
  end;
  lerajz;
end;
begin
  for k:=1 to lsz do new(lp[k]);      { helyfoglalás a heap-ben a  }
  for k:=1 to 20 do new(stcs[k]);     { dinamikus változók számára }
  repeat begin
    write('geometria (1, 2, 3 vagy 4) ?  (0 kilép)  '); readln(vlt);
    if vlt<>0 then begin
      haladv;
      gm1:=sin(2*pi/5);
      gm2:=cos(2*pi/5);
      gn1:=sin(pi/5);
      gn2:=cos(pi/5);
      alf:=arctan(2); bet:=arctan((1+sqrt(5))/2);
      bt1:=sin(2*bet); bt2:=cos(2*bet);
      grafind; rajz;
    end;
  end until vlt=0;
end.
```



2.51. ábra

Ha olyan testet akarunk ábrázolni, amely nem felel meg az ikozaéder szimmetriájának, akkor az egész gömbön kell tudnunk tájékozódni. Ahhoz, hogy ezt lehetővé tegyük, írjuk fel a *2.44. ábrán* látható Q osztópont koordinátáit a gömbi koordinátarendszer szögeinek függvényében:

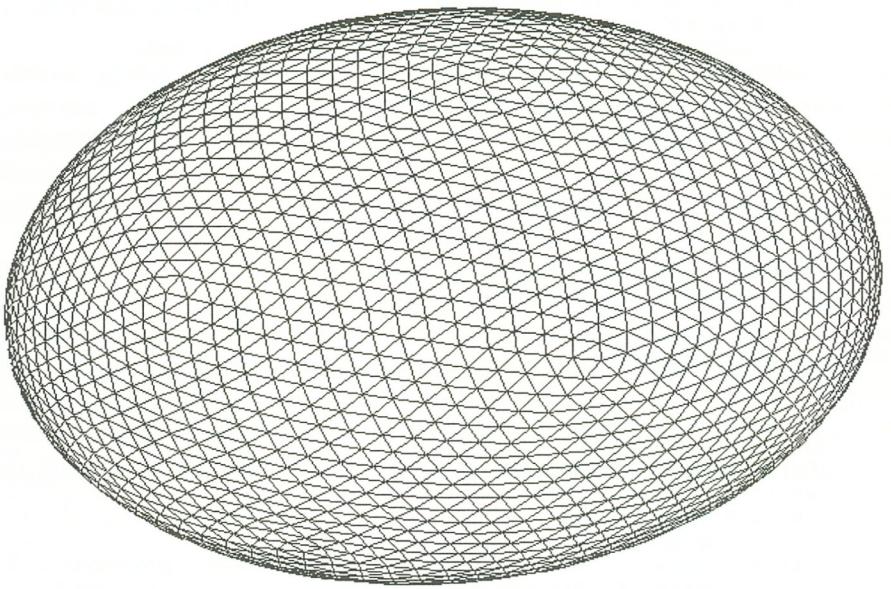
$$\begin{cases} x = r\sin\vartheta\ \cos\psi \\ y = r\sin\vartheta\ \sin\psi \\ z = r\cos\vartheta \end{cases} \tag{2.91}$$

ahol

$$\begin{cases} \vartheta = \arccos(\cos\xi\ \cos\eta) \\ \psi = \operatorname{arctg}\dfrac{\operatorname{tg}\xi}{\sin\eta} \end{cases} \tag{2.92}$$

amint az a gömbi trigonometria képleteiből vagy pedig az *OKS*, *OKL*, *OSL* és *OK'S'* derékszögű háromszögekben felírható trigonometriai összefüggésekből következik.

A *2.37. ábrán* látható ellipszoid paraméteres előállítása a (2.67) egyenletekkel történik. A (2.92) képleteket felhasználva és szem előtt tartva a háló szerkesztésénél használt forgatásokat és tükrözéseket készült a ZEPP3H.PAS program, amely a *2.52. ábrát* készítette ugyanarról az ellipszoidról. Megfigyelhető a háromszögű háló nagyobb mértékű egyenletessége.



2.52. ábra

A szóbanforgó program megjelenítő és a közelítő poliéder csúcsait számító eljárásai:

```
const
  a=6; b=4; c=3;
  n=16; lsz=n*n; csz=round((n+1)*(n+2)/2);
  s=50; dt=pi/30;
procedure lerajz;      { a felületelemeket láthatóságuk függvényében  }
var u,w:integer;       { megjelenítő eljárás - mivel az ábrázolt test }
begin                  { konvex, nincs szükség külön megjelenítési    }
  setfillstyle(1,0); bar(0,0,640,480);  { sorrend megállapítására  }
  for u:=1 to 4 do
    for w:=0 to 4 do begin
      sz:=u+4*w;
      ct:=0;
      for k:=0 to n-1 do
        for l:=0 to k do begin
          ct:=ct+1;
          lp[ct]^.uj(stcs[sz]^.gcs[hv[k]+l],
          stcs[sz]^.gcs[hv[k+1]+l+1],
          stcs[sz]^.gcs[hv[k+1]+l]);
        end;
      for k:=1 to ct do begin
        lp[k]^.lt; lp[k]^.vt;
```

```
        if (u=1) or (u=4) then begin
          if lp[k]^.lth=true then begin
            setfillstyle(1,0); fillPoly(3, lp[k]^.s); end;
        end else
          if lp[k]^.lth=false then begin
            setfillstyle(1,0); fillPoly(3, lp[k]^.s); end;
      end;
    end;
end;
procedure rajz;          { kiszámítja a közelítő }
                         { poliéder csúcsainak koordinátáit }
var
  skn, eta, teta, ksi, khi, psi: real;
  g: byte;
begin
  for g:=1 to 5 do begin
    ct:=0;
    for k:=0 to n do begin
      skn:=sin(alf*k/n);
      khi:=arcsin(skn*gn1);
      eta:=arcsin(skn*gn2/cos(khi));
      for l:=n-k to n do begin
        if k=0 then ksi:=0
               else ksi:=(-1+2*(n-l)/k)*khi;
        if k=0 then psi:=0
               else psi:=arctan(sin(ksi)/cos(ksi)/sin(eta))+gm[g];
        teta:=arccos(cos(ksi)*cos(eta));
        ct:=ct+1;
        stcs[4*g-3]^.gcs[ct].uj(s*a*cos(psi)*sin(teta),
        s*b*sin(psi)*sin(teta), s*c*cos(teta));
        stcs[4*g-2]^.gcs[ct].uj(-stcs[4*g-3]^.gcs[ct].xx,
        -stcs[4*g-3]^.gcs[ct].yy, -stcs[4*g-3]^.gcs[ct].zz);
        if k=0 then psi:=gm[g]
               else
                psi:=arctan(sin(ksi)/cos(ksi)/sin(2*bet-eta))+gm[g];
        teta:=arccos(cos(ksi)*cos(2*bet-eta));
        stcs[4*g-1]^.gcs[ct].uj(s*a*cos(psi)*sin(teta),
        s*b*sin(psi)*sin(teta), s*c*cos(teta));
        stcs[4*g]^.gcs[ct].uj(-stcs[4*g-1]^.gcs[ct].xx,
        -stcs[4*g-1]^.gcs[ct].yy,-stcs[4*g-1]^.gcs[ct].zz);
      end;
    end;
  end;
  lerajz;
end;
begin
  for k:=1 to lsz do new(lp[k]);          { helyfoglalás a heap-ben }
  for k:=1 to 20 do new(stcs[k]);
  for k:=0 to 4 do gm[k]:=2*k*pi/5;
  haladv;
  gn1:=sin(pi/5); gn2:=cos(pi/5);
  alf:=arctan(2); bet:=arctan((1+sqrt(5))/2);
  grafind; rajz;
 end.
```

2.7. Görbék a térben

A térbeli görbék képsíkra eső vetületei természetesen síkgörbék. Az 1.2 alfejezetben leírtakhoz hasonlóan a térgörbék képét is kétféleképpen jeleníthetjük meg, ponthalmaz vagy törtvonal segítségével. Az alkalmas módszer kiválasztása ebben az esetben is attól függ, hogy a görbét hogy tudjuk előállítani. A törtvonalas megjelenítés viszonylag kevesebb szakasz (ami azt jelenti, hogy kevesebb pontot kell meghatározni - a program gyorsabb) esetén is biztosíthatja a simaság illúzióját, míg a ponthalmazos megjelenítésnél már a folytonosság illúziójának igénye is nagyobb pontsűrűséget követel meg. Az utóbbi módszer előnye viszont az, hogy nem kell tudni, hogy a görbén a pontok milyen sorrendben helyezkednek el. Ezt a tulajdonságot kihasználva jeleníthető meg például két felület áthatási görbéje a paraméteres előállítás analitikus meghatározása nélkül. A felületek kontúrgörbéje (például a *2.40. ábra*) ugyancsak ponthalmazosan jeleníthető meg legkönnyebben.

2.7.1. Paraméteres előállítás

A paraméteres előállítású görbéket a legkönnyebb ábrázolni, ugyanis ezek esetében a paraméter minden megengedett értékére rendelkezésünkre állnak a megfelelő pontok koordinátáinak analitikus képletei:

$$\begin{cases} x = x(t) \\ y = y(t) \\ z = z(t) \end{cases} \quad ; \qquad t \in I \subset \Re \tag{2.93}$$

Az ábrázolás úgy történik, hogy a paramétert kis Δt értékekkel növelve, rendre kiszámítjuk a görbe pontjainak és ezek képsíkra eső vetületeinek koordinátáit, és meghúzzuk a megfelelő szakaszokat, amelyek tulajdonképpen a görbe húrjainak vetületei. A Δt növekedésnek megfelelő húr hossza:

$$\Delta s = \sqrt{[x(t+\Delta t) - x(t)]^2 + [y(t+\Delta t) - y(t)]^2 + [z(t+\Delta t) - z(t)]^2} \tag{2.94}$$

Ha a paraméternövekedést menet közben újraszámítjuk úgy, hogy a kapott húrok megközelítőleg egyenlőek legyenek, egyenlő ívhosszas rajzolásról beszélünk. Ha a kívánt ívhossz adott (Δs_0), akkor a

$$\Delta t' = \frac{\Delta s_0}{\Delta s} \Delta t \tag{2.95}$$

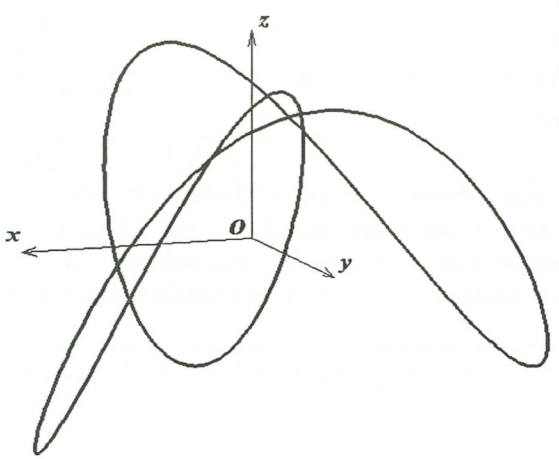
képlettel dolgozhatunk.

Az egyenlő ívhosszas szaggatott vonalas rajzolás lehetővé teszi a görberészek térbeli helyzetének becslését, ugyanis a képsíkkal párhuzamos húrok a leghosszabbak, a többiek pedig annál rövidebbek, minél nagyobb szöget zárnak be a képsíkkal.

Az alábbi ábrán az

$$\begin{cases} x = \cos t - \sin 2t \\ y = \cos t - \sin t \quad ; \quad t \in [0, 2\pi] \\ z = \sin 3t \end{cases}$$

előállítású görbe látható.



2.53. ábra

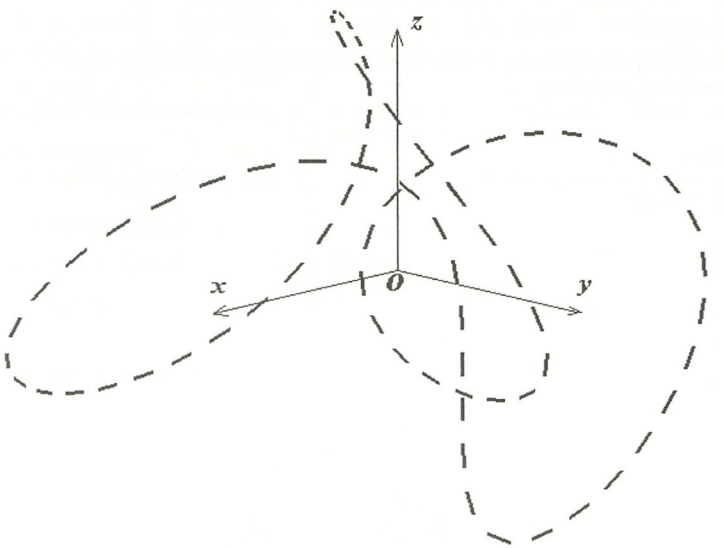
Az ábrázolást végző eljárás:

```
procedure rajzol;
var
  t, nv1, nv2: real;
  x, y, z: real;
  kx, ky: integer;
  procedure rzk;
  begin
    x:=fx(t); y:=fy(t); z:=fz(t);
                        { kiszámítja a görbe pontjainak koordinátáit }
    kx:=kx0+round((-bp*x+ap*y)/nv1); { képsíkra vetítés }
    ky:=ky0-round((-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2);
  end;
begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  bar(0,0,getmaxx,getmaxy);
  t:=t0; rzk; moveto(kx,ky);     { t0 és t1 a paraméterintervallum }
                                 { korlátai }
  repeat begin
    t:=t+dt; rzk; lineto(kx,ky);          { húr megjelenítése }
  end; until t>=t1;
end;
```

Az

$$\begin{cases} x = 3\sin t \cos 3t \\ y = \cos 2t - 3\sin t \\ z = 2\sin 3t + \cos t \end{cases} \quad ; \qquad t \in [0,2\pi]$$

előállítású görbe szaggatott vonalas képe:



2.54. ábra

a rajzoló eljárás pedig:

```
const
  t0=0; t1=2*pi;
  a=70; ds0=20; eps=1;
  kx0=320; ky0=240;

var
  ap, bp, cp: real;
.....................................................
```

