μlógia 44

ELTE TTK Informatikai Tanszékcsoport

1. kiadás

Készült az *NJSzT* gondozásában
200 példányban
Felelõs kiadó: Dr. Kátai Imre
Sorozatszerkesztõ: Szlávi Péter - Zsakó László

Zsakó László (szerk.)

Fejezetek a számítógépi grafikából

Szerzők:
Bán-Széll Előd, Barna Edina, Katona Krisztina,
Szukics Sarolta, Tari Judit, Volf Annamária,
Varga Zsuzsanna, Zalán Eszter

Előszó

Az 1999/2000-es tanévtől kezdődően minden informatika tanárszakos évfolyam legjobb teljesítményt elérő hallgatói külön foglalkozásokon vesznek részt.

Ezeken a foglalkozásokon nagyon sok olyan anyag születik, amely szélesebb érdeklődésre tarthat számot.

Ebben a jegyzetben a 2001/2002-es tanévben született, számítógépi grafikával kapcsolatos tananyagok jelennek meg, melynek szerzői:

Bán-Széll Előd

Barna Edina

Katona Krisztina

Szukics Sarolta

Tari Judit

Volf Annamária

Varga Zsuzsanna

Zalán Eszter

Budapest, 2003

A sorozat szerkesztői

Tartalom

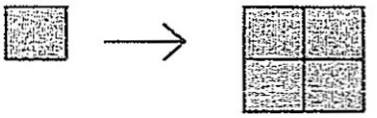
I. Elemi rasztergrafikai algoritmusok

Raszteres képeket képpontonként tárolunk, az egyes képpontok fényességének megadásával.

1. Raszterképek nagyítása, kicsinyítése

Az elemi nagyítási algoritmusok a képet mindig kétszeresére nagyítják, a kicsinyítési algoritmusok pedig a felére kicsinyítik. Az alábbi algoritmusokban az eredeti képet az A NxM-es mátrix tartalmazza, a transzformált képet pedig a B (2*N)x(2*M)-es, illetve a C (N/2)x(M/2)-es mátrix.

A. Nagyítás pontsokszorozással



A módszer alapelve: az eredeti kép minden egyes képpontját négyszerezzük meg.

```
Nagyítás:
  Ciklus I=1-től N-ig
    Ciklus J=1-től M-ig
      B(2*I-1,2*J-1):=A(I,J); B(2*I,2*J-1):=A(I,J)
      B(2*I-1,2*J):=A(I,J); B(2*I,2*J):=A(I,J)
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

B. Nagyítás pontátlagolással

A módszer alapelve: az eredeti kép minden egyes képpontját egy pontnégyessel helyettesítsük: az egyik az eredeti, a másik három pedig az eredeti három szomszéddal vett átlag lesz.

```
Nagyítás:
  Ciklus I=1-től N-ig
    Ciklus J=1-től M-ig
      B(2*I-1,2*J-1):=A(I,J)
      B(2*I,2*J-1):=(A(I,J)+A(I+1,J)) div 2
      B(2*I-1,2*J):=(A(I,J)+A(I,J+1)) div 2
      B(2*I,2*J):=(A(I,J)+A(I+1,J+1)) div 2
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

C. Kicsinyítés pontelhagyással

A módszer alapelve: az eredeti kép minden egyes képpont négyeséből csak a bal felső sarkot hagyjuk meg.

```
Kicsinyítés:
  Ciklus I=1-től N/2-ig
    Ciklus J=1-től M/2-ig
      B(I,J):=A(2*I-1,2*J-1)
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

D. Kicsinyítés pontátlagolással

A módszer alapelve: az eredeti kép minden egyes képpont négyezése helyére az átlagukat tesszük.

```
Kicsinyítés:
  Ciklus I=1-től N/2-ig
    Ciklus J=1-től M/2-ig
      B(I,J):=(A(2*I-1,2*J-1)+A(2*I-1,2*J)+
               A(2*I,2*J-1)+A(2*I,2*J)) div 4
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

2. Raszterképek megjelenítése

A. Ekvidenzitás képek

A legegyszerűbb megjelenítési módszer az elsőrendű ekvidenzitás kép, amiben rögzítünk egy intervallumot, amihez fehér színt rendelünk, az azon kívül esőkhöz pedig feketét (háttérszínt).

$$SZ(I,J)=\begin{cases} fehér, & ha\ AH \le A(I,J) < FH \\ fekete, & ha\ \ egyébként \end{cases}$$

```
Rajzolás(AH,FH):
  Ciklus I=1-től N-ig
    Ciklus J=1-től M-ig
      Ha AH≤A(I,J) és A(I,J)<FH akkor SZ(I,J):=fehér    {1}
                                különben SZ(I,J):=fekete  {0}
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

Az intervallum felső határát alapesetben nem rögzítik, így csak az alsó határral kell hasonlítani.

Ennek módosítása a kevert rendű ekvidenzitás kép, amikor nem két-, hanem többárnyalatos képet állítunk elő, ekkor az egyenlő nagyságú beosztás esetén az algoritmus így módosul (MAXA a legmagasabb, sehol sem szereplő fényesség):

```
Rajzolás(MAXA):
  Ciklus I=1-től N-ig
    Ciklus J=1-től M-ig
      SZ(I,J):=[A(I,J)*K/MAXA]
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

B. Adaptív vágás

Bármilyen lineáris vágást is alkalmazunk, mindenképpen fontos információt veszíthetünk: a kép egyes, eltérő átlagfényességű részei rossz minőségűek lesznek. Ezen azzal lehet segíteni, ha a kép különböző részeire különböző küszöbértékeket alkalmazunk; világos területen nagyobbat, sötéten pedig kisebbet.

Az egyik legegyszerűbb módszer: határozzuk meg minden pont adott környezetének átlagos fényességét, majd rendeljük a ponthoz fehér színt, ha az az átlagosnál fényesebb volt, s feketét, ha az átlagosnál sötétebb.

```
Rajzolás:
  Ciklus I=1-től N-ig
    Ciklus J=1-től M-ig
      Ha SZ(I,J)>átlag(I,J) akkor SZ(I,J):=fehér
                            különben SZ(I,J):=fekete
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

3. Adott tulajdonságú elemek kiemelése a képen

Az A mátrixban tárolunk egy NxM-es képet, melyben minden képpontot egy egy-byte-os értékkel adunk meg. Az egyes képpontokat a szomszédaik értékétől függően transzformálva, különböző jellegű ábrákat kapunk:

```
Képfeldolgozás:
  Ciklus I=2-től N-1-ig
    Ciklus J=2-től M-1-ig
      B(I,J):=128+(A(I,J)-A(I,J-1)) div 2 [1]
      C(I,J):=|A(I,J)-A(I,J-1)|+|A(I,J)-A(I-1,J)| div 2 [2]
      D(I,J):=128+(2*A(I,J)-A(I,J-1)-A(I-1,J)) div 4
      E(I,J):=128+(4*A(I,J)-A(I,J+1)-A(I+1,J)-A(I,J-1)
                   -A(I-1,J)) div 8
    Ciklus vége
  Ciklus vége
Eljárás vége
```

B mátrix: balról jobbra haladva az emelkedő felületek lesznek fényesek, a sík felületek átlagosak, a lejtők pedig sötétek (mintha a felületet nyugati irányból világítanánk meg)

---

[1] Elméletileg jobb: `B(I,J):=128+(A(I,J+1)-A(I,J-1)) div 2`

[2] Elméletileg jobb: `E(I,J):=|A(I,J+1)-A(I,J-1)|+|A(I+1,J)-A(I-1,J)| div 2`

C mátrix: azok a pontok lesznek fényesek, amelyeknél az eredeti képen a fényesség jelentősen változott, s azok lesznek sötétek, ahol az eredeti képen nem volt változás (így az "éleket" emeljük ki).

Matematikai alapja: Ha a kép fényességét kétváltozós függvénynek tekintjük, akkor ott lesz él, ahol a gradiens vektor abszolút értéke nagy:

$|grad \quad f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$, ahol a differenciálhányadosokat differenciahányadosokkal közelítjük, pl.: $\frac{\partial f}{\partial x} \cong \frac{A(i,j) - A(i,j-1)}{\Delta x}$, illetve matematikai szempontból jobb: $\frac{\partial f}{\partial x} \cong \frac{A(i,j+1) - A(i,j-1)}{\Delta x}$. A módszer egyes verzióiban a négyzetgyököt elhagyhatjuk, a négyzetre emelés helyett abszolút értéket vehetünk (bár ez túl élesre veszi a 45 fokos dőlésű éleket), ...

Ennek másik változata a Roberts féle keresztoperátor, ami a négy sarokszomszédot vizsgálja: $\max(|A(i+1,j-1) - A(i-1,j+1)|, |A(i+1,j+1) - A(i-1,j-1)|)$

D mátrix: balról jobbra és felülről lefelé haladva az inkább emelkedő felületek lesznek fényesek, a sík felületek átlagosak, az inkább lejtő felületek pedig sötétek (mintha a felületet északnyugati irányból világítanánk meg)

E mátrix: azok a pontok lesznek az átlagosnál fényesebbek, amelyek a 4 szomszédjuk átlagánál fényesebbek (domború felület), azok lesznek sötétebbek, amelyek a szomszédaik átlagánál sötétebbek (homorú felület), a többiek pedig átlagos fényességűek lesznek. Az élek mentén az alacsony fényességűek még sötétebbek, a magas fényességűek még világosabbak lesznek.

Matematikai alapja a Laplace operátor, ami az adott függvény gradiensének divergenciáját számolja:

$div \quad grad \quad f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$, ahol a differenciálhányadost differenciahányadossal helyettesítve, pl. a következőt kapjuk: $\frac{\partial^2 f}{\partial x^2} = \frac{A(i+1,j) + A(i-1,j)) - 2 * A(i,j)}{\Delta x^2}$.

4. A képfeldolgozás alapalgoritmusai

A. Hisztogram számítás

Az árnyalatos képek fontos jellemzője, hogy melyik árnyalat milyen arányban fordul elő a képen. Ennek szokásos ábrázolása egy hisztogram, ahol az I-edik oszlop magassága az I fényességű pontok számával arányos.

```
Hisztogram(H):
  Ciklus I=1-től N-ig
    Ciklus J=1-től M-ig
      H(A(I,J)):=H(A(I,J))+1
    Ciklus vége
  Ciklus vége
Eljárás vége
```

Az így kapott hisztogramról több jellemző is megállapítható.

- A szürkeségi felbontás szám megegyezik a hisztogramon található nem nulla oszlopok számával. (Ez nem a maximális fényességérték, ha sok közbülső fényességet nem használunk.)
- Eldönthető, hogy kiegyensúlyozott tónusú-e a kép (minden oszlop kb. egyforma magasságú-e). Sötét tónusú (alulexponált) képeknél a hisztogram alsó részén vannak jelentősen magasabb oszlopok, világos tónusúnál (túlexponált) pedig a felső részén.

B. Lineáris szűrés

Sok képen véletlen zajok jelennek meg, amelyek a kép minőségét határozottan rontják, azaz minden egyes valódi értéket megváltoztathatott egy véletlen érték. A szűrés feladata ezen véletlen hatások minél jobb hatásfokú megszüntetése.

Ennek legegyszerűbb változatában minden egyes képpont értékét helyettesítjük önmaga és közvetlen 8 szomszédja átlagával:

$$B(i,j)=\frac{1}{9}\sum_{k=i-1}^{i+1}\sum_{l=j-1}^{j+1}A(k,l)$$

Valószínűségszámításból ismert, hogy ha N pont átlagát számoljuk, akkor a zajcsökkentés mértéke $\sqrt{N}$, azaz esetünkben háromszoros javulást érhetünk el, a zaj hatását szétkenjük.

Egyetlen probléma, hogy ez a módszer egyben a kép élességét rontja, az éles határvonalakat elkeni.

C. Rank szűrés

Ennél a módszernél átlagszámítás helyett a szomszédos pontokkal (N db) más műveletet végzünk. Első lépésként vegyük a környező pontok fényességértékét és rendezzük nagyság szerint sorba. Ezután válasszuk ki a nagyság szerint K-adik elemet, s ezzel helyettesítsük az eredeti pontot. Ha K=1, akkor éppen a legsötétebb pontot választjuk, ha K=N, akkor pedig a legfényesebbet.

Különösen érdekes egy speciális rank szűrő, a medián szűrő, amikor K=N/2, azaz éppen a nagyság szerint középső értéket választjuk. Ez a módszer a kiugró zajcsúcsokat tökéletesen eltünteti.

A módszer egyetlen hibája, hogy a szomszédsági környezet növelésére nagyon érzékeny, hiszen a környezetben szereplő pontok számának négyzetével arányos a rendezés ideje. Ezen segíthet egy, a logaritmikus keresés módszerére emlékeztető közelítő algoritmus (feltéve, hogy $N=2^M$):

```
Középső(T,N,K):
  K:=0; B:=N
  Ciklus
    B:=B/2; K:=K+B  {a következő bit beállítása}
    Számlálás(KI,KE){KI legyen a K-nál kisebb, KE pedig a
                     kisebb vagy egyenlő elemek száma}
    Ha KI≥N/2 akkor K:=K-B {a bit letörlése, ha kell}
  amíg KI<N/2 és KE≥N/2
  Ciklus vége
Eljárás vége
```

A műveletigény itt már csak a bitek számával (külső ciklus) és a szomszédos pontok számával (számlálás) arányos.

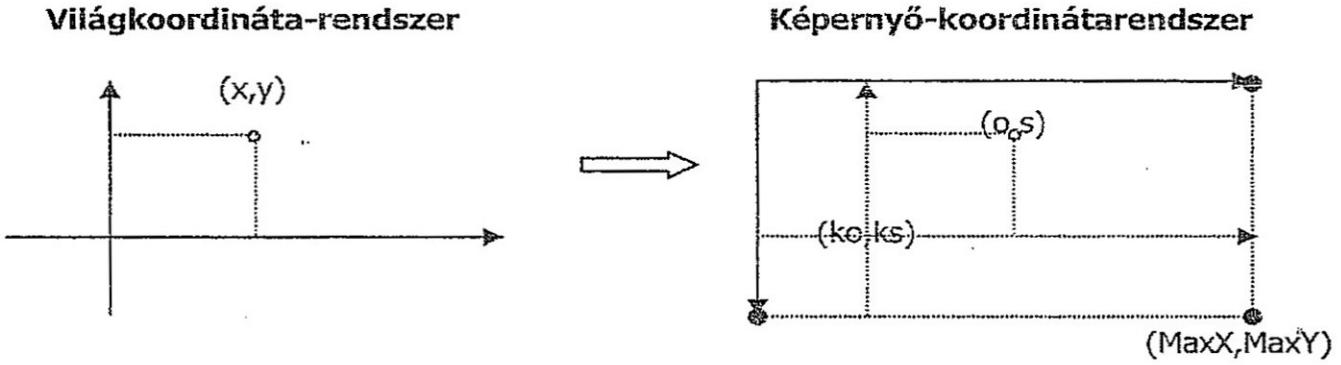
II. Elemi grafikai algoritmusok

Megjegyzés: minden algoritmusnál normál koordináta-rendszert tételezünk fel, ezért a 'Rajzol' eljárásnak kell még gondoskodnia a képre transzformálásról.

1. Pont rajzoló eljárás

A képernyőn a „normál" koordinátarendszer:

- Origó a bal-felső sarokban.
- A pixel az egység.
- Csak egész koordinátájú pontokkal jellemzett görbékkel, ívekkel foglalkozunk.



Tegyük fel, hogy a képernyő sorainak számát az SS, oszlopainak számát az OS változó tartalmazza, a képernyő origója a bal felső sarokban van, a koordináták jobbra, illetve lefelé növekszenek. Legyen a valódi koordinátatengely origója a képernyő KS. sor KO. oszlopában és irányítása legyen a szokásos! A rajzoló eljárás a paraméterként kapott valós koordináta értékeket kerekítse egészre, s a pontot csak akkor ábrázolja, ha a képernyőn van. Ebben a megoldásban a valós koordinátarendszer és a képernyő-koordináta-rendszer léptéke azonos. (Ha nem azonos lenne, akkor X-et és Y-t meg kell szorozni a megfelelő léptékkel.)

```
Eljárás PontRajzolás(Konstans x,y:Valós):
  s:=Kerekít(ks-y); o:=Kerekít(ko+x)[i]
  Ha s∈[0,MaxY] és o∈[0,MaxX] akkor Pont(o,s)
Eljárás vége.
```

2. Szakasz rajzolása

A: egyszerű algoritmus

A szakasz végpontjai legyenek (X1,Y1), illetve (X2,Y2)! Tegyük fel, hogy X2>X1! A két ponton húzható egyenes egyenlete: $y=(y_2-y_1)/(x_2-x_1)*(x-x_1)+y_1$

A megoldás lényege:

---

[i] Érdemes eljátszadozni a `Kerekít` függvény helyett `Egészrész`-szel.

1. vegyük sorra x lehetséges (egész) értékeit $[x_1, x_2]$ között, és
2. rajzoljuk ki az (x,y(x)) pontot!

```
Eljárás SzakaszRajzolás(Konstans x1,y1,x2,y2:Egész):
  Változó
    x,y,it:Valós
  it:=(y2-y1)/(x2-x1)   [iránytangens]
  Ciklus x=x1-től x2-ig [ii]
    y:=(x-x1)*it+y1; PontRajzolás(x,y)
  Ciklus vége
Eljárás vége.
```

Problémák:

1. $x_1=x_2$–re nem működik.
2. it≤1 (legfeljebb 45° lejtésszög) esetén „folytonos" pixelek sorozata a szakasz, it>1 (több, mint 45° lejtésszög) esetén „szakadozott" pixelek sorozata.
3. a lejtéssel változó fényerő:
   Pl.:it=0, akkor a „fénysűrűség"=(N-fénypont)/(N pixelnyi hossz)=1,
   it=1, akkor a „fénysűrűség"=(N-fénypont)/($\sqrt{2}$ *(N pixelnyi hossz))=$\sqrt{2}/2\approx$ ≈0,71.

Az alábbiakban először az első két problémára keresünk megoldást.

B: első javítás

Válasszuk szét az it≤1, it>1 és $x_1=x_2$ eseteket! A 2. esetben cseréljük föl x és y szerepét, a 3. esetben speciálisan erre az esetre specifikálunk egy eljárást.

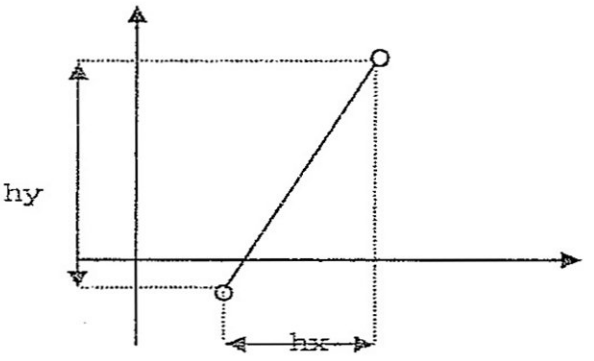
```
Eljárás SzakaszRajzolás(Konstans x1,y1,x2,y2:Egész):
  Változó
    x,y,it:Valós
  Ha x1=x2 akkor
    FüggőlegesRajzolás(x1,y1,y2)
   különben
    it:=(y2-y1)/(x2-x1)   [iránytangens]
    Ha it≤1 akkor
        Ciklus x=x1-től x2-ig
          y:=(x-x1)*it+y1; PontRajzolás(x,y)
        Ciklus vége
      különben
        Ciklus y=y1-től y2-ig
          x:=(y-y1)/it+x1; PontRajzolás(x,y)
        Ciklus vége
    Elágazás vége
  Elágazás vége
Eljárás vége.
```

[ii] Ha $x_1 \le x_2$.nem lenne feltehető, akkor a ciklus Sgn (x_2-x_1) -esével szervezzük

C: folytonos vonallal

Válasszuk meg úgy az x-irányú lépésközt, hogy az megfelelő legyen minden esetben (a függőleges esetet kivéve). Ez elérhető, ha az x-irányú eltérés (hx) és az y-irányú eltérés (hy) maximumával normáljuk a lépésközöket.

```
h:=Max(|hx|,|hy|)
lx:=hx/h     - x-lépésköz
ly:=hy/h     - y-lépésköz
   ⇒
lx,ly≤1
   ⇒
Max(lx,ly)=1
```



Hány lépést (k) kell tenni?

$x_2=x_1+k*lx \Leftrightarrow k=(x_2-x_1)/lx \Leftrightarrow k=(x_2-x_1)/(hx/h) \Leftrightarrow$
$\Leftrightarrow k=(x_2-x_1)*h/hx \Leftrightarrow k=(x_2-x_1)*h/(x_2-x_1) \Leftrightarrow k=h$

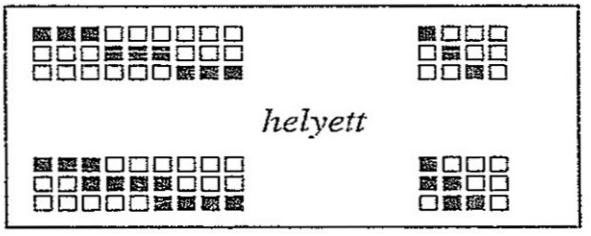
Hasonlóan (és nem meglepő módon) ugyanezt kapjuk az y-irányban is.

```
Eljárás SzakaszRajzolás(Konstans x1,y1,x2,y2:Egész):
  Változó
    lx,ly,x,y:Valós
  hx:=x2-x1; hy:=y2-y1
  Ha Abs(hx)>Abs(hy) akkor h:=Abs(hx) különben h:=Abs(hy)
  Ha h=0 akkor
    PontRajzolás(x1,y1)
   különben
    lx:=hx/h   [x-irányú lépésköz]
    ly:=hy/h   [y-irányú lépésköz]
    x:=x1; y:=y1; PontRajzolás(x1,y1)   [kezdőpont]
    Ciklus k=1-től h-ig
      x:+lx; y:+ly; PontRajzolás(x,y)
    Ciklus vége
  Elágazás vége
Eljárás vége.
```

D: javítás a fényerő-problémán

Ötlet:

kiegészíteni *további* (azonos színű, fényességű) pontokkal a „töréseknél". (A színezés csak az új pontok kiemelését szolgálja.)

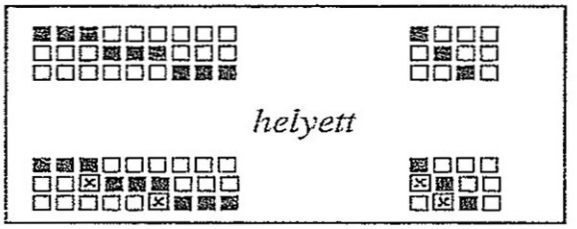


Ami után a probléma „megfordul":

Ui. ha it=1, akkor h pont helyett h+(h-1) pont lesz,
így a fénysűrűség= ((2*h-1)-fénypont)/($\sqrt{2}$ *(h pixelnyi hossz))≈
≈ ((2*h)-fénypont)/($\sqrt{2}$ *(h pixelnyi hossz))=$\sqrt{2}$ *(h-fénypont)/(h pixelnyi hossz)=$\sqrt{2}$ ≈1,41

Ötlet:

kiegészíteni *kisebb* (számítható) fényerejű pontokkal.



3. Kör rajzolása

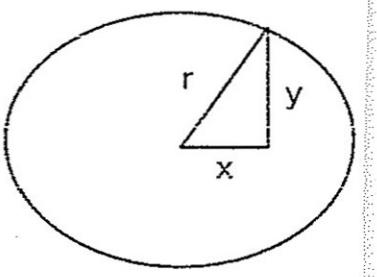
A feladat körívet rajzolni. Vegyük észre, hogy

1. a kör szimmetriája miatt, ha az `(x,y)` pont rajta van az íven , akkor az `(-x,y)`, `(x,-y)`, `(-x,-y)` `pontok` is rajta lesznek. (Sőt további szimmetriatengelyei is vannak, amelyek kihasználhatók!)
2. az (x_0,y_0) középpontú kör a `(0,0)` középpontú eltolásával egyszerűen megkapható, amelyet ismét rábízhatunk a `PontRajzol` eljárásra.

A: a körvonal pontjai: x,gyök(r*r-x*x)

Egy origó középpontú, r sugarú kör pontjaira a következő azonosság igaz: x*x+y*y= r*r! Ezt felhasználva készíthetjük el az első megoldást!

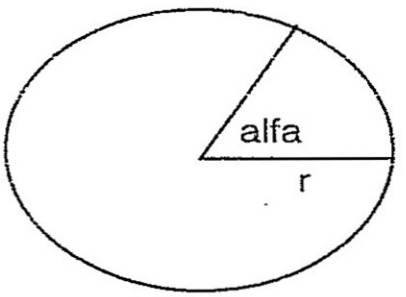
```
Eljárás KörRajzolás(Konstans r:Egész):
  Ciklus x=0-tól r-ig
    y:=Egész(Négyzetgyök(r*r-x*x))
    PontRajzolás(x,y); PontRajzolás(-x,y)
    PontRajzolás(x,-y); PontRajzolás(-x,-y)
  Ciklus vége
Eljárás vége.
```



Az ív „meredek" érintőjű részén szakadozik.

B: a körvonal pontjai: r*cos(alfa),r*sin(alfa)

Ezen a problémán segíthetünk azzal, hogy „egyenletesen" járjuk be a negyed körívet. Nem x-szerint haladunk, hanem origóból azonos ($\Delta\alpha$) szögelfordulással haladva A körvonal azon pontjának koordinátái, amely az x-tengellyel alfa szöget zár be, a következő azonossággal számolhatók: x=r*cos(alfa), y=r*sin(alfa)



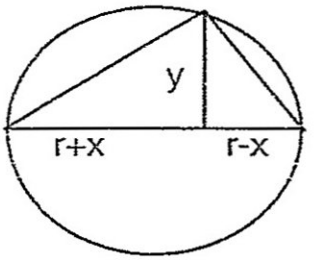

```
Eljárás KörRajzolás(Konstans r:Egész):
  Változó Δα,α: Valós
```

```
  Δα:=???
  Ciklus α=0-tól π/2-ig Δα-asával
    x:=r*cos(α); y:=r*sin(α)
    PontRajzolás(x,y); PontRajzolás(-x,y)
    PontRajzolás(x,-y); PontRajzolás(-x,-y)
  Ciklus vége
Eljárás vége.
```

Meggondolandó a $\Delta\alpha$ választása! Ha túl kicsire választjuk, akkor sokszor fogja ugyanazt a pontot kiszínezni, ha meg túl nagyra, akkor meg kimaradnak pontok.

Legyen $\Delta\alpha$-nyi fordulat az r-sugarú íven kb. 1 pixelnyi! Azaz $\Delta\alpha/(2*\pi) = 1/(2*r*\pi)$ $\Rightarrow \Delta\alpha = 1/r$

C: a körvonal pontjai: x,gyök((x+r)*(r-x))



Ha a kör átmérőjének két végpontját összekötjük a körvonal egy tetszőleges pontjával, akkor egy derékszögű háromszöget kapunk. Ennek a derékszögű háromszögnek a magassága adja meg az x koordinátához az y koordinátát. A derékszögű háromszög magasságára az alábbi ábra alapján a következő képlet adható: y*y=(r+x)*(r-x)

```
Eljárás KörRajzolás(Konstans r:Egész):
  Ciklus x=0-tól R-ig
    y:=Egész(Négyzetgyök((R+x)*(R-x))
    PontRajzolás(x,y); PontRajzolás(-x,y)
    PontRajzolás(x,-y); PontRajzolás(-x,-y)
  Ciklus vége
Eljárás vége.
```

Mindhárom megoldás esetén az az alapvető probléma, hogy nem biztos, hogy a körvonal folytonos vonal lesz, illetve elképzelhető, hogy nagyon sok pontot rajzolunk egymásra.

D: folytonos vonallal

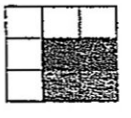
Egy, a görbék rajzolására általánosan alkalmazható ötlet:

1. kiindulás a görbe egy alkalmas kezdőpontjából,
2. válasszunk valamilyen elképzelhető haladási (rajzolási) irányt (balra/jobbra, fel/le),
3. az irányba eső szomszédos (mondjuk: 5) pontokat vizsgáljuk meg: melyik tér el legkevésbé a görbétől. S arra lépjünk tovább!

Pl.:

A kör esetén megint csak az első negyedbeli ívét használjuk föl „vezérlőként".

```
Eljárás KörRajzolás(Konstans r:Egész):
  x:=0; y:=r; rr:=r*r
  Ciklus amíg y≥0
    PontRajzolás(x,y); PontRajzolás(-x,y)
    PontRajzolás(-x,y); PontRajzolás(-x,-y)
    Elágazás
      (x+1)*(x+1)+y*y≤rr esetén
            x:+1              [vízszintesen]
      (x+1)*(x+1)+(y-1)*(y-1)≤rr esetén
            y:-1; x:+1        [vízszintesen és le]
```



```
        egyéb esetben
            y:-1              [le]
      Elágazás vége
    Ciklus vége
  Eljárás vége.
```

Megjegyzés:

Az világos, hogy az algoritmusbeli feltételek –sorrendjükkel együtt– feltételezik, hogy a „vezérlő szerepű" körív-darab éppen az *1. negyedbe eső negyed körív*, sőt, hogy a rajzolás az (0,r)-től kezdődik. (Ui.: y≥0; és elegendő 5 helyett a rajzon jelzett 3 „próbapont"; a „legkevésbé eltérés"-t helyettesíthetjük a „a körlapba először belépés"-sel.) Észre kell vegyük, hogy az általános algoritmust csak „ε-pontossággal" követi a fenti, ui. elképzelhető, hogy bár a jobbra lépés lenne a legjobb, mi mégis a „jobbra-le" szomszédot választjuk, mert ő az első még bennmaradó. Ez a hiba azonban a kör „körségét" nem befolyásolja.

E: A körvonal követésével

```
Körvonalpont(x,y):
  Rajzol(x,y); Rajzol(-x,y); Rajzol(-x,-y); Rajzol(x,-y)
  Rajzol(y,x); Rajzol(-y,x); Rajzol(-y,-x); Rajzol(y,-x)
Eljárás vége.
```

```
Körrajz(r):
  x:=0; y:=r [a tetején kezdjük, Kelet felé egy nyolcadot]
  d:=1-r; dE:=3: dSE:=-2*r+5
  Körvonalpont(x,y)
  Ciklus amíg y>x
    Ha d<0 akkor  [Keletre lépés]
      d:=d+dE; dE:=dE+2; dSE:=dSE+2; x:=x+1          [ y:=y ]
    különben   [Délkeletre lépés]
      d:=d+dSE; dE:=dE+2; dSE:=dSE+4; x:=x+1; y:=y-1
      Körvonalpont(x,y)
  Ciklus vége
Eljárás vége.
```

4. Alakzatok színezése

Adott színű (AktSzín), tetszőleges görbével határolt, zárt tartomány kiszínezése (adott másik színnel), ismert belső pontból kiindulva.

Itt már a képernyő-koordinátarendszerben dolgozunk, így további transzformálásra és kerekítésre nincs szükség. Emiatt a programozási nyelvek eredeti Pont(X,Y) függvényét használhatjuk rajzolásra.

A. Rekurzív festés pontonként

Ez az eljárás egy belső pontból kiindulva festi be az alakzatot. A rekurzió miatt nagy alakzatokhoz túl nagy a memóriaigénye.