```
procedure rajzol;
var
  t, tk, dt, nv1, nv2: real;
  x, y, z, ds: real;
  kx, ky: integer;
  f: byte;
  procedure rzk;          { koordinátaszámítás és vetítés }
  begin
    x:=fx(t); y:=fy(t); z:=fz(t);
    kx:=kx0+round((-bp*x+ap*y)/nv1);
    ky:=ky0-round((-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2);
  end;
begin
  dt:=pi/80;
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  bar(0,0,getmaxx,getmaxy);
  t:=t0; rzk; moveto(kx,ky); f:=1;
  repeat begin
    tk:=t+dt;
     { aktuális ívhossz }
    ds:=sqrt(sqr(fx(tk)-x)+sqr(fy(tk)-y)+sqr(fz(tk)-z));
        if abs(ds-ds0)>eps then begin    { lépésigazítás }
      dt:=ds0/ds*dt; tk:=t+dt; end;
    t:=tk; rzk;
    if f=1 then begin lineto(kx,ky); f:=0; end { szaggatott vonalas }
                                           { rajzolás }
           else begin moveto(kx,ky); f:=1; end;
  end; until t>=t1;
end;
```

2.7.2. *Differenciálegyenletes felírás*

Számos műszaki alkalmazás kapcsán (például vektorterek erővonalai) találkozunk térgörbék differenciálegyenletes előállításával, amikor is nem ismerjük a görbe pontjainak koordinátáit, hanem csak ezek deriváltjait valamilyen paraméter függvényében és egy kiinduló pontot. Ilyenkor a kiinduló pont koordinátáit növeljük az itt számított deriváltakkal arányosan úgy, hogy a megfelelő húr a kívánt mértékű legyen, majd az így kapott pontot új kiindulópontnak tekintve megismételjük az eljárást. Ha *ds* a kívánt ívhossz (illetve a vele közelítően egyenlő húrhossz), akkor a

$$\begin{cases} x = x_0 + \dfrac{\dot{x}_0}{\sqrt{\dot{x}_0^2 + \dot{y}_0^2 + \dot{z}_0^2}} ds \\ y = y_0 + \dfrac{\dot{y}_0}{\sqrt{\dot{x}_0^2 + \dot{y}_0^2 + \dot{z}_0^2}} ds \\ z = z_0 + \dfrac{\dot{z}_0}{\sqrt{\dot{x}_0^2 + \dot{y}_0^2 + \dot{z}_0^2}} ds \end{cases} \tag{2.96}$$

képletekkel számíthatjuk az új pont koordinátáit.

Ha potenciáltér erővonalait kívánjuk ábrázolni, akkor az ívhossz lesz a paraméter. Ha Φ (x,y,z) a potenciálfüggvény, akkor a térerősség-vektor koordinátái:

$$\begin{cases} E_x(x_0, y_0, z_0) = \dfrac{\partial \Phi}{\partial x}(x_0, y_0, z_0) \\ E_y(x_0, y_0, z_0) = \dfrac{\partial \Phi}{\partial y}(x_0, y_0, z_0) \\ E_z(x_0, y_0, z_0) = \dfrac{\partial \Phi}{\partial z}(x_0, y_0, z_0) \end{cases} \tag{2.97}$$

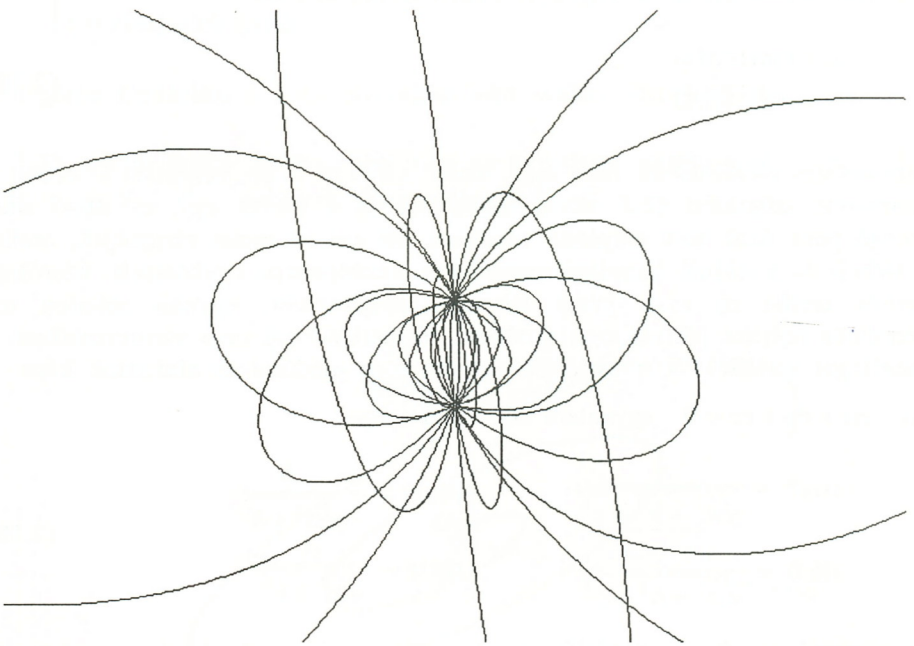
és az erővonal (amelynek a térerősségvektor minden pontjában érintője) új pontjának koordinátái:

$$\begin{cases} x = x_0 + \dfrac{E_x}{E} ds \\ y = y_0 + \dfrac{E_y}{E} ds \\ z = z_0 + \dfrac{E_z}{E} ds \end{cases} \quad ; \qquad E = \sqrt{E_x^2 + E_y^2 + E_z^2} \tag{2.98}$$

Az alábbi ábrán a

$$\Phi = -\frac{1}{\sqrt{x^2+y^2+z^2}} + \frac{1}{\sqrt{x^2+y^2+(z-b)^2}}$$

potenciáltér néhány erővonala látható. Ez két pontszerű, ellentétes előjelű töltés által létesített elektromos erőtérnek felel meg, ahol az egyik töltés a (0,0,0) pontban, a másik pedig a (0,0,b) pontban van.



2.55. ábra

Az ábrát az **EROTER.PAS** program rajzolta.

2.7.3. Síkgörbék a térben

Előfordulhat, hogy a térben adott helyzetű síkon levő görbét kívánunk ábrázolni. Ha a görbe a síkon poláris koordinátarendszerben van megadva, azaz

$$r = r(\omega) \tag{2.99}$$

előállítású, ahol r a kezdőponthoz viszonyított helyzetvektor hossza, ω pedig a helyzetvektornak az x tengellyel bezárt szöge, ami az

$$\begin{cases} x = r(\omega)\cos\omega \\ y = r(\omega)\sin\omega \end{cases} \tag{2.100}$$

paraméteres előállításnak felel meg, akkor célszerű a sík helyzetét a térben az Euler-féle szögekkel (*2.3. ábra*) jellemezni és a görbét egy, az illető síkon mozgó pont által leírt pályának tekinteni. Itt azt az esetet tárgyaljuk, amikor a térbeli és a síkbeli koordinátarendszerek kezdőpontjai egybeesnek. Merőleges vetítés esetén ez nem jelent lényeges megszorítást, ugyanis minden más kölcsönös helyzet párhuzamos eltolással (transzlációval) erre visszavezethető és merőleges vetítésnél a transzlációk folytán nem módosul az alakzatok képe.

Az $ax + by + cz = 0$ egyenletű sík Euler-szögei:

$$\begin{cases} \cos\vartheta = \dfrac{c}{\sqrt{a^2+b^2+c^2}} \\ \sin\vartheta = \dfrac{\sqrt{a^2+b^2}}{\sqrt{a^2+b^2+c^2}} \end{cases} ; \quad \begin{cases} \cos\psi = -\dfrac{b}{\sqrt{a^2+b^2}} \\ \sin\psi = \dfrac{a}{\sqrt{a^2+b^2}} \end{cases} \tag{2.101}$$

és viszont, a ϑ nutációs és ψ precessziós szögekkel megadott sík egyenletének együtthatói az

$$\begin{cases} a' = \sin\vartheta\,\sin\psi \\ b' = -\sin\vartheta\,\cos\psi \\ c' = \cos\vartheta \end{cases} \tag{2.102}$$

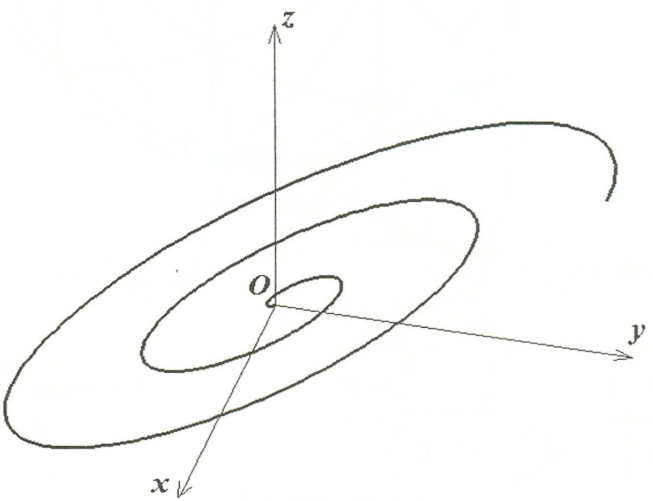
számokkal arányos mennyiségek.

Ha a g nutációs tengely egybeesik a görbe síkjának x tengelyével, akkor az ω tiszta forgásszög kezdeti értéke 0, ha nem, akkor a szóbánforgó két tengely közötti szög lesz az ω_0 kezdeti érték. A (2.99) előállítású görbe térbeli koordinátarendszerhez viszonyított paraméteres egyenlete a (2.23) transzformációs képlet alapján, ahol ξ helyére $r(\omega)$ kerül, η és ζ pedig nulla:

$$\begin{cases} x = r(\omega)(\cos\psi\cos\omega - \sin\psi\cos\vartheta\sin\omega) \\ y = r(\omega)(\sin\psi\cos\omega + \cos\psi\cos\vartheta\sin\omega) \\ z = r(\omega)\sin\vartheta\sin\omega \end{cases} \tag{2.103}$$

és a görbe ábrázolása a 2.7.1 alpontban leírt módon végezhető el.

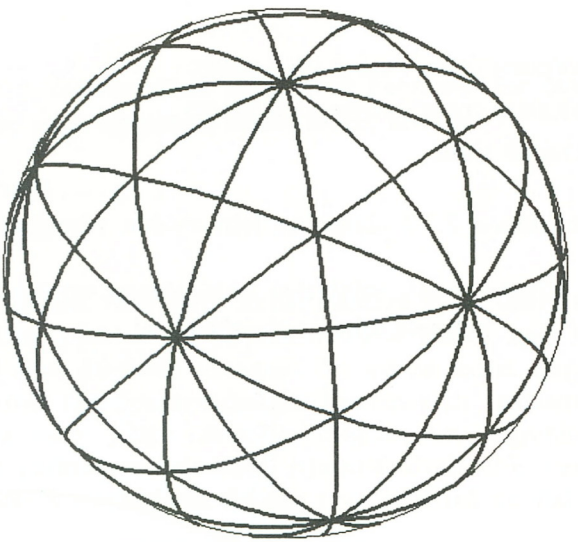
A *2.56. ábrán* látható, $r=\omega$ előállítású archimédeszi spirális a $\vartheta = \pi / 3$, $\psi = -\pi / 4$ helyzetű síkon helyezkedik el.



2.56. ábra

A *2.57. ábrán* látható gömbön azokat a főköröket rajzoltuk meg, amelyek a gömbbe írt ikozaéder négy-négy csúcsán mennek át (és az ugyanabba a gömbbe írt dodekaéder négy-négy csúcsán is). Az ikozaéder csúcsokon öt-öt főkör halad át, a dodekaéder csúcsokban pedig három-három főkör találkozik.

A gömb nem átlátszó, ezért csak azokat az elemi íveket jelenítettük meg, amelyek környezetében a felület normálisa felénk mutat. A képsíkban fekvő középpontú gömb esetében ez a feltétel ekvivalens azzal, hogy az illető elemi ív a képsík fölött helyezkedjék el.



2.57. ábra

A főkörök a

l	0	1	2
ϑ	β	$\frac{\pi}{2}$	$\frac{\pi}{2}-\beta$
ψ	$2k\frac{\pi}{5}+(l-1)\frac{\pi}{2}$		

$$k \in \{0,1,\ldots,4\}$$

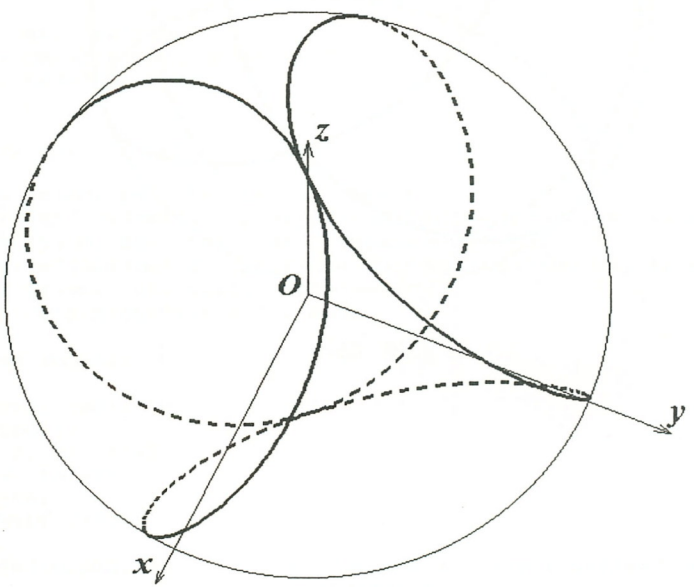
; $\beta = \operatorname{arctg}\frac{1+\sqrt{5}}{2}$

helyzetű síkokban fekszenek. Az ábrázolt főkörök mindegyikéhez jól meghatározott ϑ és ψ érték tartozik. A program a lemezmellékleten LABDA.PAS néven található meg.

Ha egyetlen görbe ábrázolásakor a nutációs vagy a precessziós szög is változik (ω -val együtt), vagy mind a kettő, akkor olyan görbéket kapunk, amelyek nem egy síkban fekszenek. A *2.58* és *2.59 ábrákon* látható görbék állandó r és ϑ mellett

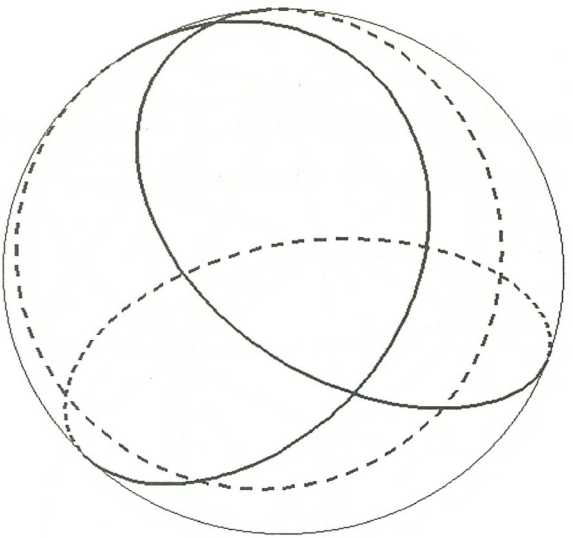
$$\psi = \frac{p}{m}\omega \quad ; \qquad \omega \in [0, 2m\pi]$$

előállításúak, ahol az ábrázolt görbék esetén $p = 1$ és $m = 2$. A *2.58. ábra* $\vartheta = \frac{\pi}{2}$, a *2.59. ábra* pedig $\vartheta = \frac{\pi}{3}$ értékekre készült.



2.58. ábra

A görbék gömbön helyezkednek el, ugyanis a helyvektor abszolút értéke (hossza) állandó. A gömböt áttetszőnek tekintettük, ezért a görbének a gömb túloldalán levő részeit szaggatott vonallal rajzoltuk.



2.59. ábra

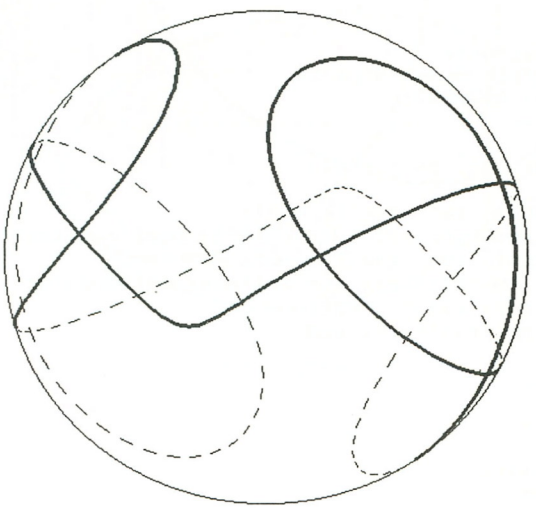
A következő három ábrán az

$$\begin{cases} r = R \\ \vartheta = \pi - \dfrac{q}{m}\omega \quad ; \qquad \omega \in [0, 2m\pi] \\ \psi = \dfrac{p}{m}\omega \end{cases}$$

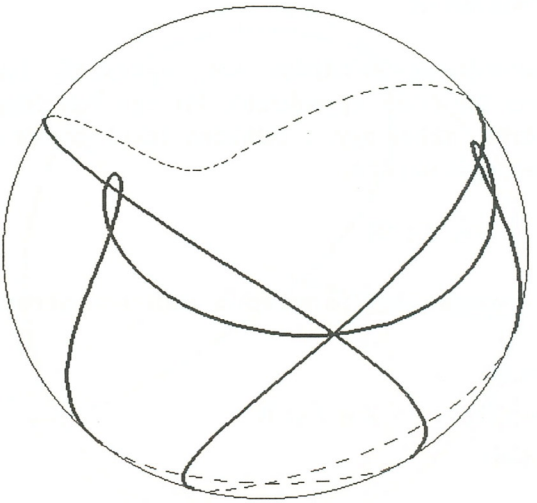
előállítású görbék láthatók. Az m, p, q állandók értékei a megfelelő ábra alatt olvashatók. A rajzok a következő eljárás segítségével készültek:

```
const
  n=400; m=2; p=1; q=4;
  om0=0; om1=2*m*pi; dom=2*m*pi/n;
  a=200; kx0=320; ky0=240;
.....
var
  ap, bp, cp, om, tet, psi: real;

function fx(om, psi, tet: real): real;
begin fx:=a*(cos(om)*cos(psi)-sin(om)*sin(psi)*cos(tet)); end;
function fy(om, psi, tet: real): real;
begin fy:=a*(cos(om)*sin(psi)+sin(om)*cos(psi)*cos(tet)); end;
function fz(om, tet: real): real;
begin fz:=a*sin(om)*sin(tet); end;

procedure rajzol;
var
  nv1, nv2: real;
  k: integer;
  x, y, z, kz: real;
  kx, ky: integer;
  fl: byte;
  procedure rzk;
  begin
    x:=fx(om,psi,tet);     { kiszámítja a görbe következő pontjának }
                           { koordinátáit }
    y:=fy(om,psi,tet);
    z:=fz(om,tet);
    kx:=kx0+round((-bp*x+ap*y)/nv1);          { képsíkra vetítés }
    ky:=ky0-round((-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2);
    kz:=(ap*x+bp*y+cp*z)/nv2;       { képsíkhoz viszonyított magasság }
  end;
begin
  fl:=0;
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  bar(0,0,getmaxx,getmaxy);
  om:=om0; tet:=pi; psi:=0; rzk; moveto(kx,ky);
```

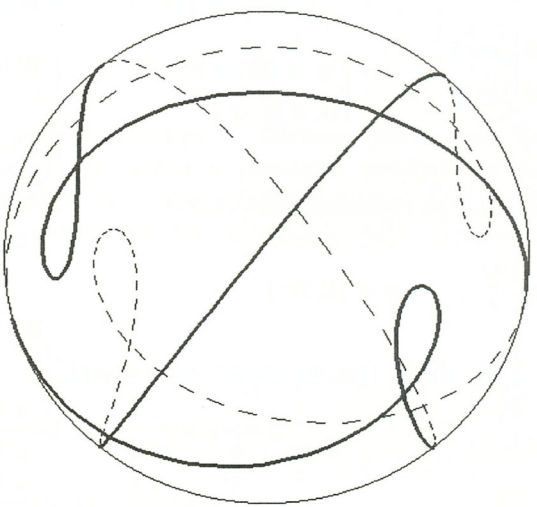
```
  repeat begin
    om:=om+dom; tet:=pi-q*om/m; psi:=p*om/m; rzk;
        if kz>=0 then begin          { húr megjelenítése: ha látható }
                                     { tartományon }
          setlinestyle(0,0,3); fl:=0; end
                        { fekszik, folytonos vastag vonal, ha nem,    }
          else setlinestyle(0,0,1);
                         { szaggatott, vékony vonal (tulajdonképpen }
                         { minden második húr kerül megrajzolásra)  }
        if fl=0 then begin lineto(kx,ky); fl:=1; end
        else begin moveto(kx,ky); fl:=0; end;
  end; until om>=om1;
  setlinestyle(0,0,1); circle(kx0, ky0, a);
                                     { a gömb képkontúrja - egy kör }
end;
```



2.60. ábra
m=1 p=2 q=4



2.61.ábra
m=1 p=2 q=3



2.62. ábra
m=2 p=1 q=4

2.7.4. Görbék felületeken

A felületek paraméteres előállítása két, egymástól független paraméter használatát tételezi fel (2.65 egyenletek). Ha egy összefüggést adunk meg a két paraméter között, akkor egy, a felületen fekvő görbét értelmezünk. Ha a szóbanforgó összefüggés explicit,

$$v = v(u)\,; \qquad u \in I \subset \Re \tag{2.104}$$

alakú, akkor az eredmény egy paraméteres előállítású görbe:

$$\begin{cases} x = x(u, v(u)) \\ y = y(u, v(u))\,; \\ z = z(u, v(u)) \end{cases} \qquad u \in I \subset \Re \tag{2.105}$$

amelyet a 2.7.1 alpontban leírt módon ábrázolunk. A *2.58...62.ábrák* tulajdonképpen gömbfelületre rajzolt görbék.

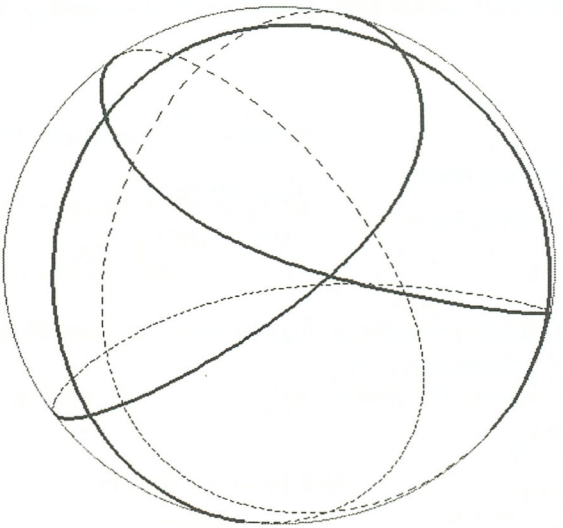
Az

$$\begin{cases} x = r \sin\vartheta \cos\psi \\ y = r \sin\vartheta \sin\psi \\ z = r \cos\vartheta \end{cases}\,; \qquad \begin{cases} \psi \in [0, 2\pi] \\ \vartheta \in [0, \pi] \end{cases}$$

paraméteres előállítású gömbfelületen a

$$\vartheta = \frac{5}{3} \operatorname{arctg} \frac{3\psi}{4}\,; \qquad \psi \in [0, 2\pi]$$

összefüggéssel a *2.63. ábrán* látható görbét értelmeztük.



2.63. ábra

Ha a paraméterek közötti összefüggés implicit,

$$F: D \subset \Re^2 \to \Re \quad ; \qquad F(u,v) = 0 \tag{2.106}$$

alakú, akkor úgy járunk el, hogy a felületet sűrűn vett paramétervonalak mentén végigpásztázzuk, és azokat a pontokat, amelyekben teljesül a (2.106) összefüggés levetítjük és megjelenítjük. A görbe felületű testek kontúrgörbéinek egyenlete (2.71) tulajdonképpen a paraméterek közötti implicit összefüggés:

$$\begin{vmatrix} a_p & b_p & c_p \\ \dfrac{\partial x}{\partial u}(u,v) & \dfrac{\partial y}{\partial u}(u,v) & \dfrac{\partial z}{\partial u}(u,v) \\ \dfrac{\partial x}{\partial v}(u,v) & \dfrac{\partial y}{\partial v}(u,v) & \dfrac{\partial z}{\partial v}(u,v) \end{vmatrix} = 0 \tag{2.107}$$

Az itt vázolt módszert alkalmazva készült a tóruszfelület *2.40. ábrán* látható képkontúrja.

Az

$$\begin{cases} x = (R + r\sin\varphi)\cos\psi \\ y = (R + r\sin\varphi)\sin\psi \\ z = r\cos\varphi \end{cases} ; \quad \begin{cases} \varphi \in [0,2\pi] \\ \psi \in [0,2\pi] \end{cases}$$

egyenletű tóruszfelület paraméterei közötti $\varphi = a\psi$ összefüggéssel a felületen egy csavarvonalat értelmezünk. A

$$\begin{cases} \varphi_1 = \dfrac{2k+1}{2}\psi \\ \varphi_2 = \dfrac{2k+1}{2}\psi + \pi \end{cases} ; \quad \psi \in [0,2\pi], \quad k \in \mathbf{N}$$

összefüggések tulajdonképpen ugyanannak a görbének két darabját adják meg. A kettéosztás eredményeképpen a két görbeág azonos ψ értéknek megfelelő pontjai a tórusz ψ -nek megfelelő leírókörén (r) átmérősen szembenfekvő pontok. Összekötve őket egy olyan torzfelületet kapunk, amelynek csak egy oldala van, akárcsak a Moebius-szalagnak. Moebius-szalag úgy készíthető, hogy téglalap alakú papírcsík egyik végét 180-kal megforgatva a papírcsík ilyen helyzetű két végét összeragasztjuk. Így egy úgynevezett egyoldalú felület jön létre, vagyis a felületet teljesen be lehet festeni anélkül, hogy a festő ecsetet felemelnénk.

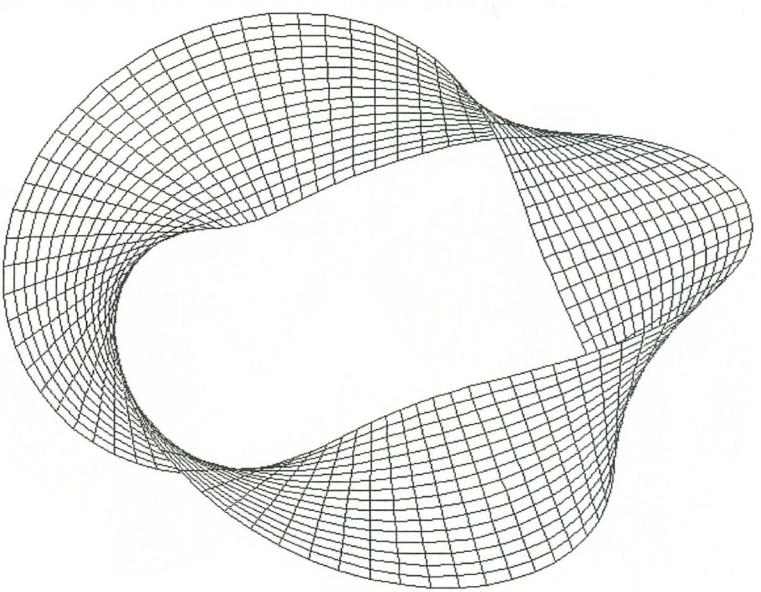
A felület paraméteres egyenlete a szerkesztési módjából adódik:

$$\begin{cases} x = (1-u)\left(R + r\cos\dfrac{2k+1}{2}\psi\right)\cos\psi + u\left(R - r\cos\dfrac{2k+1}{2}\psi\right)\cos\psi \\ y = (1-u)\left(R + r\cos\dfrac{2k+1}{2}\psi\right)\sin\psi + u\left(R - r\cos\dfrac{2k+1}{2}\psi\right)\sin\psi \\ z = (1-u)r\sin\dfrac{2k+1}{2}\psi - ur\sin\dfrac{2k+1}{2}\psi \end{cases} ; \quad \begin{cases} \psi \in [0,2\pi] \\ u \in [0,1] \end{cases}$$

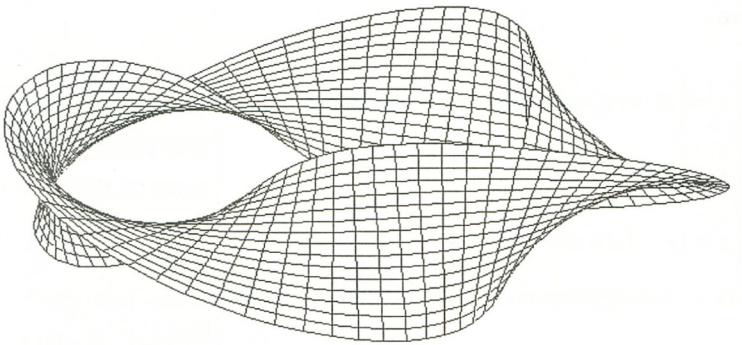
ahol ψ és u a paraméterek, k pedig egy adott szám. A fenti egyenletet rendezve:

$$\begin{cases} x = \left(R + r(1-2u)\cos\dfrac{2k+1}{2}\psi \right)\cos\psi \\ y = \left(R + r(1-2u)\cos\dfrac{2k+1}{2}\psi \right)\sin\psi \\ z = (1-2u)r\sin\dfrac{2k+1}{2}\psi \end{cases} ; \quad \begin{cases} \psi \in [0,2\pi] \\ u \in [0,1] \end{cases}$$

A felület *k=1*-re a *2.64* és *2.65 ábrákon* látható két különböző vetítési irány esetén,

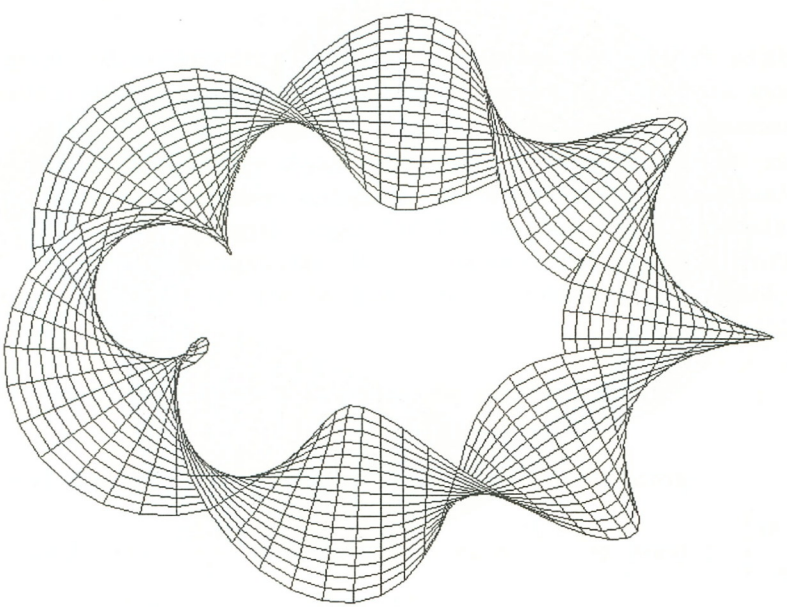


2.64. ábra

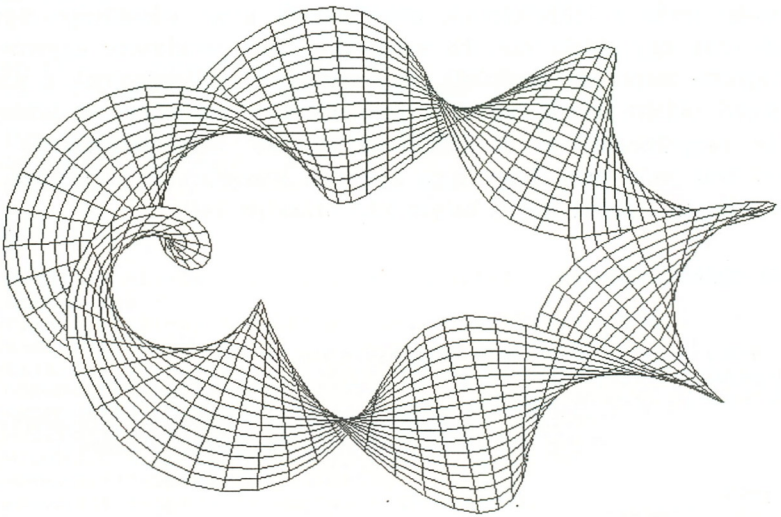


2.65. ábra

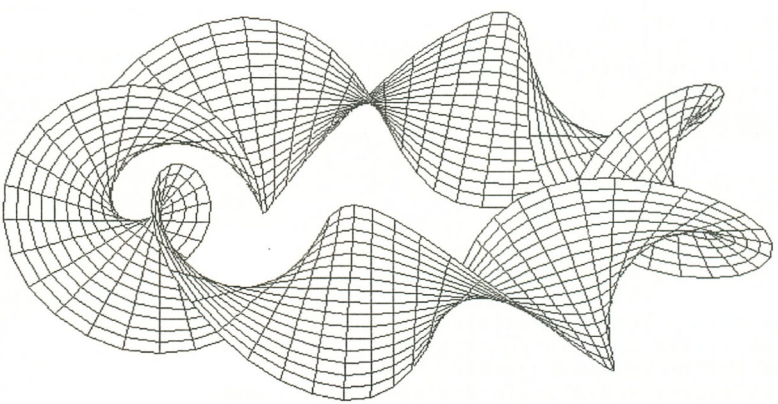
k=3 -ra pedig a *2.66...68 ábrákon,* három különböző vetítési irány esetén.



2.66. ábra



2.67. ábra



2.68. ábra

Az ábrázolás során a felületelemek irányításának nincs jelentősége, ugyanis a felületnek csak egy oldala van. Ez mértanilag nem paradoxon, ugyanis mikor egy pályagörbe menti sáv bezáródik, ugyanannak a felületelemnek a normálisa az ellenkező oldalra kerül, mivel a felületelemek csúcsainak a körbenjárási iránya is megváltozik. A megjelenítési sorrend meghatározásánál viszont tekintettel kell lenni arra, hogy igen könnyen létrejöhet téves takarás, amely a felület jellegénél fogva mégis helyes kép illúzióját kelti.

A rajzoló eljárás:

```
const
  m=16; n=80; rr=11; r=4; kx0=320; ky0=240; s=20;
  pim=2*pi/m; pin=2*pi/n;
  ap=-4; bp=-3; cp=1.4; l0=0;
                    { l0 a pályagörbék menti paraméter kezdőértéke }
                    { a vetítési iránytól függően változhat}
type
  pt=object
    xp, yp: integer;
    procedure uj(ux, uy: integer); end;

  nsz=object
    c1, c2, c3, c4: pt;
    procedure uj(u1, u2, u3, u4: pt); end;

var
  s1, s2, s3, s4: pt;
  hl: nsz;
  x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4: real;
  nv1, nv2: real;
  k, l: integer;
  kx1, ky1, kx2, ky2, kx3, ky3, kx4, ky4 : integer;

procedure pt.uj(ux, uy: integer);
begin xp:=ux; yp:=uy; end;

procedure nsz.uj(u1, u2, u3, u4: pt);
begin c1:=u1; c2:=u2; c3:=u3; c4:=u4; end;
                { a tórusz felületén kígyózó csavarvonal egyenletei }
function fx1(l: integer): real;
begin fx1:=(rr+r*sin(3*l*pin/2))*cos(l*pin); end;
function fy1(l: integer): real;
begin fy1:=(rr+r*sin(3*l*pin/2))*sin(l*pin); end;
function fz1(l: integer): real;
begin fz1:=r*cos(3*l*pin/2); end;
                                                { a felület egyenletei }
function fx2(l: integer): real;
begin fx2:=(rr+r*sin(3*l*pin/2+pi))*cos(l*pin); end;
function fy2(l: integer): real;
  begin fy2:=(rr+r*sin(3*l*pin/2+pi))*sin(l*pin); end;
```

```
function fz2(l: integer): real;
begin fz2:=r*cos(3*l*pin/2+pi); end;

function fx(k,l: integer): real;
begin fx:=(1-k/m)*fx1(l)+k/m*fx2(l); end;
function fy(k,l: integer): real;
begin fy:=(1-k/m)*fy1(l)+k/m*fy2(l); end;
function fz(k,l: integer): real;
begin fz:=(1-k/m)*fz1(l)+k/m*fz2(l); end;

procedure rzhl;
begin
  x1:=fx(k,l); x2:=fx(k-1,l); x3:=fx(k-1,l-1); x4:=fx(k,l-1);
 { felületelemek }
  y1:=fy(k,l); y2:=fy(k-1,l); y3:=fy(k-1,l-1); y4:=fy(k,l-1);
 { csúcsai }
  z1:=fz(k,l); z2:=fz(k-1,l); z3:=fz(k-1,l-1); z4:=fz(k,l-1);
  kx1:=round(s*(-bp*x1+ap*y1)/nv1);                    { vetítés }
  ky1:=round(s*(-cp*(ap*x1+bp*y1)+sqr(nv1)*z1)/nv2);
  kx2:=round(s*(-bp*x2+ap*y2)/nv1);
  ky2:=round(s*(-cp*(ap*x2+bp*y2)+sqr(nv1)*z2)/nv2);
  kx3:=round(s*(-bp*x3+ap*y3)/nv1);
  ky3:=round(s*(-cp*(ap*x3+bp*y3)+sqr(nv1)*z3)/nv2);
  kx4:=round(s*(-bp*x4+ap*y4)/nv1);
  ky4:=round(s*(-cp*(ap*x4+bp*y4)+sqr(nv1)*z4)/nv2);
  s1.uj(kx0+kx1,ky0-ky1); s2.uj(kx0+kx2,ky0-ky2);    {csúcs-objektumok}
  s3.uj(kx0+kx3,ky0-ky3); s4.uj(kx0+kx4,ky0-ky4);    { aktualizálása }
  hl.uj(s1,s2,s3,s4);
                  { négyszög (felületelem) objektum aktualizálása }
end;
procedure rajzol;
begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  for l:= l0 to l0+round(n/2) do
    for k:= 1 to m do begin
      rzhl; fillpoly(4,hl);              { felületelem megjelenítése }
    end;
  for l:= l0+n downto l0+round(n/2) do
    for k:= 1 to m do begin
      rzhl; fillpoly(4,hl);
    end;
end;
```

2.7.5. Felületek áthatása

Két felület általában egy térbeli görbében metszi egymást. A térgörbék implicit egyenletrendszere:

$$F,G\colon M \subset \mathfrak{R}^3 \to \mathfrak{R} \quad ; \quad \begin{cases} F(x,y,z)=0 \\ G(x,y,z)=0 \end{cases} \tag{2.108}$$

tulajdonképpen két implicit előállítású felület áthatási görbéjének egyenletrendszerét jelenti.

Abból kiindulva, hogy egy felületen a felület implicit egyenlete előjelet vált, az áthatási görbét úgy rajzolhatjuk, hogy az egyik felületet valamilyen módon paraméterezzük, a paramétervonalak mentén végigpásztázzuk, és azokat a pontokat, amelyeknek koordinátáira a másik felület egyenlete a nulla értéket adja, vagy előjele megváltozik (azaz a paramétervonalon két egymás utáni pont között a másik felület egyenlete nullát adva teljesül), megjelenítjük.

A fentiek alapján az áthatási görbe egyenletrendszere

$$\begin{cases} \text{F:} \begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u,v) \end{cases} \\ G(x,y,z)=0 \end{cases} \tag{2.109}$$

alakú lesz, ha a (2.108) rendszer első egyenletét írjuk át paraméteres alakba. Azokat a pontokat kell megjeleníteni, amelyekre:

$$G(x(u,v),y(u,v),z(u,v)) \cdot G(x(u+\delta,v),y(u+\delta,v),z(u+\delta,v)) \le 0 \tag{2.110}$$

a v = áll. paramétervonalak mentén, vagy

$$G(x(u,v),y(u,v),z(u,v)) \cdot G(x(u,v+\delta),y(u,v+\delta),z(u,v+\delta)) \le 0 \tag{2.111}$$

az u = áll. paramétervonalak mentén, ahol δ egy alkalmasan megválasztott kicsi szám. A paramétervonalak mentén apró lépésekben kell haladni, hogy

közelítőleg helyes eredményt kapjunk. A paramétervonalak sűrűsége pedig a kapott görbkép folytonosságát határozza meg.

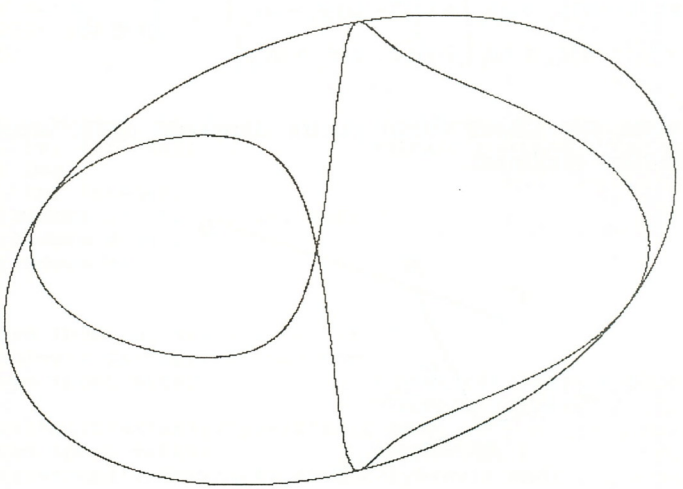
A *2.69. ábrán* látható egy ellipszoid képkontúrja és egy hiperbolikus paraboloiddal való áthatási görbéje, amelynek implicit előállítása tehát:

$$\begin{cases} \dfrac{x^2}{a^2}+\dfrac{y^2}{b^2}+\dfrac{z^2}{c^2}-1=0 \\ z=x^2-y^2 \end{cases}$$

Az áthatási görbe rajza úgy készült, hogy az ellipszoidot a (2.67) előállításnak megfelelő paramétervonalai mentén végigpásztáztuk mind szélességi, mind meridián irányban és megjelenítettük azokat a pontokat, amelyekben az

$$F(x,y,z)=z-x^2-y^2$$

függvény értéke 0 vagy előjelet vált.



2.69. ábra

A *2.71. ábrán* egy gömbnek a bele írt szabályos tetraéder éleire illeszkedő három hiperbolikus paraboloiddal való áthatási görbéi láthatók. Két kitérő tartóegyenesű szakaszra illeszkedő hiperbolikus paraboloid parametrikus előállítása a következő módon kapható meg (*2.70. ábra*):

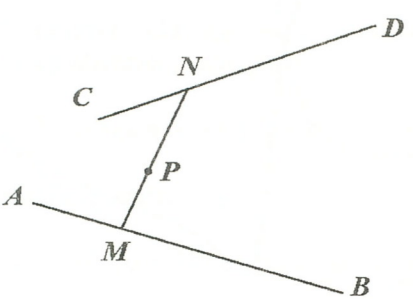
$$\begin{cases} x_m = (1-u)x_a + ux_b \\ y_m = (1-u)y_a + uy_b \\ z_m = (1-u)z_a + uz_b \end{cases} ; \quad \begin{cases} x_n = (1-u)x_c + ux_d \\ y_n = (1-u)y_c + uy_d \\ z_n = (1-u)z_c + uz_d \end{cases} ; \quad u \in [0,1]$$

$$\begin{cases} x_p = (1-v)x_m + vx_n \\ y_p = (1-v)y_m + vy_n \\ x_p = (1-v)x_m + vx_n \end{cases} ; \quad v \in [0,1]$$

amit az egész térre kiterjesztve:

$$\begin{cases} x = (1-v)\left[(1-u)x_a + ux_b\right] + v\left[(1-u)x_c + ux_d\right] \\ y = (1-v)\left[(1-u)y_a + uy_b\right] + v\left[(1-u)y_c + uy_d\right] \\ z = (1-v)\left[(1-u)z_a + uz_b\right] + v\left[(1-u)z_c + uz_d\right] \end{cases} ; \quad \begin{cases} u \in \Re \\ v \in \Re \end{cases}$$

amely a négy, nem egy síkban fekvő pontra illeszkedő egyik hiperbolikus paraboloid paraméteres előállítása.



2.70. ábra

A gömbbel való áthatási görbét a következőképpen rajzoltuk meg:
- $u\in$ *[0,1]* -re, *v=0* -ból a negatív értékek felé indulva, állandó *u* mellett, meghatározzuk az illető egyenes gömbbel való metszéspontját (ahol a gömb implicit egyenlete teljesül vagy előjelet vált)
- új *u*-ra megkeressük a következő metszéspontot, majd a kettő vetületeit összekötjük
- ilyen módon meghúzzuk az *A* és *B* pontok közötti görbeívet
- a fenti eljárásban *v=1* -ből növekvő értékek felé haladva megrajzoljuk a *C* és *D* pontok közé eső görbeívet
- *u* és *v* szerepét felcserélve megrajzolhatók az *A* és *C* valamint a *B* és *D* pontokat összekötő ívek

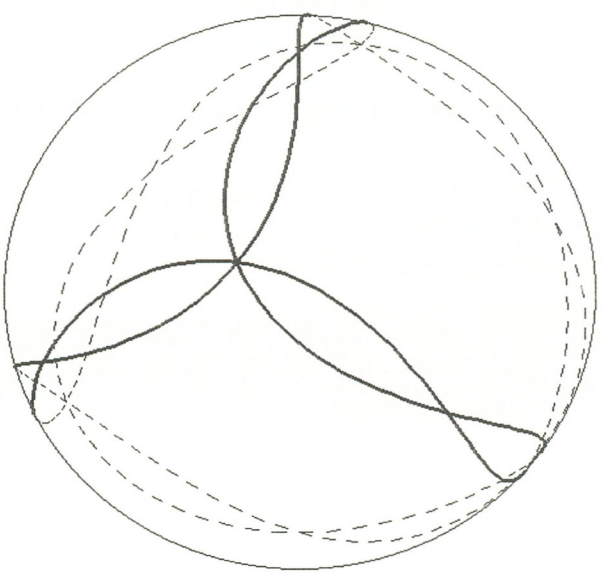
Az itt vázolt ábrázolási módszer az áthatási görbéket húrtörtvonalak segítségével rajzolja, nem pedig ponthalmazosan. Ennek következtében a program gyorsabb (nincs szükség olyan sűrű paramétervonal-hálóra), mi több, lehetővé válik a gömb túlsó oldalán fekvő ívek szaggatott vonalas ábrázolása.

Az ábrázolást végző eljárás:

```
const
  m=50; n=500;
  kx0=320; ky0=240;
  a=130;

type
  tpont = object                    { a paramétervonalon mozgó pont }
    xx, yy, zz: real;               { adatait rögzítő objektum }
    rr: real;
    kx, ky: integer;
    procedure uj(ux, uy, uz: real);
    procedure sugar;
    procedure vetit;
  end;

procedure tpont.uj(ux, uy, uz: real);
begin xx:=ux; yy:=uy; zz:=uz; end;
procedure tpont.sugar;              { a pont távolsága a gömb }
                                    { középpontjától }
begin rr:=sqrt(xx*xx+yy*yy+zz*zz); end;
procedure tpont.vetit;              { felülnézet }
begin kx:=round(kx0+xx); ky:=round(ky0+yy); end;

var
  k, l: integer;
  aa, bb, cc, dd: tpont;
  r: real;
  cs: array[1..4] of tpont;
```

```
  am, bm, ap, bp: tpont;
  fl1, fl2: byte;

procedure rajzol;
  function fx: real;          { négy pontra illeszkedő  }
                              { hiperbolikus paraboloid }
  begin                       { parametrikus egyenletei }
    fx:=(1-l/n)*((1-k/m)*aa.xx+k/m*bb.xx)+l/n*((1-k/m)
        *cc.xx+k/m*dd.xx); end;
  function fy: real;
  begin
    fy:=(1-l/n)*((1-k/m)*aa.yy+k/m*bb.yy)+l/n*((1-k/m)
        *cc.yy+k/m*dd.yy); end;
  function fz: real;
  begin
    fz:=(1-l/n)*((1-k/m)*aa.zz+k/m*bb.zz)+l/n*((1-k/m)
         *cc.zz+k/m*dd.zz); end;

  procedure metszpont;        { paramétervonalak gömbbel való   }
  begin                       { metszéspontjainak meghatározása }
    l:=0;
    repeat begin
      l:=l-1;
      am.uj(fx, fy, fz); am.sugar;
    end; until am.rr>r;
    l:=n;
    repeat begin
      l:=l+1;
      ap.uj(fx, fy, fz); ap.sugar;
    end; until ap.rr>r;
  end;

  procedure szakasz;          { húrszakasz rajzolása }
  begin
    if am.zz>=0 then begin setlinestyle(0,0,3); fl1:=0; end
                else setlinestyle(0,0,1);
    if fl1=0 then begin line(bm.kx, bm.ky, am.kx, am.ky); fl1:=1; end
             else fl1:=0;
    if ap.zz>=0 then begin setlinestyle(0,0,3); fl2:=0; end
                else setlinestyle(0,0,1);
    if fl2=0 then begin line(bp.kx, bp.ky, ap.kx, ap.ky); fl2:=1; end
             else fl2:=0;
  end;

  procedure iv;           { az áthatási görbe két-két csúcs közé eső }
  begin                   { ívének megrajzolása }
    k:=0; fl1:=0; fl2:=0;
    bm:=aa; bp:=cc; bm.vetit; bp.vetit;
    repeat begin
      k:=k+1; metszpont; am.vetit; ap.vetit;
      szakasz;
      bm:=am; bp:=ap;
    end until k=m-1;
    am:=bb; ap:=dd; am.vetit; ap.vetit; szakasz;
  end;
```

```
begin                    { áthatási görbe megrajzolása }
  aa:=cs[1]; bb:=cs[2]; cc:=cs[3]; dd:=cs[4]; iv;
  aa:=cs[1]; bb:=cs[2]; cc:=cs[4]; dd:=cs[3]; iv;
  aa:=cs[1]; bb:=cs[3]; cc:=cs[2]; dd:=cs[4]; iv;
  aa:=cs[1]; bb:=cs[3]; cc:=cs[4]; dd:=cs[2]; iv;
  aa:=cs[1]; bb:=cs[4]; cc:=cs[2]; dd:=cs[3]; iv;
  aa:=cs[1]; bb:=cs[4]; cc:=cs[3]; dd:=cs[2]; iv;
  setlinestyle(0,0,1); circle(kx0, ky0, round(r));
                       { gömb képkontúrja }
end;

begin
  r:=a*sqrt(3);
  cs[1].uj(a,a,a); cs[2].uj(-a,-a,a); cs[3].uj(a,-a,-a); cs[4].uj(-
a,a,-a);
.....................................
  rajzol;
.....................................
end.
```



2.71. ábra

A *2.72. ábrán* látható felületet olyan egyenesek alkotják, amelyek egy, az ábrán függőleges helyzetű z koordináta-tengelyen levő $[-r, r]$ intervallumnak megfelelő egyenes szakaszra:

$$\begin{cases} x = 0 \\ y = 0 \\ z = 2\sigma r(1-2v) \end{cases} \quad ; \quad v \in [0,1], \quad \sigma \in \{-1,1\}$$

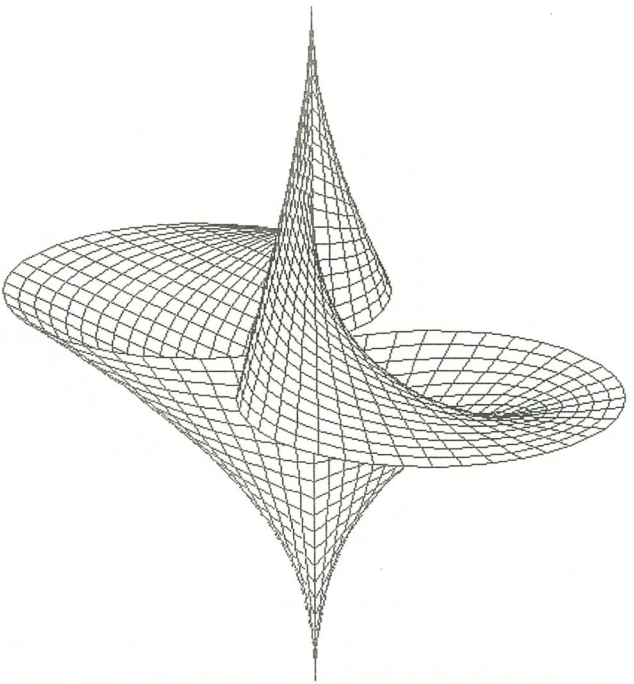
és egy két r sugarú, érintkező körből álló vezérgörbére, ahol a körök középpontjai a (0,r,0) illetve a (0,–r,0) pontok:

$$\begin{cases} x = r \sin 2\pi v \\ y = \sigma r(1-\cos 2\pi v) \\ z = 0 \end{cases} \quad ; \quad v \in [0,1], \quad \sigma \in \{-1,1\}$$

támaszkodnak. A felület ábrázolt részének paraméteres előállítása:

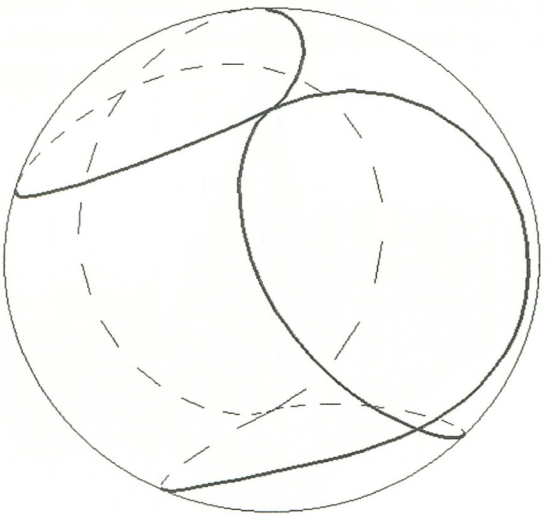
$$\begin{cases} x(u,v) = r u \sin 2\pi v \\ y(u,v) = \sigma r u(1-\cos 2\pi v) \\ z(u,v) = \sigma 2r(1-u)(1-2v) \end{cases} \quad ; \quad \begin{cases} u \in [0,1] \\ v \in [0,1] \end{cases}$$

ahol σ egy előjel, amely az egyik körre illeszkedő felületdarab esetén pozitív, míg a másiknál negatív. Az u=áll. paramétervonalak zárt görbék, míg az v=áll. paramétervonalak egyenesek.



2.72. ábra

Az egész szerkesztést egy *2r* sugarú gömbbe írjuk, majd az előbbi példához hasonlóan a torzfelületet kiterjesztjük negatív, illetve 1-nél nagyobb pozitív *u* értékekre és megrajzoljuk a gömbbel való áthatási görbéjét, amely a *2.73. ábrán* látható.



2.73. ábra

A szóban forgó felületnek az y illetve a z tengelyek önáthatási vonalai.

2.8. Nem geometriai tulajdonság szemléltetése

Számos műszaki alkalmazás kapcsán felvetődik bizonyos mennyiségek görbe felületeken való ábrázolásának feladata. Ilyen például a hőmérséklet-eloszlás alkatrészek felületén. A feladat megoldásának legszemléletesebb módja a szintvonalas illetve - ha színes ábrát készítünk - a szintsávos ábrázolás.

A 2.65 paraméteres egyenletekkel előállított G felületen értelmezzük az ábrázolandó $f : G \to \Re$ függvényt. Az α névleges értékű szintvonal képlete:

$$M(\alpha) = \left\{(x,y,z) \in G \;\middle|\; f(x,y,z) = \alpha\right\} \tag{2.112}$$

ami tulajdonképpen egy implicit összefüggés a felület u és v paraméterei között, tehát egy, a felületen fekvő görbe egyenlete.

Ha az ábrázolandó függvény alsó és felső korlátját *mn* illetve *mx* jelölik, akkor *p+1* egyenlő közű szintvonal névleges értékei:

$$\alpha_i = \left(1 - \frac{i}{p}\right) mn + \frac{i}{p} mx; \qquad i \in \{0,1,\ldots,p\} \tag{2.113}$$

A szintvonalak rajzolása úgy történik, hogy a felületet paramétervonalai mentén végigpásztázzuk és azokat a pontokat, amelyekben a 2.112 képletben szereplő egyenlőség megközelítőleg teljesül, megjelenítjük. Gyakorlatilag azokról a pontokról van szó, amelyekben

$$f(x(u,v),y(u,v),z(u,v)) \cdot f(x(u+\delta,v),y(u+\delta,v),z(u+\delta,v)) \le 0 \tag{2.114}$$

a v = áll. paramétervonalak mentén, vagy

$$f(x(u,v),y(u,v),z(u,v)) \cdot f(x(u,v+\delta),y(u,v+\delta),z(u,v+\delta)) \le 0 \tag{2.115}$$

az u = áll. paramétervonalak mentén. Minél kisebb a δ paraméterlépés, annál folytonosabb lesz a kapott szintvonalkép.

A *2.74. ábrán* az

$$\begin{cases} x = r\sin\vartheta\cos\psi \\ y = r\sin\vartheta\sin\psi \\ z = r\cos\vartheta \end{cases} ; \quad \begin{cases} \vartheta \in [0,\pi] \\ \psi \in [0,2\pi] \end{cases}$$

előállítású gömbfelületen értelmezett

$$f(x,y,z) = \frac{\sqrt{a^2x^2 + b^2y^2 + c^2z^2}}{r}$$
$$a > b > c$$

függvény szintvonalai láthatók.

A szintvonalak képletei:

$$\sqrt{(a\sin\vartheta\cos\psi)^2+(b\sin\vartheta\sin\psi)^2+(c\cos\vartheta)^2}=\left(1-\frac{i}{p}\right)c+\frac{i}{p}a,$$

$$i\in\{0,1,\ldots,p\}$$

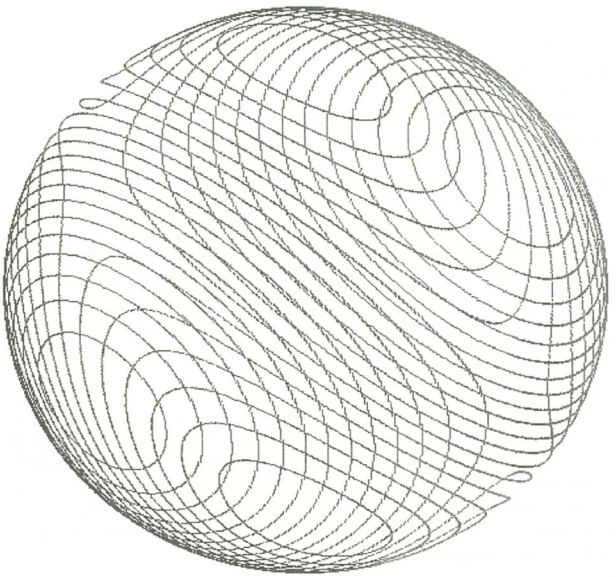
A *2.75. ábrán* a gömb tömör, nem átlátszó, ezért a túloldalán levő szintvonalak nem látszanak. Paraméteres előállítású felületek esetén a felület normálisvektora (2.69) és a vetítősugár irányvektora (2.35) közötti skaláris szorzat előjele határozza meg a felületpont láthatóságát. Az (u,v) paraméterpárnak megfelelő pont akkor látható, ha

$$\begin{vmatrix} a_p & b_p & c_p \\ \frac{\partial x}{\partial u}(u,v) & \frac{\partial y}{\partial u}(u,v) & \frac{\partial z}{\partial u}(u,v) \\ \frac{\partial x}{\partial v}(u,v) & \frac{\partial y}{\partial v}(u,v) & \frac{\partial z}{\partial v}(u,v) \end{vmatrix} > 0 \qquad (2.116)$$

Gömb esetében a normálisvektor kollineáris a sugárral, tehát a 2.116 láthatósági feltétel az

$$a_p x+b_p y+c_p z>0$$

lesz.

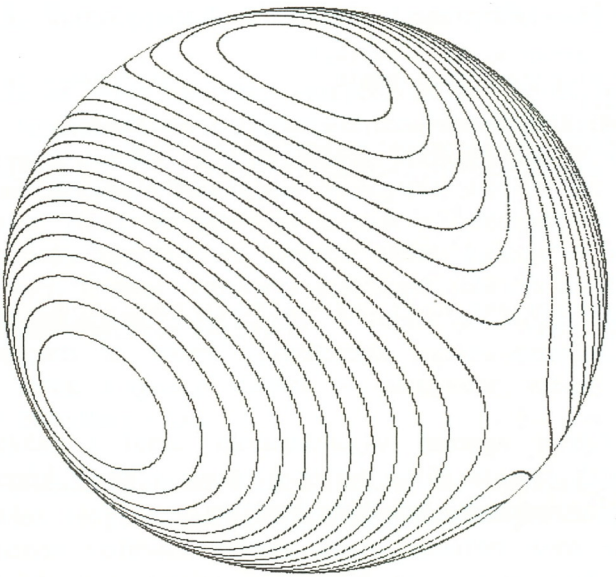


2.74. ábra

A *2.74. ábrát* a lemezmellékleten található GBSZVATL.PAS nevű program rajzolta, míg a *2.75. ábrát* a GBSZINTV.PAS program, amelyből az alábbi részletek származnak:

```
const
  m=1200; n=800; pim=2*pi/m; pin=pi/n;
  kx0=320; ky0=240; s=50;
  a=6; b=4; c=3;
  rg=4.6;
  vsz=20; svsz=(a-c)/vsz; { szintvonalak száma és névleges értéke }
  ap=1; bp=1; cp=1;

function fx(k,l: integer): real; { a gömb paraméteres előállítása }
begin fx:=rg*cos(k*pim)*sin(l*pin); end;
function fy(k,l: integer): real;
begin fy:=rg*sin(k*pim)*sin(l*pin); end;
function fz(k,l: integer): real;
begin fz:=rg*cos(l*pin); end;

function fr(k,l: integer): real;          { az ábrázolandó függvény }
begin
fr:=sqrt(sqr(a*cos(k*pim)*sin(l*pin))+sqr(b*sin(k*pim)*sin(l*pin))
              +sqr(c*cos(l*pin))); end;
```

```
procedure rajzol;
var
  j, k, l: integer;
  dl, dt: array[0..vsz] of real;
  nv1, nv2, f1, f2: real;
  x, y, z, u: real;
  kx, ky: integer;

procedure rzpt;       { szintvonalpont megjelenítése }
begin
  x:=fx(k,l); y:=fy(k,l); z:=fz(k,l);
  if ap*x+bp*y+cp*z>=0 then begin    { láthatósági feltétel }
    kx:=round(s*(-bp*x+ap*y)/nv1);
    ky:=round(s*(-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2);
    putpixel(kx0+kx,ky0-ky,15);
  end;
end;

begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  setcolor(7); circle(kx0, ky0, round(rg*s)); setcolor(15);
  { gömb határköre }
  for j:=0 to vsz do dt[j]:=0;
  for k:=0 to m do          { pásztázás délkörök mentén }
    for l:=0 to n do begin
      u:=fr(k,l);
      for j:=0 to vsz do begin
        dl[j]:=c+svsz*j-u;
        if l>0 then
          if dl[j]*dt[j]<=0 then rzpt;     { szintvonalkeresztezés }
                                           { feltétele }
      end;
      dt:=dl;
    end;
  for l:=0 to n do              { pásztázás szélességi körök mentén }
    for k:=0 to m do begin
      u:=fr(k,l);
      for j:=0 to vsz do begin
        dl[j]:=c+svsz*j-u;
        if k>0 then
          if dl[j]*dt[j]<=0 then rzpt;     { szintvonalkeresztezés }
                                       { feltétele }
      end;
      dt:=dl;
    end;
end;
```



2.75. ábra

Ugyanazt a feladatot szintsávok segítségével a GBSZTSAV.PAS nevű program oldja meg. Amikor szintsávokat ábrázolunk, a felület minden pontját megjelenítjük, csak a pontok színe változik, attól függően, hogy az illető pontban az ábrázolt függvény értéke melyik szintsávban helyezkedik el.

A program megjelenítő eljárása:

```
const
  a=6; b=4; c=3;
  sv=11/(a-c);        { szintsávszélesség fordítottja }
.................................

procedure rajzol;
var
  k, l: integer;
  nv1, nv2: real;
  x, y, z, u: real;
  kx, ky: integer;
```

```
procedure rzpt;
begin
  x:=fx(k,l); y:=fy(k,l); z:=fz(k,l);
  if ap*x+bp*y+cp*z>=0 then begin          { láthatósági feltétel }
    u:=fr(k,l);
    kx:=round(s*(-bp*x+ap*y)/nv1);         { vetítés }
    ky:=round(s*(-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2);
    putpixel(kx0+kx,ky0-ky,1+round(sv*(u-c)));      { szintsávnak }
                                  { megfelelő színű pontot rajzol }
  end;
end;

begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*sqrt(sqr(nv1)+sqr(cp));
  for k:=0 to m do                  { a teljes felület lefedése }
    for l:=0 to n do rzpt;
end;
```

Elegendő a felület egyszeri végigpásztázása, azzal a feltétellel, hogy a paramétervonalak annyira sűrűn legyenek, hogy a gömb képének megfelelő képernyőpontok mindegyike befestésre kerüljön.

3. Középpontos vetítés (perspektíva)

A látás folyamán az emberi szemben középpontos, más néven centrális vetítés történik. Kissé leegyszerűsítve a folyamatot, azt mondhatjuk, hogy a vetítősugarak egy ponton mennek át, a szemlencse középpontján. Mivel a szemlencse méretei a megfigyelt alakzatokéhoz viszonyítva elenyészőek, a szóbanforgó egyszerűsítés nem jelent lényegbevágó eltérést a valóságtól. A kép a szem belsejében, a retinán képződik, és fordított állású, kicsinyített mása annak a képnek, amely egy, a szem tengelyére merőleges, átlátszó ernyőre lenne rajzolható. Itt hallgatólagosan feltételeztük a retina sík voltát, ami ugyancsak nem hiba, ugyanis a retina görbületéből eredő torzulást az agy a képfeldolgozás során kiigazítja (amint a szemlencse valódi méretéből és alakjából eredő torzulásokat is).

Középpontos vetítéssel készült ábrákat főként az építészetben készítenek. Ezenkívül minden fénykép valójában perspektíva. Tulajdonképpen soha nem látunk párhuzamos vetítéssel készült képet, azonban apró tárgyak szemlélésekor az eltérés nem annyira nyilvánvaló, mint nagyméretű alakzatok esetén.

3.1. A vetítés elve

A perspektív vetítési rendszer alapelemei a vetítési középpont (szem) és a képsík. Ha a képsík függőleges, akkor álló képsíkú perspektíváról beszélünk, ha nem, akkor dőlt képsíkú perspektíváról. Ha magas toronyból szemlélünk egy várost, akkor az amit látunk madárperspektíva. Ha a toronyra a töve közeléből nézünk fel, békaperspektíváról van szó.

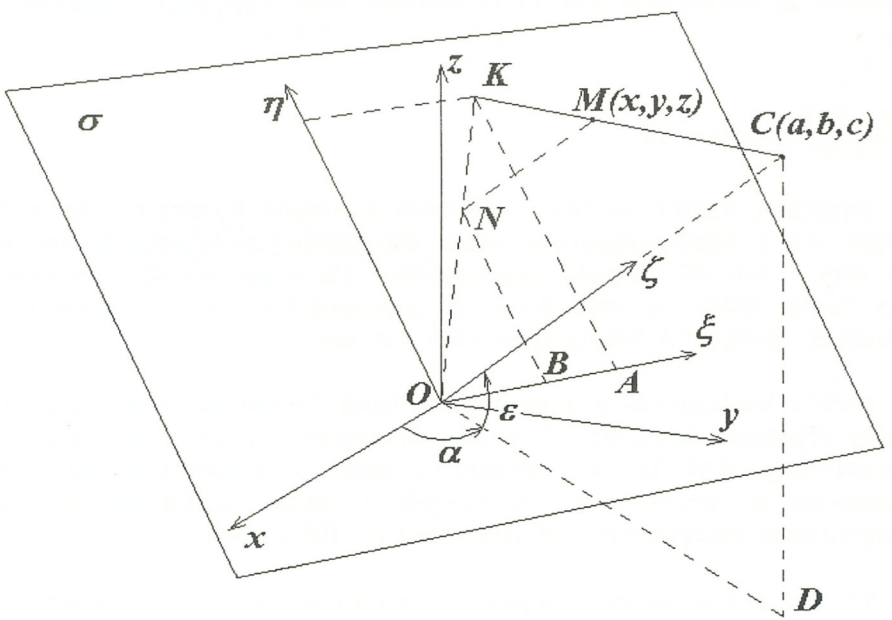
A vetítési középpontból a képsíkra merőlegest fősugárnak, ennek talppontját pedig főpontnak nevezzük. A vetítési középpont képsíktól való távolsága a szemtávolság. Álló képsíkú perspektíva esetén a képsík főponton átmenő vízszintese a horizontvonal. A perspektív vetítési rendszerről és ennek megválasztási szempontjairól bővebben lásd [3] 105-123.old.

A *3.1. ábrán* C a vetítési középpont (centrum), σ a képsík, O a főpont, M az ábrázolandó alakzat egy pontja, CK az ehhez tartozó vetítősugár. A CO

fősugár merőleges a képsíkra, így a vetítési középponttal a képsíkot is megadtuk, amelynek egyenlete az *xyz* rendszerben tehát:

$$ax + by + cz = 0 \tag{3.1}$$

A képsík ilyetén való megválasztása nem jelent megszorítást, ugyanis párhuzamos eltolással az ábrázolandó alakzat bármikor olyan helyzetbe hozható, hogy az *xyz* koordinátarendszer megfeleljen az eleve adott képsík-centrum elrendezésnek. Egy új, ξ η ζ koordinátarendszert határozunk meg úgy, hogy kezdőpontja ugyancsak *O* legyen, a ζ tengely a centrum felé mutasson, az η tengely a *z* tengely képsíkra eső merőleges vetülete, ξ pedig a képsíkban levő, az előző kettőre merőleges tengely, amely a centrumból nézve jobbra mutat. Ezt a rendszert használtuk a párhuzamos vetítés esetében is (*2.1. ábra*) csak ott nem határoztuk meg explicit módon a ζ tengelyt, ugyanis a vetület képsíkbeli koordinátáinak meghatározásához nem volt szükség a pont képsíktól való távolságára.



3.1. ábra

Az M pont koordinátái az új rendszerben:

$$\begin{cases} \xi_m = \dfrac{-bx + ay}{\sqrt{a^2 + b^2}} \\ \eta_m = \dfrac{-c(ax + by) + (a^2 + b^2)z}{d\sqrt{a^2 + b^2}} \\ \zeta_m = \dfrac{ax + by + cz}{d} \end{cases} \tag{3.2}$$

ahol d a szemtávolság:

$$d = \sqrt{a^2 + b^2 + c^2} \tag{3.3}$$

Az MNK és COK illetve NBO és KAO hasonló háromszögek oldalaira felírható hasonlósági arányok segítségével meghatározhatók a K pont képsíkbeli koordinátái:

$$\begin{cases} \xi_k = \dfrac{d\xi_m}{d - \zeta_m} \\ \eta_k = \dfrac{d\eta_m}{d - \zeta_m} \end{cases} \tag{3.4}$$

Az M ponthoz tartozó vetítősugár irányvektora (nem egységvektor):

$$\vec{p}_m = (x - a)\vec{i} + (y - b)\vec{j} + (z - c)\vec{k} \tag{3.5}$$

Az irányított sokszögek láthatóságának meghatározásához és a görbe felületű testek képhatárának meghúzásához lesz rá szükség.

A centrum *xyz* rendszerre vonatkozó térbeli helyzetét a szemtávolság mellett meghatározó azimut (α) és emelkedés (ε) képletei:

$$\begin{cases} \alpha = \begin{cases} \arctan\dfrac{b}{a} & \text{ha} \quad a >= 0 \\ \pi + \arctan\dfrac{b}{a} & \text{ha} \quad a < 0 \end{cases} \\ \varepsilon = \arctan\dfrac{c}{\sqrt{a^2 + b^2}} \end{cases} \tag{3.6}$$

A *3.1. ábrán* a *D* pont a *C* centrum *xy* síkra eső merőleges vetületét jelöli.

Álló képsíkú perspektíva esetén

$$c = 0 \quad ; \qquad d = \sqrt{a^2 + b^2} \quad ; \quad \varepsilon = 0 \tag{3.7}$$

és a (3.2) transzformáció

$$\begin{cases} \xi_m = \dfrac{-bx + ay}{\sqrt{a^2 + b^2}} \\ \eta_m = z \\ \zeta_m = \dfrac{ax + by}{\sqrt{a^2 + b^2}} \end{cases} \tag{3.8}$$

alakú, amit a (3.4) vetítési képletbe behelyettesítve meghatározható a vetület helyzete a képsíkban felvett ξ η koordinátarendszerben.

Felülnézeti perspektív kép készítésekor:

$$a = b = 0 \quad ; \quad c = d \quad ; \quad \varepsilon = \frac{\pi}{2} \tag{3.9}$$

és a két koordinátarendszer egybeesik:

$$\begin{cases} \xi_m = x \\ \eta_m = y \\ \zeta_m = z \end{cases} \tag{3.10}$$

a vetítés képletei pedig:

$$\begin{cases} \xi_k = \dfrac{dx}{d-z} \\ \eta_k = \dfrac{dy}{d-z} \end{cases} \tag{3.11}$$

A párhuzamos vetítéshez hasonlóan itt is kétféleképpen járhatunk el. Kisebb, vagy térben tetszőleges helyzetet felvehető tárgyakról (például űrhajók, műholdak) célszerű felülnézetet készíteni, majd a tárgyat elforgatni, hogy a kívánt irányból szemlélhessük. A tárgyforgatási képletek:

- tetszőleges tengely körül: 2.27
- koordinátatengelyek körül: 2.28, 2.29 és 2.30 (2.2.3 alpont).

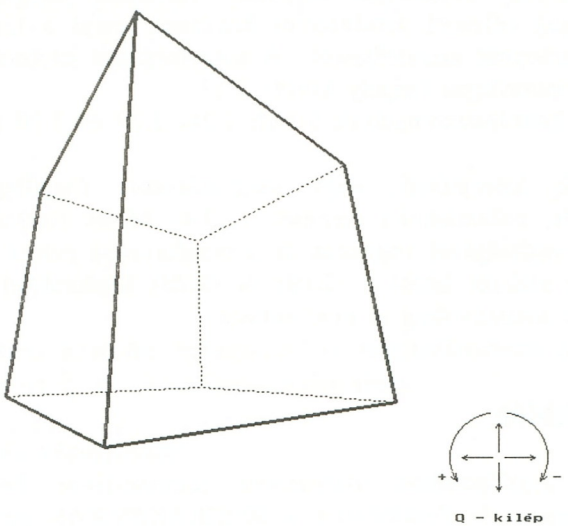
Nagyobb kiterjedésű, vagy meghatározott függőleges tengelyű alakzatok (épületek, műemlékek) vetítését a (3.4) képlet (természetesen 3.2 és 3.3-mal együtt) segítségével végezzük és a megfigyelési pontot (centrum) mozgatjuk az ábrázolt alakzat körül a (2.18) és (2.20) képletekkel (2.2.2 alpont). Szükség esetén a szemtávolság is módosítható.

3.2. Példák

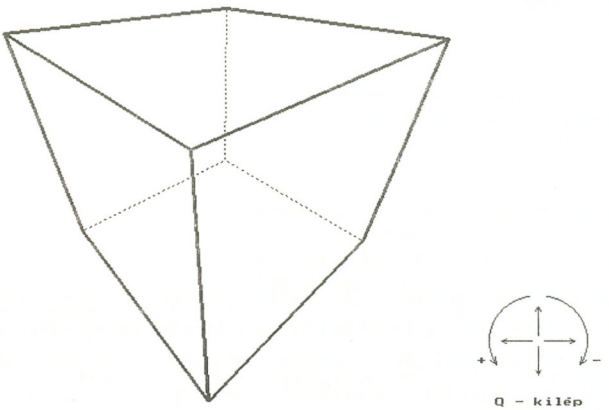
a) A kezdőponthoz viszonyítva szimmetrikus helyzetű kockáról készít perspektív felülnézetet a KOCKAKPV.PAS program, lehetővé téve az alakzat forgatását is.

A kocka látható éleit vastag, folytonos, a nem láthatóakat pedig vékony, szaggatott vonallal kívánjuk meghúzni. Ezt úgy végezzük el, hogy vastag, folytonos vonaltípus mellett megjelenítjük a látható lapokat, majd vékony, szaggatott vonallal meghúzzuk a kocka minden élét (a már megrajzolt éleket ez nem befolyásolja). A kocka konvex poliéder, ezért az egyes lapok megjele-

nítésének feltétele az, hogy a vetítési középpontból a lap külső oldalát lássuk, ezt pedig a vetítősugár irányvektora és a lap kifelé mutató normálisvektora közötti skaláris szorzat előjeléből tudjuk meghatározni. Centrális vetítésnél egy alakzat különböző pontjaihoz különböző vetítősugár tartozik, síkidomok esetében azonban a láthatóság ennek ellenére egyértelműen eldönthető. Elegendő a láthatósági feltételt a síkidom egy pontjára megvizsgálni, az eredmény a síkidom minden pontjában ugyanaz lesz. Jelen feladat esetében kézenfekvő a kocka lapjának egyik csúcsára végezni el a számításokat. A normálisvektor a síkidom minden pontjában ugyanaz ((2.33) és (2.34) képletek a 2.3.2 alpontban). A vetítősugár irányvektorának koordinátáit a (3.5) képletből kapjuk meg. Ha az említett skaláris szorzat negatív, akkor a lap külső oldalát látjuk (a normálisvektor felénk mutat), ha pozitív, akkor a belső oldalát, vagyis az adott helyzetben az illető lap a centrumból nem látható. A *3.2* és *3.3 ábrákon* a kocka perspektív képe látható, két helyzetben.



3.2. ábra



3.3. ábra

A megjelenítést végző eljárás:

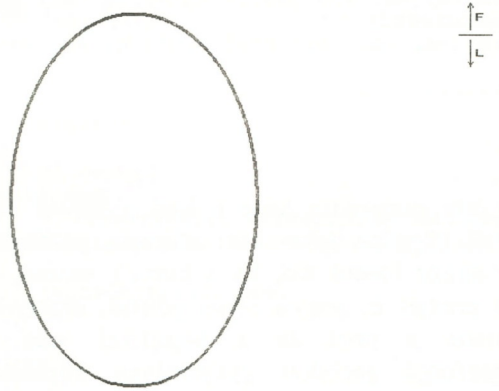
```
const
  a=100; d=250;                    {kocka élhossza és a szemtávolság}
  e: array[1..6,1..4] of byte = {csúcsok-lapok összerendelés}
     ((1, 2, 3, 4), (1, 5, 8, 2), (2, 8, 7, 3), (3, 7, 6, 4),
     (4, 6, 5, 1), (5, 6, 7, 8));
  f: array[1..12,1..2] of byte =          {csúcsok-élek összerendelés}
     ((1, 2), (2, 3), (3, 4), (4, 1), (1, 5), (2, 8), (3, 7), (4, 6),
     (5, 6), (6, 7), (7, 8), (8, 5));
  kx0=320; ky0=240;
.....................................................

type
  pont = object
    x, y: integer;
    procedure uj(ux, uy: integer);
  end;
  csucs = object
    xx, yy, zz: real;
    csp: pont;
    procedure uj(ux, uy, uz: real);
    procedure vt;
  end;
  nsz = object                {lap adatait tároló objektum}
    cs: array[1..4] of csucs;
    s: array[1..4] of pont;
    lth: boolean;
    procedure uj(k: byte);
```

```
    procedure lt;
  end;

var
  gcs: array[1..8] of csucs;
  lp: array[1..6] of nsz;
  k, l: integer;
.....................................

procedure pont.uj(ux, uy: integer);
begin x:=ux; y:=uy; end;
procedure csucs.uj(ux, uy, uz: real);
begin xx:=ux; yy:=uy; zz:=uz; end;
procedure csucs.vt;              {felülnézet középpontos vetítésben}
begin csp.uj(round(kx0+d*xx/(d-zz)), round(ky0-d*yy/(d-zz))); end;
procedure nsz.uj(k: byte);
begin for l:=1 to 4 do cs[l]:=gcs[e[k,l]]; end;
procedure nsz.lt;
var
  al, bl, cl, ap, bp, cp: real;
begin
  al:=cs[1].yy*(cs[2].zz-cs[3].zz)+cs[2].yy*(cs[3].zz-cs[1].zz)
                                    { a normálisvektor }
      +cs[3].yy*(cs[1].zz-cs[2].zz); { koordinátái      }
  bl:=cs[1].zz*(cs[2].xx-cs[3].xx)+cs[2].zz*(cs[3].xx-cs[1].xx)
                                  +cs[3].zz*(cs[1].xx-cs[2].xx);
  cl:=cs[1].xx*(cs[2].yy-cs[3].yy)+cs[2].xx*(cs[3].yy-cs[1].yy)
                                  +cs[3].xx*(cs[1].yy-cs[2].yy);
  ap:=-cs[1].xx;  bp:=-cs[1].yy;  cp:=b-cs[1].zz;
  { a vetítősugár iránytényezői }
  if al*ap+bl*bp+cl*cp > 0 then lth:=true else lth:=false;
  { láthatósági feltétel }
end;