```
Eljárás RekPont(Konstans x,y:Egész):
  Pont(x,y)
  Ha Üres(x-1,y) akkor RekPont(x-1,y)
  Ha Üres(x,y-1) akkor RekPont(x,y-1)
  Ha Üres(x+1,y) akkor RekPont(x+1,y)
  Ha Üres(x,y+1) akkor RekPont(x,y+1)
Eljárás vége.
```

Érdemes meggondolni, mekkora verem szükséges (maximálisan) a rekurzió során.

B. Sort alkalmazó festés pontonként

Ez az eljárás az előző egy gazdaságosabb változata, egy hullámfrontot ábrázoló sor adatszerkezetben.

Az alkalmazható sorműveletek:

```
Eljárás  SorUres{legyen}(Változó s:TSor)
Függvény UresSor{?}(Konstans s:TSor):Logikai
Eljárás  Sorba(Konstans elem:TSorElem, Változó s:TSor)
Eljárás  Sorból(Változó s:Sor, elem:TSorElem)
Függvény SorHossz(Konstans s:Sor):Egész

Eljárás SorPont(Konstans x,y:Egész):
  Változó s:TSor

  SorÜres(s)
  Pont(x,y); Sorba(x,y,s)
  Ciklus amíg nem ÜresSor?(s)
    Sorból(s,x,y)
    Ha Üres(x-1,y) akkor Pont(x-1,y); Sorba(x-1,y,s)
    Ha Üres(x,y-1) akkor Pont(x,y-1); Sorba(x,y-1,s)
    Ha Üres(x+1,y) akkor Pont(x+1,y); Sorba(x+1,y,s)
    Ha Üres(x,y+1) akkor Pont(x,y+1); Sorba(x,y+1,s)
  Ciklus vége
Eljárás vége.
```

D. Vermet alkalmazó festés pontonként

Ez az eljárás az előző egy másik változata, vermet használ a még feldolgozandó belső pontok ábrázolására.

Az alkalmazható veremműveletek:

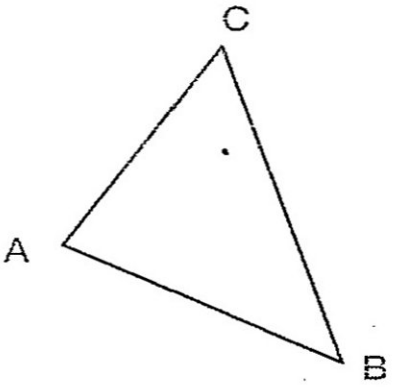
```
Eljárás  VeremÜres{legyen}(Változó v:TVerem)
Függvény UresVerem{?}(Konstans v:TVerem):Logikai
Eljárás  Verembe(Konstans elem:TVeremElem, Változó v:TVerem)
Eljárás  Verembol(Változó v:Verem, elem:TVeremElem)
Függvény VeremMelyseg(Konstans v:Verem):Egész

Eljárás VeremPont(Konstans x,y:Egész):
  Változó v:TVerem
```

```
  VeremÜres(v)
  Pont(x,y); Verembe(x,y,v)
  Ciklus amíg nem ÜresVerem?(v)
    Veremből(v,x,y)
    Ha Üres(x-1,y) akkor Pont(x-1,y); Verembe(x-1,y,v)
    Ha Üres(x,y-1) akkor Pont(x,y-1); Verembe(x,y-1,v)
    Ha Üres(x+1,y) akkor Pont(x+1,y); Verembe(x+1,y,v)
    Ha Üres(x,y+1) akkor Pont(x,y+1); Verembe(x,y+1,v)
  Ciklus vége
Eljárás vége.
```

E. Háromszögfestés

Egy háromszöget könnyű festeni, ha megadjuk azokat a vízszintes szakaszokat, amelyekkel teljesen befesthető. Ezen szakaszok végpontjai az oldalegyenesek egyenleteiből számolhatók, azaz nincs szükség a képmemória leolvasására.

```
Háromszögfestés(A,B,C):
  Ciklus S=C.Y-tól A.Y-ig
    O1:=(A-C egyenes) az S. sorban
    O2:=(B-C egyenes) az S. sorban
    Vonal(O1,S,O2,S)
  Ciklus vége
  Ciklus S=A.Y+1-től B.Y-ig
    O1:=(A-B egyenes) az S. sorban
    O2:=(B-C egyenes) az S. sorban
    Vonal(O1,S,O2,S)
  Ciklus vége
Eljárás vége.
```



F. Rekurzív festés szakaszonként

Egy alakzat kiszínezéséhez meg kell adni egy belső pontját. Ekkora színezést a következőképpen végezzük el. Húzzuk meg azt a függőleges vonalat, amely ezen a ponton keresztül halad, s a kiszínezendő ábra legközelebbi határvonaláig tart. Ez a vonal a kiszínezendő ábrát (legalább) 2 részre vágja: a vonaltól balra, illetve jobbra levő részre. (Kettőnél több részről akkor van szó, ha a vonal érintette valahol a kiszínezendő ábrát.) A feladat ezek után e két rész kiszínezése. Ezt úgy oldjuk meg, hogy végigmegyünk a vonal mindkét oldalán, s ha belső, még kiszínezetlen pontot találunk, akkor a fenti eljárás végrehajtjuk ebből a pontból kiindulva.

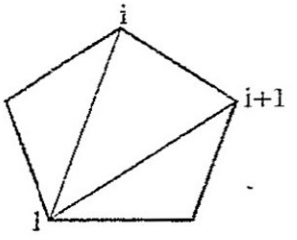
```
Eljárás RekSzakasz(Konstans x,y:Egész):
  Változó i,y1,y2:Egész

  Vonalvég(-1,x,y,y1); Vonalvég(+1,x,y,y2)
  VonalRajzolás(x,y1,x,y2)
  Ciklus i=y1-től y2-ig
    Ha Üres(x-1,i) akkor RekSzakasz(x-1,i)
    Ha Üres(x+1,i) akkor RekSzakasz(x+1,i)
  Ciklus vége
Eljárás vége.
```

```
Eljárás Vonalvég(Konstans merre,x,y:Egész, Változó z:Egész):
  z:=y
  Ciklus amíg Üres(x,z+merre)
    z:=z+merre
  Ciklus vége
Eljárás vége.
```

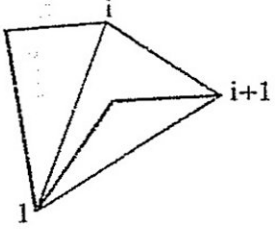
A feladat nemrekurzív változata ennél jóval bonyolultabb lesz. Itt verembe tesszük a megrajzolandó vonal végpontjai koordinátáit, miközben az előbb megrajzolt szakasz oldalait járjuk be, s csak ezután fogunk hozzá a verem tetején levő szakasz tényleges feldolgozásához.

```
Eljárás RekSzakasz(Konstans x,y:Egész):
  Változó i,y1,y2:Egész
  Vonalvég(-1,X,Y,Y1): Vonalvég(1,X,Y,Y2)
  Verembe(X,Y1,Y2)
  Ciklus amíg a verem nem üres
    Veremből(X,Y1,Y2)
    Vonalrajzolás(X,Y1,X,Y2)
    Szomszéd oszlop(X-1,Y1,Y2)                    [bal szomszéd]
    Szomszéd oszlop(X+1,Y1,Y2)                    [jobb szomszéd]
  Ciklus vége
Eljárás vége.

Eljárás Szomszéd oszlop(Konstans X,Yk,Yv):
  I:=Yk
  Ciklus amíg I≤Yv
    Ha üres(X,I) akkor Vonalvég(-1,X,I,Y3); Vonalvég(1,X,I,Y4)
                       Verembe(X,Y3,Y4); I:=Y4+1
                 különben I:=I+1
    Elágazás vége
  Ciklus vége
Eljárás vége.
```

G. Sokszög festése háromszögenként



Konvex sokszög mindig felbontható háromszögekre egy tetszőleges csúcsából kiindulva, így alkalmazható rá a felbontás (1., i-edik és i+1-edik csúcs), majd a háromszögfestés.



Konkáv sokszög esetén is alkalmazható a fenti eljárás, de ekkor a sokszögön kívüli területeket is befest ez az eljárás. Megállapíthatjuk ugyanakkor, hogy a sokszögön kívüli területeket mindig páros sokszor, a belülieket pedig páratlan sokszor festi. Ekkor XOR műveletet alkalmazva a festéskor, végeredményként éppen a befestett alakzatot kapjuk.

H. Sokszög festése metsző vonalakkal

A képernyő széléről a másik széle felé haladva számoljuk, hogy hány határoló szakaszon haladunk át! A belső pontokban ez a szám mindig páros, a külső pontokban pedig páratlan. Kivételt jelentenek sajnos a vízszintes szakaszokat, illetve alsó vagy felső csúcspontokat tartalmazó sorok. A megoldás: ne a képpontokon, hanem pontosan két

képpont között húzzuk a képzeletbeli vonalat, s ez alapján figyeljünk a párosságra, páratlanságra!

Egy másik megoldásnál számoljuk ki az összes (nem vízszintes) határoló szakasz és képernyő vízszintes szakaszai metszéspontjait, majd ezeket képernyő soronként rendezzük sorba! Ekkor minden esetben páratlan sorszámú ponttól a következő páros sorszámúig kell egy vízszintes szakaszt rajzolni a képernyőre.

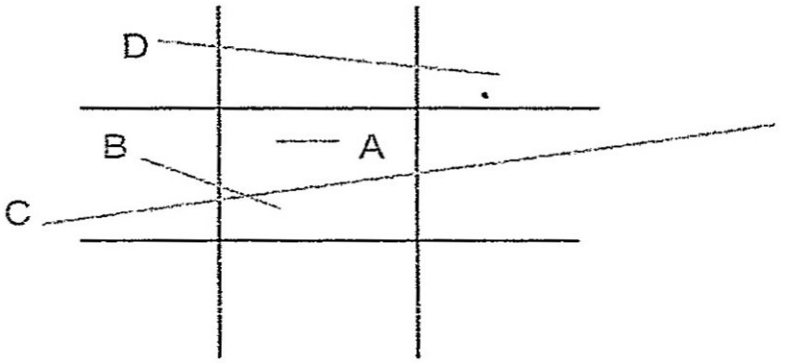
5. Vágás (Cohen-Sutherland)

Ha az alakzat nagyobb, mint a képtartomány*, amelyben megjelenítendő, akkor a kívül eső részeket el kell hagyni, azaz az alakzatról „le kell vágni", röviden szólva: az alakzatot *vág*ni kell.

Tegyük föl, hogy az alakzat (egyenes) *szakasz*okra bontható. (A tartományok vágása hasonlóan megoldható, bár további problémákat is fölvet.) Az alábbiakban szakaszok vágását vizsgáljuk.

Az alakzat és a képernyő viszonya a következő lehet:

Az ábrán az A szakasz teljes egészében látszik a képen, a B-nek csak az egyik része látható, a C-nek csak egy középső darabja látszik, a D pedig egyáltalán nem látszik. Ez a négy eset sokféleképpen fordulhat elő. A feladatunk ezek vizsgálata, s a szükséges vágások elvégzése lesz.



Egészítsük ki a képet az egész képmezőre, s rendeljünk egy négy bites kódot minden mezőhöz:

1001	0001	0011
1000	0000	0010
1100	0100	0110

A kódolás célja az, hogy elkerüljük nagyon sok logikai feltétel vizsgálatát.

Ha egy szakasz mindkét végpontjának kódja 0000, akkor a teljes szakasz látszik, ha a végpontok kódjával végrehajtott logikai ÉS művelet eredménye nem 0, akkor a szakasz biztosan nem látszik (a kódokat ezért választottuk ilyennek). Ha nem ezek az esetek állnak fenn, akkor a szakaszt a képernyő egyik élével alkotott metszéspontjánál kettévágjuk. Az így kapott egyik szakasz biztosan a képernyőn kívül lesz, a másikra pedig újra el kell végezni a fenti eljárást (ugyanis a 0001-ből 1000-ba húzott szakasz vagy metszi a képernyő egyik élét, vagy nem).

Megjegyzés: A nagybetűs logikai műveletek egész számokra vonatkozó bitenkénti műveleteket jelentenek.

Legyen a képernyő felső sorának y-koordinátája KF, alsó sorának pedig KA, a baloldali szél x-koordinátája KB, a jobboldalié pedig KJ!

---

* Szokás szerint a képernyővel megegyező állású téglalapra gondolunk ***képtartomány*** esetén.

Legyen a képernyő felső sorának y-koordinátája KF, alsó sorának pedig KA, a baloldali szél x-koordinátája KB, a jobboldalié pedig KJ!

```
Vágóalgoritmus(X1,Y1,X2,Y2):
  Végpontok kódjának előállítása(C1,C2)
  Ciklus amíg nem ( (C1=0 és C2=0) vagy (C1 ÉS C2)≠0 )
    Ha C1=0 akkor Csere(C1,X1,Y1,C2,X2,Y2)
    Ha (C1 ÉS 1)=1 akkor Vágás a felső szélen
    Ha (C1 ÉS 2)=2 akkor Vágás a jobbszélen
    Ha (C1 ÉS 4)=4 akkor Vágás az alsó szélen
    Ha (C1 ÉS 8)=8 akkor Vágás a balszélen
    Végpontok kódjának előállítása(C1,C2)
  Ciklus vége
  Ha C1=0 és C2=0 akkor Szakasz rajzolás(X1,Y1,X2,Y2)
Eljárás vége.

Végpontok kódjának előállítása(C1,C2):
  C1:=0; C2:=0
  Ha Y1>KF akkor C1:=C1 VAGY 1
  Ha Y2>KF akkor C2:=C2 VAGY 1
  Ha X1>KJ akkor C1:=C1 VAGY 2
  Ha X2>KJ akkor C2:=C2 VAGY 2
  Ha Y1<KA akkor C1:=C1 VAGY 4
  Ha Y2<KA akkor C2:=C2 VAGY 4
  Ha X1<KB akkor C1:=C1 VAGY 8
  Ha X2<KB akkor C2:=C2 VAGY 8
Eljárás vége.
```

A vágáshoz mindig két szakasz metszéspontjának koordinátáit kell kiszámítani.

```
Vágás a felső szélen:
  X1:=X1+(X2-X1)*(KF-Y1)/(Y2-Y1); Y1:=KF
Eljárás vége.

Vágás az alsó szélen:
  X1:=X1+(X2-X1)*(KA-Y1)/(Y2-Y1); Y1:=KA
Eljárás vége.

Vágás a jobbszélen:
  Y1:=Y1+(Y2-Y1)*(KJ-X1)/(X2-X1); X1:=KJ
Eljárás vége.

Vágás a balszélen:
  Y1:=Y1+(Y2-Y1)*(KB-X1)/(X2-X1); X1:=KB
Eljárás vége.
```

III. Képtömörítés

A JPG képtömörítési szabvány az ISO által elfogadott eljárás, ami lehetővé teszi felhasználóinak a nagy tömörítési arányt a legkevesebb minőségromlás mellett. Ennek a módszernek az alaplépéseit ismerhetjük meg ebből a leírásból; a diszkrét koszinusz transzformációt, a kvantálást, és a kódolást.

A tömörítés aránya egy 24 bites kép esetén elérheti a 100:1-hez arányt, de szinte észlelhetetlen minőségromlással is 10:1 - 30:1 arány érhető el. Ezért azt lehet mondani: a képtömörítés nem csak adatfeldolgozási probléma. Mivel a végső felhasználó többnyire ember, ezért a nagy tömörítést lehetővé tevő hiba tényleges hatása az emberi látórendszer zajérzékenységétől függ. Ezzel a kérdéssel empirikus tudományágak foglalkoznak (pszichofizika, kísérleti pszichológia stb.), az eredményeket viszont a képfeldolgozásban is régóta eredményesen alkalmazzák - a leghétköznapibb példa erre a televíziós műsorszórás.

Hogy ebben a témakörben aránylag ilyen hosszú időt áldozunk a kérdésre, az indokolja, hogy a videó-digitalizálással nyert roppant méretű adatmennyiség számítástechnikai kezelése még ma sem egyszerű feladat. Újabb megoldások gyakran keletkeznek. Hogy valami módon csökkenjen a fájl mérete, meg kell próbálnunk az adatok között válogatni, csak az „értékesebbeket, hasznosabbakat" meghagyni, esetleg valamiféle, fontosság szerinti újrarendezést megvalósítani. Ezek során gyakran kényszerülünk – a technika mai állását is figyelembe vevő – kompromisszumokra. A megfelelő eljárás kiválasztása a témával foglalkozók egyik kulcsproblémája.

A számítástechnikában már régebben alkalmaznak a hosszú fájlokkal való munkák során méretcsökkentést eredményező tömörítési eljárásokat. Az eljárásokat kétféleképpen csoportosíthatjuk: vannak veszteségmentes és veszteséges tömörítések. Magától értetődő, hogy kívánatosabb lenne az előbbit használni, de az is köztudott dolog, hogy igazán látványos (akár néhányszor tízszeres!) méretcsökkenést csak akkor lehet produkálni, ha a matematikusok által kidolgozott algoritmusokat egyéb megfontolásokkal kombináljuk. A nagy adatbázisokat nem tekintve az elektronikus képfeldolgozás során vált először szükségessé számítógépes tömörítési eljárások használata. A nyomdai reprodukáláshoz szükséges nagymennyiségű képpont a hozzátartozó színinformációkkal együtt már olyan nagy fájlméreteket produkálnak, hogy mindenképpen célszerű valamiféle tömörítési eljárás alkalmazásán gondolkodni. Képekre a fentebb említett kompromisszumos megoldáskeresések során a Joint Photographic Expert Group képtömörítési eljárása (a JPEG) lett széleskörben elfogadva. Algoritmusuk az ismétlődő redundáns információk megkeresésén és kiszűrésén alapul, jellemzője a változtatható értékű, nagy (maximum kb. tízszeres) arányt elérő veszteséges tömörítés. Ebben a tömör mondatban nagyon lényeges dolgok vannak megfogalmazva. A legfontosabb szó a veszteséges. Másik fontos dolog a változtatható tömörítési arány. Legyünk őszinték! Azt még egy kevésbé

képzett felhasználó is megérti, hogy veszteségkor valami elvész. De azt már sokkal kevesebben teszik megfontolás tárgyává – talán még olyanok sem mindig, akik JPEG fájl-formátumú képeket készítenek –, hogy az információk elvesztésének szabályozása egyértelműen a kép keletkezése során megadott paraméterekkel történik. Másképp fogalmazva: a (megmaradó) minőség (vagy az ezzel fordított arányban álló tömörítési arány) valamiféle skálán általunk megválasztható. Anélkül, hogy részletesen elmagyaráznánk a tömörítés során alkalmazott eljárást, annyit érdemes tudnunk, hogy a minőségi elvárásainkhoz beállított paraméter szerint a redundáns információk képzése bizonyos nagyságú (pl. 4x4, 6x6, 8x8) képpontmátrixokból vett adatok alapján történik. Gondolom mindenki előtt világos, hogy annál több lesz a veszteség, de kisebb a fájlméret, minél nagyobb mintamátrix alapján történik a további (kiátlagoló) számolás. Csak veszteségmentes tömörítéssel olyan mértékű fájlméret csökkentést nem tudnánk elérni, mint a JPEG használatával! A kompromisszum egyik oldalán a kényelmesebben kezelhető kevesebb adat, a másik oldalon viszont az elvesztett és vissza már nem szerezhető adatok vannak.

A digitális kép általában igen nagy, de redundáns adathalmazként fogható fel. Ez utóbbi annyit jelent, hogy a képek által tartalmazott lényegi információt jóval kevesebb független adattal lehet jellemezni, mint ahány adatból maga a kép áll. Lényegében ez adja meg a lehetőséget a kép tömörítésére, ami tehát nem a kép méretének csökkentését jelenti, hanem ugyanannak a képnek a tárolásához szükséges adatok mennyiségének csökkentését. A felhasználási területek többsége a képi adatok kismértékű változását is megengedi (ha a felhasználás célkitűzéseit ez a változás nem veszélyezteti), s ez a redundancia kihasználásán túlmenően további tömörítésre nyújt lehetőséget. Így pl. abban az esetben, ha a képek szerepe csupán a látvány létrehozása (egzakt feldolgozásra nincs szükség), akkor az emberi látás sajátságainak figyelembevételén alapuló veszteséges, de jelentős látványromlással nem járó képtömörítési eljárások is alkalmazhatók. (pl. JPEG)

A képtömörítés tipikusan az a művelet, amire érdemes célhardvert készíteni. (Az algoritmusok a szükséges szabványosítás után már nem változnak, a célhardver egyéb készülékekkel egybeépíthető, így azok eladhatóságát javítja, a köznapi felhasználási igények miatt a célhardver nagy szériában gyártható, azaz kellően olcsó lehet.)

Néhány alaptípus:

Az amerikai Intel cég i750 (video processor) VLSI áramkörével 1/160 arányú mozgókép-tömörítés is elérhető, s így pl. szabványos optikai lemezre (CD-ROM-ra) 72 percnyi mozgó videó felvétel rögzítésére is van lehetőség, JPEG és MPEG kódoló, dekódoló eszközök. Léteznek valós idejű (real-time) változatai is, melyek videojelet szolgáltató képforrással együttműködve folyamatosan szolgáltatják az 1/15-1/20 arányban tömörített fekete-fehér, illetve színes képeket.

Megjegyzendő, hogy képek kódolása alatt nem csupán tömörítéseiket, hanem pl. illetéktelen használat elleni titkosításukat is értjük. Természetesen erre a kényes felhasználási területre is készültek célberendezések.

A legtöbb grafikus fájl tartalmaz ismétlődő információkat (például nagyobb kiterjedésű, azonos színű háttér, stb..) és nagy a méretük is. Így viszonylag hamar felmerült a képek tömörítésének kérdése.

A *veszteséges tömörítés* mellett a visszaállított kép minősége rosszabb, mint az eredetié. Ez azonban a legtöbb esetben nem okoz problémát, mivel az emberi szem nem érzékeli, vagy nem olyan mértékűnek érzékeli a romlást. Ennél a módszernél a minőségromlás mértékétől függően átlagosan 1:10 arányú méretcsökkenés érhető el. Ilyen típusú például a JPG ((Joint Picture Expert Group), amely az egyik legelterjedtebb képtárolási formátum. Ennek a formátumnak egyik változata az ún. progresszív JPEG, amely a gif interlaced módjához hasonlóan a megjelenő kép fokozatosan finomodik. Ez utóbbi változat hátránya a nagy számolás igény.

1. Veszteséges tömörítők

A JPEG a szabványos veszteséges tömörítési eljárás. Nevét arról a bizottságról kapta, amely elsőként fogadta el ezt a szabványt. Először ismerkedjünk meg a veszteséges tömörítési eljárás fogalmával. Az ilyen eljárások kidolgozására a CD-ROM-ok tömeges megjelenése váltotta ki az igényt. Ahhoz ugyanis, hogy CD-ROM-on videoképeket lehessen tárolni, mintegy 180:1 arányú tömörítést kell elérni. Ennek az az oka, hogy egy tömörítetlen videó lejátszásához 27 Mbájt/s-os adatátviteli sebesség szükséges a CD-ROM meghajtók szabványos 150 Kbájt/s-os sebességével szemben. A veszteségmentes tömörítés esetében azonban a kompressziós arány alacsony, rendszerint 2:1-hez. A legtöbb tömörítő rendszer ezért veszteséggel dolgozik, azaz a tömörítő eljárásban az eredeti kép egy része elvész. Ennek ellenére az ilyen eljárásoknak van létjogosultsága, mivel a fejlesztők feltételezik, hogy a legtöbb felhasználó észre sem veszi a kisebb résszletek elvesztését, különösen nem egy gyors film megtekintésekor.

A veszteséges rendszerek többnyire úgy működnek, hogy elhagyják vagy megváltoztatják például a képfrekvenciát, a képméretet vagy a pixel színmélységét. Megőrzik viszont azokat az információkat (például a képélességet, kontrasztot, a szinek és mozgások folyamatosságát), amelyek a kép szubjektív megítélését befolyásolják, tehát azokat a jellemzőket amelyek egy hétköznapi TV-n vagy videón is vezérelhetők.

A legtöbb digitális kompressziós rendszer a redundáns adatokat is felismeri, egy képen belül éppúgy, mint az egymást követőkön. Ezeknek a tényezőknek a kihasználása nagyon nagy tömörítési arányhoz vezet, akár a kívánt 180:1-hez is.

A JPEG először pixelek blokkjaira bontja szét a képet, és matamatikai ismétlődések halmazaként ábrázolja valamennyi blokk mintáit és színeit. Végezetül kiszűri az ismétlődéseket, hogy csökkentse a blokkokban lévő ismétlődéseket, és így a kép tárolásához szükséges helyet is.

A digitális kép általában igen nagy, de redundáns adathalmazként fogható fel. Ez utóbbi annyit jelent, hogy a képek által tartalmazott lényegi információt jóval kevesebb független adattal lehet jellemezni, mint ahány adatból maga a kép áll. Lényegében ez adja meg a lehetőséget a kép tömörítésére, ami tehát nem a kép méretének csökkentését jelenti, hanem ugyanannak a képnek a tárolásához szükséges adatok mennyiségének csökkentését. A felhasználási területek többsége a képi adatok kismértékű változását is megengedi (ha a felhasználás célkitűzéseit ez a változás nem veszélyezteti), s ez a redundancia kihasználásán túlmenően további tömörítésre nyújt lehetőséget. Így pl. abban az esetben, ha a képek szerepe csupán a látvány létrehozása (egzakt feldolgozásra nincs szükség), akkor az emberi látás sajátságainak figyelembevételén alapuló veszteséges, de jelentős látványromlással nem járó képtömörítési eljárások is alkalmazhatók. (pl. JPEG)

Ebben a leírásban a képek veszteséges tömörítéséről olvashatsz. Csak olyan képekkel foglalkozunk, ahol a pixelek színkomponensenként vannak tárolva, nem pedig palettából indexelve. A tárgyalt algoritmus a JPEG és MPEG kódolásának is alapja.

A színmélység lehet bármilyen, ez nem befolyásolja az algoritmust, de kisebb mélységgel jobb tömörítés érhető el, mert a komponensegyüttthatók jobban kvantálhatóak. Pl. egy 24 bites kép esetén a pixelek komponensei 8 bitet foglalnak el, míg egy 15 bites képnél csak 5-öt. Természetesen az eljárás használható grayscale képek kódolására is. Ilyenkor csak egy komponenst kódolunk a szokásos háromhoz képest.

Amikről szó lesz:

- kódolások csoportosítása
- színmodell választás
- transzformációs kódolás (DCT)
- kvantálás
- a kódolás befejező lépései
- számítási komplexitás
- kódolási hibák
- introk veszteséges kódolással

1.1. Kódolások csoportosítása

- veszteséges, veszteségmentes
  Veszteségmentes tömörítéssel kb. 1.5-2-szeres tömörítés érhető el, ez az arány veszteségessel sokkal nagyobb, hátránya, hogy a kódolás után kitömörített kép nem lesz azonos az eredetileg kódolttal.
- pixelalapú, blokkalapú
  A pixelalapú kódolás egy ütemben egy pixelt kódol, míg a blokkalapú egy tipikusan 8x8-as blokkot, amivel nagyobb nyereséget kapunk.

A következőkben a veszteséges, blokkalapú kódolást tárgyaljuk meg.

1.2. Színmodell választás

A képek kódolásakor fontos szerepet játszik, hogy milyen modellt választunk a képek leírására. A szokásos RGB színrendszerben a látható színek nagy része előállítható, de ez nem elég hatékony, mivel a komponensek (r – vörös, g – zöld, b - kék) tárolására közel ugyanannyi bitet kell használni. Célszerű olyan ábrázolási módot választani, amely a veszteséges tömörítéskor elkövetett hibával szemben kevésbé érzékeny, illetve amelyben a minél nagyobb tömörítési arány érhető el.

Az emberi látás sokkal kevésbé érzékeny az ún. krominancia összetevőkre (szín), mint az ún. luminancia összetevőre (fényesség). Ezért érdemes az RBG helyett az YUV modellt választani, amelyben a színösszetevők adatai a látás szempontjából fontosabb és kevésbé fontos adatokra válnak szét. Egy képponthoz itt is három értéket/komponenst rendelünk:

- y – a színjel világossága, fénysűrűség,
- u – kék színkülönbségi jel,
- v – vörös színkülönbségi jel.

A konvertálás az RGB színrendszerből az alábbi képletek alapján lehetséges:

$$y=0,3*r+0,59*g+0,11*b$$
$$u=(b-y)/2,03$$
$$v=(r-y)/1,14$$

Ugyenezek a képletek felírhatók mátrixos formában is:

$$\begin{vmatrix} y \\ u \\ v \end{vmatrix} = \begin{vmatrix} 0,299 & 0,587 & 0,114 \\ -0,147 & -0,289 & 0,437 \\ 0,615 & -0,515 & -0,100 \end{vmatrix} \begin{vmatrix} r \\ g \\ b \end{vmatrix}$$

Hasonlóan felírható a konvertálás inverzének mátrixa is:

$$\begin{vmatrix} r \\ g \\ b \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1{,}140 \\ 1 & -0{,}394 & -0{,}581 \\ 1 & 2{,}028 & 0 \end{vmatrix} \begin{vmatrix} y \\ u \\ v \end{vmatrix}$$

Az YUV modellben az u és v komponenseknek nincs fényességtartalmuk, csak az adott jel színéről hordoznak információt. Mivel ezek az emberi szem számára kevésbé fontosak, ezeknél a komponenseknél nagyobb hibát engedhetünk meg a tömörítés során.

1.3. Transzformációs kódolás

A transzformációs kódolás alapja, hogy a képet N*N-es blokkokra osztjuk, és a blokkokat a frekvenciasíkra transzformáljuk. Így N*N együtthatót kapunk. Ezt a transzformációt többféle képlet alapján is végezhetjük.

A diszkrét transzformációk és inverzeik általános képlete:

$$y_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} * a_{ijkl}$$

$$x_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{n} y_{kl} * b_{ijkl} \quad ,$$

ahol x_{ij} az eredeti, y_{kl} a transzformált kép képpontjainak mátrixa. Az a és b együtthatókat az adott transzformáció határozza meg.

A tömörítés mértéke attól függ, hogy ezeket az együtthatókat milyen pontossággal számoljuk ki. Míg az eredeti kép pontjai ugyanannyi információt hordoztak, addig a kapott értékek között vannak fontosak és kevésbé fontosak (kevesebb információt hordozók). Ezek elhagyása nem okoz olyan változást, ami az emberi szem számára látható lenne.

1.4. Diszkrét koszinusz transzformáció

A diszkrét transzformációk közül a legelterjedtebb a diszkrét koszinusz transzformáció (DCT). Ennek képlete:

$$DCT(k_1,k_2) = \alpha(k_1) * \alpha(k_2) * \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} DCT^{-1}(i,j) * \cos(\frac{2i+1}{2N}\pi * k_1) * \cos(\frac{2j+1}{2N}\pi * k_2)$$

$$DCT^{-1}(i,j) = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} \alpha(k_1) * \alpha(k_2) * DCT(k_1,k_2) * \cos(\frac{2i+1}{2N}\pi * k_1) * \cos(\frac{2j+1}{2N}\pi * k_2)$$

ahol $\alpha(n) = \begin{cases} \frac{1}{\sqrt{N}}, k = 0 \\ \sqrt{\frac{2}{N}}, k = 1, \ldots, N-1 \end{cases}$

A következő ábrákon egy-egy képrészlet (8*8-as blokk), és azok kódoltjai láthatóak az intenzitási együtthatók megadásával:

Egy fehér négyzet és transzformáltja:

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

65280	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

A homogén, egyszínű felületek transzformáltja sok egymás utáni 0-t tartalmaz, ami jól tömöríthető.

255	0	255	0	255	0	255	0
0	255	0	255	0	255	0	255
255	0	255	0	255	0	255	0
0	255	0	255	0	255	0	255
255	0	255	0	255	0	255	0
0	255	0	255	0	255	0	255
255	0	255	0	255	0	255	0
0	255	0	255	0	255	0	255

32640	0	0	0	0	0	0	0
0	530.2	0	625.4	0	936	0	2665
0	0	0	0	0	0	0	0
0	625.4	0	737.7	0	1104	0	3144
0	0	0	0	0	0	0	0
0	936	0	1104	0	1652	0	4705
0	0	0	0	0	0	0	0
0	2665	0	3144	0	4705	0	13399

A DCT előnye, hogy a használt együtthatók valósak. Az elvégzendő műveletek száma igen nagy, ezért a képletek közvetlen számításokra nem alkalmasak. A transzformáció hatékonyan számítható közelítése például az RVFFT (Real Valued Fast Fourier Transform) eljáráson alapul.

1.5. Kvantálás

A kvantálás egy, az eredeti halmaznál lényegesen kisebb elemszámú halmazra való olyan leképezés, ami az eredeti érékhez a legközelebbi értéket rendeli hozzá a halmazból egy távolságkritérium (úgy nevezett norma) alapján. Például ilyen leképezés a lebegőpontosról, a legközelebbi egészre. (A JPEG 11 bit pontossággal kvantál.) Ez a művelet nyilván valóan veszteséges és tömörít.

A kvantálás végezhető önálló értékre (skalár kvantálás), ennél lényegesen hatékonyabb viszont, ha több értéket egy blokkba foglalunk össze (vektor kvantálás: képek esetében tipikusan 2 x 2, 3 x 3, 4 x 4-es vektorok). A kvantálás a frekvenciatartományban lényegesen hatékonyabb, mint az eredeti képen - a zajérzékenységet figyelembe véve (vagyis, hogy az emberi látórendszer a nagyobb frekvenciájú jelek esetén kevésbé érzékeny a torzításra) az egyre nagyobb frekvenciákat egyre durvábban kvantáljuk.

A kvantálás egy N x N-es kvantálási mátrix alapján történik. (A fent említett módszer alapján történő tervezését, a mátrix Human Visual System alapján történő tervezésnek nevezzük.)