procedure rajz;
begin
  for k:=1 to 8 do gcs[k].vt;   {   csúcsok vetítése   }
  for k:=1 to 6 do begin lp[k].uj(k); lp[k].lt; end;
  { lapok aktualizálása }
  bar(0,0,640,480); setlinestyle(0,0,3);
  for k:=1 to 6 do
    if lp[k].lth=true then begin
      for l:=1 to 4 do lp[k].s[l]:=lp[k].cs[l].csp;
      fillpoly(4, lp[k].s);     { látható lap megjelenítése }
    end;
  setlinestyle(1,0,1);
  for k:=1 to 12 do  { kocka éleinek rajzolása szaggatott vonallal }
    line(gcs[f[k,1]].csp.x, gcs[f[k,1]].csp.y,
         gcs[f[k,2]].csp.x, gcs[f[k,2]].csp.y);
end;
```

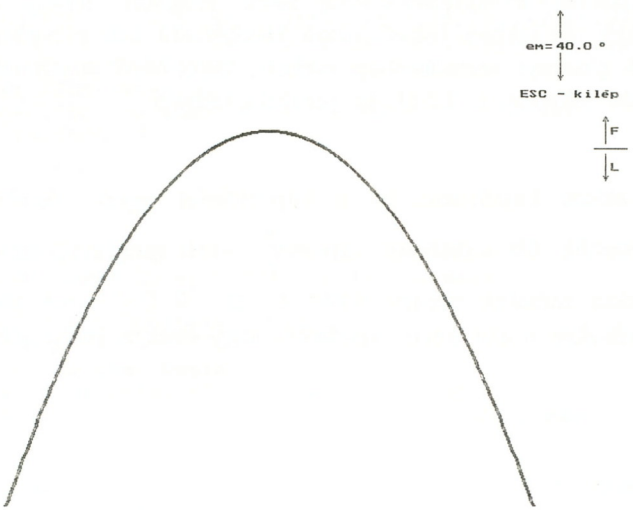
```
begin
.............................
  gcs[1].uj(a, a, a); gcs[2].uj(-a, a, a); gcs[3].uj(-a, -a, a);
     { csúcsok }
  gcs[4].uj(a, -a, a); gcs[5].uj(a, a, -a); gcs[6].uj(a, -a, -a);
     { koordinátái }
  gcs[7].uj(-a, -a, -a); gcs[8].uj(-a, a, -a);
  rajz;
.............................
end.
```

b) Egy kör perspektív képe a kör, a képsík és a centrum kölcsönös helyzetétől függően lehet kör, ellipszis, parabola vagy hiperbola is. Utóbbi eset akkor fordul elő, ha a körnek vannak a megfigyelő *"háta mögött"* levő pontjai is, vagyis olyan pontok, amelyekhez tartozó vetítősugáron a centrum a pont és a képsíkkal való döféspont között van. A szóbanforgó görbéket gyűjtőnéven kúpszeleteknek is nevezik, mivel kúpfelület síkkal való áthatásaként is előállíthatók. Mivel a centrum és az ábrázolandó kör egy kúpfelületet határoz meg, a képsík pedig az ezt metsző sík, az említett görbék létrejötte könnyen belátható. Ezt szemlélteti a KORKPV.PAS nevű program, amely egy 50 pixelnyi sugarú, *xy* síkban fekvő, origó középpontú kör perspektív képét rajzolja 38.3 pixelnyi szemtávolság mellett, különböző emelkedés esetén. A *3.4. ábrán* ellipszis, a 3.5.ábrán parabola látható.

Parabola akkor keletkezik, ha a kúpfelületet egyik alkotójával párhuzamos síkkal metsszük. Ez a helyzet $\operatorname{tg}\varepsilon = \frac{d}{r}$ -nek megfelelő emelkedés esetén áll elő, az adott méretek mellett tehát $\varepsilon \cong 40°$ -os emelkedésre. Ennél kisebb értékekre a kör képe hiperbola, nagyobbakra pedig ellipszis lesz.

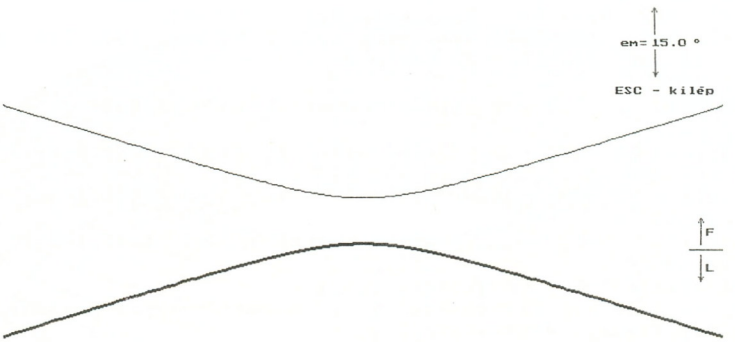


3.4. ábra



3.5. ábra

A *3.6. ábrán* látható hiperbolának csak a vastag vonallal rajzolt ága látszik a centrumból, a vékonyan rajzolt ág a centrumban álló megfigyelő háta mögött levő körív centrális vetülete.



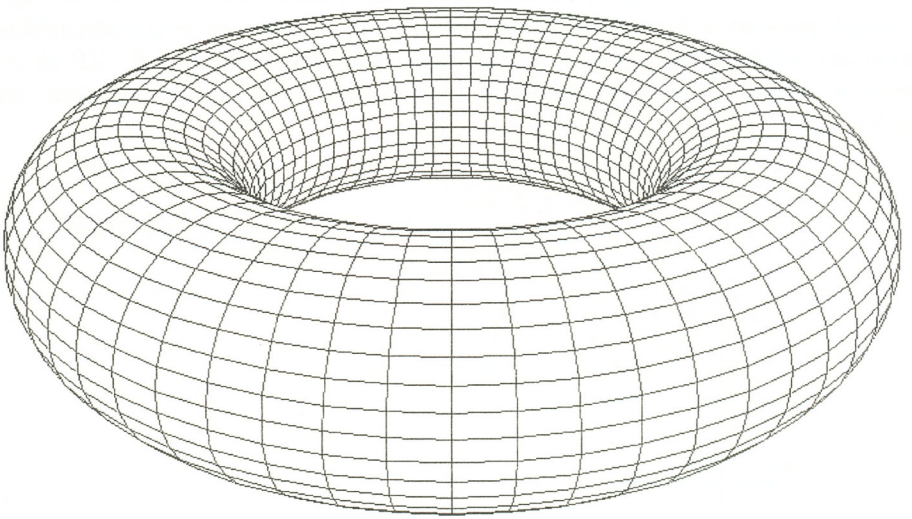
3.6. ábra

c) A *3.7. ábrán* egy tórusz perspektív képe látható. A tórusz torokköre az *xy* síkban fekszik, középpontja az origóban található. Az ábrát az alábbi, a GYURUKPV.PAS nevű programban található eljárás rajzolta:

```
const
  m=36; n=60; pim=2*pi/m; pin=2*pi/n;
  kx0=320; ky0=200;
  rr=220; r=80;
  ap=-720.0; bp=0; cp=360.0;

type
  pt=object                              { képernyőpont-objektum }
    xp, yp: integer;
    procedure uj(ux, uy: integer); end;
  nsz=object                             { felületelemkép-objektum }
    c1, c2, c3, c4: pt;
    procedure uj(u1, u2, u3, u4: pt); end;
procedure pt.uj(ux, uy: integer);
begin xp:=ux; yp:=uy; end;
procedure nsz.uj(u1, u2, u3, u4: pt);
begin c1:=u1; c2:=u2; c3:=u3; c4:=u4; end;

function fx(k,l: integer): real;{ a felület paraméteres előállítása }
begin fx:=(rr+r*sin(k*pim))*cos(l*pin); end;
function fy(k,l: integer): real;
begin fy:=(rr+r*sin(k*pim))*sin(l*pin); end;
function fz(k,l: integer): real;
begin fz:=r*cos(k*pim); end;
```

```
procedure rajzol;
var
  k, l: integer;
  nv1, nv2, d, al, bl, cl: real;
  x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4: real;
  kx1, ky1, kx2, ky2, kx3, ky3, kx4, ky4 : integer;
  lx1, ly1, lz1, lx2, ly2, lz2, lx3, ly3, lz3, lx4, ly4, lz4: real;
  s1, s2, s3, s4, t1, t2, t3, t4: pt;
  hl, hm: nsz;
procedure rzhl;       { egy felületelemet rajzoló eljárás }
begin
  x1:=fx(k,l); x2:=fx(k-1,l); x3:=fx(k-1,l-1); x4:=fx(k,l-1);
   { felületelem }
  y1:=fy(k,l); y2:=fy(k-1,l); y3:=fy(k-1,l-1); y4:=fy(k,l-1);
   { sarkainak  }
  z1:=fz(k,l); z2:=fz(k-1,l); z3:=fz(k-1,l-1); z4:=fz(k,l-1);
   { koordinátái }
  al:=(y2-y1)*(z3-z1)-(y3-y1)*(z2-z1);
  bl:=(z2-z1)*(x3-x1)-(z3-z1)*(x2-x1);{ normálisvektor koordinátái }
  cl:=(x2-x1)*(y3-y1)-(x3-x1)*(y2-y1);
  if al*(x1-ap)+bl*(y1-bp)+cl*(z1-cp) < 0 then begin
    { láthatósági feltétel }
    lx1:=(-bp*x1+ap*y1)/nv1;
    ly1:=(-cp*(ap*x1+bp*y1)+sqr(nv1)*z1)/nv2;
    lz1:=(ap*x1+bp*y1+cp*z1)/d; { változócsere }
    lx2:=(-bp*x2+ap*y2)/nv1;
    ly2:=(-cp*(ap*x2+bp*y2)+sqr(nv1)*z2)/nv2;
    lz2:=(ap*x2+bp*y2+cp*z2)/d;
    lx3:=(-bp*x3+ap*y3)/nv1;
    ly3:=(-cp*(ap*x3+bp*y3)+sqr(nv1)*z3)/nv2;
    lz3:=(ap*x3+bp*y3+cp*z3)/d;
    lx4:=(-bp*x4+ap*y4)/nv1;
    ly4:=(-cp*(ap*x4+bp*y4)+sqr(nv1)*z4)/nv2;
    lz4:=(ap*x4+bp*y4+cp*z4)/d;
    kx1:=round(d*lx1/(d-lz1));
    ky1:=round(d*ly1/(d-lz1));        { képkoordináták kiszámítása }
    kx2:=round(d*lx2/(d-lz2));
    ky2:=round(d*ly2/(d-lz2));
    kx3:=round(d*lx3/(d-lz3));
    ky3:=round(d*ly3/(d-lz3));
    kx4:=round(d*lx4/(d-lz4));
    ky4:=round(d*ly4/(d-lz4));
    s1.uj(kx0+kx1,ky0-ky1); s2.uj(kx0+kx2,ky0-ky2);
    { képpont-objektumok }
    s3.uj(kx0+kx3,ky0-ky3); s4.uj(kx0+kx4,ky0-ky4);
    { betöltése          }
    t1.uj(kx0-kx1,ky0-ky1); t2.uj(kx0-kx2,ky0-ky2);
    t3.uj(kx0-kx3,ky0-ky3); t4.uj(kx0-kx4,ky0-ky4);
    hl.uj(s1,s2,s3,s4); hm.uj(t1,t4,t3,t2);
    { felületelemkép-objektumok   }
    fillpoly(4,hl); fillpoly(4,hm);   { megjelenítés }
  end;
end;
begin
  nv1:=sqrt(sqr(ap)+sqr(bp));
  d:=sqrt(sqr(nv1)+sqr(cp));                { szemtávolság }
```

```
  nv2:=nv1*d;
  for l:= 1 to round(n/2) do
    for k:= 1 to m do rzhl;
end;
```

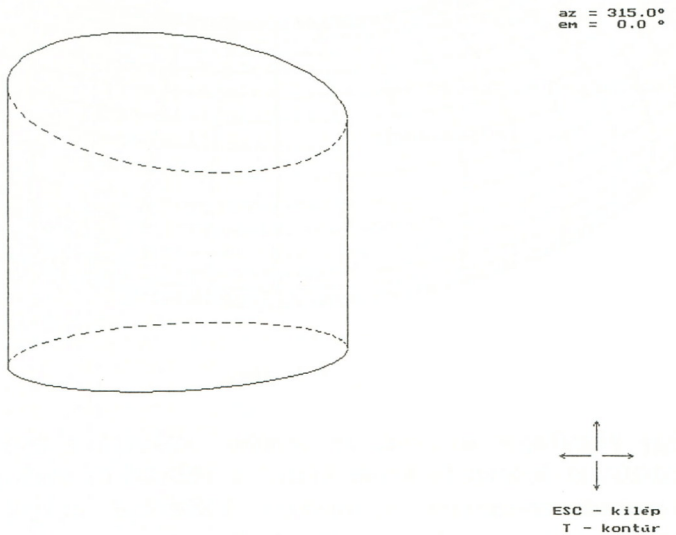


3.7. ábra

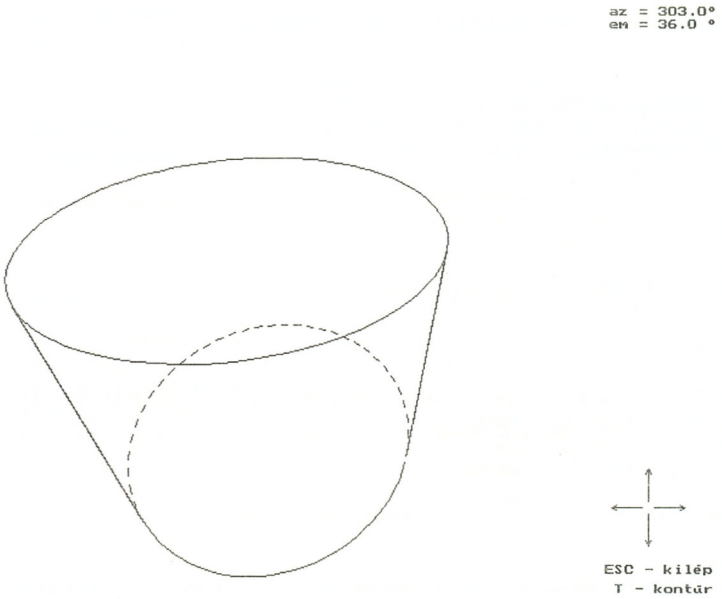
d) Egy függőleges tengelyű, az origóhoz viszonyítva excentrikus, egyenes körhenger perspektív képét készíti a HENGKPV.PAS program, az *xyz* koordinátarendszerhez viszonyított különböző azimut és emelkedés esetén. A henger alapköre 50 pixellel az *xy* sík alatt, fedőköre pedig 100 pixellel az *xy* sík fölött, ezzel párhuzamos síkban fekszik, tengelye pedig a *z* tengellyel párhuzamos és az *xy* síkot a (0,-80) pontban metszi. A program az alapkörök látható íveit folytonos, a nem láthatóakat szaggatott vonallal rajzolja. A láthatóság eldöntésére a következő algoritmust használja:

- ha a körnek a centrumból a külső oldala látszik, akkor a kör teljes egészében látható.
- ha a körnek a centrumból a belső oldala látszik, akkor azok az elemi ívek, amelyekben a palást normálisvektora a centrum felé mutat, láthatóak, a többiek nem láthatóak.

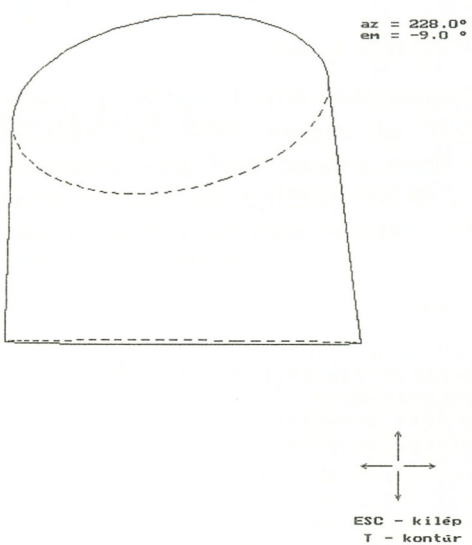
A program a hengerpalást képkontúrját is meghúzza. Az alapkörökkel párhuzamos köröknek azokat a pontjait jeleníti meg, amelyekben a vetítősugár megközelítőleg merőleges a normálisvektorra. Mivel ez az eljárás viszonylag lassú, a palástkontúr megrajzolására csak külön parancs esetén kerül sor. Így a centrum gyorsan a kívánt helyzetbe hozható az azimut- és emelkedésmódosító parancsok segítségével, majd a palástkontúr is meghúzható. A *3.8*, *3.9* és *3.10 ábrákon* látható a henger különböző megfigyelési irány mellett készített perspektív képe.



3.8. ábra



3.9. ábra



3.10. ábra

Az alapköröket és a palástkontúrt rajzoló eljárások:

```
const
  r=100; d=300; x0=0; y0=-80; z01=-50; z02=100;
  n=100; pin=2*pi/n; nk=1000; pink=2*pi/nk; nl=500; eps=100;
  kx0=320; ky0=240;

var
  ap, bp, cp, nv1, nv2, z0, az, em: real;
  cn, al, bl, cl: real;
  x, y, z: real;
  kx, ky: integer;
  k, l: integer;
  ka, ke: string;
  fl: byte;

function fx(t: real): real;{ az alapkörök parametrikus előállítása }
begin fx:=x0+r*cos(t); end;
function fy(t: real): real;
begin fy:=y0+r*sin(t); end;

procedure azimut;          { azimut kiszámítása }
begin
  if bp>=0
    then if ap=0 then az:=pi/2 else if ap>0 then az:=arctan(bp/ap)
                                            else az:=pi+arctan(bp/ap)
    else if ap=0 then az:=3*pi/2 else if ap>0 then
az:=2*pi+arctan(bp/ap)
                                                 else
az:=pi+arctan(bp/ap);
  az:=180*az/pi; str(az:4:1,ka);
end;

procedure emelkedes;              { emelkedés kiszámítása }
begin
  em:=arctan(cp/nv1);
  em:=180*em/pi; str(em:4:1,ke);
end;

procedure vetit;                  { képsíkravetítés }
var
  ksi, eta, zet: real;
begin
  ksi:=(-bp*x+ap*y)/nv1;
  eta:=(-cp*(ap*x+bp*y)+sqr(nv1)*z)/nv2;
  zet:=(ap*x+bp*y+cp*z)/d;
  kx:=kx0+round(d*ksi/(d-zet));
  ky:=ky0-round(d*eta/(d-zet))
end;

procedure rajz;
begin
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*d;
  x:=fx(0); y:=fy(0); z:=z0;
```

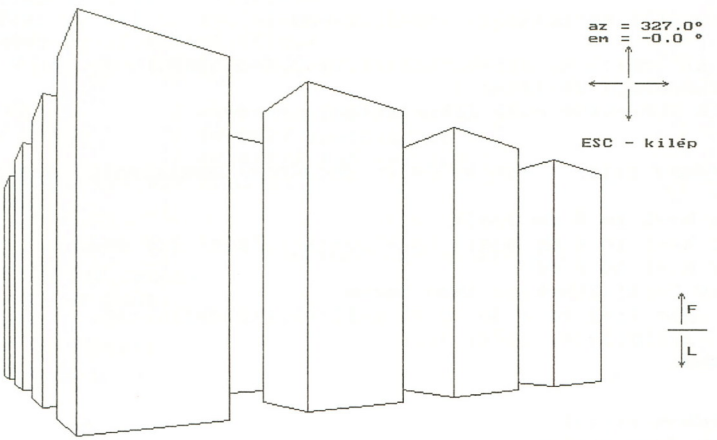
```
  vetit; moveto(kx, ky); fl:=0;
  for k:=1 to n do begin
    if (cp-z0)*cn>0 then fl:=0;   { alapkör láthatósági feltétele }
    x:=fx(k*pin); y:=fy(k*pin);
    if (ap-x)*x+(bp-y)*(y-y0)>0 then fl:=0; { elemi alapkörív }
                                            { láthatósági feltétele}
    vetit;
    if fl=0 then begin lineto(kx,ky); fl:=1; end { láthatóságtól}
                                              {függő megjelenítés }
            else begin moveto(kx,ky); fl:=0; end;
  end;
end;

procedure rajzk;  { hengerpalást képhatárának rajzolása }
begin
  for l:=1 to nl-1 do begin
    z:=(1-l/nl)*z01+l/nl*z02;
    for k:=1 to nk do begin
      x:=fx(k*pink); y:=fy(k*pink);
      if abs((ap-x)*x+(bp-y)*(y-y0))<eps then begin
                                              vetit;
                                              putpixel(kx,ky,15);
                                            end;
    end;
  end;
end;

procedure rajzol;
begin
  z0:=z01; cn:=-1; rajz;          { alapkörök rajzolása }
  z0:=z02; cn:=1; rajz;
end;
```

e) Néhány hasáb alakú épülettömbből álló *"városnegyed"* perspektív képét rajzolja a VAROSKPV.PAS nevű program. Az *"épületek"* megjelenítési sorrendjét úgy határoztuk meg, hogy minden megfigyelési irány esetén a vetítési középponttól távolabb eső alakzatok előbb, a közelebbiek később kerüljenek megrajzolásra, hogy a helyes takarások valósuljanak meg.

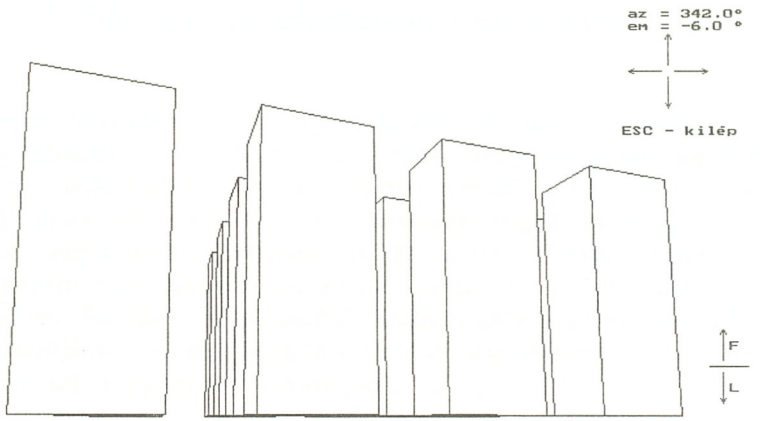
```
const
  a=20; b=30; c=80;
  u=20; v=20;
  d=400; z0=40;
  e: array[1..6,1..4] of byte =   { csúcsok-lapok összerendelés }
     ((1, 2, 3, 4), (1, 5, 8, 2), (2, 8, 7, 3), (3, 7, 6, 4),
     (4, 6, 5, 1), (5, 6, 7, 8));
.....................................