```
JPEG default kvantálási tábla y komponensre
{
   16.0,   11.0,   10.0,   16.0,   24.0,   40.0,   51.0,   61.0,
   12.0,   12.0,   14.0,   19.0,   26.0,   58.0,   60.0,   55.0,
   14.0,   13.0,   16.0,   24.0,   40.0,   57.0,   69.0,   56.0,
   14.0,   17.0,   22.0,   29.0,   51.0,   87.0,   80.0,   62.0,
   18.0,   22.0,   37.0,   56.0,   68.0,  109.0,  103.0,   77.0,
   24.0,   35.0,   55.0,   64.0,   81.0,  104.0,  113.0,   92.0,
   49.0,   64.0,   78.0,   87.0,  103.0,  121.0,  120.0,  101.0,
   72.0,   92.0,   95.0,   98.0,  112.0,  100.0,  103.0,   99.0
}

JPEG default kvantálási tábla u,v komponensre
{
      17,   18,   24,   47,   99,   99,   99,   99,
      18,   21,   26,   66,   99,   99,   99,   99,
      24,   26,   56,   99,   99,   99,   99,   99,
      47,   66,   99,   99,   99,   99,   99,   99,
      99,   99,   99,   99,   99,   99,   99,   99,
      99,   99,   99,   99,   99,   99,   99,   99,
      99,   99,   99,   99,   99,   99,   99,   99,
      99,   99,   99,   99,   99,   99,   99,   99
}
```

1.6. Blokkméret választása

A kódolási egységként használt blokkok méretének megválasztásakor több szempontot is figyelembe kell venni:

- a számítógép a 2 hatványaival gyorsabban, hatékonyabban számol,
- minél kisebb a blokkméret,
- annál homogénebb lesz a kép,
- annál több a kisfrekvenciás együttható, ezért finomabban kell kvantálni, így a tömörítés mértéke kisebb lesz.

Ezeknek a szempontoknak a segítségével a különböző blokkméretekről a következőket állapíthatjuk meg:

- 4*4-es blokk: durva kvantálás kell a jó tömörítéshez, ami sokat ront a kép minőségén,
- 8*8-as blokk: jól tömöríthető úgy, hogy közben a minősége is jó marad,
- 16*16-os blokk: a tömörítés mértéke csak kis mértékben javítható, viszont eközben jelentősen nő a számításigény.

Kedvező tulajdonságai miatt a 8*8-as blokkméret terjedt el.

A kép blokkokra való széttagolása különböző hibákat is okozhat a kapott kép minőségében:

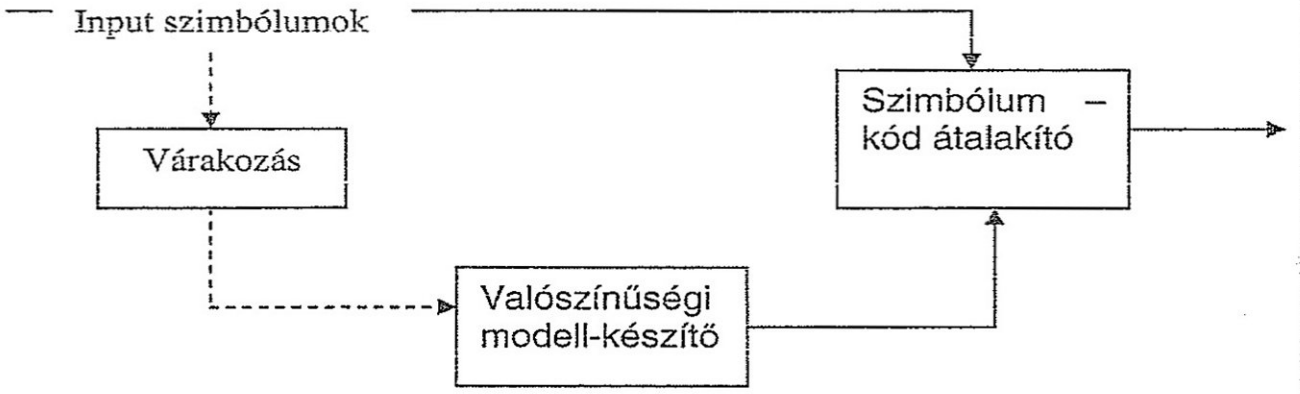
- blokkosodási hiba: a blokkok határai megjelennek,
- Gibbs-oszcilláció vagy mosquito-zaj: ha egy blokkhatáron megy át egy él, akkor az él körül elmaszatolódás vehető észre.

2. Nem információvesztő képtömörítési módszerek

2.1. Általánosságban a veszteségmentes képtömörítésről (különös tekintettel az entrópiás kódolásra)

A nem veszteséges tömörítések azokhoz a tömörítési módszerekhez tartoznak, amelyek képesek a tömörített állományból pontosan az eredeti adatsorozatot előállítani. Sok alkalmazás esetén – például az orvostudományban – elvárjuk, hogy a képfile tárolása során ne vesszen el adat.

Nézzük meg, hogy milyen előzetes tennivalókat kell elvégeznünk a rendelkezésre álló bemenő adatokkal!



A kódolásnál fontos szempont lehet, hogy a gyakran előforduló szimbólumok kódja rövid legyen, a ritkán előfordulóké már lehet hosszabb. Ehhez tudnunk kell, hogy milyen a bemenő szimbólumok (*input szimbólumok*) eloszlása. Ennek megbecslését a *modellkészítő* végzi el, a későbbiekben az általa becsült eredmények felhasználásával történik a *szimbólum – kód átalakítás*. A kódolásnak azt a formáját, amikor a szimbólum – kód átalakítást és a modellkészítést egyaránt használjuk, *entrópiás kódolásnak* nevezzük.

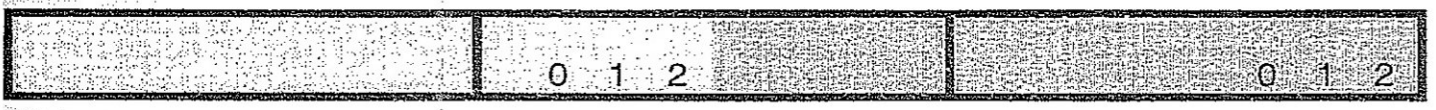
Az ábrán látható valószínűségi modell kétféleképpen adható meg:

- a bemenő adatok figyelembe vételével
- a bejövő adatokról előzetesen kialakított feltevés segítségével

Nézzünk egy-egy példát a két esetre! A bemenő szimbólumok legyenek karakterek. Ha figyelembe vesszük a bemenő adatokat (a rajzon szaggatott vonallal jelölve), akkor a valószínűségi modellkészítő megszámolja, hogy az egyes karakterből mennyi van a bejövő szimbólumsorozatban, és ez alapján ad pontos eloszlást a bemenő adatokról. A másik esetben az eloszlás becslését az input-sorozat vizsgálata nélkül adjuk meg. Ha tudjuk, hogy a magyar nyelvben milyen eloszlást mutatnak az egyes karakterek, akkor ezt az eloszlást fogadjuk el érvényesnek a bejövő karaktersorozatra is. Ezzel a módszerrel nem pontos, csak becsült eloszlást kapunk. Előnye, hogy az eloszlás gyorsabban megkapható, mint a dinamikus (első) modell esetén, hátránya, hogy nem lesz feltétlenül pontos, és ez a későbbiekben azt eredményezheti, hogy kisebb hatékonysággal tömörít. (Azaz minél pontosabb a becslés, annál jobb lesz a tömörítés.) Fontos megemlíteni, hogy a tömörítés nem mindig garantált! Ha a valószínűségi modell nagyon pontatlan, akkor elképzelhető, hogy a kimenő file mérete megnő. (Ezzel együtt az eredeti bemenet továbbra is veszteség nélkül visszaállítható.)

Eddig azt néztük végig, hogy az entrópiás kódolás hogyan csinál a bemenő szimbólumokból kimenő szimbólumokat. A valóságban azonban a bemenő adataink nem szimbólumok. Egy kép mérete 256*256 pixeltől 64000*64000 pixelig terjedhet. A teljes bemenő adat a legkisebb méret esetén is 64000 egység. Ezt nem tekinthetjük egyetlen szimbólumnak. A gyakorlatban a képet szimbólumok sztringjeként (sorozataként) értelmezzük. A 256*256 pixeles kép esetében feltesszük, hogy minden pixel értéke egy 0 és 255 közötti egész szám (azaz a kép 8 bit/pixeles). Ekkor a kép olyan szimbólumok sorozata, amelynek tagjait a {0,1,2,...,255} halmazból választjuk.

Más problémával is szembe kell néznünk. A veszteség nélküli tömörítések az adatokat 8, 16, 32 vagy 64 bites egységekben tudják kezelni. Ha a képünk 12 bit/pixeles (azaz az abc-je: 0,1,...,2^{12}-1), akkor valahogy át kell alakítani. Nézzünk erre egy megoldási lehetőséget! Tekintsük a 12 bit/pixeles adatsorozatot adatfolyamnak, majd csoportosítsuk a biteket 8-asával, 16-osával, vagy ahányasával szeretnénk:



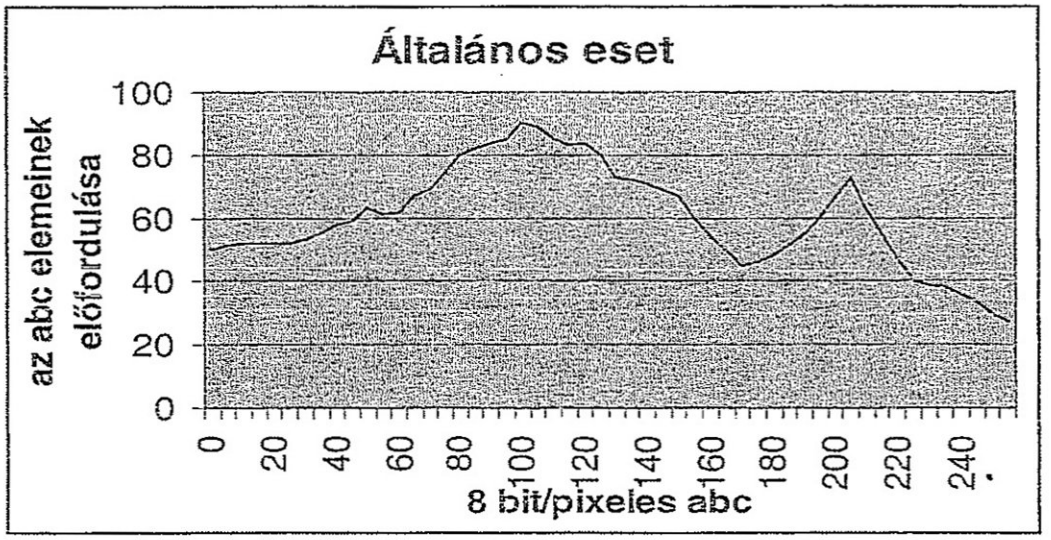
A fenti transzformáció elvégzésével nem csak azt értük el, hogy megfelelő egységekre tördeltük az adatfolyamot, hanem csökkentettük az abc méretét is. A 12 bit/pixeles képek abc-je ugyanis 2^{12} elemű, a 8 bit/pixeleseké viszont csak 2^8 tagból áll.

Foglalkozunk még egy előzetes átalakítással, amely növeli a tömörítés mértékét. Az eljárás neve különbségi kódolás, és az a lényege, hogy aszimmetrikussá teszi a szimbólumok statisztikáját, és az így kapott eloszlás entrópiás kódolás esetén jobb tömörítést tesz majd lehetővé. A különbségi kódolás esetén kihasználjuk a képfile-ok azon tulajdonságát, amely szerint a szomszédos pixelek között erős korreláció áll fenn, azaz nagyon hasonlók lehetnek. Tegyük fel, hogy a pixelek a következő sorrendben követik egymást: X_1, X_2, X_3,...,X_n. Ekkor a pixe-

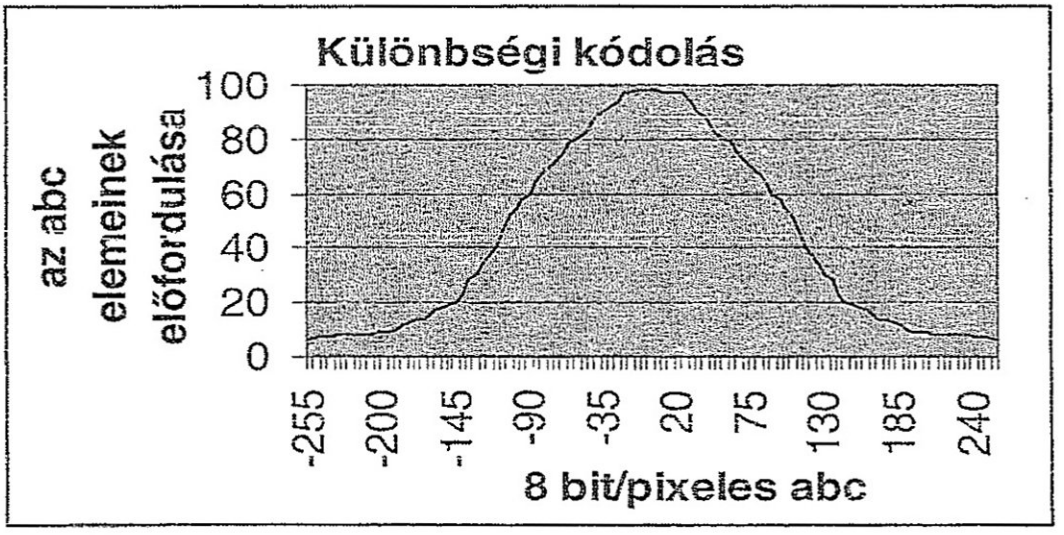
lek tömörítése helyett a szomszédosak különbségeinek sorozatát tömörítsük. A tömörítendő adatok sorozata tehát legyen a következő:

$$Y_i:=X_i-X_{i-1} \qquad i=1,2,... \qquad X_0=0$$

Egy általános kép esetén a kétféle eloszlás így néz ki (az első ábra x_i-k, a második y_i-k eloszlását mutatja):



Most nézzük meg, hogy miért jó a különbségi kódolással aszimmetrikussá teszzük a szimbólumok eloszlását! Legyen egy s_i szimbólum előfordulásának valószínűsége p_i. A kódolási elképzelésből adódik, hogy a szimbólum – kód átalakításkor a kódszó hossza körülbelül



$\log_2 1/p_i$ bit lesz. Ha ez nagyjából minden i-re azonos, azaz a kép pixelei között elhanyagolható a különbség, akkor p_i 1/255 lesz minden i-re. Ez azt jelenti, hogy minden kódszó 8 bites lesz, vagyis valójában nem végeztünk tömörítést. Itt az volt a probléma, hogy a p_i-k lehettek közel azonosak. Különbségi kódolással viszont ezt ki tudjuk küszöbölni, így jobb lehet a tömörítés.

2.2. Huffmann kódolás (JPEG)

A Huffmann-kódolás az entrópiás kódolások közé tartozik, tehát teljesül rá, hogy a gyakran előforduló jeleknek rövid, a ritkábban előfordulóknak pedig hosszú bitsorozat felel meg. Ezen kívül a Huffmann-kódok prefixek, azaz nincs olyan kód, ami egy másiknak a kezdőszelete. Ennek következménye, hogy elválasztójelek használata nélkül is végezhetjük a kódolást, így is dekódolható az eredeti információ. (Megjegyezzük, hogy minden prefix kód egyértelműen visszaállítható, de van olyan nem prefix kód is, amely egyértelműen visszaállítható. Például legyenek a kódok a következő abc elemei: 0, 01, 011, 0111, 01111, ... Itt a 0 jelzi az új kód kezdetét.)

A kódgeneráláshoz felépítünk egy bináris fát a szimbólumok eloszlásának segítségével. (Részletesen lásd µ14 38. o.) A kód hosszát az adja meg, hogy a fában hányadik szinten helyezkedik el a szimbólumot tartalmazó levél. A Huffmann-kódolás átlagos szóhossza:

$$l_{avg}=\Sigma l_i * p_i,$$

ahol l_i a kódhossz, p_i pedig az adott szimbólum előfordulásának valószínűsége. Shannon szerint az optimális kódhosszt a következő képlettel számolhatjuk ki:

$$H(S)= \Sigma p_i * \log_2 1/ p_i$$

A Huffmann-kód átlagos hossza csak kevéssel tér el az optimális kódhossztól, mivel teljesül rá a következő összefüggés:

$$H(S) \leq l_{avg} \leq H(S)+1$$

Érdemes megjegyezni, hogy ha van egy Huffmann-kódunk, akkor annak a komplemense is Huffmann-kód. (Komplemensnek ebben az esetben azt a kódolást nevezzük, amikor a kódban szereplő 1-eseket 0-vá, a 0-kat pedig 1-esekké változtatjuk.) Ennek a tulajdonságnak gyakorlati haszna is van. Tegyük fel, hogy egy Huffmann-kóddal kódolt információ olyan zajos csatornán halad keresztül, ahol az 1-0 átalakulás valószínűbb, mint a 0-1, akkor a két komplemens Huffmann-kódból választhatjuk azt, amelyikben több 0 van mint1.

A Huffmann-kód hátránya, hogy a visszatömörítéshez mellékelnünk kell a kódfát. A jelnek ott van vége, ahol a fában már nem tudunk tovább menni.

A fenti kódvisszafejtésnek az a hátránya, hogy egy adott kódot annyiszor kell visszakeresni a fában, ahányszor előfordul a kódolt adatsorban. Ezt a hátrányos tulajdonságot küszöböli ki a Look-up-table dekódolás. Ennek lényege, hogy a memóriában felépítünk egy 2^L sort tartalmazó táblázatot. (L a leghosszabb kódszó bitjeinek száma, másképp a kódfa mélysége.) A táblázatot a következő szabály szerint töltjük ki:

Legyen c_i a kódolt állományban szereplő kódszó, s_i a c_i által kódolt szimbólum (vagyis az, amit c_i dekódolásával kapunk), l_i pedig jelölje c_i hosszát. Készítsünk olyan L-bites (Look-up-table-beli) címeket, amelyek első l_i bitje c_i, a maradék $L-l_i$ bitet pedig töltsük ki 0-val és 1-gyel

minden lehetséges módon. Az így kijelölt sorok tartalmazzák az s_i szimbólumot. Az s_i szimbólumnak ezek szerint 2^{L-l_i} címe lesz a Look-up-table-ben.

Nézzünk egy példát!

A kódfa mélysége (azaz L) legyen 4, az s_i szimbólum legyen az 'a' karakter, s_i kódja (c_i) pedig legyen 01. Ekkor a Look-up-table következő soraiba írjuk be 'a'-t: 0100 (4), 0101 (5), 0110 (6), 0111 (7).

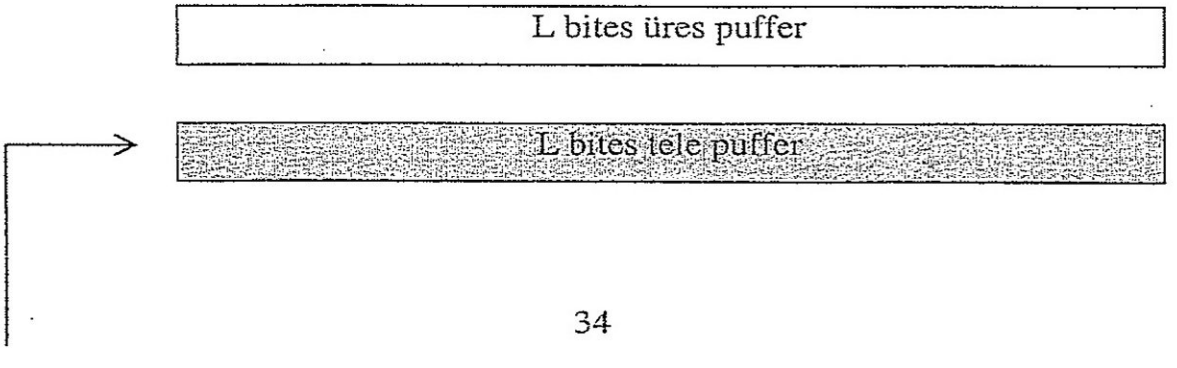
A kitöltött Look-up-table tehát a következőképpen néz ki:

2^{L-l1} db sor	'a'
	...
	'a'
2^{L-l2} db sor	'b'
	...
	'b'
2^{L-l3} db sor	'c'
	...
	'c'

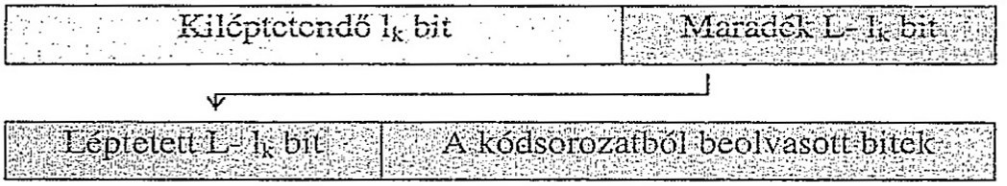
Most nézzük meg, hogy mi értelme volt a Look-up-table kitöltésének! Egyik előnye, hogy a kódfában minden szimbólumot csak egyszer kell megkeresni. Ha megtaláltuk a szimbólumot, akkor az őt kódoló kódszó segítségével meghatározzuk, hogy a Look-up-table melyik soraiba kell őt beírni. A továbbiakban dekódolásra már nem a kódfát, hanem a táblázatunkat használjuk. A dekódolást az alábbi módszerrel végezzük:

- beolvasunk L bitet a pufferbe (L a kódfa mélysége)
- a puffer tartalmát a Look-up-table egy címének (sorának) tekintjük, az adott sor tartalma lesz a kódolt szimbólum. A szimbólum kódjának hossza legyen l_k. (A szimbólumokhoz tartozó kódhosszokat szintén állapítsuk meg a táblázat felépítésekor, és tároljuk valahol.)
- Dobjuk el a pufferből az első l_k bitet, a maradékot toljuk a puffer elejére. Az így megüresedett helyekre olvassuk be a kód soron következő bitjeit.
- Ezt ismételjük addig, amíg a teljes kódnak a végére nem értünk. Ha elfogyott a kódsorozat, akkor megkaptuk az eredeti szimbólumsorozatot.

A fenti lépéseket mutatja az alábbi ábra:



A puffer által tartalmazott számot sorszámnak tekintve megkapjuk a Look-up-table valamelyik sorát, írjuk ki ennek a sornak a tartalmát. (Ez a keresett szimbólum)



A Look-up-table használatával a dekódolás folyamata gyorsabb lesz, viszont a táblázat felépítése sok munkával jár, ráadásul nő a helyfoglalás is.

2.3. LZW kódolás (GIF, TIFF)

Az LZW nem az entrópiás kódolások közé tartozik. Nem jelekhez, hanem jelcsoportokhoz rendel kódot, ezen kívül itt minden kód azonos hosszúságú. A kódolás egy táblázat segítségével történik. Előnye, hogy a dekódoláshoz nem kell csatolnunk a táblázatot, mivel az a dekódoló folyamat közben felépül. Az LZW kódolás menete részletesen megtalálható a Mikrológia sorozat 14. kötetének 41. oldalán.

A GIF és a TIFF file-ok használnak LZW kódolást. Nézzük meg, hogy a GIF-ben használt LZW algoritmus miben különbözik az általános LZW algoritmustól!

- Definiálva van egy speciális Clear ('üres') kód, minden betömörítő/kitömörítő paramétert és táblázatot a kezdeti állapotra állít. A Clear kód értéke 2^<kódméret>. Például ha a jelzett kódméret 4 (azaz a kép 4 bit/pixeles), akkor a Clear kód értéke 16. A Clear kód az adatfolyamban bárhol szerepelhet, és azt kívánja az LZW algoritmustól, hogy úgy folytassa a számítást, mintha egy új adatfolyam kezdődne.
- Az Információ Vége kód az image adatfolyam végét határozza meg. Az LZW folyamat megáll, amikor ez a kód kerül sorra. Az Információ Vége kód értéke <Clear kód>+1.
- Az első elérhető tömörített kód értéke <Clear kód>+2.
- A kimenő kódok változó hosszúságúak (3 és 12 bit közötti hosszúságok fordulnak elő), ezért ezeket a kódokat 8-bites byte-sorozattá kell alakítani. Ez az 1. fejezetben megismert módon történik. Miután a byte-okat létrehoztuk, blokkokba kell csoportosítani őket úgy, hogy minden blokkot előzzön meg egy karakterszámláló byte. A nulla byte-számlálójú blokk zárja le az adott image raszter-adatfolyamát. Ez a blokkfelépítés teszi lehetővé a dekódoló program számára, hogy a blokkszámláló figyelésével átugorhassa az aktuális képadatokat amennyiben ez szükséges.

2.4. A PCX által használt tömörítés

A PCX-ben használt tömörítő eljárás alapvetően a szomszédos byte-ok összehasonlításával dolgozik. Megszámolja, hogy hányszor követi egymást ugyanaz a szimbólum, majd ezt az ismétlődés számával és a szimbólummal kódolja. A következőkben a példákban szereplő számok hexadecimális számokat jelölnek.

Ha öt egymás utáni byte azonos (01 01 01 01 01), akkor a PCX kimenetén ehelyett az öt byte helyett mindössze két byte szerepel (C5 01). A sorszámot egy speciális flag segítségével (itt C) különböztetjük meg az adatoktól. Amennyiben egy adott byte csak egyszer szerepel (azaz nincs ismétlődés), akkor a byte kódja önmaga. Például legyen a bemenő adat a következő: 01 01 01 01 04 01 01. Ekkor a kimeneten a C5 01 04 C2 01 kód jelenik meg. Az algoritmus két esetben akadhat el:

- Ha egy byte értéke nagyobb vagy egyenlő C-vel. Például ha a C2 C2 C2 C2 C2 a bemenő adat, akkor a kód C5 C2 lesz. Ahhoz, hogy az ismétlődést jelölő C-t ne keverjük össze az adat C-jével, figyelnünk kell a C előfordulási számának paritását. Ezzel megoldódik a gondunk, mivel ha a C páratlan előfordulási számú, akkor sorszámot, különben pedig adatot jelöl.
- A másik problémás helyzet akkor áll elő, amikor több mint 15 egymás után következő byte azonos. Mivel az ismétlődést jelző flag mögött álló számnak bele kell férnie négy bitbe (azaz legfeljebb 15 lehet az értéke), ezért a 15. ismétlődő byte után kódoljuk az addig elolvasott információt, majd újra kezdjük a számolást. Például ha 17 azonos byte követi egymást (01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01), akkor a kód CE 01 C3 01 lesz.

A PCX által használt kódolás a tokenizálás mintájára történik. Erről részletesen a 14-es Mikrológia 36. oldalán lehet bővebben olvasni.

2.5. Az RLE tömörítés

Az RLE tömörítésnek két fajtája van: a nyolc és a négy bites RLE. A két sűrítési módszer alapjaiban megegyezik, csak egy kis eltérés van közöttük.

A nyolc bites RLE tömörítés

A kódolás itt akkor hatékony, ha egy szín sokszor követi egymást. A képadatban két byteot találunk.

- Ha az első byte értéke nem 0, akkor a második byte által meghatározott színnel kiteszünk az első byte értékének megfelelő darabszámú pixelt a képernyőre.
- Ha az első byte 0, akkor a második byte vezérlési feladatokat lát el:
- 00: sor vége jel
- 01: BMP file vége
- 02: rajzolási pozíció beállítása (relatív megadás)
- Az ezután következő két byte közül az első azt mondja meg, hogy mennyivel jobbra, a második pedig azt, hogy mennyivel lejjebb folytassuk a rajzolást.
- 03 vagy nagyobb (FF-ig): pixelek abszolút megadása (a szám azt jelenti, hogy hány pixelt adunk meg egymás után), a sort általában (páratlan darabszám esetén) egy 00 byte zárja

Az alábbi táblázat néhány példán keresztül mutatja be a fenti szabályok gyakorlati megvalósulását.

8 bites RLE-vel tömörített adat	Értelmezett adat
03 04	04 04 04
06 2D	2D 2D 2D 2D 2D 2D
00 02 09 03	mozgatás 9 pixellel jobbra és 3-mal lefelé
02 48	48 48
00 03 55 1A FF 00	55 1A FF
0A 12	12 12 12 12 12 12 12 12 12 12
00 04 98 AF A0 8C	98 AF A0 8C
00 00	vége a sornak
00 01	vége a file-nak

A négy bites RLE tömörítés

Az előzőnél valamivel bonyolultabb tömörítési módszer. Itt egy byte nem egy, hanem két pixel színét határozza meg. A tömörítés akkor hatékony, ha két szín váltakozva követi egymást. A byte-ok jelentése ugyanaz, mint a nyolc bites tömörítésnél. Az előző táblázatban látható példák segítségével hasonlítsuk össze a két eljárást! (Kiemeltük azokat a sorokat, amelyeknek első oszlopa eltér az előző táblázat megfelelő sorának első oszlopától.)

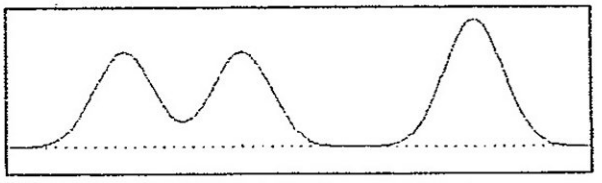
4 bites RLE-vel tömörített adat	Értelmezett adat
03 04	0 4 0
06 2D	2 D 2 D 2 D
00 02 09 03	mozgatás 9 pixellel jobbra és 3-mal lefelé
02 48	4 8
00 06 55 1A FF 00	5 5 1 A F F
0A 12	1 2 1 2 1 2 1 2 1 2
00 08 98 AF A0 8C	9 8 A F A 0 8 C
00 00	vége a sornak
00 01	vége a file-nak

3. Fourier transzformáció

A (fekete-fehér) képfeldolgozásban általában minden pontnak külön határozzuk meg a szürkeségét. Sokszor azonban jobban megismerhetjük a kép felépítését, szerkezetét, ha frekvenciák szerint adjuk meg az adatokat. Az így kapott függvényes formátum azért előnyös, mert ennek kezelésére egy sor eljárás áll rendelkezésünkre a matematikában. A legelterjedtebb módszer, amely a képpontokkal megadott képet átalakítja frekvencia adatokra, a Fourier Transzformáció. Amikor kiszámoljuk egy kép Fourier transzformáltját, akkor a képen lévő jelek erősségét határozzuk meg függvény formájában, és nem csupán értékek tömbjével dolgozunk.

3.1. A Fourier-elmélet

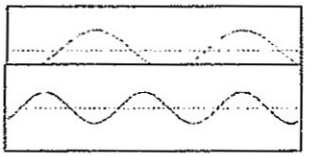
A Fourier-elmélet olyan matematikai módszer, amely szerint a legtöbb függvény ábrá-



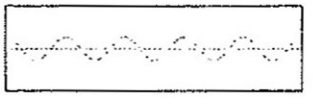
zolható több (akár végtelen sok) szinusz- és koszinuszfüggvény összegeként.

Példaként először nézzük, hogyan állíthatjuk elő az alábbi két kisebb és egy nagyobb lokális maximumot tartalmazó függvényt szinuszfüggvényekből.

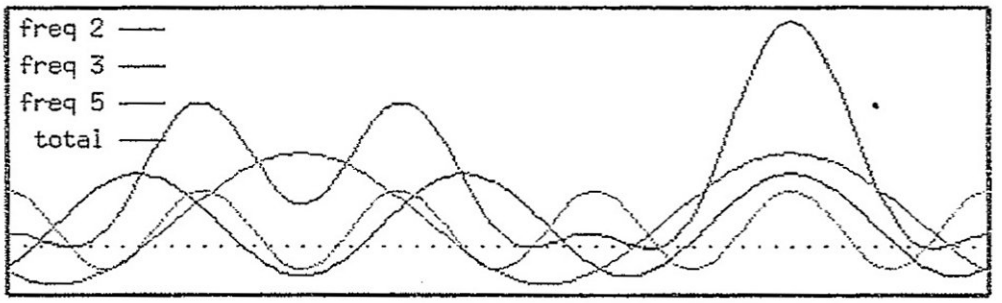
Az első szinuszfüggvény, amivel próbálkozunk, kétszer ismétlődik a vizsgált tartományban, azaz a frekvenciája 2. Az első csúcs helyettesíti a két kisebb lokális maximumot, a második a nagyobbat.



A második szinuszfügvény frekvenciája 3, a harmadiké 5. (Figyeljük meg, hogy a függvények az x tengely mentén eltolódhatnak!)

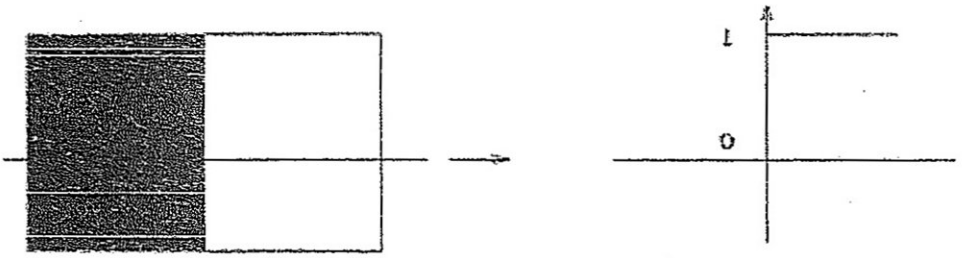


Az ábrán láthatjuk, hogy a három függvény összege már elég jól közelíti az eredetit.



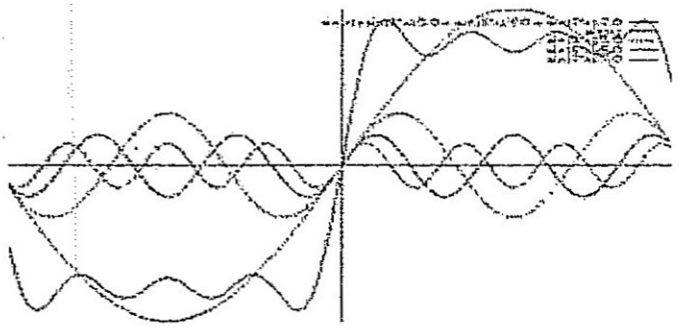
3.2. Képek Fourier transzformáltja

Térjünk vissza a képfeldolgozáshoz, és nézzük meg az alábbi kép egyik vízszintes metszetének Fourier transzformáltját!



Az y tengelytől balra a metszet teljesen fekete (legyen a függvényérték itt 0), jobbra fehér (értéke: 1). Ezt az úgynevezett lépcsős függvényt közelíthetjük a sinx, sin3x, sin5x, ... függvények végtelen sorával. $f(x) = \sin x + 1/3 \sin 3x + 1/5 \sin 5x + 1/7 \sin 7x + \ldots$, ahogy azt a következő ábra is mutatja.

A különböző frekvenciájú szinuszfüggvények együtthatója más és más. Ezekkel határozzuk meg az eredeti függvényt. Így egy függvény Fourier transzformáltjának megadásához a frekvenciákhoz tartozó együtthatókat kell kiszámolnunk. Az előző példánál F(1)=1, F(3)=1/3, F(5)=1/5 stb. (A függvények Fourier transzformáltjait mindig nagybetűvel jelöljük, változói



pedig a frekvenciák.)

3.3. Matematikai alapok

De hogyan tudjuk kiszámolni az együtthatókat? A Fourier transzformált meghatározásakor valójában a függvényt ortogonális bázisfüggvények összegére bontjuk fel, ugyanazon a módon, ahogy az Euklideszi térben egy pontot megadunk a bázisvektorok segítségével. A bázisfüggvények különböző frekvenciájú szinusz- és koszinuszfüggvények. A megfelelő frekvenciájú függvényekhez tartozó skalárok adják a Fourier transzformált adott frekvenciabeli értékét.

Először a háromdimenziós térben gondoljuk végig egy pont meghatározásának menetét. Itt lehetnek bázisvektorok **i**(1,0,0), **j**(0,1,0), **k**(0,0,1). Ezekkel pl. a v(3,5,7) így írható fel: **v**=3**i**+5**j**+7**k**. Az együtthatókat úgy kapjuk meg, hogy vesszük a megfelelő bázisvektornak és **v**-nek a skaláris szorzatát. (3=**v**·**i**, 5=**v**·**j**, 7=**v**·**k**)

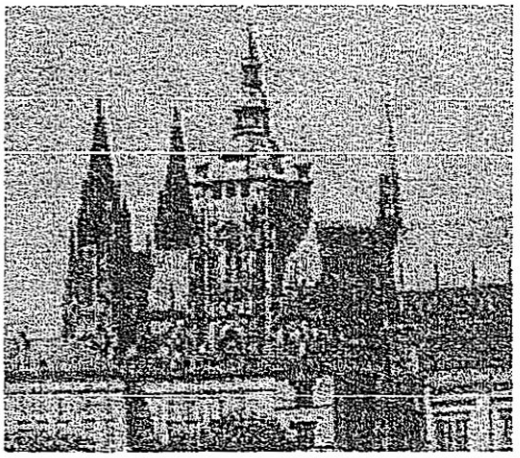
Ugyanilyen módszerrel számolhatjuk ki a függvények Fourier transzformáltját is. A függvények vektorterében ortogonális bázist alkotnak a $\{\cos(2\pi \cdot ux) - i\sin(2\pi \cdot ux), u \in R\}$ alakú függvények, ahol az u frekvencia tetszőleges valós értéket felvehet. Ebben a vektortérben a két függvény skaláris szorzata a szorzatuk integrálja (ez megfelelő, hiszen az integrál is egy számot ad). Tehát az f függvény Fourier transzformáltja minden u frekvenciára a következő:

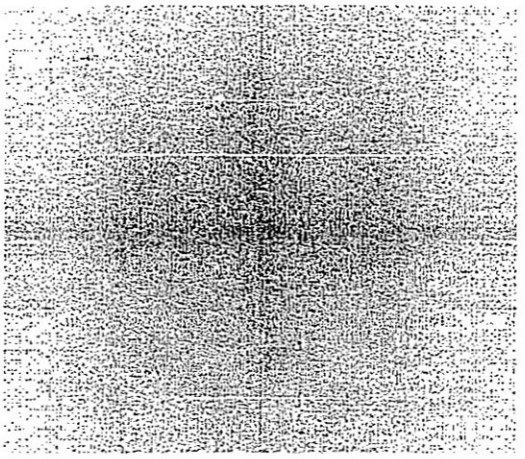
$$F(u) = \int f(x) \cdot (\cos(2\pi \cdot ux) - i \sin(2\pi \cdot ux))\, dx .$$

Az inverz függvénye pedig (vagyis amikor visszakapjuk f-et a transzformáltból):

$$f(x) = \int F(u) \cdot (\cos(2\pi \cdot ux) + i \sin(2\pi \cdot ux))\, du .$$

Az eredeti kép mellett látható a Fourier transzformáltja.

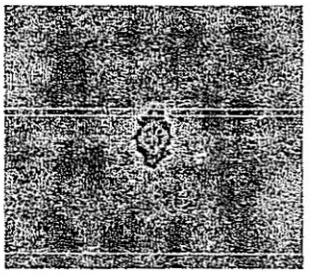


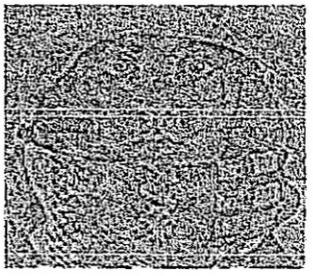


3.4. A fázis szerepe

Általában F(u) komplex mennyiség, még akkor is, amikor f(x) valós volt. Ez azt jelenti, hogy nemcsak az egyes frekvenciákhoz tartozó szorzó nagysága a fontos, hanem ezek fázisa is (azaz, hogy az x tengely mentén mekkora a bázisfüggvények eltolása). A legelső példán láttuk, hogy bár a két kisebb és egy nagyobb lokális maximumot tartalmazó függvény felírható csak szinusz függvényekkel, de a függvények fázisa különböző.

A hölgy képének vettük a Fourier transzformáltját, majd vissza is alakítottuk a képet. Az első esetben csak a transzformált valós részével számoltunk, a másodikban csak a képzetessel. A képeken jól látszik, hogy a fázis tartalmazza az információ jelentős részét, hiszen ezen a téma nagyjából felismerhető.





3.5. Diszkrét Fourier transzformáció

Ahhoz, hogy képekről származó információkat tudjunk transzformálni, a diszkrét Fourier transzformációra (DFT) van szükségünk, hiszen a képet diszkrét képpontok adják. Ekkor képpontokkal egyező számú frekvenciához számolhatjuk ki Fourier transzformáltat.

Természetes gondolat, hogy ilyenkor integrálás helyett elég a szorzatot szummáznunk.

A diszkrét Fourier transzformáció képlete:

$$F(u)=\frac{1}{N}\sum_{x=0}^{N-1}(f(x)\cdot(\cos\frac{2\pi\cdot xu}{N}-i\sin\frac{2\pi\cdot xu}{N}))$$

Az inverz diszkrét Fourier transzformáció képlete pedig:

$$f(x)=\sum_{x=0}^{N-1}(F(u)\cdot(\cos\frac{2\pi\cdot xu}{N}+i\sin\frac{2\pi\cdot xu}{N}))$$

A *DFT algoritmusa*

A transzformáció és inverze nem sokban tér el egymástól, ezért egy algoritmusban írhatjuk le őket. A `dir=1` adja a transzformációt, `dir=-1` az inverzét.

A pontok számát `N`-nel jelöljük, az értékük valós részét az `x1`, képzetes részét az `y1` vektorban tároljuk. A Fourier transzformáltat az `x2` és `y2` vektorban kapjuk meg. (Az inverz számolásánál `x1`, `y1`-ből vesszük a transzformáltat és `x2`, `y2`-be tesszük a visszaalakított függvényt.)

```
DFT:
  Ciklus i=0-tól N-1-ig
    x2[i]:=0; y2[i]:=0
    arg:=-dir*2*pi*i/N              //a sin és cos argumentuma
    Ciklus k=0-tól N-1-ig
      cosarg:=cos(k*arg); sinarg:=sin(k*arg)
      x2[i]:=x2[i]+x1[k]*cosarg-y1[k]*sinarg
      Y2[i]:=y2[i]+x1[k]*sinarg+y1[k]*cosarg
    Ciklus vége
  Ciklus vége
  Ha dir=1 akkor                      //csak az előre irányban
    Ciklus i:=1-től N-1-ig      //osztunk N-nel minden tagot
      x2[i]:=x2[i]/N
    Ciklus vége
  Elágazás vége
Eljárás vége
```

3.6. Gyors Fourier transzformáció

A diszkrét Fourier transzformációt gyakran a jóval hatékonyabb gyors Fourier transzformációval (FFT) helyettesítik. Használatának egyetlen feltétele, hogy a pontok száma kettő hatvány legyen, $N=2^n$.

A transzformáció képletében a függvényt az N-edik egységgyökkel szorozzuk. A gyors Fourier transzformációban az egységgyökök tulajdonságait használjuk ki (jelöljük az N-edik egységgyököt ω_N-nel, és M:=N/2). A szummát két részre bontjuk. Egyikben az M szerinti páros, másikban a páratlan változójú részek lesznek. Ekkor u alsó értékeire (1 ... M-1):

$$F(u)=\frac{1}{2}(\frac{1}{M}\sum_{x=0}^{M-1}f(2x)\omega_M^{xu}+\frac{1}{M}\sum_{x=0}^{M-1}f(2x+1)\omega_M^{xu}\omega_{2M}^{u})$$

u felső értékeire (M ... 2M-1) pedig:

$$F(u)=\frac{1}{2}(\frac{1}{M}\sum_{x=0}^{M-1}f(2x)\omega_M^{xu}-\frac{1}{M}\sum_{x=0}^{M-1}f(2x+1)\omega_M^{xu}\omega_{2M}^{u}).$$

Ez rekurzív algoritmushoz vezet, hiszen a fenti képletek két szummája az induló függvény szummájával azonos, csak az elemszám feleződött. A kettéosztást, és ezzel a tényezők felezését addig folytatjuk, amíg már a szumma helyett egyetlen tag szerepel.

[f(0),f(1),f(2),f(3),f(4),f(5),f(6),f(7),f(8)]

[f(0),f(2),f(4),f(6)] [f(1),f(3),f(5),f(7)]

[f(0),f(4)] [f(2),f(6)] [f(1),f(5)] [f(3),f(7)]

Az ábra szintjei mutatják a rekurzió menetét. Tehát N=8 esetén az ábra legalsó sorában látható párok megfelelő egységgyökökkel szorzott tagjait kell először összeadnunk illetve kivonnunk.

A rekurzív algoritmus helyett általában iteratívat használnak. Az f(x) értékek megfelelő sorrendbe tevése után rögtön az egységgyökkel való szorzás, majd az összeadás illetve a különbségképzés végezhető el.

A helyes sorrend meghatározásához a bináris bitfordító eljárást alkalmazhatjuk. Ennek elve az, hogy ha kettes számrendszerben a tagok sorszámát megfordítjuk bitenként, akkor éppen a rekurzió által létrehozott sorrendet kapjuk. (Gondoljunk arra, hogy például a 011. helyen az áll, amely először a bal oldali, majd kétszer a jobb oldali csoportba került. Tehát páros, de a fele és annak a fele kettővel osztva egyet ad maradékul, azaz 110.)

{000, 001, 010, 011, 100, 101, 110, 111}={0, 1, 2, 3, 4, 5, 6, 7}

{000, 100, 010, 110, 001, 101, 011, 111}={0, 4, 2, 6, 1, 3, 5, 7}

Az FFT algoritmusa

Az x vektorban van a 2^n pont értéke komplex számként. Az algoritmusban a rekurziónál mutatott ábra alsó szintjétől haladunk felfelé.

```
FT:
 bitfordítás(x)
 ωm:=-1                                          //második egységgyök
 Ciklus s=1-től n-ig                            //a szinteken fölfelé
   m:=2^s; ω:=1
   Ciklus j:=0-tól m/2-1-ig
     Ciklus k:=j-től 2^n-ig m-esével
       a:=ω*x[k+m/2]                                 //az első szumma
       b:=x[k]                                      //a második szumma
       x[k]:=a+b                                    //k alsó értékeire
       x[k+m/2]:=a-b                               //k felső értékeire
     Ciklus vége
     ω:=ω*ωm          //m-edik egységgyök következő hatványa
   Ciklus vége
   ωm:=sqrt(ωm)                                    //2m-edik egységgyök
 Ciklus vége
 Ciklus i:=1-től N-1-ig        //osztunk N-nel minden tagot
   x2[i]:=x2[i]/N
 Ciklus vége
Eljárás vége
```

3.7. Két dimenziós Fourier transzformált

Képek transzformálásához a Fourier transzformált kétdimenziós változatára van szükségünk. Ennek képlete:

$$F(u,v)=\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cdot(\cos\frac{2\pi\cdot i(xu+yv)}{N})-i\sin(\frac{2\pi\cdot i(xu+yv)}{N})=$$

$$=\frac{1}{N}\sum_{x=0}^{N-1}((\cos\frac{2\pi\cdot ixu}{N}-i\sin\frac{2\pi\cdot ixu}{N})\sum_{y=0}^{N-1}(\cos\frac{2\pi\cdot iyv}{N}-i\sin\frac{2\pi\cdot iyv}{N}))$$

, ahol a kép NxN képpontból áll.

Szerencsére a felbonthatóság miatt tekinthetjük ezt úgy is, mint két egydimenziós Fourier transzformált. A jobb oldali szumma x-et konstansnak tekintve egydimenziós. A bal oldali pedig szintén egydimenziós, melyben szorzóként felhasználjuk az előzőleg kapott összeget.

Tehát a kétdimenziós Fourier transzformáció kiszámolásának menete a következő:

- x minden értékére előállítunk egy egydimenziós Fourier transzformáltat, vagyis f(x,y) oszlopait, aztán
- az előbb meghatározott adatok segítségével egy újabb egydimenziós transzformáltat alkalmazva megkapjuk a kétdimenziós Fourier transzformáltat.

3.8. A Fourier transzformáció alkalmazása

A képfeldolgozásban a Fourier transzformációt leginkább a képek javításában használják. A kép torzulását okozhatja az optikai lencse hibája, rossz fókusz beállítás, a kamera vagy a

tárgy bemozdulása esetleg a kép átvitele közben keletkezett zavar (pl. ha a kép a világűrből jön). Feladatunk ilyenkor az, hogy a hibás képből (valamilyen függvény segítségével) előállítsuk az eredeti vagy az eredetihez leginkább hasonlító képet.

A kép javítása Fourier transzformáció segítségével

A torzított képből a javított kép meghatározása bonyolult, nehézkes. Ilyenkor vesszük hasznát a Fourier transzformációnak, hiszen a hibás kép Fourier transzformáltja az eredeti kép és a torzító függvény transzformáltjainak szorzata. Tehát bonyolult függvénykapcsolatok helyett, csupán a transzformált függvények szorzatáról van szó. $F(u,v)=E(u,v)\cdot H(u,v)$, ahol E az eredeti kép, H a hiba, F pedig a torzult kép Fourier transzformáltja.

A Fourier transzformálttal a képet frekvenciákra bontva adjuk meg, és így a kép területeit nem tudjuk megkülönböztetni. Ezért ez a módszer akkor alkalmazható, ha feltehetjük, hogy a kép minden része ugyanúgy torzult. Ez az eset áll fenn, például amikor a kép homályos rossz fókuszálás vagy bemozdulás miatt.

Tehát a kép javításához a következő lépéseket kell megtennünk:

- Vegyük a torzult kép Fourier transzformáltját.
- Vegyük a hibát leíró függvény Fourier transzformáltját.
- Képezzük a $\frac{torzultkép\ FT-ja}{hibafüggvény\ FT-ja}$ hányadost.
- A hányadosra alkalmazzuk az inverz Fourier transzformáltat, és így megkapjuk a javított képet.

Zajos átvitel esetén fellépő hibák javítása

Egyszerű a feladatunk, ha pontosan ismerjük a torzulást okozó függvényt, ám a legtöbb esetben először a hiba mikéntjét kell megbecsülnünk.

Ha a zajos átviteli csatorna okozza a hibát a képeknél, akkor ez úgy jelentkezik, hogy a képpontok fényerőssége kis területen belül gyorsan változik, néhol intenzív, majd mellette alacsony, és fordítva. Vagyis a torzulás a kép Fourier transzformáltjának magas frekvenciájú komponenseiben jelenik meg. Így ha kivesszük a magas frekvenciájú komponenseket, akkor ezzel a hibát csökkentjük.

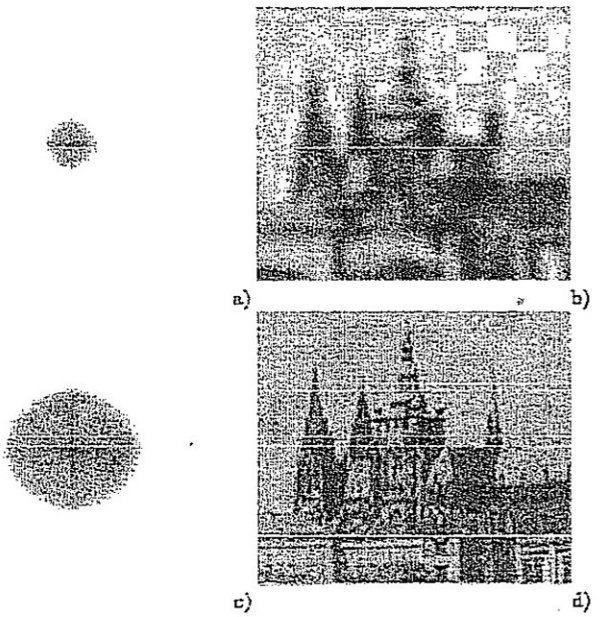
Tehát a kép javított változatának transzformáltját kapjuk, ha a torzult kép transzformáltját megszorozzuk egy szűrőfüggvénnyel, amely nem engedi át a magas frekvenciájú komponenseket. $J(u,v)=F(u,v)*Sz(u,v)$.

Ideális alul áteresztő szűrő

Az ideális alul áteresztő szűrő csak az alacsony frekvenciájú komponenseket engedi át, sem. Ennek függvénye: $Sz(u,v)=\begin{cases}1, & ha\sqrt{u^2+v^2}\le\omega\\ 0, & különben\end{cases}$, ahol ω a vágófrekvencia. Az ω sugáron belül minden frekvencia megmarad, és minden egyéb eltűnik.

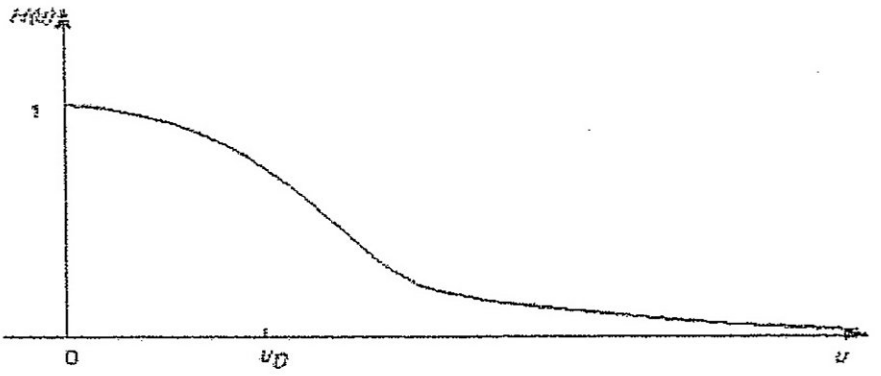
Ennek a szűrőnek a hibája a képen található éles határvonalaknál mutatkozik meg. Ilyen helyeken a kép hirtelen változik világosból sötétbe vagy fordítva, és ezért itt ugyanúgy a magas frekvenciájú komponensek dominálnak, mint a torzulás, amit enyhíteni szeretnénk. A módszer minden magas frekvenciát kiszűr, tehát ez az eljárás az éles határokat elmossa. Minél alacsonyabb a vágófrekvencia, annál kevesebb információt enged át a szűrő, annál inkább homályos a kép.

Ezt mutatják a fenti képek. Az eredeti kép és Fourier transzformáltja a Matematikai alapok című részben található. A transzformált inverzével visszaalakítottuk a képeket, de előtte a magas frekvenciákat kiszűrtük Az a) és c) ábra mutatja, hogy mekkora részt engedtünk át a transzformált kép frekvenciáiból, vagyis hogy mekkora volt a vágófrekvencia. A felső képen a vágófrekvencia alacsonyabb, kevesebb információ marad, ezért a kép homályosabb.



Az n-ed rendű alul áteresztő szűrők

Az ideális alul áteresztő szűrő hibáját enyhítik az n-ed rendű alul áteresztő (Butterwoth) szűrők, amelyeket szintén gyakran használnak. Ezeknél $Sz(u,v)=\dfrac{1}{1+\left[(u^2+v^2)/\omega^2\right]^n}$, ahol n-t nevezzük a szűrő rendjének. Ezek n-től függően meghagyják a magas frekvenciájú információk egy részét ezáltal csökkentve a homályosságot (ha n=∞, akkor az ideális alul áteresztő szűrőt kapjuk). Az ábra a másodrendű Butterworth szűrő egydimenziós változatát mutatja.



IV. Grafikus fájlok

1. Miért van sokféle képformátum?

A raszteres adatmodell nem más, mint „szabályos rácsba szervezett négyzetes képelemek (pixelek) értékeinek mátrixa".

Ez az egyszerű struktúra **négy ok** miatt eredményez számtalan formátumot:

- 1. milyen értékeket vehetnek fel a pixelek,
- 2. milyen tömörítési eljárást alkalmazunk,
- 3. a pixelek elrendezését (hány pixel van egy sorban és hány sor van a képben), illetve a kép általános adatait hogyan adjuk meg,
- 4. a formátumok különböznek a kódolás típusa szerint is;

Tekintsük át ezeket az okokat egyenként:

1) **Bináris képek** (és bináris kódolás) esetén minden pixelnek egy bitet kell fenntartanunk.
   - *tónusos fekete-fehér képek* esetén 256 szürkeségi fokozatot szoktak alkalmazni, ami 8 bittel írható le,
   - *színes képek* esetén számtalan variáció létezik a biztosítandó színgazdagságra: pld. 24 bit esetén 16 millió különféle színt lehet a pixelekhez rendelni,
   - a *multispektrális távérzékelt képeknél* gyakorlatilag az érzékelt csatornák számának megfelelő számú képet kell egyszerre rögzíteni, amit vagy úgy csinálnak, hogy minden pixelnél egymásután rögzítik a különböző csatornák intenzitás értékeit, azaz 4 csatorna és 256 intenzitás fokozat esetén minden pixelt 32 biten rögzítenek, vagy a sorokat fűzik egymásután vagy a spektrális képeket;
2) Korábbi órákon több tömörítési eljárás **elvével** ismerkedtünk meg. A tényleges tömörítést azonban ezek és más, eddig nem ismertetett, elvek alapján működő konkrét algoritmusok valósítják meg, következésképpen minden formátumhoz specifikált tömörítő algoritmus vagy algoritmusok tartoznak, ez utóbbi esetben az átvitt fájlban specifikálni kell a konkrét algoritmust is.
3) Egyszerű kép esetén általában a fájl fejléce tartalmazza:
   - az oszlopok és sorok számát,
   - a verzió számot,
   - a színek számát,
   - a tömörítési módszer kódját;

Az egyszerű kép számtalan képi formátum (GIF, TIFF, JPEG, BMP stb.) bármelyikében továbbítható, feltéve, hogy a pixelértékek kielégítik a kérdéses formátum specifikációját.

4) A kódolás vagy bináris vagy szöveges (ASCII) lehet, s ugyanaz a formátum gyakran mind a két kódolási formát ismeri.

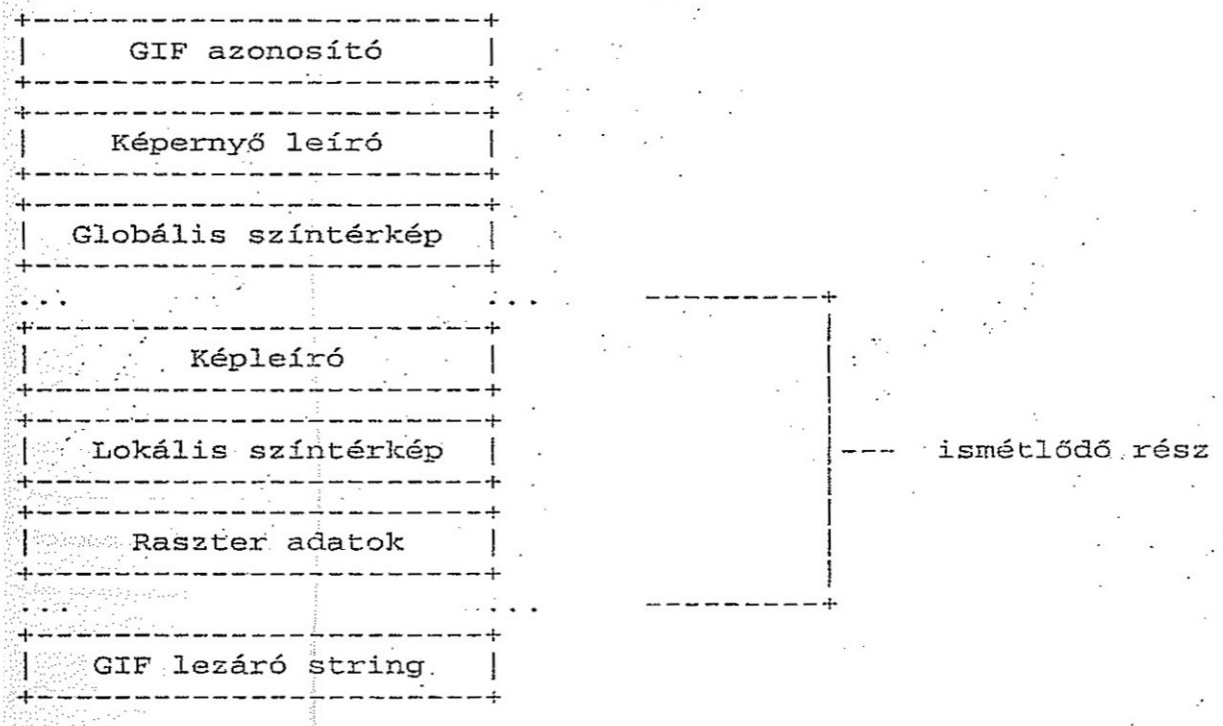
2. A GIF file-formátum (Graphics Interchange Format)

A GIF a CompuServe által bejegyeztetett és használt szabványos grafikai leíró formátum lehetővé teszi kiváló minőségű, nagyfelbontású képek megjelenítését az eljárás azzal a céllal került megtervezésre, hogy egységes alapját képezze minden, a CompuServe által kiadott jövőbeni grafikai terméknek

A GIF file formátum 8 bites (256 színű), vagy annál kisebb felbontású képekhez alkalmas.

A GIF adatszerkezete

A GIF fájlszerkezete a következő:

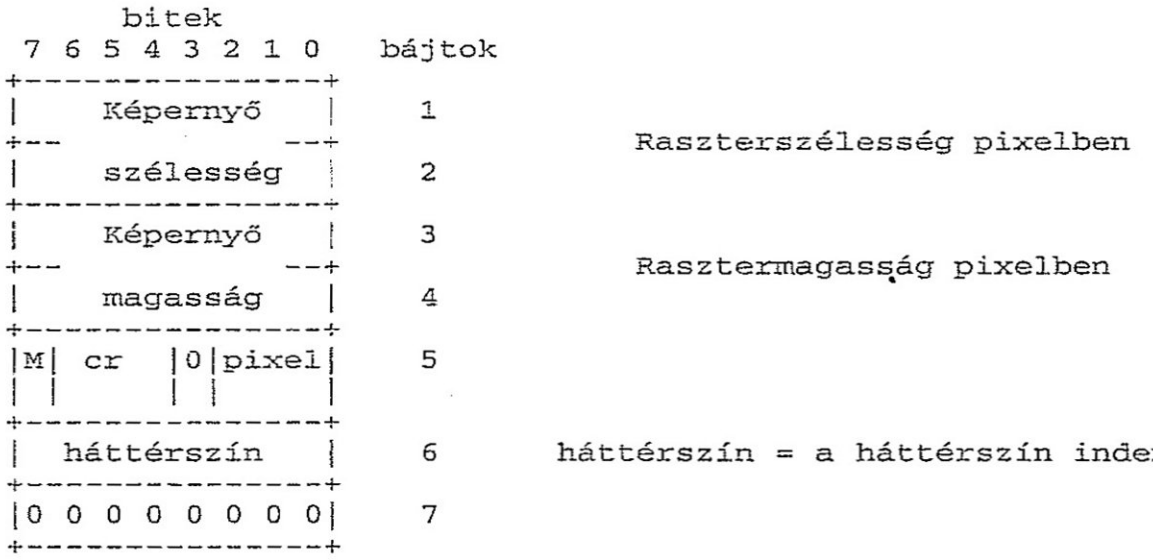


GIF azonosító

- a fájl elején található karaktersorozat
- jelzi, hogy az őt követő adatok szabványos GIF leíró struktúra elemei
- hat karakterből áll, például : G I F 8 7 a
- Az utolsó három betű a használt GIF definíció verziószáma, mely általában hivatkozásként használatos azon GIF formátumú képet tartalmazó dokumentumokban, ahol verziófüggőséggel is számolni kell

Képernyő leíró = Screen Descriptor

- az ábrázolás általános paramétereit adja meg
- definiálja:
  - a kép méretcit
  - a szükséges logikai képernyőméretet
  - a háttér színét
  - megadja, hogy létezik-e globális színtérkép
  - valamint, hogy hány biten ábrázolja a színeket a leírás ezen információk 7 bájton helyezkednek el, a következőképpen:




```
ha M = 1, a globális színtérkép követi a deszkriptort
cr+1 = színtárolásra használt bitek száma
pixel+1 = # bit/pixel (1 pixel leírására használt bitek száma)
```

A logikai képernyő szélessége és magassága egyaránt meghaladhatja a fizikai képernyő méreteit. Az ilyen képek kezelése általában implementáció-függő, sőt egyes esetekben a hardver előnyös jellemzőit is kihasználhatja (például a Macintosh scrollozható ablakai). Legegyszerűbb esetben a képet le lehet vágni a fizikai képernyő méretére.

Az előbbi "pixel" érték egyben meghatározza a képen használható színek maximális számát. A "pixel" argumentum értéke 0 és 7 közötti, ami tehát 1..8 bitet jelent. Eszerint a használható színek száma 2-től (fekete-fehér) 256-ig terjedhet. Az 5. bájt 3. bitje későbbi definíciókra van fenntartva, értéke mindig nulla.

Globális színtérkép = Global Color Map

- Használata opcionális, de ajánlott azon esetekben, amikor pontos színvisszaadás kívánatos.
- A színtérkép meglétét a deszkriptor 5. bájtjának "M" mezője jelzi.
- Egy GIF fájlban több kép is szerepelhet, s ezek mindegyikéhez külön színtérkép is tartozhat, normál esetben viszont ez a globális színtérkép kerül felhasználásra a

különböző platformokon fellépő hardverkorlátozások miatt. Az egyes képekhez tartozó külön leíró struktúrákban (Image Descriptors) az "M" flag általában nullára van állítva.

- Ha a GIF fájl tartalmaz globális színtérképet, annak definíciója közvetlenül a képernyő leíró után van.
- A képernyő leíró utáni színtérkép bejegyzések száma 2^n, ahol n a pixelenként használt bitek száma (bit/pixel); ezek 3 bájtos struktúrák, amelyekben a piros, zöld, kék komponensek intenzitása szerepel.
- A színtérkép blokkszerkezete:

```
        bitek
 7 6 5 4 3 2 1 0    bájtok
+-----------------+
|piros intenzitás |     1     Piros intenzitás a 0 indexű színben
+-----------------+
|zöld intenzitás  |     2     Zöld intenzitás a 0 indexű színben
+-----------------+
|kék intenzitás   |     3     Kék intenzitás a 0 indexű színben
+-----------------+
|piros intenzitás |     4     Piros intenzitás az 1 indexű színben
+-----------------+
|zöld intenzitás  |     5     Zöld intenzitás az 1 indexű színben
+-----------------+
|kék intenzitás   |     6     Kék intenzitás az 1 indexű színben
+-----------------+
:                 :           (Ugyanígy tovább az összes színre)
```

  - A kép minden egyes pontja azzal a színnel jelenik meg a képernyőn, amely a színtérképen legközelebb áll az adott pont eredeti színéhez.
  - A színkomponensek intenzitása 0 és 255 közötti érték; ahol a fehér színt például a (255, 255,255), a feketét pedig a (0,0,0) kombináció adja. Azokon képernyőkön, amelyek támogatják a komponensenkénti 8 bitnél kevesebb biten való színábrázolást, csak a felső bitek kerülnek felhasználásra. Ilyen hardver esetén a színtérkép létrehozásakor az egyes komponensek értékeit át kell konvertálni 8 bites formátumra a következőképpen:

    <színtérkép_érték> = <komponens_érték> *255 / (2**<bitek_száma> - 1)

  - Ez pontos színeltolást biztosít bármely megjelenítő eszköz számára. Azon esetekben, amikor GIF képet hozunk létre közvetlenül hardverről, és külön színpaletta készítésére nincs lehetőség, akkor egy fix palettát kell megadni, amely az adott hardverrel rendelkezésre álló képernyőszíneken alapszik. Ha globális színtérkép nincs kijelölve, akkor egy alapértelmezett színtérkép generálódik, amely leképez minden lehetséges bemenő színértéket (color index) az n féle hardver szín valamelyikére.

Kép Leíró = Image Descriptor

- Megadja a kép elhelyezkedését és kiterjedését a képernyőleíró által definiált területen belül.
- Tartalmaz két flag-et: egyet arra, hogy tartozik-e a képhez lokális színtérkép, egyet pedig a képpontok megjelenítési sorrendjének meghatározására.

- Minden képleíró egy szeparátor karakterrel kezdődik - Ez az elválasztó jel akkor játszik szerepet, amikor egy GIF fájl több képet is tartalmaz. Ez a jel mindig a "," (vessző, hexában: 0x2c), mely után közvetlenül a képleíró következik.
- Több kép esetén a képleíró struktúra vége és a következő szeparátor közti karaktereket a rendszer nem veszi figyelembe.
- A képleíró struktúra felépítése:

```
   bitek
 7 6 5 4 3 2 1 0   bájtok
+---------------+
|0 0 1 0 1 1 0 0|    1     "," - szeparátor karakter
+---------------+
|      Kép      |    2     A kép kezdete pixelben a képernyő
+--           --+          bal szélétől számítva
|   bal sarok   |    3
+---------------+
|      Kép      |    4     A kép kezdete pixelben a képernyő
+--           --+          felső szélétől számítva
|  felső sarok  |    5
+---------------+
|      Kép      |    6
+--           --+          A kép szélessége pixelben
|    szélesség  |    7
+---------------+
|      Kép      |    8
+--           --+          A kép magassága pixelben
|    magasság   |    9
+---------------+
| M I 0 0 0 pix |   10
+---------------+
```

```
M = 0 - a globális színtérkép érvényes, "pix"-et nem használja
M = 1 - van lokális színtérkép, "pix"-et használja
I = 0 - a képpontok megjelenítése szekvenciális sorrendű
I = 1 - a képpontok megjelenítése "interlaced" sorrend
pix + 1 - # bit/pixel (1 pixel leírására használt bitek száma)
```

- A kép elhelyezkedésére és méretére vonatkozó értékeknek természetesen a képernyő leíróban megadott méreteken belül kell lenniük, nem szükséges viszont, hogy a kép teljesen kihasználja ezeket a méreteket.

Lokális színtérkép

- Megadása opcionális, tulajdonképpen jövőbeni használatra van definiálva.
- Ha a képleíró struktúra 10. bájtjának M bitje 1-re van állítva, akkor lokális színtérkép követi a képleírót, amely csak erre az egyetlen képre vonatkozik. A 10. bájt alsó három bitje (a "pix" mező) csak ebben az esetben - vagyis lokális színtérkép megléte esetén - használatos. Ez nemcsak a kép pixel méretét adja meg, hanem a következő színtérkép bejegyzések számát is.

Raszter adatok

- A kép tárolási formátuma az egyes képpontokhoz (pixel) tartozó színértékek (color index) sorozata.

- A kép egy sorának pontjai balról jobbra haladva következnek egymás után, a sorok pedig a kép tetejétől az alja felé haladva.
- Ha a képleíró 10. bájtjának "I" bitje 1-re van állítva (Interlace mód), akkor a kép sorai egy négy szakaszból álló struktúrával vannak leírva, ahol az egyes szakaszok egymástól távol eső sorokat tartalmaznak. Az első szakasz minden nyolcadik sort ír le a kép legfelső sorától kezdve. A második szakasz felülről az ötödik sortól kiindulva tartalmaz minden nyolcadikat. A harmadik szakasz minden negyedik sort foglal magába a harmadik sortól kezdve, s végül a negyedik szakasz fejezi be a kép kitöltését az összes hátra maradt sor leírásával, a kép második sorával indulva.
- A szerkezet grafikus leírása:

```
Sorok    1.szakasz     2.szakasz     3.szakasz     4.szakasz     Eredmény
-------------------------------------------------------------------------
  0      **1a**                                                   **1a**
  1                                                 **4a**       **4a**
  2                                  **3a**                      **3a**
  3                                                 **4b**       **4b**
  4                    **2a**                                     **2a**
  5                                                 **4c**       **4c**
  6                                  **3b**                      **3b**
  7                                                 **4d**       **4d**
  8      **1b**                                                   **1b**
  9                                                 **4e**       **4e**
 10                                  **3c**                      **3c**
 11                                                 **4f**       **4f**
 12                    **2b**                                     **2b**
  :
```

- Az egyes képpontokhoz tartozó értékek színindexek sorozata, melyek a színtérkép egy-egy színére képeződnek le, s végül ez a szín jelenik meg a képernyőn. Ez a sorozat kerül tehát a GIF adatszerkezet végére, majd az egész struktúra tömörítésre kerül, mégpedig úgynevezett LZW tömörítési algoritmus szerint.

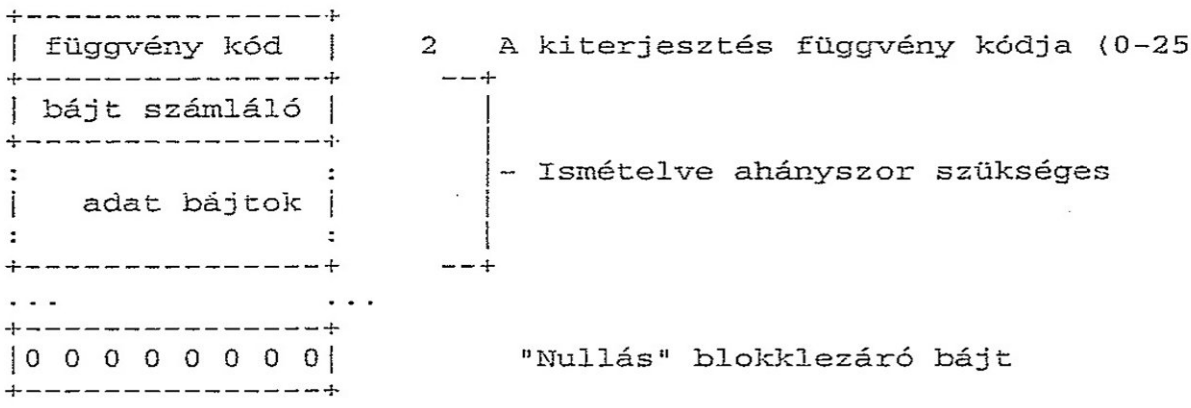
GIF lezáró string = GIF terminator

- A GIF fájlok olvasás közbeni szinkronizációjának biztosítása végett a GIF fordító abbahagyja a fájl dekódolását, ha egy kép után ";" karaktert (0x3b hexában) talál. A szabályok szerint a GIF fordító ilyenkor leáll, és valamilyen jelre vár, hogy a felhasználó készen áll-e a folytatásra. Ez a jel lehet egy ENTER a billentyűzeten vagy egy klikkelés az egérrel.

GIF kiterjesztés blokkok = GIF extension blokks

- Hogy a korábbi verziójú GIF fordítók képesek legyenek a későbbiek által létrehozott képek dekódolására, a fordítónak meg kell adni a GIF fájlon belüli kiterjesztések csomagolási mechanizmusát. Annak érdekében, hogy a fejlesztési irányok ellenőrzöttek és átjárható legyenek, a CompuServe-nek definiálnia és dokumentálnia kell speciális GIF kiterjesztéseket. Erre szolgálnak a kiterjesztés blokkok, melyek ugyanúgy vannak becsomagolva, mint a raszter adatok, viszont nincsenek betömörítve.
- Egy blokk szerkezete:

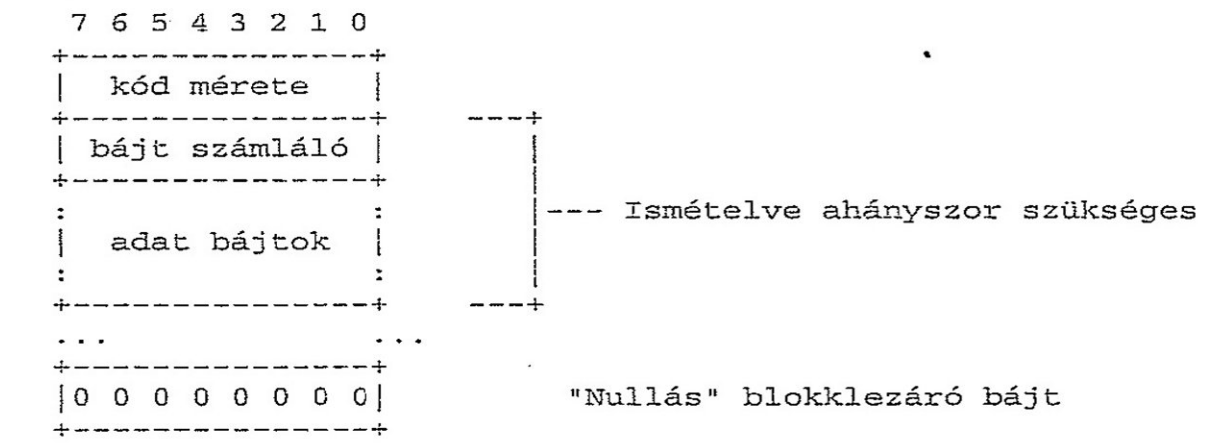
```
 7 6 5 4 3 2 1 0   bájtok
+---------------+
|0 0 1 0 0 0 0 1|    1   "!" - GIF kiterjesztés blokk elejét jelző karakter
```



A kiterjesztés blokk állhat közvetlenül a képleíró előtt, vagy a GIF lezáró string előtt. Minden GIF fordítónak fel kell ismernie a kiterjesztés blokk létezését, és ha a dekódolás sikertelen, el kell azt olvasnia.

Képek csomagolása és tömörítése

- A képet leíró raszter adatok a következő szerkezetűek:



A több lépésből álló konverzió során a fordító a képpontok színindexeinek sorozatát karaktersorozattá alakítja. Ezek a lépések a következők:

1) A kód méretének megállapítása - meghatározza az aktuális adatok leírásához szükséges bitek számát.

2) Tömörítés - a képpontok sorozatát tömörítési kódsorozattá alakítja.

3) Bájtsorozatok felépítése - A kompressziós kódok egy adott részéből stringet - 8 bites bájt sorozatot - állít elő.

4) Csomagolás - A bájtokból blokkokat hoz létre, a blokkok elejére karakterszámlálót helyez.

1) A kódméret meghatározása - A raszter adatok folyamának első bájtja az adatfolyamban lévő képpontok értékeinek leírásához minimálisan szükséges bitek számát adja meg. Általános esetben ez az érték ugyanaz, mint a színek megadására használt bitek száma. Bizonyos algoritmikus korlátok miatt azonban a fekete-fehér képek esetén, ahol egy színleíró bit van a minimális kódméretet 2 bitben kell megjelölni. Ennek eredményeként a tömörítési kód is egy bittel hosszabb lesz.

2) Tömörítés - Az LZW algoritmus egy adatsorozatot olyan kódsorozattá alakít, amely tartalmazhatja magukat a "nyers" adatokat, vagy előre meghatározott értékek sorozatát. Szöveges karaktereket például véve, a kimenő kód állhat karakterekből vagy olyan kódértékekből, amelyek egy-egy karakterre vagy szövegre utalnak. A GIF fájlokban használt LZW algoritmus néhány tekintetben eltér a szabványostól:

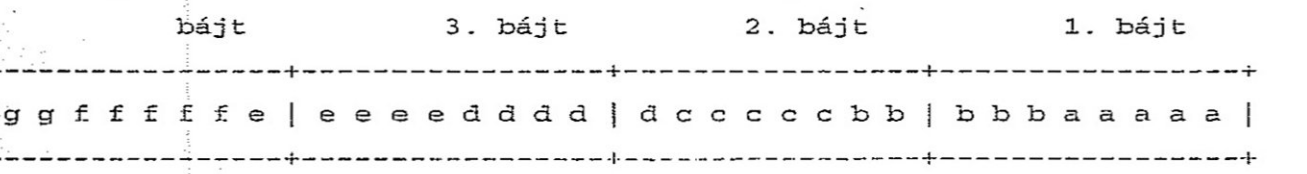
   a) Definiálva van egy speciális "Clear" kód, mely visszaállítja az összes kompressziós/dekompressziós paramétert és táblát a kiindulási értékére. Ennek a kódnak az értéke $2^{<\text{kódméret}>}$. Ha például a kódméret 4, a Clear kód értéke $2^4 = 16$. A Clear kód a képet leíró adatfolyamban bárhol előfordulhat, ezért ahhoz, hogy a kód végrehajtódjék, az LZW algoritmusnak azt úgy kell értelmeznie, mintha új adatfolyam kezdődött volna. Kódoláskor az adatfolyamnak egy Clear kód kell legyen az első eleme.

   b) Definiálva van egy "End of Information" kód, amely explicite jelzi az adatfolyam végét. Az LZW dekódolás leáll, ha ilyen kódot talál, ezért egy kép adatfolyamában ennek lennie az utolsó kódnak. Ennek értéke <Clear kód> + 1.

   c) Az első lehetséges tömörítési kód értéke: <Clear kód> + 2.

   d) A kimenő kódok hossza változó lehet <kódméret>+1bit/kód-tól 12 bit/kód-ig. Ennek értelmében a maximális kódérték 4095 (hexában FFFF). Amikor az LZW kód értéke meghaladja az aktuális kódhosszat, a kódhossz értéke eggyel megnő. Az ilyen kódok ki- és becsomagolása az új kódhossznak megfelelően módosul.

3) 8 bites bájtok felépítése - Mivel a keletkező LZW kódok hossza 3 és 12 bit között váltakozó lehet, ezeket újabb átalakítással 8 bites bájtokká kell szervezni. A kódokból ezért egy összefüggő bitfolyamot képeznek jobbról balra haladva, majd ebből egyszerűen 8 bites szeleteket képeznek. Tegyük fel, hogy egy 5 bites LZW kódot kell feldolgozni 8 bites karaktersorozattá. Ekkor az eljárás a következő:



4) Csomagolás (Package) - Az elkészült bájtokat maximálisan 255 elemű blokkokká szervezik, a blokkok elejére pedig egy karakterszámláló bájt kerül. Egy nulla számlálójú blokk zárja az adott kép raszter adatfolyamát.

3. A TIFF file-formátum (Tag Image File Format)

A TIFF címke alapú fájlformátum, melyet kifejezetten digitális képek hatékony adatcseréjének támogatására fejlesztettek ki az Aldus cég a nyolcvanas évek végén, majd miután a cég egyesült az ADOBE céggel a formátum fejlesztését az ADOBE keretein belül folytatta.

A TIFF:

- mezői elsősorban az elektronikus kiadványszerkesztés és a hozzá kapcsolódó alkalmazások számára definiáltak;

- célja, hogy a meglévő digitális képszerkesztés terén már meglévő gyakorlati technikákat egybegyűjtse és kodifikálja; azon okból, hogy a jövőbeni bővítések minél kisebb problémát okozzanak, a formátumot úgy alakították ki, hogy az egyszerűen bővíthető legyen;
- nem nyomtató vagy lapleíró nyelv, dokumentum leíró szabvány; tervezésénél az elsődleges cél egy olyan környezet létrehozása volt, amelyben a képleíró adatok egyes applikációk közötti cseréje jól megvalósítható;
- nem támogatja az objektum-orientált grafikát és szöveget, csakis digitális képek leírására alkalmas;
- az őt körülvevő operációs rendszerről csupán annyit tételez fel, hogy az támogat valamilyen "fájlszerű" adatszerkezetet, amely 8 bites bájtok sorozataként áll elő; a megengedett legnagyobb TIFF fájl mérete 2^{32} bájt.
- Nagyon sok applikációs program, amely képes TIFF formátum olvasására, nem támogatja az abban definiált tulajdonságok mindegyikét. Egyes rendszerek az alapfunkcióknál alig valamivel többet támogatnak. Éppen ezért a következőkben leírtak egyes szoftverekre csak korlátozott mértékben feleltethetők meg.

Szerkezete, struktúrája

Egy TIFF fájl három fő részre osztható. Az első a fejrész (*file header*), a második a fájlban szereplő mezők jegyzéke, majd ezt követik maguk a mezők.

A TIFF fájl fejlécének szerkezete:

Fejléc
bájt sorrend
Verziószám
0. IFD
Offszetje

A fejléc a következő információkat tartalmazza:

0-1 bájt: Az első gépi szó a bájtok tárolási sorrendjét adja meg mind a 16 bites mind a 32 bites egész számok esetére.

Ha ez az érték 'II' [(hex 4949) Intel], akkor a bájtok helyértéke a tárolásban jobbról balra csökken, 'MM' [hex 4D4D) Motorola] esetén pedig a megszokott módon balról jobbra.

2-3 bájt: A második gépi szó a verziószám, amely nem változhat. A jelenlegi szám 42 (2A hex).

4-7 bájt: Az első képfájl jegyzék (Image File Directory - IFD) eltolódását (offszetjét) adja meg bájtban a fájl elejéhez képest. A jegyzék ugyanis bárhol lehet a fájlon belül a fejléc után, de mindig gépi szó határán kell kezdődnie (azaz az eltolódásnak a gépi szó egész számszorosának kell lennie).

A fájlban több IFD is lehet, és fájlon belüli elhelyezkedésük nem kötött. Az eltolási érték tulajdonképpen az IFD első bájtjának sorszáma. Ez mindig szóha-

tárra esik azaz páros szám. Legalább egy IFD kell a TIFF fájlba és minden IFD-nek legalább egy eleme (**Entry**) kell, hogy legyen.

Képfájl jegyzék (Image File Directory)

IFD Bejegyzés számláló
2 0. bejegyzés
14 1. bejegyzés
26 2. bejegyzés
38
2+ következő IFD
n*12 eltolása

Az IFD elején egy 2 bájtos számláló áll, amely a bejegyzések számát tartalmazza. Ezt követik az egyenként 12 bájtos bejegyzések, majd végül a következő IFD eltolódása.

A bejegyzések (azaz IFD egy elemének) szerkezete látható a következő ábrán:

A 0-1 bájt tartalmazza a mező címkéjét, a 2-3 bájt adja meg a hossz típusát, a 4-7 bájt a mező hosszát. A 8-11 bájtok a mező értékének eltolódását adják meg, azaz a címke értéke első bájtjának sorszáma, mely mindig páros szám illetve, ha az érték 4 bájton elfér, akkor magát az értéket. Értelemszerűen az értékeknek is gépi szó határán kell kezdődniük.

Az egyes bejegyzések címke szerinti növekvő sorrendben következnek, az értékek, amikre a bejegyzések mutatnak, viszont tetszőleges sorrendben lehetnek.

A "Hossz" tag értéke mindig az adott adattípus egységében van megadva. A valós, bájtban mért hossz tehát sizeof(Típus)*Hossz..

A definiált hossz típusok a következők:

1 = BYTE 8 bites előjel nélküli egész (unsigned int)

2 = ASCII 8 bites bájtok sora ASCII karakterek számára.

Az utolsó bájt értéke 0.

3 = SHORT 16 bites előjel nélküli egész.

4 = LONG 32 bites előjel nélküli egész.

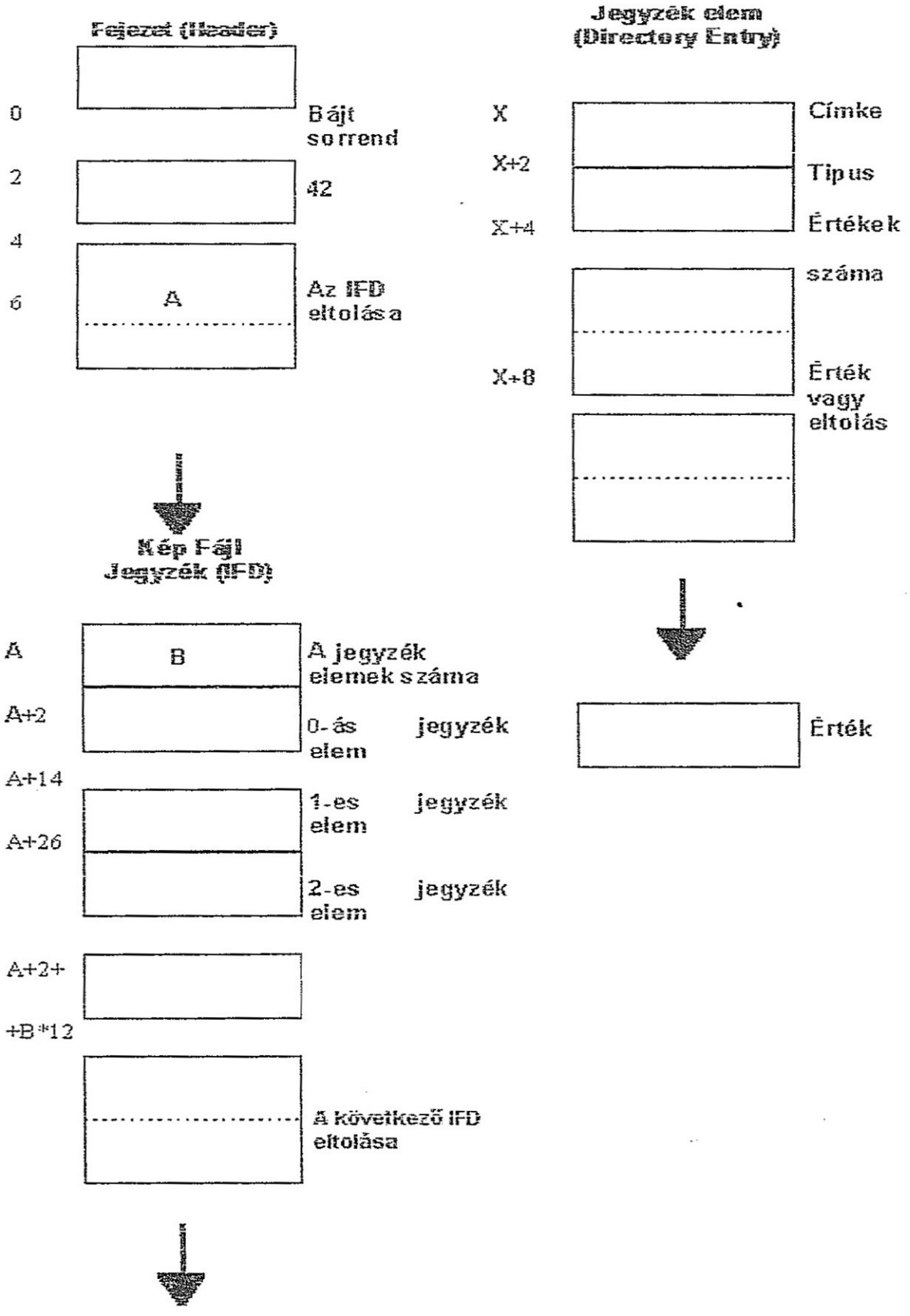
5 = RATIONAL Két LONG érték; az első a tört számlálója, a második a nevező.

12=DOUBLE 8 bájton ábrázolt kettős pontosságú valós szám

ASCII típusú mező esetén a Hossz értékében az utolsó nullás bájt is benne van.

Mezők

A mezők leírása tartalmazza a mező nevét, a címke értékét, a mező típusát, a mezőhöz rendelt adatelemek (Value) számát, egy rövid megjegyzést a mező leírására, és esetlegesen default értéket is a mező számára.



A TIFF fájlban az egyes mezők saját címkével azonosíthatók. Ez lehetővé teszi azt, hogy bizonyos mezők az applikációtól függően vagy benne vannak a fájlban vagy akár el is hagyhatók, vagyis lehetnek olyan TIFF fájlok, amelyek alig tartalmaznak néhány

mezőt, míg lehetnek olyanok, amelyekben viszont nagyon sokféle szerepel. A felvázolt struktúra alapján látható, hogy a TIFF igen rugalmas formátum, mivel a tárolt információ milyensége rugalmasan bővíthető címkék segítségével vezérelhető.

A TIFF esetében egy alapcímke halmazt minden író és olvasó programnak értelmeznie kell, e mellett, mind a specifikáció gondozói, mind a felhasználók kiegészíthetik a rendszert úgynevezett bővítésekkel, illetve privát címkékkel. A privát címkék 32768 sorszámmal kezdődhetnek.

Hogy valamilyen elképzelésünk is legyen a címkék milyenségéről, bemutatunk néhányat az alapcímkék közül. Az eddigi TIFF verziókban a következő mezők kerültek definiálásra:

Általános leírás (general description)

```
** Subfile Type (Fájlrész) **
        Címke           = 255 (FF)
        Típus           = SHORT
        N               = 1
```

Általános leírás az adott fájlrészben tárolt adatról. Az eddig definiált értékek:

1 = teljes felbontású kép adatok - *ImageWidth*, *ImageLength*, *StripOffset* a szükséges mezők;

2 = csökkentett felbontású kép adatok - *ImageWidth*, *ImageLength*, *StripOffset* a szükséges mezők;

3 = Egy több oldalból álló kép egy oldala.

Defaultérték nincs.

```
** ImageWidth  (Képszélesség) **
        Címke           = 256 (100)
        Típus           = SHORT
        N               = 1
```

A kép szélessége pixelben. Defaultérték nincs.

```
** ImageLength (Képhosszúság) **
        Címke           = 257 (101)
        Típus           = SHORT
        N               = 1
```

A kép magassága pixelben. Defaultérték nincs.

```
** RowsperStrip (Sor/Szalag) **
        Címke           = 278 (116)
        Típus           = SHORT or LONG
        N               = 1
```

Egy szalagban lévő sorok száma. A gyorsabb elérés érdekében a képsorok ún. szalagokba vannak rendezve a tömörített adatállományban.

Defaultérték: 2**32-1, ami gyakorlatilag végtelen nagy. Ez azt jelenti, hogy ekkor a kép összes sora egyetlen szalagba tartozik.

```
[StripsperImage (Szalag/Kép)]
        N                = 1
```

Az egy képben lévő szalagok száma. Ez nem igazi mező, mivel értékét közvetve számíthatjuk két másik mezőből:

```
StripsperImage = (ImageLength + RowsperStrip - 1) /
RowsperStrip
```

```
** StripOffset (SzalagOffszet) **
    Címke            = 273 (111)
    Típus            = SHORT or LONG
    N                = StripsperImage, ha a PlanarConfiguration mező
                             és
                     = SamplesperPixel * StripsperImage, ha a
                        PlanarConfiguration mező 2
```

Az adott szalag bájt-offszetjét adja meg a fájl elejétől számítva. Enélkül a kép adatait nem találja meg olvasás közben a konverter. Defaultérték nincs.

```
** StripByteCounts (SzalagBájtSzámláló) **
    Címke            = 279 (117)
    Típus            = LONG
    N                = StripsperImage, ha a PlanarConfiguration mező
                             és
                     = SamplesperPixel * StripsperImage, ha a
                        PlanarConfiguration mező 2
```

Az adott szalagban lévő bájtok száma. Defaultérték nincs.

```
** SamplesperPixel (Adat/Képpont) **
        Címke            = 277 (115)
        Típus            = SHORT
        N                = 1
```

Az egy képponthoz tartozó adatok száma. Monochrom leírás esetén 1, színes leírás esetén 3. Defaultérték: 1

```
** BitsperSample (Bit/Adat) **
        Címke            = 258 (102)
        Típus            = SHORT
        N                = SamplesperPixel
```

Az adatelemet leíró bitek száma. Defaultérték: 1

```
** PlanarConfiguration (SíkKonfiguráció) **
        Címke            = 284 (11C)
        Típus            = SHORT
        N                = 1
```

1 = a képpontokhoz tartozó adatok összefüggően vannak tárolva, vagyis egyetlen képsík van.

2 = Az egyes adatelemek külön adatsíkon vannak tárolva. A *StripOffset* és *StripByteCounts* mezők értékei kétdimenziós tömbbe rendezettek. A tömb *SamplesperPixel* számú sorból és *StripsperImage* számú oszlopból áll, feltöltése soronként történik.

Defaultérték nincs.