type
  pont = object                    { képernyőpont-objektum }
    x, y: integer;
    procedure uj(ux, uy: integer);
  end;
  csucs = object             { alakzatcsúcs-objektum }
    xx, yy, zz: real;        { térbeli koordináták }
    csp: pont;               { képsíkra eső vetület }
    procedure uj(ux, uy, uz: real);
    procedure vt;
  end;
  nsz = object                     { alakzatlap-objektum }
    cs: array[1..4] of csucs;
    s: array[1..4] of pont;
    lth: boolean;
    procedure uj(k: byte);
    procedure lt;
  end;

var
  ap, bp, cp, nv1, nv2, az, em: real;
  scs, gcs: array[1..8] of csucs;
  lp: array[1..6] of nsz;
  g, h, k, l, ky0: integer;
.....................................

procedure pont.uj(ux, uy: integer);
begin x:=ux; y:=uy; end;
procedure csucs.uj(ux, uy, uz: real);
begin xx:=ux; yy:=uy; zz:=uz; end;
procedure csucs.vt;          { képsíkravetítés }
var
  ksi, eta, zet: real;
begin
  ksi:=(-bp*xx+ap*yy)/nv1;
  eta:=(-cp*(ap*xx+bp*yy)+sqr(nv1)*zz)/nv2;
  zet:=(ap*xx+bp*yy+cp*zz)/d;
  csp.uj(kx0+round(d*ksi/(d-zet)), ky0-round(d*eta/(d-zet)));
end;
```

```
procedure nsz.uj(k: byte);
begin for l:=1 to 4 do cs[l]:=gcs[e[k,l]]; end;
procedure nsz.lt;          { láthatóság megállapítása }
var
  al, bl, cl: real;
begin                      { normálisvektor koordinátái }
  al:=cs[1].yy*(cs[2].zz-cs[3].zz)+cs[2].yy*(cs[3].zz-cs[1].zz)
                                  +cs[3].yy*(cs[1].zz-cs[2].zz);
  bl:=cs[1].zz*(cs[2].xx-cs[3].xx)+cs[2].zz*(cs[3].xx-cs[1].xx)
                                  +cs[3].zz*(cs[1].xx-cs[2].xx);
  cl:=cs[1].xx*(cs[2].yy-cs[3].yy)+cs[2].xx*(cs[3].yy-cs[1].yy)
                                  +cs[3].xx*(cs[1].yy-cs[2].yy);
  if al*(cs[1].xx-ap)+bl*(cs[1].yy-bp)+cl*(cs[1].zz-cp) < 0
{ láthatósági feltétel }
  then lth:=true else lth:=false;
end;

procedure rajz; { egy épületet (hasábot) megjelenítő eljárás }
begin
  for k:=1 to 8 do gcs[k].vt;
  for k:=1 to 6 do begin lp[k].uj(k); lp[k].lt; end;
  for k:=1 to 6 do
    if lp[k].lth=true then begin
      for l:=1 to 4 do lp[k].s[l]:=lp[k].cs[l].csp;
      fillpoly(4, lp[k].s);
    end;
end;

procedure rajzol;
  procedure epulet;
  begin              { hasábok származtatása az alaphasábból }
                    { párhuzamos eltolással }
    for k:=1 to 8 do
      gcs[k].uj(scs[k].xx+(2*g+1)*(u+a),
                scs[k].yy+(2*h+1)*(v+b), scs[k].zz+z0);
    rajz;            { megjelenítés }
  end;
  procedure utcah;          { épületsor származtatása }
  begin
    if az<180 then for h:=-2 to 1 do epulet
              else for h:=1 downto -2 do epulet;
  end;
  procedure utcag;          { épületsor származtatása }
  begin
    if (az<90) or (az>270) then for g:=-2 to 1 do epulet
                           else for g:=1 downto -2 do epulet;
  end;
 begin { megjelenítés a megfigyelési iránytól függő sorrendben }
  nv1:=sqrt(sqr(ap)+sqr(bp)); nv2:=nv1*d;
  if (az<45) or (az>315) then for g:=-2 to 1 do utcah;
  if (az>135) and (az<225) then for g:=1 downto -2 do utcah;
  if (az>=45) and (az<=135) then for h:=-2 to 1 do utcag;
  if (az>=225) and (az<=315) then for h:=1 downto -2 do utcag;
end;
begin
```

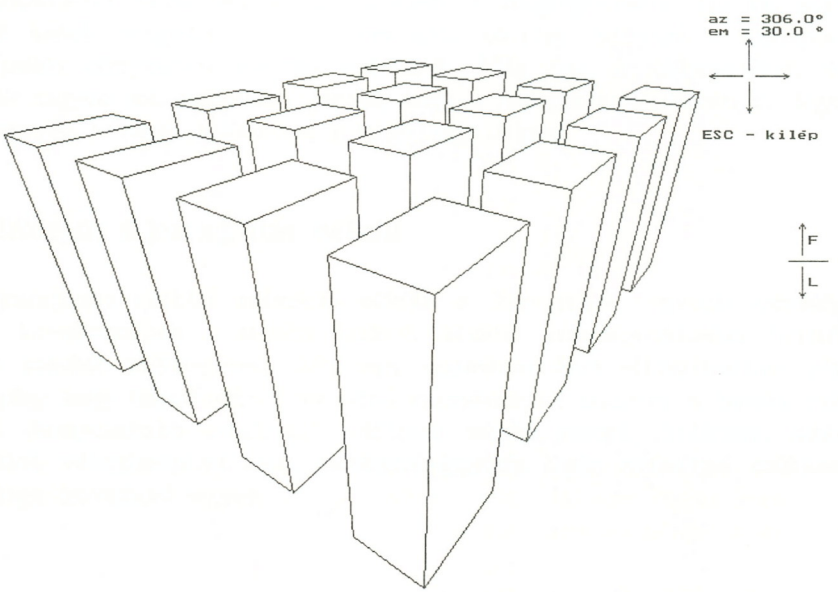
```
.....................................
  scs[1].uj(a, b, c); scs[2].uj(-a, b, c);
 {az alaphasáb csúcsainak koordinátái }
  scs[3].uj(-a, -b, c); scs[4].uj(a, -b, c);
  scs[5].uj(a, b, -c); scs[6].uj(a, -b, -c);
  scs[7].uj(-a, -b, -c); scs[8].uj(-a, b, -c);
  rajzol;
.....................................
end.
```



3.11. ábra



3.12. ábra



3.13. ábra

4. Mozgás érzékeltetése (animáció)

A számítógépes ábrázolás egyik legérdekesebb feladata a mozgó képek készítése. Az ábrázolt alakzat illetve a vetítési irány mozgatási képleteinek ismeretében (a párhuzamos vagy a centrális vetítés keretében), a feladat megoldásának nincs elvi akadálya. A számítógép teljesítőképessége azonban megszabja a megjelenítési sebesség felső határát. Bonyolultabb térbeli alakzatok újrarajzolással történő mozgatása esetén ez a felső határ eléggé alacsony is lehet. Áthidaló megoldásként bizonyos esetekben célravezető lehet a mozgás illúziójának keltése színkódváltoztatás segítségével, ha az ábrázolt tárgy felületére színes rajzolatot alkalmazunk.

Az újrarajzolásos animációkészítés központi kérdése az egymás után következő képek követési sebessége. Ha a megjelenítési sebesség elég nagy, akkor még az egylapos animáció (az újrarajzolás folyamata nincs elrejtve) is elfogadható eredményt ad, ha fekete háttérre rajzolunk, amint az a lemezmelléklet SOKSZOG.PAS nevű programja esetében is megfigyelhető. Ha az egymást követő képek megjelenítése hosszabb időt követel, kötelező a kétlapos, az újrarajzolás elrejtésével történő animáció. Még így is előfordulhat, ha a rajzolás nagyon sok számítást igényel, hogy a mozgás szakadozott, de legalább a képernyőn minden pillanatban a teljes rajz látható.

4.1. Mozgás újrarajzolás nélkül

Az újrarajzolás nélküli animáció előnye a lényegesen nagyobb sebesség és ennek következtében a tágabb határok közötti sebességmódosítás lehetősége (***delay*** utasítás segítségével). Míg egy szintsávos ábra elkészítéséhez például viszonylag nagy teljesítményű személyi számítógépek számára is hosszú másodpercek (bonyolultabb előállítású felületek esetén percek) kellenek, addig a színkódok változtatásával elért hullámzás-áramlás hatás sebességét csökkenteni kell, hogy követhető legyen.

VGA alapszabványban mindössze 16 színárnyalat lehet egyszerre a képernyőn. Dolgozhatunk több színárnyalattal is, folyamatosan újradefiniálva a színeket. A színhullámzás illúzióját keltő programok egy része erre a célra a következő eljárást használja:

```
var
  a: array[1..55,1..3] of byte;
  i, j, k: Integer;
  pl: PaletteType;

begin
  for k:=1 to 10 do begin  { RGB színkeverési arányok definiálása}
    a[k,1]:=63; a[k,2]:=(k-1)*7; a[k,3]:=0;
    a[k+9,1]:=(10-k)*7; a[k+9,2]:=63; a[k+9,3]:=0;
    a[k+18,1]:=0; a[k+18,2]:=63; a[k+18,3]:=(k-1)*7;
    a[k+27,1]:=0; a[k+27,2]:=(10-k)*7; a[k+27,3]:=63;
    a[k+36,1]:=(k-1)*7; a[k+36,2]:=0; a[k+36,3]:=63;
    a[k+45,1]:=63; a[k+45,2]:=0; a[k+45,3]:=(10-k)*7; end;
  i:=45;
  getpalette(pl);                  { a kezdőszínek keverése }
  for k:=15 downto 1 do begin
    j:=k+i;
    if j>54 then j:=j-54;
    setRGBpalette(pl.colors[k],a[j,1],a[j,2],a[j,3]);
  end;
.....................................
      { itt található a szintsávokat ábrázoló eljárás }
.....................................
repeat begin
    getpalette(pl); { a színek folyamatos újrakeverése }
    for k:=15 downto 1 do begin
      j:=k+i; if j>54 then j:=j-54;
      setRGBpalette(pl.colors[k],a[j,1],a[j,2],a[j,3]); end;
    i:=i+1; if i=55 then i:=1;
  end; until keypressed;
.....................................
end.
```

Görbe felületek poliéderes közelítéssel való ábrázolásakor a felületelemek bizonyos rendszer szerinti kiszínezésével és a színkódváltoztatás megfelelő késleltetésével bonyolult mozgás illúziója kelthető. Ezt a módszert jól szemlélteti a GYURUSZM.PAS nevű program, amely tóruszfelületre rajzol különböző színű csigavonalakat, majd ezeket a színek sorrendjének változtatásával mozgatva a felület csavarodásának vagy forgásának illúzióját kelti:

```
.................................
var
  p: palettetype;
  vlt: char;
.................................
```

```
procedure rajzol;
.....
      { a rajzolást mértani vonatkozásban támogató eljárások }
.....
begin
  with p do getpalette(p); { színpaletta definiálása }
    setrgbpalette(p.colors[1],63,0,0);
    setrgbpalette(p.colors[2],63,40,0);
    setrgbpalette(p.colors[3],63,63,0);
    setrgbpalette(p.colors[4],48,63,0);
    setrgbpalette(p.colors[5],0,63,0);
    setrgbpalette(p.colors[6],0,63,40);
    setrgbpalette(p.colors[7],0,63,63);
    setrgbpalette(p.colors[8],0,32,63);
    setrgbpalette(p.colors[9],0,0,63);
    setrgbpalette(p.colors[10],32,0,63);
    setrgbpalette(p.colors[11],63,0,63);
    setrgbpalette(p.colors[12],63,0,32);
    setrgbpalette(p.colors[13],0,0,0);
    setrgbpalette(p.colors[14],63,63,63);
    setrgbpalette(p.colors[15],0,0,0); setallpalette(p);
.....
  { felületelemek különböző színű megjelenítése}
  for l:= 1 to round(n/2) do begin
    case vlt of
    '1','2','4','5':for k:= 1 to m do begin          { a felületen }
                                                { csavarvonalban elhelyezett }
          rzhl;                  { felületelemek lesznek azonos színüek  }
          if al*ap+bl*bp+cl*cp > 0 then begin
            u:=1+round(12*((k+l-1)/12-trunc((k+l-1)/12)));
            setfillstyle(1,u); fillpoly(4,hl);
            if k>=1 then
              v:=1+round(12*((k-1)/12-trunc((k-1)/12))) else
              v:=12-round(12*((1-k-1)/12-trunc((1-k-1)/12)));
            setfillstyle(1,v); fillpoly(4,hm); end;
        end;
    '3','6':begin    { a felületen tört csavarvonalban elhelyezett }
                     {     felületelemek lesznek azonos színüek     }
        for k:= 1 to round(m/2) do begin
          rzhl;
          if al*ap+bl*bp+cl*cp > 0 then begin
            u:=1+round(12*((k+l-1)/12-trunc((k+l-1)/12)));
            setfillstyle(1,u); fillpoly(4,hl);
            if k>=1 then
              v:=1+round(12*((k-1)/12-trunc((k-1)/12))) else
              v:=12-round(12*((1-k-1)/12-trunc((1-k-1)/12)));
            setfillstyle(1,v); fillpoly(4,hm); end;
        end;
        for k:=round(m/2) to m do begin
          rzhl;
          if al*ap+bl*bp+cl*cp > 0 then begin
            u:=1+round(12*((m-k+l-1)/12-trunc((m-k+l-1)/12)));
            setfillstyle(1,u); fillpoly(4,hl);
            if m-k>=1 then
              v:=1+round(12*((m-k-1)/12-trunc((m-k-1)/12))) else
```

```
              v:=12-round(12*((l-m+k-1)/12-trunc((l-m+k-1)/12)));
            setfillstyle(1,v); fillpoly(4,hm); end;
        end;
      end;
    end;
  end;
end;

procedure szinforg; { színek permutációja a palettában }
begin
  for q:=12 downto 1 do p.colors[q+1]:=p.colors[q];
  p.colors[1]:=p.colors[13]; setallpalette(p);
end;

begin
  repeat begin
    clrscr; writeln('változat (1, 2, 3, 4, 5 vagy 6; 0 - kilép) ?');
    repeat vlt:=readkey; until (ord(vlt)>47) and (ord(vlt)<55);
.............................
    grafind;
    setcolor(15); if ord(vlt)>51 then setcolor(0);
    if vlt<>'0' then begin
      rajzol;
      repeat begin
        case vlt of
        '1','4':begin szinforg; delay(20); end;
              { folyamatos csavarodás illúziója }
        '2','3','5','6':begin
              {   oszcilláló csavarodás (2 és 5)     }
              {   illetve forgás (lengés;3 és 6)     }
              for i:=1 to 50 do begin
                szinforg; delay(70-round(i*(51-i)/10)); end;
              for i:=1 to 50 do begin
                for q:=1 to 12 do p.colors[q]:=p.colors[q+1];
                p.colors[13]:=p.colors[1]; setallpalette(p);
                delay(70-round(i*(51-i)/10)); end;
          end;
        end;
      end; until keypressed; readln;
    end; closegraph;
  end; until vlt='0';
end.
```

4.2. Újrarajzolás törlés nélkül

Előfordulhat, hogy a következő kép rajzolásához nem kell az előzőt letörölni a képernyőről, ugyanis az új kép a régit teljesen lefedi, vagy a régi kép megmaradó részleteire egyenesen szükség van a kívánt hatás eléréséhez. Ezt az esetet jól szemlélteti a lemezmellékleten található RZ4.PAS program, amely különböző színű és alakú kitöltött ellipsziseket rajzol egymás fölé.

4.3. Újrarajzolás hátsó lapon és megjelenítés

A szoros értelemben vett animáció úgy készíthető, hogy két lapot használunk. Az egyiket aktív lapnak nevezzük és rajta rajzolunk (ez a hátsó lap), a másikat pedig látható lapnak (ez az első lap). Mikor a rajzolás folyamata a hátsó lapon befejeződött, a lapokat felcseréljük, megjelenítve ezáltal az eredményt, majd az új hátsó lapon elkezdjük a következő kép rajzolását. VGA alapszabványban a kétlapos ábrázolás ***vgamed*** grafikus üzemmódban végezhető.

A lemezmellékleten található legtöbb animációs program a következő rajzlapkezelési eljárást alkalmazza:

```
  procedure forg;
  begin
  ..................................................
    if f=1 then f:=0 else f:=1;
    setactivepage(f);       { hátsó lap beállítása }
      forgat;     { vetítési viszonyokat módosító (mozgató) eljárás }
      rajzol;       { rajzoló eljárás - itt készül az új kép }
      setvisualpage(f);     { hátsó lapon elkészült kép megjelenítése }
  ..................................................
  end;

begin
  ..................................................
  setactivepage(1);
  rajzol;             { rajzoló eljárás - itt készül az első kép }
  setvisualpage(1);
  f:=1;
  forg;                       { mozgatást vezérlő eljárás }
  ..................................................
end.
```

Ha az újrarajzolás nem túl sok időt igényel, akkor ezzel az eljárással folyamatosan változó kép készíthető.

A LENGOCSG.PAS program csillagszerű poliéderek folyamatosan módosított tengely körüli forgatását végzi:

```
.....................................
procedure forgteng; { forgástengely koordinátáinak paraméteres }
                    { előállítása }
const
  lt=pi/6;
begin
  t:=t+lt; if t>2*pi then t:=t-2*pi;
  af:=cos(t)*cos(t/2)-sin(t)*sin(t/2)/2;
  bf:=cos(t)*sin(t/2)+sin(t)*cos(t/2)/2;
  cf:=sin(t)*sqrt(3)/2;
end;

procedure forgat;
begin
  forgteng;
  f:=1;
  repeat begin
    if f=1 then f:=0 else f:=1;
    setactivepage(f);
    for k:=1 to 32 do begin { csúcsok forgatás utáni koordinátái }
      gcs[k].uj(a11*gcs[k].xx+a12*gcs[k].yy+a13*gcs[k].zz,
                a21*gcs[k].xx+a22*gcs[k].yy+a23*gcs[k].zz,
                a31*gcs[k].xx+a32*gcs[k].yy+a33*gcs[k].zz);
    end;
    rajz;                          { rajzoló eljárás }
    setvisualpage(f); { hátsó lapon elkészült rajz megjelenítése }
  end; until keypressed;
end;

procedure egyutthato;       { adott tengely körüli forgatási képlet }
                            { együtthatói }
var nvz: real;
begin
  nvz:=af*af+bf*bf+cf*cf;
  a11:=cos(dt)+af*af*(1-cos(dt))/nvz;
  a12:=(-cf*sqrt(nvz)*sin(dt)+af*bf*(1-cos(dt)))/nvz;
  a13:=(bf*sqrt(nvz)*sin(dt)+af*cf*(1-cos(dt)))/nvz;
  a21:=(cf*sqrt(nvz)*sin(dt)+af*bf*(1-cos(dt)))/nvz;
  a22:=cos(dt)+bf*bf*(1-cos(dt))/nvz;
  a23:=(-af*sqrt(nvz)*sin(dt)+bf*cf*(1-cos(dt)))/nvz;
  a31:=(-bf*sqrt(nvz)*sin(dt)+af*cf*(1-cos(dt)))/nvz;
  a32:=(af*sqrt(nvz)*sin(dt)+bf*cf*(1-cos(dt)))/nvz;
  a33:=cos(dt)+cf*cf*(1-cos(dt))/nvz;
end;
```