```
** Compression (Tömörítés) **
   Címke          = 259 (103)
   Típus          = SHORT
   N              = SamplesperPixel, ha a PlanarConfiguration
                    mező értéke 1 vagy 2.
```

1 = Nincs tömörítés, de olyan szorosan csomagolja össze az adatokat, hogy ne legyenek használatlan bitek egyetlen bájtban sem, kivéve a sorvégeket.

2 = CCITT Group 3, 1-dimenziós módosított Huffman kódolással tömörít. A *BitsperSample* értéke 1 kell legyen.

3 = Fax-kompatibilis CCITT Group 3 tömörítés, szabványos leírását lásd Standardization of Group 3 facsimile apparatus for document transmission, Recommendation T. 4, Volume VII, Fascicle VII.3, Terminal Equipment and Protocols for Telematic Services, The International Telegraph and Telephone Consultative Committee (CCITT), p. 16-31.

4 = Fax-kompatibilis CCITT Group 3 tömörítés, szabványos leírását lásd Standardization of Group 3 facsimile apparatus for document transmission, Recommendation T. 4, Volume VII, Fascicle VII.3, Terminal Equipment and Protocols for Telematic Services, The International Telegraph and Telephone Consultative Committee (CCITT), p. 40-48.

Az adattömörítés csak a képpont adatokra vonatkozik, az egyéb TIFF információkra nincs befolyással. Defaultérték: 1

```
** Group3Options **
      Címke          = 292 (124)
      Típus          = LONG
      N              = 1
```

Ez a mező 32 db flagként használt bitből áll. A használaton kívüli bitek értéke 0. A 0. bit értéke 1, ha 2-dimenziós a kódolás. Ellenkező esetben 1-dimenziós. Az 1. bit értéke 1, ha tömörítés nélküli a kódolás. A 2. bit értéke 1, ha plusz bitek lettek hozzáadva a kódhoz annak érdekében, hogy az EOL kódok vége mindig bájthatárra essen. Defaultérték: 0, az általános 1-dimenziós kódolás miatt.

```
** FillOrder **
      Címke          = 266 (10A)
      Típus          = SHORT
      N              = 1
```

Az adatelemek sorrendje a bájton belül.

1 = először a magas helyiértékű bitek kerülnek feltöltésre.

2 = először az alacsony helyiértékű bitek kerülnek feltöltésre.

Defaultérték: 1

```
** Thresholding (Küszöbérték) **
      Címke          = 263 (107)
```

```
        Típus           = SHORT
        N               = 1
```

1 = "line art" scan mód, BitsperSample = 1

2 = "halftone" vagy "dithered" scan mód, általában folyamatos tónusú képek, például fotók esetén. BitsperSample = 1

3 = Hibajelzés.

Defaultérték: 1

```
** CellWidth (Cella Szélesség) **
        Címke           = 264 (108)
        Típus           = SHORT
        N               = 1
```

1 bites adatelemek (samples) esetén a dithering/halftoning mátrix szélessége. Értéke csak akkor használt, ha a Thresholding = 2. Defaultérték nincs.

```
** CellLength (CellaHossz) **
        Címke           = 265 (109)
        Típus           = SHORT
        N               = 1
```

1 bites adatelemek (samples) esetén a dithering/halftoning mátrix hossza. Értéke csak akkor használt, ha a Thresholding = 2. Defaultérték nincs.

Fényerősségi adatok

```
** MinSampleValue **
        Címke           = 280 (118)
        Típus           = SHORT
        N               = SamplesperPixel
```

Az adatelem minimális értéke. Defaultérték: 0.

```
** MaxSampleValue **
        Címke           = 281 (119)
        Típus           = SHORT
        N               = SamplesperPixel
```

Az adatelem maximális értéke. Defaultérték: 2**(BitsperSample) - 1.

```
** PhotoMetricInterpretation **
        Címke           = 262 (106)
        Típus           = SHORT
        N               = 1
```

0 = MinSampleValue értéke felel meg a fehér színnek, MaxSampleValue értéke a feketének.

1 = MinSampleValue értéke felel meg a fekete színnek, MaxSampleValue értéke a fehérnek.

2 = RGB modell, minden szín a három alapszín (piros, zöld, kék) keverékeként van megadva.

MinSampleValue ekkor a minimális, MaxSampleValue a maximális intenzitásnak felel meg. Ha PlanarConfiguration = 1, a színadatok hármas tömbök formájában vannak tárolva, ha PlanarConfiguration=2, a piros, zöld, kék értékek 3 külön síkon tárolódnak.

```
** GreyResponseUnit **
        Címke           = 290 (122)
        Típus           = SHORT
        N               = 1
```

1 = az adott szám tízes nagyságrendet reprezentál

2 = az adott szám százas nagyságrendet reprezentál

3 = az adott szám ezres nagyságrendet reprezentál

4 = az adott szám tízezres nagyságrendet reprezentál

5 = az adott szám százezres nagyságrendet reprezentál

Defaultérték: 2.

```
** GreyResponseCurve **
        Címke           = 291 (123)
        Típus           = SHORT
        N               = 2**BitsperSample
```

A GreyResponseUnit és a GreyResponseCurve mezők célja kiegészítő fotometriai információk szolgáltatása szürke tónusú (gray scale) képekhez. A szürke átviteli jellegörbe a szürke árnyalatok erősségi fokát adja meg a minimális és maximális értékek között, az optikai sűrűség alapján. A GreyResponseUnit mező a görbe értékek pontosságát adja meg.

```
** ColorResponseUnit **
        Címke           = 290 (122)
        Típus           = SHORT
        N               = 1
```

1 = az adott szám tízes nagyságrendet reprezentál

2 = az adott szám százas nagyságrendet reprezentál

3 = az adott szám ezres nagyságrendet reprezentál

4 = az adott szám tízezres nagyságrendet reprezentál

5 = az adott szám százezres nagyságrendet reprezentál

Defaultérték: 2.

```
** ColorResponseCurve **
        Címke           = 291 (123)
        Típus           = SHORT
        N               = 2**BitsperSample (Red)
                          + 2**BitsperSample (Green)
                          + 2**BitsperSample (Blue)
```

A három színintenzitás átviteli jelleggörbe az alapszínekre. A ColorResponseUnit mező a görbe értékek pontosságát adja meg.

```
** XResolution **
        Címke           = 282 (11A)
        Típus           = RATIONAL
        N               = 1
```

Egy ResolutionUnit-ban (lásd később) az X irányba eső képpontok száma. Természetesen a kép nem csak ebben a méretben jeleníthető meg, az alkalmazástól függ, hogy használja-e ezt a javasolt paramétert. Defaultérték nincs.

```
** YResolution **
        Címke           = 283 (11B)
        Típus           = RATIONAL
        N               = 1
```

Egy ResolutionUnit-ban (lásd később) az Y irányba eső képpontok száma. Defaultérték nincs.

```
** ResolutionUnit (Felbontási mértékegység) **
        Címke           = 296 (128)
        Típus           = SHORT
        N               = 1
```

1 = Nincs mértékegység. Akkor használatos, ha a kép nem téglalap alakú és nincs kitüntetett irány a méret meghatározáshoz

2 = inch a mértékegység.

3 = centiméter a mértékegység.

Defaultérték: 2.