```
begin
  t:=0;
.........................
  forgteng;
  egyutthato;
.................................
      { csúcsok koordinátáinak kiszámítása a kezdő helyzetben }
.................................
 setactivepage(1); rajz; setvisualpage(1); f:=1;
{ itt készül az első kép.}
  forgat;
end.
```

5. A lemezmelléklet ismertetése

A lemezmellékleten megtalálhatóak a könyvben leírt ábrázolási algoritmusokat és eljárásokat szemléltető programok forráskódjai Turbo Pascal 7.0 nyelvezetben, illetve néhány program végrehajtható formátumban is, hogy fordító nélkül is lehessen őket futtatni.

A bonyolult ábrákat készítő, nagyszámú műveletet igénylő programok numerikus processzor nélkül eléggé lassan futnak, ami főként az alakzat mozgatásával történő animáció esetén lehet zavaró.

A forráskódokat tematikus alkönyvtárakba csoportosítottuk, a következőképpen:

1. KOZOS

Ebben az alkönyvtárban olyan programok találhatók, amelyekre minden program futtatásához szükség van. Célszerű az alkönyvtárat elérő útvonalat (path) beírni a fejlesztési környezet Options / Directories megfelelő (Object directories és Unit directories) ablakaiba, hogy a Turbo Pascal fordító bárhonnan megtalálja.

GRINDHI.PAS — a grafikus meghajtó vgahi üzemmódban (640×480 pontos képernyőfelbontás) való elindítására szolgáló unit.

GRINDMED.PAS — a grafikus meghajtó vgamed üzemmódban (640×350 pontos képernyőfelbontás) való elindítására szolgáló unit. Újrarajzolással dolgozó animációs programok használják, ugyanis lehetővé teszi a két képernyős (egy aktív és egy látható) rajzolást.

2. SIKGORBE

Különböző előállítású síkgörbéket ábrázoló programok:

CSOKOR.PAS	paraméteres előállítású görbéket rajzol.
EXPL.PAS	explicit görbék egyenlő ívhosszas, szaggatott vonalas ábrázolása.
IMPL.PAS	implicit görbék ábrázolása.
LISSAJ.PAS	Lissajoux-görbéket rajzol.
PARAM.PAS	paraméteres előállítású görbék egyenlő ívhosszas, szaggatott vonalas ábrázolása.

3. SIKIDOM

Síkidomokat ábrázoló programok.

ROZSAFF.PAS	görbe oldalú síkidom ábrázolása.
SOKSZOG.PAS	sokszög ábrázolása és síkban való mozgatása (párhuzamos eltolás, forgatás és tükrözés)

4. FELULET

Euler-Monge előállítású felületek (kétváltozós függvények) ábrázolása. Lehetséges a vetítési irány interaktív módosítása. Választani lehet animációs (újrarajzolás elrejtése) és szokványos megjelenítés között.

F3D3HF.PAS	felületek szemléletes ábrázolása háromszögű háló segítségével.
F3D4HF.PAS	felületek szemléletes ábrázolása négyszögű háló segítségével.

5. FELULETU

Ugyancsak felületek ábrázolását végző programok, ezúttal unit-ok felhasználásával.

F3D3HFU.PAS	háromszögű hálós ábrázolás főprogramja.
F3D3HUAD.PAS	az ábrázoláshoz szükséges adatok.
F3D3HURZ.PAS	megjelenítést végző unit.
F3D4HFU.PAS	négyszögű hálós ábrázolás főprogramja.
F3D4HUAD.PAS	az ábrázoláshoz szükséges adatok.
F3D4HURZ.PAS	megjelenítést végző unit.
F3DU_ALL.PAS	grafikus üzemmód-specifikus állandók beállítása.
F3DU_FV.PAS	az ábrázolandó kétváltozós függvények.
GRIND.PAS	grafikus meghajtó elindítása a felhasználó által választott üzemmódban.

6. SZVLESVL

Felületek szintvonalas-esésvonalas ábrázolása.

SZEV1.PAS	szintvonalak és esésvonalak rajzolása.
SZEV2.PAS	szintvonalak és esésvonalak rajzolása.
SZEV3.PAS	szintvonalak és esésvonalak rajzolása.
SZEV4.PAS	szintvonalak és esésvonalak rajzolása.
SZINTSAV.PAS	szintsávok ábrázolása.
SZINTVON.PAS	szintvonalak rajzolása.

7.POLIEDER

Poliéderek ábrázolása és forgatása.

ATLCSG.PAS	csillag alakú poliédereket rajzol és forgat a takart élek szaggatott vonalas meghúzásával.
CSGFF.PAS	ugyanazt a feladatot végzi mint a fenti program, de a forgatást az újrarajzolás elrejtésével hajtja végre.
DODEKOCK.PAS	dodekaéder származtatása kockából.
FORG12FF.PAS	dodekaédert rajzol és forgat.

FORG20FF.PAS	ikozaédert rajzol és forgat.
FRGCSG.PAS	színes csillagot forgat a felhasználó által megadott tengely körül az újrarajzolás elrejtésével (animáció).
HASAB.PAS	két hasáb összeillesztéséből származó, nem csillagszerű poliédert rajzol és forgat.
IKOZKOCK.PAS	ikozaéder származtatása kockából.
IKOZTETR.PAS	ikozaéder tetraéderbe írás.
ROMBOED.PAS	30 egybevágó rombuszból álló poliédert rajzol és forgat.

8. GORFELTE

Görbe felületű testek határvonalas és poliéderes közelítésű ábrázolása.

GOMBFF.PAS	gömb és gömbből származtatott csillagszerű felületek ábrázolása háromszögű háló (gömbmozaik) segítségével.
GYKONTUR.PAS	tórusz határvonalas ábrázolása.
GYURUFF.PAS	tórusz ábrázolása négyszögű háló használatával.
ZEPP3H.PAS	ellipszoid ábrázolása háromszögű háló használatával.
ZEPPELIN.PAS	ellipszoid ábrázolása négyszögű háló használatával.

9. TERGORBE

Különböző előállítású térbeli görbék ábrázolása.

EROTER.PAS	két, ellentétes előjelű, pontszerű töltés elektromos erőtere (differenciálegyenletes felírású görbék ábrázolása).
GBEUL2V.PAS	az Euler-féle szögek változtatásával (kivéve a nutációs szöget, amely állandó marad) generált görbék.
GBEUL3V.PAS	mindhárom Euler-féle szög változtatásával generált görbék.
LABDA.PAS	különböző síkokban fekvő körök ábrázolása.
PARAMT.PAS	paraméteres előállítású térgörbét rajzol.
PMSZAGGV.PAS	paraméteres előállítású térgörbe egyenlő ívhosszas szaggatott vonalas rajzolása.
PRMTFT.PAS	görbék paraméteres előállítású felületeken.
SPIRAL.PAS	síkgörbe (archimédeszi spirális) a térben.

10. ATHATAS

Görbe felületek áthatási görbéinek ábrázolása.

ATH.PAS	ellipszoid és hiperbolikus paraboloid áthatása.
GB4HPATH.PAS	gömb és négy hiperbolikus paraboloid áthatása.

11. GFTSZV

Szintvonalak görbe felületeken (nem geometriai tulajdonságok szemléltetése).

GBSZINTV.PAS	szintvonalak gömbön.
GBSZTSAV.PAS	szintsávok gömbön.
GBSZVATL.PAS	szintvonalak átlátszó gömbön.

12. TORZFEL

Torzfelületek ábrázolása és más felületekkel való áthatásuk.

GBTZFATH.PAS	gömb és torzfelület áthatási görbéje.
GOMBCSAV.PAS	gömbbe írt csavarfelület.
GYCSAV.PAS	tóruszba írt csavarfelület.

13. KOZEPP

Középpontos (centrális) vetítéssel (perspektíva) készült ábrák.

GYURUKPV.PAS	tórusz perspektív képe.
HENGKPV.PAS	henger perspektív képe. A vetítési középpont mozgatása.
KOCKAKPV.PAS	kocka perspektív képe. Tárgy forgatása.
KORKPV.PAS	kör centrális vetületei (kúpszeletek).
VAROSKPV.PAS	több alakzatból álló rendszer perspektívája, a megjelenítési sorrend megfigyelési iránytól függő beállításával.

14. ANIMSZ2D

Mozgás érzékeltetése újrarajzolás nélkül, színkódok változtatásával, síkbeli alakzatok és domborzatjellegű felületek szintsávos ábrázolása esetén.

ROZSA.PAS	színhullámzás görbe oldalú síkidomok között.
RZ1.PAS	Lissajoux-görbét megjelenítő húrok változó színű folyamatos újrarajzolása.
RZ2.PAS	paraméteres előállítású, zárt síkgörbe, egyenlő ívhosszas, folyamatos rajzolása, változó színű húrokkal.
SZHME1.PAS	színhullámzás ellipszis alakú keretben.
SZHME2.PAS	színhullámzás ellipszis alakú keretben.
SZHME3.PAS	színhullámzás ellipszis alakú keretben.
SZHMN1.PAS	színhullámzás téglalap alakú keretben.
SZHMN2.PAS	színhullámzás téglalap alakú keretben.
SZHMN3.PAS	színhullámzás téglalap alakú keretben.
SZHMN4.PAS	színhullámzás téglalap alakú keretben.

15. ANIMSZ3D

Mozgás érzékeltetése újrarajzolás nélkül, színkódok változtatásával, térbeli alakzatok szemléletes képe esetén.

GBCSVSZM.PAS	színes sávok vonulása gömbbe írt csavarfelületeen.
GBSZTSVM.PAS	színhullámzás gömbre rajzolt szintsávok között.
GYCSVSZM.PAS	színes sávok vonulása tóruszba írt csavarfelületen.
GYURUSZM.PAS	színes sávok egyenletes és periodikusan változó vonulása tóruszfelületen.

16. ANIMMZ2D

Síkidomok mozgatása újrarajzolással.

RZ3.PAS	térbeli mozgás illúziója Lissajoux-görbe egyik paraméterének (fáziskülönbség) folyamatos módosításával. Újrarajzolás elrejtve.
RZ4.PAS	újrarajzolás törlés nélkül.

17. ANIMMZ3D

Térbeli alakzatok mozgatása újrarajzolással.

CSGSZN.PAS	csillagszerű poliéderek billentyűzetről vezérelt elforgatása az újrarajzolás elrejtésével.
GOMBSZMA.PAS	gömb és gömbből származtatott görbe felületű testek billentyűzetről vezérelt forgatása, az újrarajzolás elrejtésével kis felbontásban (viszonylag nagy sebesség).
GOMBSZMB.PAS	gömb és gömbből származtatott görbe felületű testek billentyűzetről vezérelt forgatása, az újrarajzolás elrejtésével.
GOMBSZMC.PAS	görbe felületű testek nagyfelbontású ábrázolása és forgatása. Nagy memóriaigény miatt nem futtatható a fejlesztési környezetből, hanem végrehajtható (EXE) file-t kell készíteni és a fejlesztési környezeten kívülről futtatni.
LENGOCSG.PAS	csillag folyamatosan módosított tengely körüli forgatása.

IRODALOMJEGYZÉK

1. Hajós György: Bevezetés a geometriába (nyolcadik kiadás)
   Tankönyvkiadó, Budapest, 1987.

2. Strommer Gyula: Geometria
   Tankönyvkiadó, Budapest, 1988.

3. Lőrincz Pál, Petrich Géza: Ábrázoló geometria (negyedik kiadás)
   Tankönyvkiadó, Budapest, 1989.

4. Vermes Imre: Geometria útmutató és példatár
   Tankönyvkiadó, Budapest, 1991.

5. Budó Ágoston: Mechanika (második, bővített kiadás)
   Tankönyvkiadó, Budapest, 1953.

6. Benkő Tiborné, Benkő László, Kiss Zoltán Tóth Bertalan:
   Objektum-orientált programozás Turbo Pascal 6.0-ban. Turbo Vision
   ComputerBooks, Budapest, 1991.

7. Benkő Tiborné, Kiss Zoltán, Dr. Tamás Péter, Tóth Bertalan:
   Programozás Borland Pascal 7.0 rendszerben
   ComputerBooks, Budapest, 1994.

TÁRGYMUTATÓ

ÁFÉSZ Nyomda Vác,

Megrendelőlap

Megrendelem az alábbi kiadványokat postai utánvéttel. Tudomásul veszem, hogy a postaköltség felszámításra kerül, és a szállítási idő 2–3 hét.
Utánnyomásoknál árváltozás lehetséges.

... *pl Füzi János:* **3 dimenziós grafika és animáció IBM PC–n** – lemezmelléklettel	**1.283.–**
... *pl Nagy G.:* **Kézikönyv az adattömörítéshez – ARJ. PKZIP & Co.** – lemezmelléklettel	**1.298.–**
... *pl Dr.Dedinszky F.:* **CLIPPER 5 – 5.0, 5.01 és segédprogramjai**	**799.–**
... *pl Nagy Z.–Spányik B.–Weisz T.:* **CorelDRAW! 5**	**795.–**
... *pl Gerő J.:* **EXCEL 4 for Windows – magyar nyelvű változathoz** – (tanfolyami tananyag)	**447.–**
... *pl Kovalcsik G.:* **EXCEL for Windows 5.0 kezdőknek * haladóknak – magyar és angol változathoz**	**1.147.–**
... *pl Dr. Kovácsné C. J. Ozsváth M.:* **EXCEL 5 függvényei** – magyar változathoz	**990.–**
... *pl Krizsák L.:* **EXCEL 5 for Windows Kis☺kos** – magyar és angol változathoz	**398.–**
... *pl Bognár J.:* **dBASE III PLUS** (javított kiadás)	**480.–**
... *pl Balogh J.–Dr.Dedinszky F.:* **FoxPRO 2.0**	**895.–**
... *pl Fehérvári A:* **LOTUS for WINDOWS és a Freelance Graphics**	**447.–**
... *pl Kovácsné C. J.–Pergelné B. I.–Benkő L.:* **Mindenkinek! a PC–ről**	**599.–**
... *pl Dr. Rubicsek Gy.:* **PC 1x1** (amit a számítógépről és eszközeiről tudni illik)	**298.–**
... *pl Tamás–Kiss–Tóth:* **MS–DOS 6 – 6.2; 6.22 kiegészítéssel**	**985.–**
... *pl Kóczy A. J.:* **MS–DOS 5.0, 6 kis☺kos**	**295.–**
... *pl Dr.Janurik T.:* **MS–DOS hibaüzenetek a 3–, 4–, 5–, 6. verziókhoz**	**150.–**
... *pl Benkő–L.–Benkő T.né:* **MS WORKS 3.0 a mindennapi életben** – magyar verzióhoz	**793.–**
... *pl Rudnai P.né:* **Novell NetWare 3.11, 3.12 felhasználóknak és rendszergazdáknak**	**945.–**
... *pl Pintér M.:* **AutoCAD tankönyv** – DOS & WINDOWS; AutoCAD LT; AutoCAD R12 angol & magyar	**899.–**
... *pl Pintér M.:* **Rajzkészítés AutoCAD R12–vel**	**590.–**
... *pl Pintér M.:* **Szilárdtestek modellezése AutoCAD Release 12–vel**	**715.–**

A 175–35–91 telefonszámon könyvszolgálatunk tájékoztatja Önt a lakó– vagy munkahelyéhez legközelebb eső szaküzletről, ahol kiadványainkat megvásárolhatja.

Ha mégis a postai utat választja, kérjük a megrendelését a levélcímünkre

COMPUTERBOOKS Kft – 1253 Bp., Pf.: 71

szíveskedjen visszaküldeni.

... *pl Székely V.*: **Képkorrekció, hanganalízis, térszámítás PC-n** – lemezmelléklettel **1.258.–**

... *pl Abonyi Zs.*: **PC hardver kézikönyv** (bővített, átdolgozás kiadás) **875.–**

... *pl dr. Kovácsné C. J. – Ozsváth M.*: **QuarkXPress for Windows** **979.–**

... *pl Pergel J.né*: **QuattroPRO 5** **770.–**

... *pl Lukács O.*: **Quick Basic programozása – feladatgyűjtemény** – lemezmelléklettel **598.–**

... *pl dr. Kovácsné C. J. – Takács T.*: **Ismerkedés az SSADM-mel** **966.–**

... *pl Borgulya I.*: **Szakértői rendszerek, technikák és alkalmazások** **1.375.–**

... *pl Benkő – Tóth – Varga*: **Programozzunk TURBO PASCAL nyelven** – lemezmelléklettel (javított, átdolgozott kiadás) **796.–**

... *pl Benkő T.né – Kiss Z. – Tamás P. – Tóth B.*: **Programozás Borland Pascal 7.0 rendszerben /DPMI, WINDOWS** – példaprogramok lemezmellékleten **1.586.–**

... *pl Benkő L. – Benkő T.né – Tóth*: **Programozzunk C nyelven kezdőknek * középhaladóknak** – lemezmelléklettel **1.199.–**

... *pl Benkő – Poppe – Benkő*: **Bevezetés a BORLAND C++ programozásába** **645.–**

... *pl László József*: **VGA kártya programozása** Pascal és Assembly nyelven – lemezmelléklettel **1.375.–**

... *pl Nagy Gábor*: **Vírusvédelem PC-n** – lemezmelléklettel **1.157.–**

... *pl dr. Tamás – Horváth – Kiss – Tóth*: **WINDOWS 3.1 felhasználóknak** **698.–**

... *pl dr. Kovácsné Cohner Judit*: **Magyar WINDOWS 3.1** **990.–**

... *pl Benkő T.né – Kuzmina J. – Kiss Z. – dr. Tamás P. – Tóth B.*: **Könnyű a WINDOWS-t programozni!?** – lemezmelléklettel **1.683.–**

... *pl Dr. Kovácsné C. J. – Ozsváth M.*: **Windows for Workgroups 3.11** – hálózattal vagy anélkül **1.115.–**

... *pl Rudnai P.né – Rudnai T.*: **Windows for Workgroups 3.11 kis☺kos** – angol és magyar változathoz **399.–**

... *pl Gerő J. – Reich G.*: **WORD for WINDOWS 2.0** – magyar nyelvű változathoz **795.–**

... *pl Gerő J.*: **WORD for WINDOWS 2.0 Kis☺kos** **199.–**

... *pl Nagy G.*: **WORD for WINDOWS 2.0 makrói és a WordBASIC használata** **652.–**

... *pl Gerő Judit – Reich Gábor*: **WORD for WINDOWS 6.0** – magyar & angol nyelvű verzióhoz **980.–**

... *pl Gerő Judit – Krizsák László*: **WORD for WINDOWS 6.0 kis☺kos** **498.–**

... *pl Bartók Nagy J. – Laufer J.*: **UNIX felhasználói ismeretek** **1.200.–**

Megrendelő neve: _____

szállítási cím: _____

város: _____

utca: _____

irányítószám: _____

NOVOTRADE 2C KFT

1136 Budapest, Balzac u. 35.
Tel/Fax: 1402-954, 1315-933
Nyitva: hétfőtől péntekig 9-től 18 óráig

SZÁMÍTÁSTECHNIKAI SZAKKÖNYVEK
SZÉLES VÁLASZTÉKA:

Kezdőknek
Haladóknak
Profiknak

Ára: 1283–Ft áfával

9 789636 180577