```
** Orientation (Irányítottság) **
        Címke           = 274 (112)
        Típus           = SHORT
        N               = 1
```

1 = A kép 0. sora a legfelső, a 0. oszlop a bal szélső.

2 = A kép 0. sora a legfelső, a 0. oszlop a jobb szélső.

3 = A kép 0. sora a legalsó, a 0. oszlop a bal szélső.

4 = A kép 0. sora a legalsó, a 0. oszlop a jobb szélső.

5 = A kép 0. sora a bal szélső, a 0. oszlop a legfelső.

6 = A kép 0. sora a jobb szélső, a 0. oszlop a legfelső.

7 = A kép 0. sora a jobb szélső, a 0. oszlop a legalsó.

8 = A kép 0. sora a bal szélső, a 0. oszlop a legalsó.

Defaultérték: 1.

4. A JPEG file-formátum (Joint Photographic Experts Group)

A JPEG a szabványos veszteséges tömörítési eljárás. Nevét arról a bizottságról kapta, amely elsőként fogadta el ezt a szabványt.

A veszteséges tömörítési eljárások kidolgozására a CD-ROM-ok tömeges megjelenése váltotta ki az igényt. Ahhoz ugyanis, hogy CD-ROM-on videó képeket lehessen tárolni, mintegy 180:1 arányú tömörítést kell elérni. Ennek az az oka, hogy egy tömörítetlen videó lejátszásához 27 Mbájt/s-os adatátviteli sebesség szükséges a CD-ROM meghajtók szabványos 150 Kbájt/s-os sebességével szemben. A veszteségmentes tömörítés esetében azonban a kompressziós arány alacsony, rendszerint 2:1-hez. A legtöbb tömörítő rendszer ezért veszteséggel dolgozik, azaz a tömörítő eljárásban az eredeti kép egy része elvész. Ennek ellenére az ilyen eljárásoknak van létjogosultsága, mivel a fejlesztők feltételezik, hogy a legtöbb felhasználó észre sem veszi a kisebb részletek elvesztését, különösen nem egy gyors film megtekintésekor.

A veszteséges rendszerek többnyire úgy működnek, hogy elhagyják vagy megváltoztatják, például a képfrekvenciát, a képméretet vagy a pixel színmélységét. Megőrzik viszont azokat az információkat (például a képélességet, kontrasztot, a színek és mozgások folyamatosságát), amelyek a kép szubjektív megítélését befolyásolják, tehát azokat a jellemzőket, amelyek egy hétköznapi TV-n vagy videón is vezérelhetők.

A legtöbb digitális kompressziós rendszer a redundáns adatokat is felismeri, egy képen belül éppúgy, mint az egymást követőkön. Ezeknek a tényezőknek a kihasználása nagyon nagy tömörítési arányhoz vezet, akár a kívánt 180:1-hez is.

A JPEG először pixelek blokkjaira bontja szét a képet, és matematikai ismétlődések halmazaként ábrázolja valamennyi blokk mintáit és színeit. Végezetül kiszűri az ismétlődéseket, hogy csökkentse a blokkokban lévő ismétlődéseket, és így a kép tárolásához szükséges helyet is.

A JPG algoritmus valahogy így működik: négyzetekre osztja a képet, majd keres hasonló, közel azonos területeket, melyeket egymással behelyettesíthet. A tömörítés minősége e négyzetek méretét (is) szabályozza. Minél nagyobbak ezek a négyzetek, annál "kockásabb", elmosottabb lesz a kép, de egyre kisebb helyet foglal. A tömörítés arányát százalékosan adhatod meg. Ez azonban nem azt jelenti, hogy pl. 70%-nál azonos méretűek lesznek a képek. Egy részletdús jelenetet (pl. fű, erdő, kavicsok) sokkal nehezebb tömöríteni, mint egy sok üres felületet tartalmazó (pl. egy ház oldala) képsort.

V. Virtual Reality Modelling Language (VRML)

1. A VRML története

1994 tavaszán Genfben az első World Wide Web Conference keretében, Tim Barners-Lee és Dave Raggett szervezésében vitatták meg a Virtuális Valóság (Virtual Reality) gondolatának megvalósítását a World Wide Web-hez illesztve. A megoldandó nyelv a Virtual Reality Markup Language nevet kapta, ez később Virtual Reality Modelling Laguage-re változott. Három fontos követelmény körvonalazódott a nyelvvel kapcsolatban:

- legyen platformfüggetlen,
- bővíthető,
- legyen használható alacsony sávszélességű kapcsolaton keresztül is.

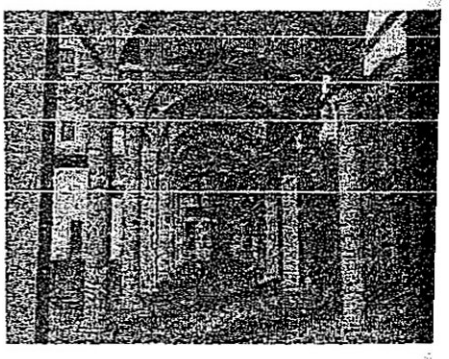
A fájl formátum választása a Silicon Graphics, Inc. Open Inventor ASCII File Format-ra esett. A VRML 1.0 1995 május 26-án jelent meg, ebből még hiányoztak az interaktivitást támogató elemek. 1996 augusztusában jelent meg a VRML 2.0, amely kibővült ezekkel.

2. A VRML felhasználási területei

A nyelv megjelenésével a virtuális világok ugrásszerűen gyarapodtak, hiszen a nyelv a felhasználók kezébe egy olyan eszközt ad, amit könnyű használni, és mindenki felépíthet vele egy saját világot, amely olyan, amilyennek ő akarja. A felhasználók egy része a munkájából merít ihletet, ebből adódóan a felhasználási területek rendkívül változatosak. Most néhány példát szeretnék bemutatni.

Építészet:

Talán ez a legszembetűnőbb az összes példa közül, hiszen a virtuális valóságról szinte mindenkinek házak, épületek is eszébe jutnak. Az interneten sok érdekes példát is lehet erre találni. Pl. Firenze egy része is rekonstruálva van (http://www.dada.it/marcoc/flowers/ címen láthatunk belőle részleteket), de pl. a metró háromdimenziós modelljét is könnyen el lehet készíteni...



Csillagászat:

Ez is jó táptalaj, hiszen a bolygókat (pl.) egyszerűen lehet ábrázolni, mérési adatokat könnyedén lehet vele feldolgozni.

Orvostudomány:

Itt a különböző szerveknek a felépítését lehet megfigyelni anélkül, hogy boncolni kellene.(Pl. szív)

Kémia:

Az atomok, ill. molekulák felépítését, kapcsolatát lehet könnyedén megrajzolni.

Matematika:

A matematika terén a legfőbb felhasználási terület a háromdimenziós függvények rajzolásában jelenti a nyelv. A különböző hasonló célra kifejlesztett programokkal szemben a VRML azt az élményt nyújtja, hogy mozoghatunk a „függvényben", és mindezek mellé, még könnyű is továbbítani.

Földrajz:

Egy adott területet adatainak alapján rajzolhatjuk meg pontos (tér)képet, és minden szemszögből megnézhetjük, annak ellenére, hogy arrafelé sem jártunk.

Kódolás:

Azon alapul, hogy minden karakterhez egy pont számhármasát rendeljük, majd minden ponthármasból egy háromszöget feszítünk. Mivel gyakorlatilag végtelen sok billentyű-koordináta pont pár van, egy lényegében megfejthetetlen kódot kapunk. Tehát az így kódolt szöveget csak az tudja visszafejteni, aki ismeri a kódot. (Cryptogram)

Egyéb:

Az Interneten találhatóak VRML áruházak, ahol akár az egész lakásunkat is berendezhetjük. Stb.

3. A VRML szerkezeti felépítése

A VRML fájl egy szöveges fájl, kiterjesztésként a .wrl-t használjuk/ják. A fájl mindig a fejléccel kezdődik. Ez az 1.0-s verzióban #VRML V1.0 ascii, míg a 2.0-s verzióban #VRML V2.0 utf8. Ezzel jelezzük, hogy a fájl VRML 1.0(2.0) specifikáció szerint épül fel, és ascii(UTF-8 ISO szabvány – nemzetközi karakterkészletek használatára) fájl.

A VRML fájlban node-ok vannak, ezek elemeket és azok jellemzőit írják le. Lényegében ezekből épül fel egy VRML fájl. A node-ok tartalmazzák a node típusát, kapcsos zárójel nyitja és zárja, a zárólelek között mezők szerepelhetnek, amelyek a node attribútumait állítják be, illetve módosítják. Ezeket a mezőket általában nem szükséges megadni, van alapértékük is.

A következőkben a VRML 1.0-ás verzióval fogok foglalkozni.

A node-ok a következőképpen csoporosíthatóak:

- geometriai formákat, és szöveget leíró node-ok
- tulajdonságok
  - transzformáció- vagy mátrixtulajdonságok
  - a geometriai formák és az ábrázolás tulajdonságai
- kamerák és megvilágítások tulajdonságai
- ún. csoportnode-ok
- és a WWWInline, ami igazából sehova sem sorolható.

A VRML fájlba megjegyzést is be lehet szúrni a # jel segítségével.

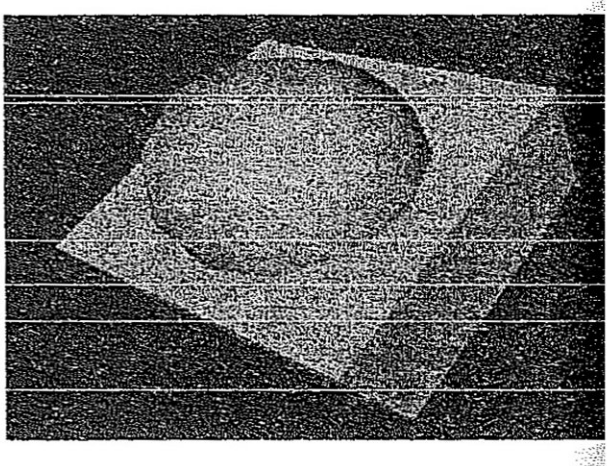
3.1. Geometriai formák és szövegek

A geometriai formákat leíró node-ok a következők: Cone (kúp), Cube (kocka), Cylinder (henger), IndexedFaceSet (lapot definiál), IndexedLineSet (élet definál), Pointset (pontokat lehet megadni vele), Sphere (gömb).

A szövegeket az AsciiText node segítségével írhatjuk ki. A szövegek formázása a FontStyle node-al lehetséges.

Cube – kocka

Ezzel a node-al téglatestet alkothatunk megadva a tulajdonságokat: szélesség (`width`), magasságot (`heigth`) és mélységet (`depth`).



Sphere – gömb

Megadott sugárral (`radius`) az aktuális rajzpontból mint középpontból rajzolja meg a gömböt

Cylinder – henger

Magasság (`height`), alapkör sugár (`radius`) és rajzolandó részek (`parts`) meghatározásaval rajzol hengert. A rajzolandó részek lehetnek: minden (ALL), felső lap (TOP), alsó lap (BOTTOM), palást (SIDES).

Cone – kúp

Magasság (`height`), alapkör sugara (`radius`), megrajzolt részek segítségével rajzolhatjuk. Megrajzolt részek lehetnek: minden rész (ALL), palást (SIDES), alaplap (BOTTOM).

Coordinate3 – pontok

Ezzel a node-al lehet pontokat megadni a háromdimenziós koordinátáik alapján. Egyetlen mezője van, ez a point. Ha több pontot adunk meg, akkor az elsőnek 0 lesz a sorszáma.

IndexedLineSet – élek megadása

Négy mezője van, a koordináta indexlista (`coordIndex`), anyag indexlista (`materialIndex`), normálvektor indexlista (`normalIndex`), textúra indexlista (`textureIndex`). A koordináta indexlistában a pontok indexeivel adjuk meg az éleket, ha több egymástól független élet szeretnénk rajzolni, akkor a –1 értékkel választhatjuk el a pontokat. A másik három indexlista sorrendben a következőkre szolgál: fényesség illetve szórt fény megadására, normálvektorok megadására (ennek a finomításokban van nagy szerepe), és a textúra megadására.

IndexedFaceSet – lapok mgadása

Ugyanaz a négy mezője van, mint az előbbinek, csak itt lapokat adunk meg a pontokkal.

PointSet – pontrajzoló

Két mezője van, a kezdőpont indexe (`startIndex`), és a rajzolandó pontok száma (`numPoints`). Ha a pontok számára –1 et írunk, akkor a kezdőindextől kezdve minden pontot kirajzol.

AsciiText – szövegek

Szöveget lehet kiíratni ezzel a node-al, négy mezője van. A `string` mezővel lehet kiírni a szöveget. A kiírandó szöveget " " jelek közé kell írni, ha több sorban akarunk kiírni, akkor vesszővel választjuk el az idézőjeleket. A másik három mező az igazítás

(justification), a szövegszélesség (width), és a sorköz mérete (spacing). Az igazítás történhet balra (LEFT), jobbra (RIGHT), és középre (CENTER).

FontStyle – szövegformázás

Ezzel a node-al lehet formázni a szövegeket. Három mezője van, a betű típusa (family), stílusa (style), és mérete (size). A size mező a betű méretét adja meg. A family mező beállításával a standard VRML fonttípusok közül választhatunk. Értékei lehetnek:

- SERIF – változtatható szélességű talpas betű (Times Roman);
- SANS – változtatható szélességű nem talpas betű (Helvetica);
- TYPEWRITER – állandó szélességű betű (Courier).

A style mező lehetséges értékei:

- NONE – normál szöveg;
- BOLD – félkövér formátum;
- ITALIC – dőlt stílus

3.2. Geometriai transzformációk

A geometriai transzformációk arra szolgálnak, hogy a rajzolás kezdetét változtathassuk. Ezek a node-ok a következők lehetnek: eltolás (Translation), elforgatás, (Rotation), léptékezés (Scale), illetve összevonhatjuk őket a Transform node-al, de használhatunk transzformációs mátrixot is (MatrixTransform).

Translation – eltolás

Egyetlen mezője van, jellemzően az is ugyanaz mint a node neve, csak kis betűvel kell írni (translation). Az eltolás értékeit az x, y, z tengelyre kell megadni.

Rotation – forgatás

Ennek is egy mezője van, és mint az előbb, itt is a node nevével egyezik meg (rotation). Négy értéket kell beírni, az első három értékkel állítjuk be, hogy melyik tengely körül forgatunk, a negyedik pedig a szöget mondja meg radiánban.

Scale – léptékezés

Egy mezője van: a ScaleFactor. Három értéket kell megadni, a három koordináta szerinti léptékezést (x, y, z szerinti sorrendben). Az egynél nagyobb szám nyújt, a kisebb csökkent.

Transform – transzformáció

Öt mezője van. Az első hármat (translation, rotation, scaleFactor), ugyanúgy kell megadni, mintha külön node-ok lennének. A negyedik a léptékezés irányát meghatározó forgatás (scaleOrientation). Ezzel lehetőségünk van tetszőle-

ges irányban megváltoztatni a léptékezést, nem csak az eddig látott X, Y, Z irányban. Az ötödik mező a középpont megadása (`center`). Ez egy vektort ad meg, melynek segítségével a léptékezést finomíthatjuk.

MatrixTransform – transzformációs mátrix

Egyetlen mezője van, a neve `matrix`. Ezután egy 4x4-es mátrixot kell megadni a geometriai ismereteinket használva.

3.3. Színek és formák

A rajzoló színt a Material node segítségével írjuk le. A színeket RGB formátumban adjuk meg, három lebegőpontos számmal (0.0 és 0.1 között). A színeket a MaterialBinding node-al tudjuk hozzárendelni a geometriai alakzatokhoz. A textúrák kezelését a Texture2, TextureCoordinate2, és a TextureTransform node-okkal tudjuk megvalósítani. Az anyag hozzárendeléshez szükségünk van az anyag képére.

Material – színek megadása

Ennek a node-nak hat mezője van: diffúzszínérték (`diffuseColor`), kibocsátott fényérték (`emissiveColor`), átlátszóság mértéke (`transparency`), környezeti színérték (`ambientColor`), tükröződszín mértéke (`specularcColor`), fényesség mértéke (`shininess`). A diffúzszínérték a testek diffúz (szórt) fényvisszaverési értékét állítja be. A VRML a fényforrások tükrözésére öszpontosít, mert ezt könnyebb számolni, ennek a színét állíthatjuk a `specularColor` mezővel. A környezeti színérték a környezet színét állítja be (este sötétebb van, stb.). A kibocsátott fényérték szabályozza, hogy milyen színt bocsát ki a test (a tükrözések nélkül). Az eddig felsorolt mezőkbe három számot kell írni az RGB kódnak megfelelően. A `transparency` és a `shininess` mezőkbe csak egyet. Az átlátszóság értéke 0.0 és 1.0 között változhat, az 1.0 a teljesen átlátszó. A `shininess` mező szabályozza a felület simaságát. Ha ez az érték 0.0, akkor a browserek nem veszik figyelembe a `specularColor` mező értékét. A Material node-ban több értéket is megadhatunk, felsorolásszerűen. Az egyes mezők nem szükséges ugyanannyi elemet tartalmazzanak, mert a sorszámozás ciklikusan újrakezdődik.

MaterialBinding – színhozzárendelés

Egyetlen mezője van a hozzárendelés típusa (`value`). Értékei lehetnek: DEFAULT, OVERALL, PER_VERTEX , PER_PART_INDEXED, PER_PART, PER_FACE_INDEXED, PER_VERTEX_INDEXED, PER_FACE. Az OVERALL típusa listák első elemének megfelelő értékeket használja a színezés során. A PER_PART, PER_FACE, PER_VERTEX típusok a listák elemeinek sorrendjében felelteti meg egymásnak a színértékeket és a test részeit (`part`), lapjait (`face`), csúcsait (`vertex` esetén). Az indexed jelzővel ellátott típusok kihasználják a sorszámozást, így egy színt többször is

használhatunk, méghozzá úgy hogy az IndexedFaceSet és IndexedLineSet materiallIndex mezőjében adjuk meg a sorszámot. A DEFAULT a VRML alapértelmezett hozzárendelését veszi figyelembe, ez általában az OVERALL típus szokott lenni.

TextureCoordinat2 – koordinátaindexek definiálása

Egy mezője van, a point, itt tudjuk felsorolni a textúraként használni kívánt képet határoló koordinátákat. Az eredeti képünk a 0.0 és 1.0 közötti X és Y koordinátájú pontok közöttvan, mi tetszőlegesen vághatunk ki belőle. Az eredeti kép határain kívülre is mehetünk, ekkor két módon tölti ki a hiányzó részt a browser: mozaikszerűen illeszti mégegyszer a képet, vagy a bal szélső pontokat húzza ki odáig.

Texture2Transform – anyagkép transzformáció

Az előbb meghatározott kitöltő anyagot transzformálhatjuk más formátumra. Négy mezője van: translation, rotation, scaleFactor, center. A transzformációk a már ismertetett módon működnek (eltolás, forgatás, léptékezés, középpont használatával).

Texture2

Ezzel a node-al lehet hozzárendelni a textúrákat a testekhez. Négy mezője van: filename, warpS, warpT, image. A filename mezőben adjuk meg a kívánt file nevet (url cím). A következő két mezőnek két értéke lehet: REPEAT, CLAMP. Az első esetben az anyag mintázata ismétlődik, a másodikban a kapcsolt mód szerint lesz kitöltve a ferület. A warpS mező az X irányú módot határozza meg, a warpT az Y tengely irányában határozza meg a kitöltés milyenségét. Az image mezőben bitenként írhatjuk le a képet, ha nem adtunk meg fájlt.

3.4. Fények és megvilágítások, kamerák

A VRML browser alaphelyzetben egy fényforrást használ a virtuális térben, az adott aktuális nézőpontból egyenesen előre irányuló sisaklámpát. Ezt a megvilágítást pont fényekkel (PointLight), irányított fényekkel (DirectionalLight), és reflektorfényekkel (SpotLight) bővíthetjük ki.

Kameratípusból kettő van, a perspektivikus kamera (PerspectiveCamera), és az ortografikus kamera (OrtographicCamera).

PointLight – pontlámpa

Pontszerű fényforrás, minden irányban egyformán világít. Négy mezője van, a kapcsoló (on), fényintenzitás (intensity), szín (color), relatív hely (location). A kapcsoló értéke TRUE vagy FALSE lehet, attól függően, hogy világít-e a fényforrás. Az intenzitás értéke 0.0 ha sötét, 1.0 ha világos, ha egynél nagyobb, akkor különösen erő

fényforrást kapunk, de értéke lehet negatív is így akár negatív fényt kibocsátó fényforrást is kaphatunk (ez sötétíti a teret, a valóságban nincs ilyen). A `color` mező a fény színét adja meg (RGB), `location` pedig a fényforrás relatív helyét.

DirectLight – irányított fényforrás

Ez a fényforrás csak egy irányba világít, a fénysugarak egy irányba mutatnak, egymással párhuzamosan. Négy mezője van. Az első három mező maradt az előző (`on`, `intensity`, `color`), a negyedik mező változott, itt az irányát adja meg a fényforrásnak (`direction`). Az irányt XYZ koordináták szerint lehet megadni.

SpotLight – reflektorfény

A reflektorfény egy irányba kibocsátott fénykúpot bocsát ki, és az ezen belüli tárgyakat világítja meg. Hat mezője van, az első négy megegyezik az irányított fényforrás mezőivel. A másik két mező a kioltási faktor (`dropOffRate`), és a fénykúp nyílásszöge (`cutOffAngle`). A nyílásszög alapértéke 45 fok megfelelő ívmértékben. A kioltási faktor határozza meg, hogy a fény intenzitása hogyan csökkenjen a kúp élei felé (a 0.0 esetén minden egyformán világítódik meg).

PerspectiveCamera – perspektivikus kamera

Ez a kameratípus érvényesül a világunkban, érvényesek a rövidülések, az összetartó vonalak. Hat mezője van, az első a helyzet (`position`), forgatási tengelyérték (`orientation`), fókusztávolság (`focalDistance`), látószög (`heightAngle`), elülső távolság (`nearDistance`), hátulsó távolság (`farDistance`). Az első két mező határozza meg a kamera helyét, és helyzetét. A fókusztávolság állítását sok browser figyelmen kívül hagyja. A látószög egy ívmérték, alapértéke 45 foknak felel meg. Az elülső és a hátulsó távolság egy-egy síkot határoz meg, ez előtt- és mögött lévő tárgyak nem láthatóak.

OrtographicCamera – ortografikus kamera

Erre a kamerára nem érvényesek a megszokott rövidülési szabályok, a testek élei nem összetartóak, hanem párhuzamosak, ebből adódóan a látható térrész egy négyzet alapú hasábnak felel meg. Ennek is hat mezője van, öt megegyezik a perspektivikus kameratípussal (`position`, `orientation`, `focalDistance`, `nearDistance`, `farDistance`). A látószög mező helyett a látónégyzet mértékét (`height`) lehet megadni.

3.5. Csoportnode-ok

Lehetőségünk van néhány node-ot egyesíteni egy csoportba, majd ezek után ezt a csoportot egyetlen node-ként kezelhetjük. Így lehetőségünk van arra, hogy egy már megrajzolt testet transzformálni. Erre négy lehetőség van: Group, Switch, TransformSeparator és a Separator node-ok. A különbség a node-ok között: Group: minden utódot

rajzol, nincs mentett állapot, Switch: semmit, egy vagy minden utódot rajzol, nincs mentett állapot, TransformSeparator: minden utódot rajzol, transzformációs jellemzőket menti, Separator, minden utódot rajzol, minden állapotot ment. Ezeken kívül van a definició (DEF) node.

Group, TransformSeparator

A Group és TransformSeparator node-oknak nincs speciális mezőjük.

Separator

A Separator nodenak van egy kötelező mezője van: `renderCulling`. Három értéke lehet: ON, OFF, AUTO. Ezzel állíthatjuk be a browsernek (illetve hagyjuk őt választani), hogy a csoport minden tagját megrajzolja-e (OFF), vagy csak a befogadó térrész koordinái és láthatóságát (ON).

Switch

A Switch node-nak is van egy mezője, `whichChild`. Ezzel állíthatjuk be, hogy hány utódot rajzolunk meg. Ha –3 az érték, akkor minden utód megjelenik, -1 érték esetén akkor egyik utód sem rajzolódik meg, ha nemnegatív, akkor annyi utódot rajzol meg (az utódok indexelése is 0-val kezdődik).

DEF – definició

Lehetőségünk, van egy node-ot vagy egy csoportot elnevezni vele, majd a USE kulcsszóval meghívhatjuk, anélkül, hogy újra begépelnénk.

3.6. Hálózathoz illesztés: wwwInline

A nyelv lehetőséget biztosít arra is, hogy a HTML-ben megismert kapcsolatok linkeket is létrehozzuk. Ezeket használva összeköthetjük a világunkat más világokkal. A kapcsolatokat WWWAnchor és a WWWInline node-okkal valósíthatjuk.

WWWAnchor

Ez a node arra szolgál, hogy egy tárgyra kattintva követhessünk linkeket. Három mezője van: a link címe (`name`), szöveg a kapcsolathoz (`description`), és a `map`. A `name` mezőbe " " jelek közé kell beírni a linket, amit követni szeretnénk. A `description` mezőben információt helyezhetünk el a kapcsolatról. A `map` mezőnek két értéke van: NONE és POINT, és arra szolgál, hogy visszaadhassuk az éppen aktuális kurzorkoordinátákat. (Ez egy virtuális fölgömbön nagyon jól használható).

WWWInline

Ezt a node-ot arra találták ki, hogy egy térrészre kattintva kövessünk egy linket. Három mezője van, az első a link címe (`name`), a második a térrész mérete (`bboxSize`), a harmadik a térrész középpontja (`bboxCenter`). A name a WWWAnchor beli name node-al egyezik meg. A térrész méretét három mérettel lehet megadni: szé-

lcsség, magasság, mélység. A középpontot három koordinátával kell megadni, XYZ tengelyek szerint.

A VRML fájlban ezeken a node-okon kívül információkat is elhelyezhetünk. Ezek az információk nem jelennek meg a rajzolás során, de lehetnek olyanok is, amelyek a browsernek nyújtanak információt. Ha egy speciális információs node-ot használunk, akkor azt a DEF node-al tehetjük meg, majd mezőként egy string –et használunk. Speciális nevek lehetnek: Title, SceneInfo, BackgroundColor stb. Ezek a nevek nagyrészben browserfüggők.

VI. A Corel programcsalád

A Corel Draw az egyik legnépszerűbb grafikai szoftver együttes. A csomag két legfontosabb eleme a Corel Draw9 és a Corel Photo-Paint9 magukban is használható rajzolóprogramok. A két nagy grafikai program mellett számos kiegészítő program is része a csomagnak.

- Corel Draw

  Vektoros rajzolóprogram. Kiválóan alkalmas vonalas grafikák, objektumokra épülő képek, műszaki rajzok és hosszabb szöveges kiadványok készítéséhez.

- Corel Photo-Paint:

  Pixelgrafikus rajzprogram. Alkalmazhatunk fototechnikai hatásokat, különleges effektusokat, foltos színezéseket, elmosódott vonalakat, stb. Könnyen készíthetünk egyéni emblémákat, illusztrációkat, grafikákat.

- Corel Trace:

  Bitképek vektorizálása alkalmas segédprogram.

- Corel Capture:

  Képlopó program.

A számítógépes grafikai tervező-, rajzolóprogramok alapvető sajátosságaik alapján (jelfeldolgozási szempontból) két fő csoportba sorolhatók: vektorgrafikus és pixelgrafikus (bittérképes) programokra.

A Draw alapvetően vektorgrafikus program, míg a Photo-Paint pixelgrafikus. Mindkettővel egyszerűen készíthetünk rajzokat, képeket; ezek létrehozásának módja, illetve a képek látványa azonban sok mindenben eltér.

1. Draw

A vektoros rajzolóprogramok szakaszokból, görbékből és ezek csatlakozási pontjaiból építik fel a rajzokat, képeket. A csomópontokat áthelyezhetjük, eltávolíthatjuk, vagy újabbakat adhatunk az alakzat körvonalához. A csomópontok helyének, számának megváltoztatásán túl módosíthatjuk a csomópontok közti vonalak alakját, görbületét. A fájl méretét nem befolyásolja, ha egy alakzatot néhány milliméterre kicsinyítünk, vagy sokszorosára nagyítunk. Egy rajz tárolásához szükséges hely alapvetően az alkalmazott csomópontok számától függ.

2. Photo-Paint

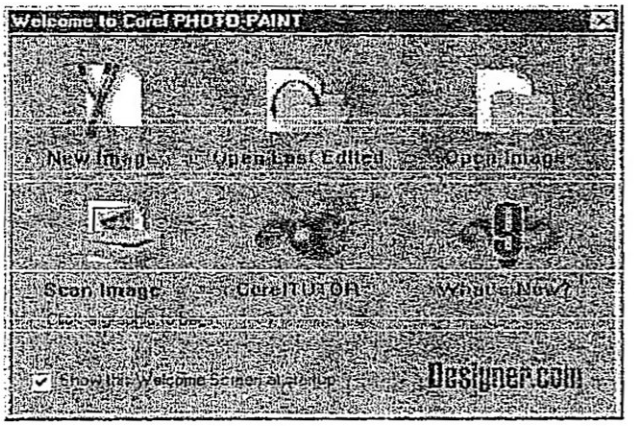
A pixelgrafikus rajzolóprogramok a képeket pontokból állítják össze. A különböző színes pontokból és azok kisebb-nagyobb halmazából bontakoznak ki az alakzatok. Az elkészített rajzok tárolásához szükséges hely nagyságát meghatározza a képpontok száma és a használt színek száma. Egy rajz képpontjainak száma függ a rajz méretétől, felbontásától. A felbontás adja meg, hogy adott távolságon (1 hüvelyken) belül, hány képpont legyen.

Természetesen lehetőségünk van a két program illetve képtípus közötti átjárhatóságra. A Corel Draw-ban szerkesztett kép bitképpé konvertálása, Bitmap/Convert To Bitmap. Ezt az immár pixeles képet szerkeszthetjük tovább a Photo-Paint segítségével, ha kiválasztjuk a Bitmap\Edit Bitmap menüpontot. A szerkesztés befejeztével (Photo-Paint bezárása) visszatérünk a Draw-ba és folytathatjuk munkánkat.

A két képtípus konvertálására alkalmas program a-Corel Trace is.

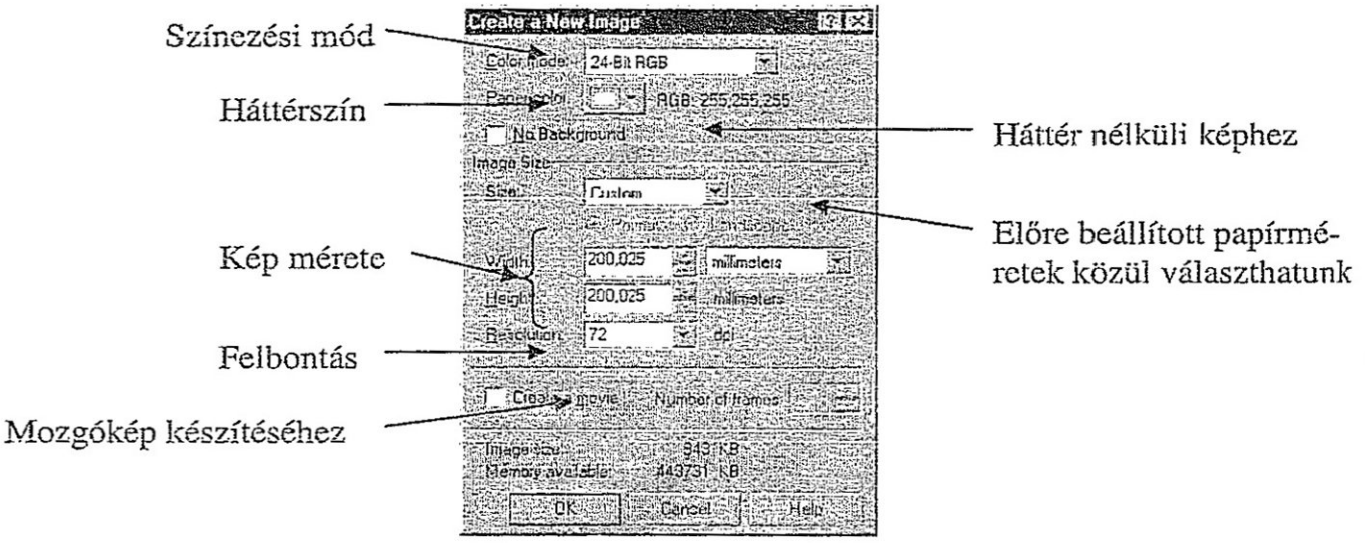
2.1. A Photo-Paint program indítása

A Corel Photo-Paint indítása után egy köszöntő ablakot kapunk, melyben lehetőségünk van választani a következők közül:



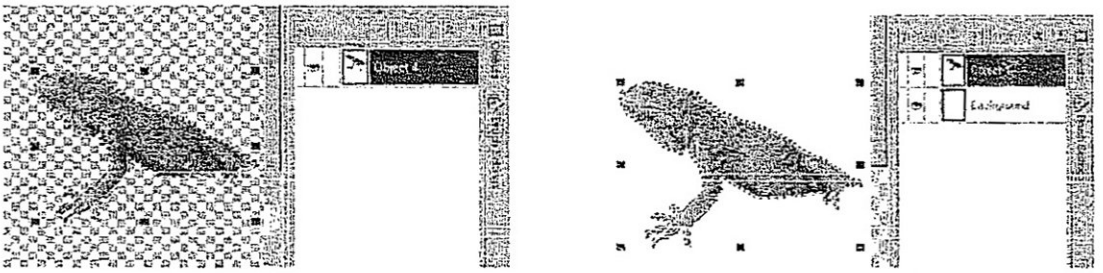
- Új üres munkalapot nyitni és ezen dolgozni tovább.
- Az utoljára szerkesztett képet nyitja meg.
- Tetszőleges kép megnyitása.
- Lehetővé teszi képek szkennelését.
- A tanító (Tutor) utasításait követve megismerkedhetünk a Photo-Paint használatának néhány fontos elemével.
- A 9-es verzió újításait mutatja be illusztrációkkal és rövid ismertetőkkel.

Amennyiben új munkalapot nyitottunk meg, egy párbeszéd ablak segítségével be kell állítanunk pár alaptulajdonságot.

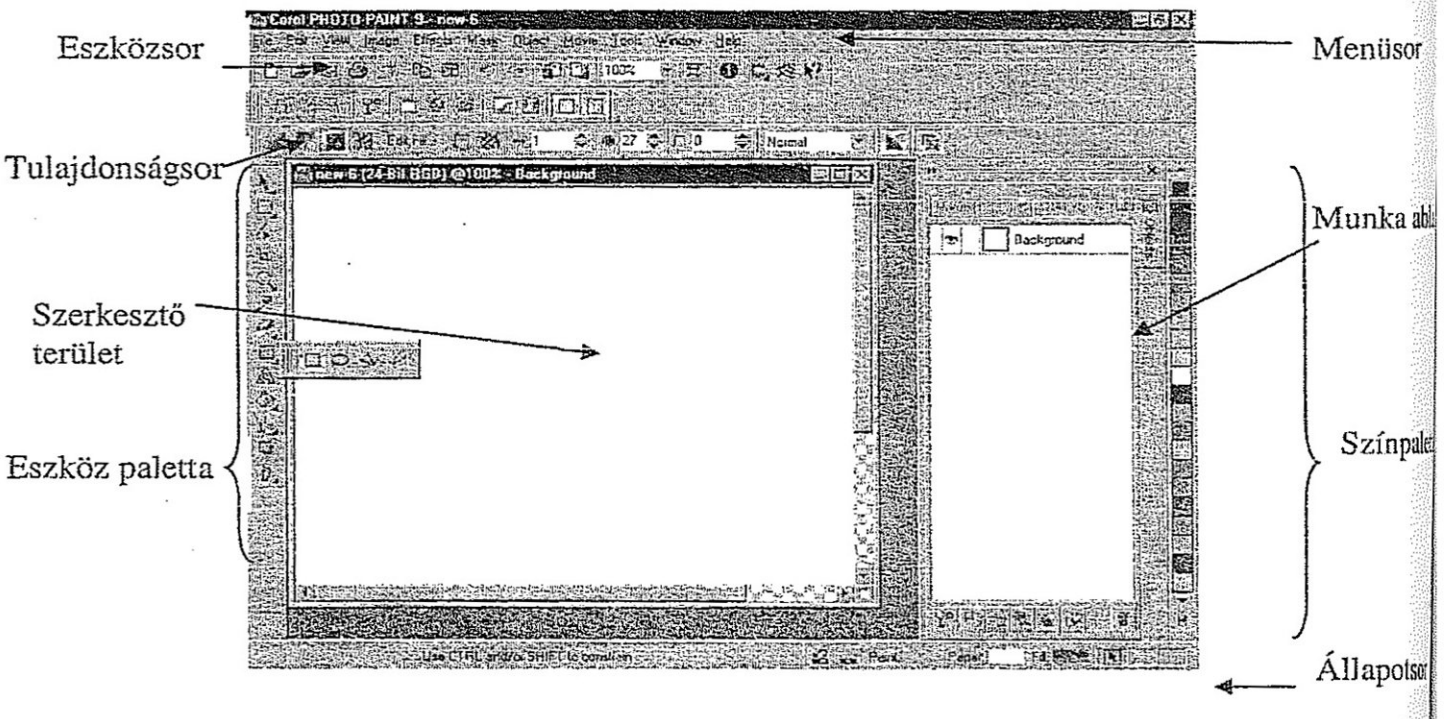


A kép létrehozásánál fontos meghatározni tartozzon-e háttér a képhez! Amennyiben háttér nélküli képet készítünk, a továbbiakban egy kockás lapon dolgozunk és mentéskor ténylegesen csak az elkészített kép (objektum) tárolódik.

Ha a képhez hátteret rendelünk, meg kell határozni a színét, és ez a szerkesztett kép elválaszthatatlan részét képezi.



Bár a program kinézete, kezelése nagyban hasonlít a megszokott Office alkalmazásokhoz, meg kell említeni pár újdonságnak számító eszközt, melyek használata nélkülözhetetlen lesz munkánk során.

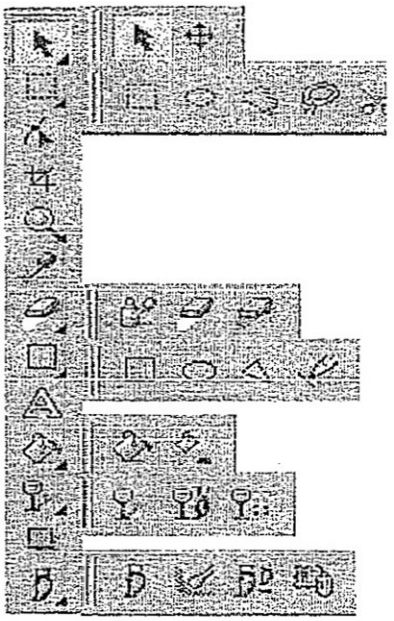


- Tulajdonságsor: az eszközhöz illetve a kijelölt objektumhoz kötődő, változó sor, melyen a gyors beállításokhoz találunk ikonokat.
- Eszközpaletta: itt találhatjuk azokat az alapvető eszközöket, amik egy kép létrehozásához, átalakításához szükségesek. Amennyiben az ikon jobb alsó sarkában egy nyilat látunk, annak lenyomása további lehetőségeket nyújt.
- Munka ablak (Docker): kategorizált parancsokat tartalmaz, plusz beállítási lehetőségeket nyújt az adott eszközhöz. Photo-Paintben az egyik legfontosabb ilyen ablak az Object Dockers, mely az objektumok kezeléséhez nélkülözhetetlen eszköz.

További lehetőségek:

- Exportálás, Importálás
- Vonalzó, Segédvonal, Segédrács: használatuk a képen történő tájékozódást, elhelyezést, illesztést könnyíti meg. A View menüponton keresztül ki, bekapcsolhatóak illetve beállíthatjuk tulajdonságaikat.
- Eszközök:

objektumok kezelése
maszkok
csomópont
nagyító
színfelszedő
visszavonás
alakzatok
szöveg
kitöltés
átláthatóság
árnyék
ecsetek



2.2. Corel és az Objektum

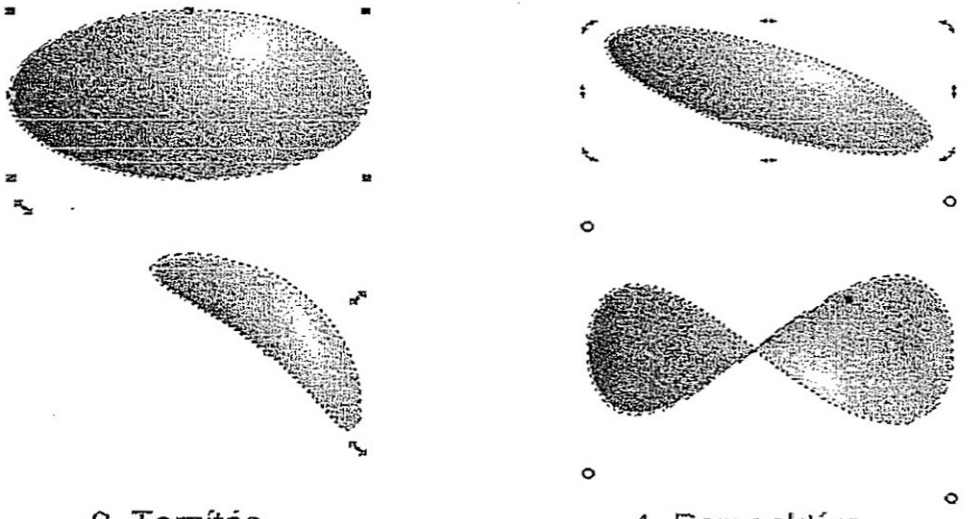
Az objektum fogalmán a Photo-Paint-ben kicsit mást értünk, mint a Corel Draw-ban. Az itt használt objektumok nem határozott körvonalú, zárt alakzatok. Ezeket az objektumot úgy képzelhetjük el, mint lapnyi méretű átlátszó fóliákat, amelyeknek egyes részei, bizonyos pontjai színezettek. Ahol az objektumfóliát nem színeztük, nem festettük be, ott az alatta lévő objektum vagy háttér látszik. Objektumokkal két alapvető műveletcsoportot kell megemlíteni:

A. Objektum transzformálása:

- Objektum kijelölése Nyíl eszközzel
- Méretezés, áthelyezés
- Forgatás, döntés
- Torzítás
- Perspektíva állítás

A transzformálásokat elérhetjük a megjelenő tulajdonságsoron, de az objektumon való többszöri kattintás is ugyanarra az eredményre vezet: Ilyenkor az objektumot körül ölelő határoló dobozok (nyilak) alakja változik meg. Ezeket a szokásos módon lehet mozgatni, melynek eredménye a megfelelő transzformálás lesz.

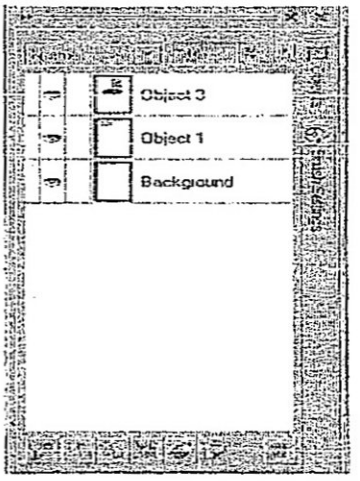
Pl.: 1. Kijelölés, áthelyezés, 2. Forgatás, döntés



3. Torzítás 4. Perspektíva

B. Objektum kezelése

- Objektumok listája:



Az Objektum munka ablakban (Object Dockers) könnyen, egyszerűen kezelhetjük az objektumokat. Az ablak középső mezőjében az objektumok listáját láthatjuk. Ha színe, nem háttér nélküli munkalapon dolgozunk, akkor az objektumlista legalsó sorában a Background feliratot láthatjuk. Egy objektumot kijelölhetünk, ha a nyíl eszköz választása után a kiköthető ablakban a bélyegképre kattintunk.

A bélyegkép mérete megváltoztatható, ha a jobb felső sarokban lévő nyílra kattintunk (Samall, Medium, Large). Egy objektumot átnevezhetünk, ha a névre jobb egérgombbal kattintunk, majd Properties/General/Name.

- Új, üres objektum készítése:

A kiköthető ablak New Image ikonjával (ill.Object/Create/New Object) illetve egy alakzat készítésekor Render To Object/Selection ikonnal új objektum készíthető.

- Másolás és törlés:

A szokásos módon végezhető el a kijelölt objektum(ok)on.

- Sorrend:

Az objektumok közti sorrend nem csak a létrehozás sorrendjét, hanem ezzel egyidejűleg a láthatóság (takarás) sorrendjét is jelenti. Az objektum sorrendben elfoglalt helyének megváltoztatásához használjuk a „Fogd és Vidd" technikát.

- Elrejtés:

Az Objektum munka ablakban a Szem ikon kikapcsolása az elrejtést, visszakapcsolása pedig a megjelenést biztosítja.

- Összekapcsolás:

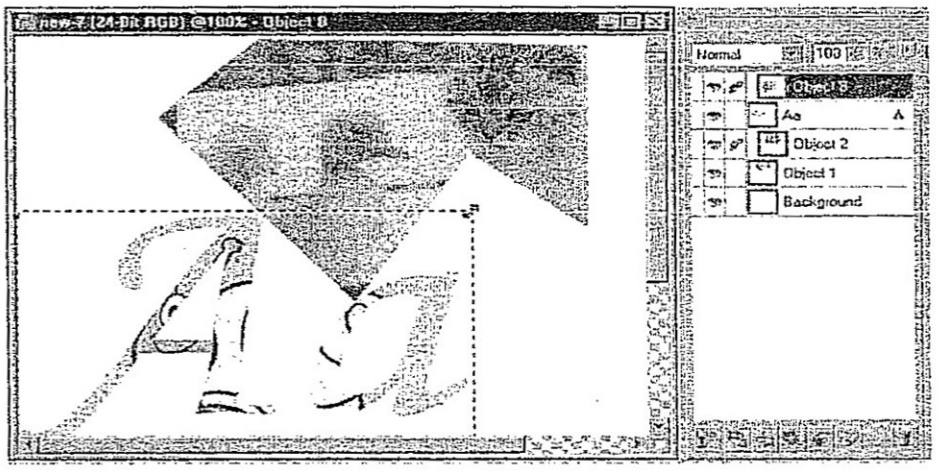
Kombinálás: az objektumokat véglegesen összekapcsoljuk (egy fóliára kerülnek), a továbbiakban külön nem kezelhetők, utólag szét nem választhatók, legfeljebb visszavonhatjuk ezt a műveletet, de ezzel egy korábbi állapotba kerülünk vissza! A kombinálással jelentősen csökkenthetjük a fájl méretét.

Lehetőségeink:

- Objektum-háttérrel
- Objektum-objektummal
- Összes objektum-háttérrel

Csoportosítás: Az objektumok együtt kezeléséhez egy kötetlenebb és használhatóbb „eszköz". A Nyíl eszközzel jelöljük ki az objektumokat, majd a tulajdonságsoron használjuk a Group gombot. A csoportba foglalt elemek egységesen módosíthatóak és bármikor szétválaszthatjuk őket.

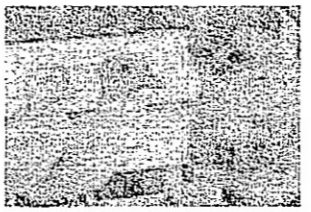
Objektumba illesztés: Egy objektumot a másik belsejébe teszünk. A „tartalom" objektumnak meg kell előznie a sorrendben a „tároló" objektumot. A rajzterületen fedje egymást a két objektum. Kattintsunk a „szem" melletti területre a tartalom objektumnál, ezzel egy gémkapcsot helyezünk el, mely a korlátozott láthatóságot eredményezi. Mozgassuk a tartalmat a megfelelő helyre.



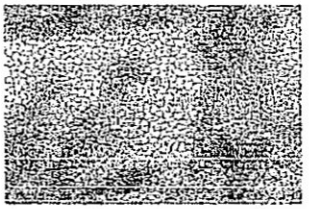
A Corel Photo-Paint-ben a leglátványosabb, különlegcs hatásokat az Effektusok alkalmazásával érhetjük el. Szinte végeláthatatlan számú lehetőség áll rendelkezésünkre, az egyes kategóriákon belül számos típus és beállítás érhető el. A teljesség igénye nélkül nézzünk pár lehetőséget:

3D effects:
Page curl..

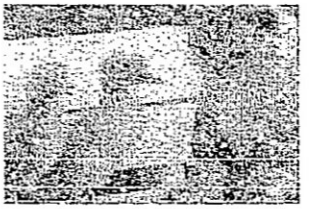
Art Strokes:
Charcoal



Creative 1:
Staried Glass.

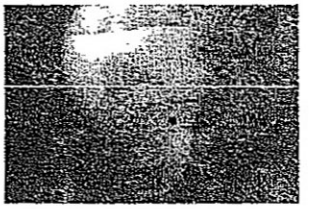


Creative 2:
Weather – snow



Render:
Light Effects

Texture: 1
Brick Wall



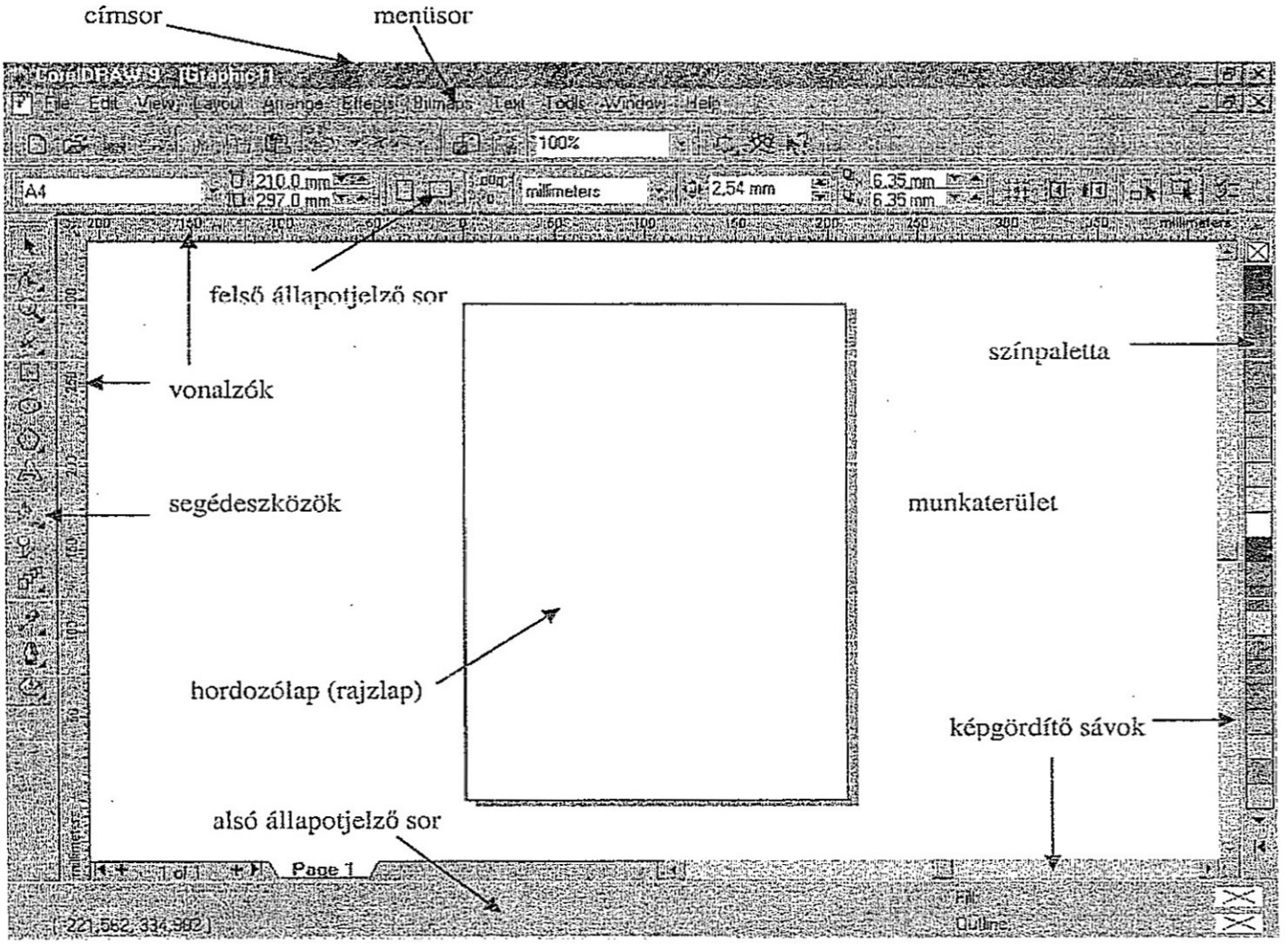
VII. Grafikus ábrák létrehozása, transzformálása (CorelDRAW 9)

A kanadai Corel cég első CorelDRAW vektorgrafikai programja még DOS alatt futott, majd hamarosan megjelent a Windows 3.0 alatt futó 1.2-es verziója. Már ez is nagy sikert aratott, de a 2.0-s verzió minőségi változást hozott. Ez a szoftver alkalmas volt saját grafikák elkészítésére, de fel volt készítve a profi nyomdai előkészítésre is.

Manapság a 9.0-s a legfrissebb verziója a CorelDRAW-nak, ez már nagyságrendekkel többet tud, mint az első verziók. (Mivel azonban most csak az alapfogalmak és funkciók ismertetése a célunk, bizonyos új lehetőségekre nem fogunk részletesen kitérni.)

1. A képernyő felépítése

A CorelDRAW képernyő a következő összetevőkből áll:



Munkaterület

A munkaterület a képernyő nagy fehér (tiszta) része. Az ezen belül elhelyezkedő lap a rajzlap, a nyomtatható területet mutatja. (Normál esetben csak a rajzlapon található objektumok kerülnek nyomtatásra.)

A *View* menüpontban választhatjuk ki, hogy milyen módon szeretnénk látni a grafikus objektumunkat a rajzlapon. (Pl. Normal módban az objektumok körvonala és kitöltése is látszik, *Simple Wireframe* üzemmódban viszont csak a körvonalak. Ez utóbbi a gyorsabb megjelenítés érdekében lehet hasznos.)

Segédeszközök

Gyorsan cserélhető, igen hasznos eszközök

Nyíl: az objektumok kijelöléséhez és megváltoztatásához (ld. 4.)
Forma: az objektumok átalakításához (ld. 7.)
Nagyító: a képernyőn látható látszólagos képméret megváltoztatásához (ld.5.)
Ceruza: vonalak, görbék szabadkézi rajzolásához (ld. 3.5.)
Négyszög: téglalapok, négyzetek rajzolásához (ld. 3.6.)
Ellipszis: ellipszisek, körök rajzolásához (ld. 3.7.)
Ötszög: sokszögek, spirálvonalak, rácsok rajzolásához (ld. 3.8.)
„A" (szöveg): szövegek, szimbólumok megjelenítéséhez (ld. 6.)
Interaktív kitöltés: színkitöltéshez (ld. 8.4.)
Interaktív minta-kitöltés: mintával való kitöltéshez (ld. 8.5.)
Interaktív objektum-módosító: objektum árnyékolásához, alakításához, stb. (ld. 8.6.)
Cseppentő: szín felvételéhez és a felvett színnel való kitöltéshez (ld. 8.2.)
Vonalkihúzó: körvonal tulajdonságok megváltoztatásához (ld. 8.7.)
Kitöltés: színkitöltéshez (ld. 8.3.)

Felső és alsó állapotjelző sorok

Tájékoztat a kijelölt objektumokról és az aktuális folyamatokról. Használatuk ajánlott, sok hasznos információt nyújtanak.

Vonalzók

Ki/be kapcsolható vonalzók, melyek az objektumok méretének és helyének meghatározásánál segítenek. A két vonalzó találkozásánál megjelenő kis gomb húzásával állíthatjuk, hogy hol legyen az origó (0,0).

Színpaletta

Az ábrák és körvonalak a képernyő jobb oldalán látható színpaletta segítségével színezhetők. Ez is ki/be kapcsolható, és megjelenése állítható.

Rács és segédvonalak

Az objektumok pontos elhelyezését segítik. Az ezekhez való igazítás ki/be kapcsolható a *View* menüpont *Snap to Grid*, illetve *Snap to Guidelines* menüpontok kiválasztásával.

2. Rajzolás

Indítás, file megnyitása és mentése

A CorelDRAW indításakor általában megjelenik egy bejelentkező panel (beállítás kérdése), amely többek közt a következő lehetőségeket kínálja fel:

- új rajz készítése
- az utolsóként készített rajz megnyitása
- rajz megnyitása
- a CorelDRAW használatának ismertetése tanfolyamként

Ebből a számunkra megfelelőt kiválasztva már hozzá is láthatunk a munkához.

Új „rajzlapot" a menü segítségével is kérhetünk, a *File* menü *New* menüpontját választva a munkaterületen egy üres rajzlap jelenik meg, amin már el is kezdhetjük a rajzolást.

Korábban elkészített rajzot a *File* menü *Open* menüpontjával is megnyithatunk. Ha nem vektorgrafikus formában van a megnyitni kívánt rajz, használjuk a *File* menü *Import* menüpontját (pl. GIF vagy JPG file-ok is így nyithatóak meg CorelDRAW-ban).

A rajzunkat elmenteni a *File* menü *Save* vagy *Save As* menüpontjaival tudjuk. Ekkor a CorelDRAW alapvetően a saját file-formátumában, CDR kiterjesztéssel menti el a rajzunkat, de ugyanitt választhatunk más vektorgrafikus file-típust is. Ha ezek nem felelnek meg az igényeinknek, akkor válasszuk a *File* menü *Export* menüppontját, és itt lehetőségünk nyílik nem-vektorgrafikus formátumban menteni (pl. GIF vagy JPG file-okba).

A különböző objektumok készítésére szolgáló segédeszközök

Ceruza: vonalak, görbék szabadkézi rajzolásához (ld. 3.5)
Négyszög: téglalapok, négyzetek rajzolásához (ld. 3.6)
Ellipszis: ellipszisek, körök rajzolásához (ld. 3.7)
Ötszög: sokszögek, spirálvonalak, rácsok rajzolásához (ld. 3.8)

Új segédeszköz kiválasztásáig a régi marad érvényben, így nem kell az adott segédeszközt újra kiválasztani az ugyanolyan objektum rajzolásához. Rajzolás közben az aktuális rajz van kiválasztva, így a vonalkihúzó, a kitöltő stb. segédeszközök opciói arra vonatkoznak.

A Nyíl segédeszköz átmeneti használata

A rajzoló segédeszközök használatakor is szükség lehet más objektumok kijelölésére. Normál esetben rá kell kattintani a Nyíl segédeszköz ikonjára, majd ki lehet jelölni a szerkeszteni kívánt objektumot.

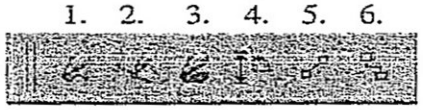
(Ezt gyorsabban is megtehetjük, rajzoló segédeszköz használata közben szóközt leütve az utolsóként rajzolt objektum kerül kijelölésre, újabb szóköz leütésre pedig a korábbi rajzolóeszköz válik aktívvá.)

A rajzlap beállításai

A rajzlap beállításához a *Layout* menü *Page Setup* pontját kell kiválasztanunk. Ezzel a lap méretét, pozícióját, orientációját, színét stb. változtathatjuk meg. Ez a funkció gyorsabban is elérhető, a rajzlap árnyékára való dupla kattintással.

A Ceruza segédeszköz

1. 2. 3. 4. 5. 6.



Szabadkézi rajzolás

Ha a ceruza mellett kinyíló kis menüből az első (*Freehand*) ikont választjuk, akkor lehetőségünk nyílik a ténylegesen szabadkézi rajzolásra. Amíg az egér bal gombját lenyomva tartjuk, addig rajzol, amint felengedjük, befejezettnek tekinti a vonalat. Ha egyenes vonalat akarunk rajzolni, akkor csak kattintanunk kell egy pontra, majd egy másikra, és így a kettő között automatikusan megjelenik egy szakasz.

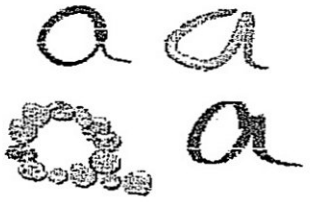
Lehetőség nyílik arra is, hogy a rajzolás mégse legyen teljesen szabadkézi. Ehhez a felső állapotsor jobb szélén lévő ikonra kell kattintanunk (*Freehand Smoothing*), és itt állíthajuk, hogy mennyire legyen valóban szabadkézi a rajz,

és mennyit finomítson rajta a szoftver (0-100 %).

Bezier módban való rajzolás

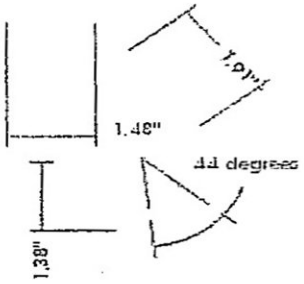
Elsősorban az egérrel való klikkelésekkel kijelölt pontokat köti össze egyenesekkel. Ha nem egyeneseket szeretnénk, hanem görbékkel összekötni a megadott pontokat, akkor klikkeléskor tartsuk lenyomva a balgombot, és mozgassuk az egeret. Ekkor szaggatott vonallal megjelenik, hogy milyen görbét tud húzni a legutolsó pont, és az általunk most kijelölt pont között. A görbe akkor kerül rögzítésre, amikor felengedjük a gombot.

Ecsetként való használat



Addig rajzol, amíg lenyomva tartjuk a balgombot, és a hatás olyan lesz, mintha egy ecsetvonást húztunk volna. A vonás formáját, stílusát, vastagságát, a szabadkéz finomságát itt is a felső állapotsorban megjelenő ikonok segítségével állíthatjuk.

„Dimenziós" vonalak



Derékszögű, illetve párhuzamos vonalak húzhatóak vele, amelyeknek megadja, a távolságát, valamint körcikkek, amelyeknek megadja a központi szögét. Itt is a felső állapotsorban található ikonok segítségével választhatunk a különböző funkciók között.

Szakaszrajzolás

A klikkeléssel kijelölt pontok közé szakaszt rajzol. Ezt a szakaszt a *Forma* segédeszköz kiválasztásával, a felső állapotsorban megjelenő ikonok segítségével tudjuk transzformálni. Pl. sokféle nyilat (vagy más egyéb formát) tud tenni a vonal végeire.

Töröttvonal-rajzolás

3 egymásra merőleges szakaszból álló vonalat rajzol. Ezt szintén a *Forma* segédeszköznél megjelenő funkciókkal tudjuk alakítani. Az előzőhöz hasonlóan itt is sokféle nyilat (vagy valamilyen ábrát) tud tenni a vonal végeire.

A Négyszög segédeszköz

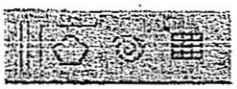
Ennek segítségével tetszőleges négyszöget rajzolhatunk. Első lépésként rajzoljunk egy tetszőleges téglalapot/négyzetet. (Kattintsunk egy pontra, majd a balgombot nyomva tartva mozgassuk az egeret a kívánt téglalapméret eléréséig, végül engedjük fel a balgombot. Ha mindeközben a Ctrl-billentyűt is lenyomva tartjuk, akkor négyzetet rajzol.) Ezután a *Forma* segédeszközt választva, a felső állapotsorban megjelenő ikonok segítségével tudjuk a téglalapot tetszőleges négyszöggé transzformálni.

Az Ellipszis segédeszköz

Ennek segítségével tetszőleges ellipszist, kört, körcikket, körívet stb. rajzolhatunk. Első lépésként rajzoljunk egy tetszőleges ellipszist/kört. (Kattintsunk egy pontra, majd a balgombot nyomva tartva mozgassuk az egeret a kívánt ellipszisméret eléréséig, végül engedjük fel a balgombot. Ha mindeközben a Ctrl-billentyűt is lenyomva tartjuk, akkor kört rajzol.) Ezután a *Forma* segédeszköz segítségével tudjuk a kört/ellipszist módosítani.

Az Ötszög segédeszköz

1. 2. 3.



Ötszög rajzolás

Ezzel a segédeszközzel tetszőleges méretű ötszöget tudunk rajzolni. A felső állapotsorban megjelenő ikonok segítségével csökkenthetjük, illetve növelhetjük a csúcsok számát, illetve arra is van lehetőség, hogy ne konvex sokszögként, hanem „csillag" alakzatként kösse össze a csúcsokat.

Spirál rajzolás

Ezzel a segédeszközzel tetszőleges méretű spirálvonalat tudunk rajzolni, amelyet a felső állapotsorban megjelenő ikonok segítségével tudunk módosítani. Például az irányát, a „sűrűségét", stb.

Rács rajzolás

Ezzel a segédeszközzel tetszőleges méretű rácsot tudunk rajzolni. A felső állapotsorban megjelenő lehetőségek segítségével tudjuk a rács oszlopainak, illetve sorainak számát növelni vagy csökkenteni.

3. Tárgyak kijelölése: a Nyíl segédeszköz

A Nyíllal a tárgyakat lehet kijelölni, amelyeket meg akarunk változtatni. A kijelölt tárgyat a menü parancsaival illetve a segédeszközökkel tetszőlegesen át lehet alakítani. A Nyilat ezen kívül a kijelölt objektumok elmozgatására, megnyújtására, nagyítására, kicsinyítésére vagy tükrözésére illetve forgatására használhatjuk.

Egy tárgyat könnyen kijelölhetünk úgy, hogy a tárgy körvonalának bármely pontjára kattintunk. A kijelölés megszűnik, ha valahová az üres mezőre kattintunk.

Több tárgy kijelölése történhet úgy, hogy egyesével kijelöljük őket az egérrel, de közben lenyomva tartjuk a Shift billentyűt.

A tárgyakat „gumikeret"-tel is kijelölhetjük, vagyis a balgombot lenyomva tartva mozgassuk a egeret. Ekkor megjelenik egy téglalap. Amint felengedjük a balgombot, mindazok az objektumok, amelyek ezen a téglalpon belül voltak, együttesen kijelölésre kerülnek.

Ha a Nyíl aktív, akkor a Tabulátorral egymás után kijelölhetjük az egymás után következő tárgyakat.

Ha egy tárgyat kijelöltünk, akkor a tulajdonságai megjelennek a felső állapotsorban.

4. Objektumok bemutatása

Nagyító segédeszköz

A Nagyítóval változtatható a nagyítás mértéke. Kijelölésekor megjelenik a felső állapotsorban egy ikonsor:

1. 2. 3. 4. 5. 6. 7. 8. 9.

Nagyítás

Ha egyszerűen csak rákattintunk a képre a funkció kiválasztása után, akkor a duplájára nagyítja a képet. Ha gumikerettel kijelöljük, hogy mit szeretnénk nagyítani, akkor a gumikeretben található objektumokat kinagyítja a lehető legnagyobbra (hogy még a képernyőn elférjenek). Ez utóbbi funkció az F4 billentyűvel egyszerűen elérhető.

Kicsinyítés

Ha kiválasztjuk ezt a funkciót, akkor a legutóbbi nagyítás előtti állapotba jutunk vissza. Ha még nem történt eddig nagyítás, akkor a felére kicsinyíti a képernyőn látható képet. A gyors kicsinyítés F3-mal történhet.

Méret 1:1

A lap és a grafika valós méretét láthatjuk a képernyőn.

A kiválasztott objektum nagyítása

Az utoljára Nyíllal kijelölt grafikát nagyítja ki a lehető legnagyobbra (úgy, hogy még elférjen a képernyőn).

Minden objektum nagyítása

Erre az ikonra kattintva azt érjük el, hogy a lapon található grafikák kinagyítódnak annyira, hogy még mindegyik teljes egészében látszódjon a képernyőn.

Nagyítás lapméretre

A képernyőn a teljes lap látszik, a lehető legnagyobbra nagyítva.

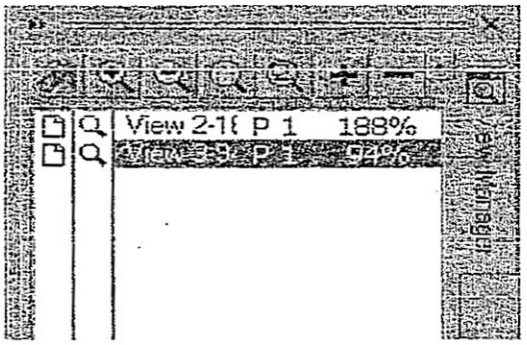
Nagyítás lapszélességnyire

A képernyőn a lap teljes szélessége látszik.

Nagyítás laphosszúságnyira

A képernyőn a lap teljes hossza látszik.

Látványkezelő



Ha erre az ikonra kattintunk, akkor a képernyő jobb oldalán megjelenik egy kis ablak. Ennek felső részében néhány kicsinyítő/nagyító funkció ikonja is megjelenik, illetve egy + és egy – jel.

A látványkezelő segítségével a különböző méretű nagyításokról és kicsinyítésekről listát készíthetünk, így meggyorsítva az adott méretű grafikák gyors elérését.

5.2. Objektumok megjelenítése a munkaterületen

5.2.1. A képgördítő sávok

A munkaterület látható részének változtatásához használjuk a képgördítő sávokat a más Windows-alkalmazásoknál megszokott módon.

5.2.2. A munkaterület frissítése

A *Window* menüpont *Refresh Window* menüponjával érhető el.

5.2.3. Kézi lap-áthelyezés 

Ha a Nagyító mellett található Kézre kattintunk, akkor a rajzlapot tudjuk mozgatni a képernyőn.

6. Szöveg elhelyezése a rajzlapon

A segédeszközök közül az „A" betűt kiválasztva, szöveget tudunk elhelyezni a rajzlapon.

6.1. Egyszerű szöveg

Az egyik lehetőség, hogy az ikonra való kattintás után egyszerűen csak rákattintunk a rajzlapra. Ekkor azon a pozíción megjelenik a kurzor, és máris elkezdhetjük begépelni a szükséges szöveget. Ekkor egy szöveges dokumentumhoz hasonlóan kezeli az egész rajzlapot a szöveg szempontjából. A begépelt szöveg a szövegszerkesztőknél megszokott módon, tetszőlegesen formázható a felső állapotsorban megjelenő szövegformázási funkciók segítségével.

6.2. Szövegdoboz

A másik lehetőség, hogy az ikonra való kattintás után rajzolunk a rajzlapon egy szövegdobozt a gumikerettel, ebben az esetben a szöveget egy „dobozban" helyezi el a rajzlapon.

Természetesen itt is elérhetők a szövegformázási funkciók.

6.3. Szöveg szerkesztése külön ablakban 

A felső állapotsorban megjelenő funkciók utolsó pontja a szöveg kényelmesebb szerkesztésére teremt lehetőséget. Erre az ikonra kattintva egy szövegszerkesztő nyílik meg egy új ablakban.

Itt nyílik mód arra is, hogy egy professzionális szövegszerkesztővel elkészített szöveget (pl. doc file-t) töltsünk be, és ha szükséges, továbbszerkesszünk.

7. Objektumok átalakítása 

Ehhez a *Forma* segédeszköz nyújt lehetőséget, amelynek több funkciója is van

1. 2. 3. 4.

7.1. Forma segédeszköz

A grafika körvonalait lehet vele módosítani a felső állapotsorban megjelenő lehetőségek segítségével.

7.2. Kés segédeszköz

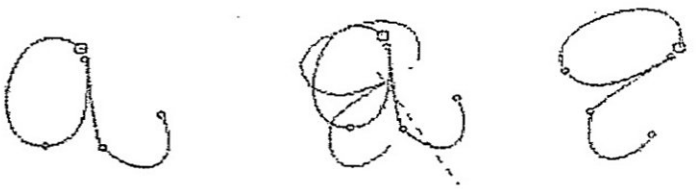
Grafikák vághatók szét vele több darabra. Itt lehetőség nyílik a választásra, hogy a szétvágás után is egy objektumként szeretnénk-e kezelni a grafikát, vagy „valóban" essen szét több darabra.

7.3. Radír segédeszköz

Ezt a funkciót kiválasztva, a balgomb lenyomásával és az egér mozgatásával bármit kiradírozhatunk a rajzlapról. (A teljesen kiradírozott grafikák megszűnnek létezni).

7.4. Szabad transzformálás segédeszköz

A grafika egészét lehet transzformálni ezzel a segédeszközzel, a felső állapotsorban megjelenő lehetőségek segítségével.



8. Munka színekkel, mintákkal és formákkal

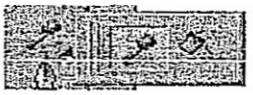
8.1. Nyitott és zárt objektumok

A CorelDRAW megkülönböztet nyitott és zárt objektumokat. Vagyis ha az objektumnak nincs egy folytonos körvonala, akkor azt nyitottnak tekinti, és színezésnél a kitöltés „túlcsordul" az objektum határain.

Ezt a problémát ki lehet küszöbölni, ha kijelöljük az összes olyan objektumot, ami részt vesz a színezésben, majd az *Arrange* menü *Combine* pontjával kombináljuk őket.

Innentől kezdve a CorelDRAW úgy tekinti, mintha a nyitott objektum két végpontja egy láthatatlan egyenessel össze lenne kötve, és így zárt lenne.

8.2. Színválasztás



A tárgyak színezésére a *Kanna*, illetve a *Cseppentő* segédeszközök használatosak. A *Cseppentő* segítségével egy korábban használt színt „vehetünk fel" a rajzunkról, a *Kannával* pedig a megadott színnel tölthetünk egy objektumot.

A kiválasztott objektum színe drótkeretes módban csak az alsó állapotsor jobb sarkában látszik.

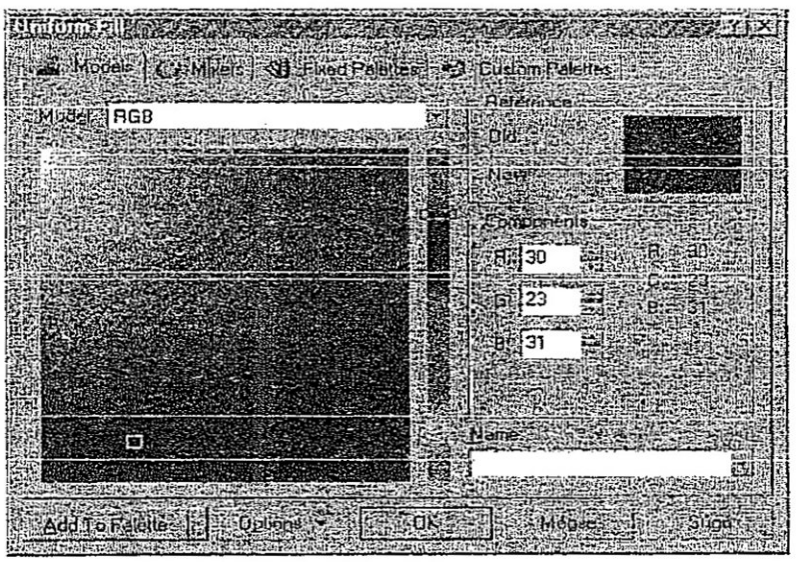
8.3. A Kitöltés segédeszköz

Többféle funkciója van, ezek a következők:

1. 2. 3. 4. 5. 6. 7.

8.3.1. Szín kiválasztása

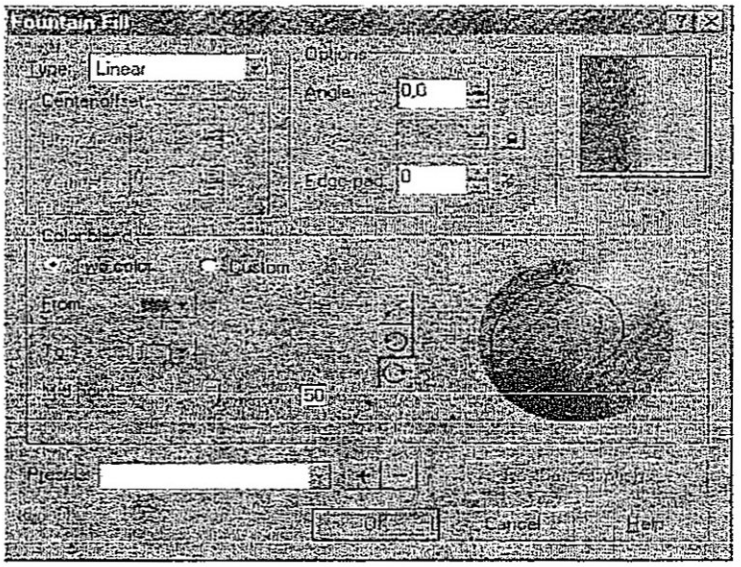
Erre a segédeszközre kattintva egy párbeszédablak jelenik meg, ahol a színpalettán nem szereplő egyedi színeket keverhetünk ki magunknak. Itt igen finom árnyalásra van lehetőség, többféle módon.



8.3.2. Színfutás kiválasztása

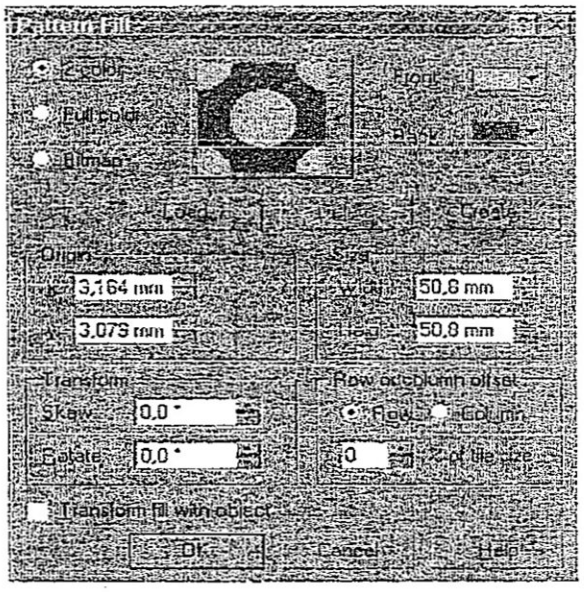
Erre a segédeszközre kattintva is egy párbeszédablak jelenik meg, ahol színárnyalási tulajdonságok beállítására van lehetőség.

Ahogy az ikonja és neve is sejtetni engedi, itt egy színt árnyal a legvilágosabbtól a legsötétebb felé. Ha két színt választunk ki, akkor az egyiket finoman átfuttatja a másikba. Ennek is többféle módja lehetséges.



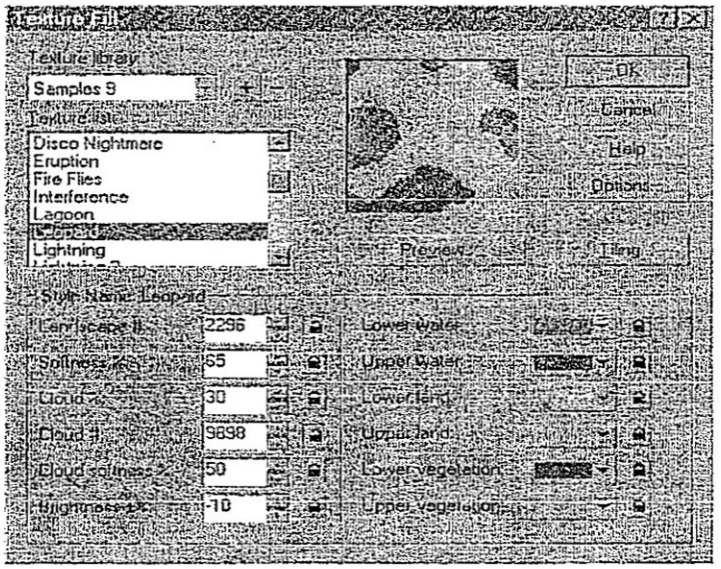
8.3.3. Minta kiválasztása

Erre a segédeszközre kattintva szintén egy párbeszédablak nyílik ki, ahol lehetőségünk van a kitöltéshez valamilyen mintát kiválasztani. Vannak a CorelDRAW-nak saját mintái, de akár bitmapeket is megadhatunk, amelyekkel mozaikszerűen kitölti az adott objektumot.



8.3.4. „Szövet" kiválasztása

Erre a segédeszközre kattintva újfent egy párbeszédablak nyílik ki, ahol mintázatok széles választékából választhatunk, és azt minden jellemzőjében pontosan az igényeinknek megfelelően alakíthatunk.



8.3.5. Postscript mintázat kiválasztása

A CorelDRAW rendelkezik PostScript kitöltő-mintákkal, amelyek csak PostScript nyomtatónk jelennek meg. Ennek a segédeszköznek a meghívásával ilyen mintákat választhatunk ki és szabhatunk át saját igényeinknek megfelelően.

8.3.6. Mintázat megszűntetése

Ez a segédeszköz az összes korábban megadott kitöltést megszűnteti.

8.3.7. Színválasztó ablak

Ennek a segédeszköznek a kiválasztásával a munkaterület jobb oldalán megjelenik egy ablak, ahol különböző sémák (pl. RGB (piros-zöld-kék); CMYK (cián-bíbor-sárga-fekete); HSB (színárnyalat-telítettség-fényerő)) alapján lehet színeket választani, és azokkal a kijelölt objektumot kitölteni, illetve a körvonalait meghúzni.

8.4. Az Interaktív kitöltés segédeszköz

A Kitöltéshez hasonlóan ennek is több funkciója van: 1. 2.

Az első a színekkel való kitöltést teszi kényelmesebbé azzal, hogy nem kell különböző párbeszéd-ablakokkal töltenünk az időt, hanem a felső állapotsorból

választhatjuk ki a Kitöltésnél már megismert funkciókat, és azok hatását közvetlenül a rajzunkon láthatjuk.

A második a körvonalak és tulajdonságaik módosítását teszi kényelmessé az előzőhöz hasonló módon.

8.5. Interaktív minta-kitöltés

Ez a segédeszköz lehetővé teszi, hogy a korábbiakban megadott „szövet"-re még mintát is illesszünk. Ehhez a felső állapotjelző sorban megjelenő lehetőségek használhatóak.

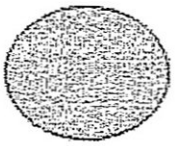
8.6. Interaktív objektum módosítók

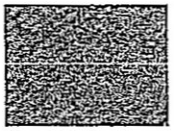
1. 2. 3. 4. 5. 6.

8.6.1. Interaktív vegyítő

Két (vagy több) különböző formát „összeolvaszt", vagyis a köztük lévő távolságot feltölti az őket összekötő formákkal, és így egy „csövet" kapunk, aminek egyik vége az egyik, másik vége a másik objektum (akárcsak az ikonján a két négyzettel). A színükkel is hasonlóan jár el, és ezek finomsága állítható a felső állapotsorban megjelenő funkciók

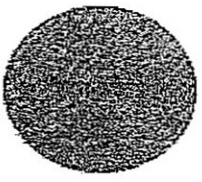
segítségével.





8.6.2. Interaktív kontúr

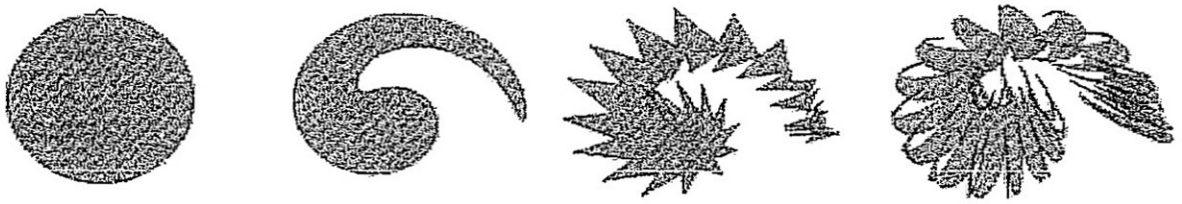
Az előzőhöz hasonló, de itt egy adott objektum körvonalait sokszorozza meg koncentrikusan befelé vagy kifelé (ahogy az ikonján a négyzet körvonalait is). Itt is lehet a vonalak színét, távolságát, illetve a kitöltés színét változtatni kontúrról kontúrra,



a felső állapotsorban megjelenő funkciók segítségével.

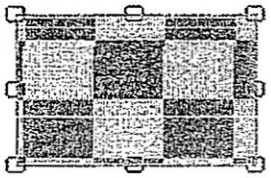
8.6.3. Interaktív torzító

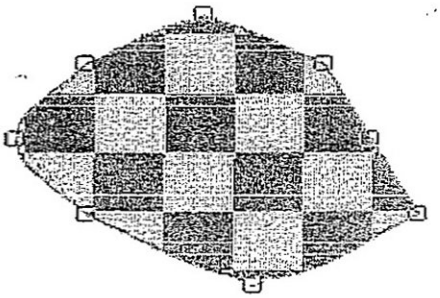
A kijelölt objektumot lehet torzítani többféle módon. A felső állapotsorban megjelenő funkciók (cikk-cakkosító, csavaró, kifordító-befordító, stb.) segítségével egészen különleges formákhoz lehet eljutni.

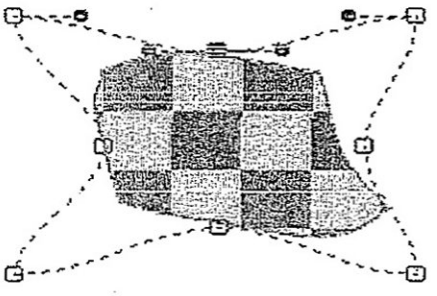


8.6.4. Interaktív beborító

Az adott objektum köré írható síkidom adható meg vele. Ezt tetszőlegesen átrajzolhatjuk, illetve a felső állapotsorban megjelenő funkciók segítségével bármilyen formára illeszthetjük az objektumunkat.

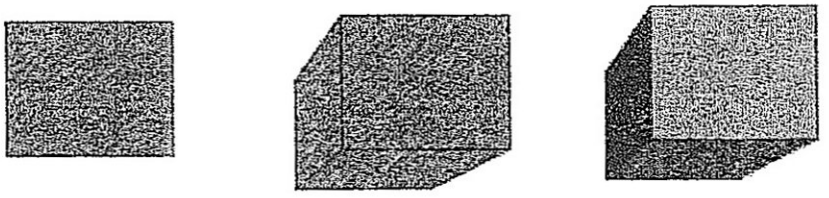






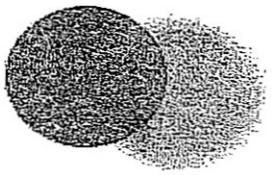
8.6.5. Interaktív testté-alakító

Az adott objektumból 3D-s ábrát készít (pl. egy négyzetből kockát). A felső állapotsorban megjelenő funkciók segítségével szabályozhatjuk az objektum mélységét, elforgathatjuk, színekkel érzékeltethetjük a térbeliségét, illetve akár három lámpával meg is világíthatjuk (ezek elhelyezkedését és fényerősségét mi magunk állíthatjuk be).



8.6.6. Interaktív árnyék-készítő

Az adott objektumnak megrajzolja az árnyékát. A felső állapotsorban megjelenő funkciók segítségével állíthatjuk az árnyék tuladjonságait (színét, vetülését, méretét, irányát, stb.).



8.7. Vonalkihúzó segédeszköz 

A segédeszköz kiválasztásakor itt is megjelenik egy helyi menü:

1. 2. 3. 4. 5.

8.7.1. Vonalkihúzó párbeszédablak

Ennek a funkciónak a kiválasztásával egy párbeszédablakhoz jutunk, ahol az objektum körvonalait megrajzoló vonalak tulajdonságait (vastagság, szín, nyílhegy formája, stb.) állíthatjuk be.

8.7.2. Vonalkihúzó-szín párbeszédablak

Itt ugyanazt a párbeszédablakot érhetjük el, amellyel már korábban is találkoztunk, és ahol egyéni színeket keverhettünk ki különféle sémák szerint.

8.7.3. Nincs kihúzás

Ha erre az ikonra kattintunk, akkor eltűnik az objektumot határoló körvonalak kihúzása.

8.7.4. – 8.7.5. Vonalvastagságok

Ezekkel az ikonokkal gyorsabban választhatunk vonalvastagságot a kihúzáshoz, a hajszálvékonytól a 24 pontosig.

A *µlógia* sorozat eddig megjelent tagjai

1. Horváth László - Szlávi Péter - Zsakó László: Modellezés és szimuláció
2. *Szlávi Péter - Zsakó László: Programozási forgácsok - KÖMAL feladatmegoldás*[2]
3. *Turcsányiné Szabó Márta: A Commodore-64 Terrapin LOGO leírása*
4. Szlávi Péter - Zsakó László: Módszeres programozás: Rekurzió
5. Szlávi Péter: A számítógépről népszerűsítő stílusban
6. Zsakó László: Módszeres programozás: Hatékonyság
7. Makány György: Programozási nyelvek: PROLOGIKA
8. *Szlávi Péter: A programkészítés technológiája*
9. Szlávi Péter - Zsakó László: Szimulációs modellek a populációbiológiában
10. Pintér László: Programozási tételek rekurzív megvalósítása
11. *Temesvári Tibor: Alkalmazói rendszerek: QUATTRO PRO 3.0*
12. Szlávi Péter - Zsakó László: Módszeres programozás: Adatfeldolgozás
13. Krammer Gergely: Turbo Grafika
14. Pap Gáborné - Szlávi Péter - Zsakó László: Módszeres programozás: Szövegfeldolgozás
15. Pintácsi Imréné - Siegler Gábor - Zsakó László: Szimulációs modellek a kémiában
16. Horváth László - Szabadhegyi Csaba - Szlávi Péter - Zsakó László: Függvényábrázolás
17. Szabadhegyi Csaba - Szlávi Péter - Zsakó László: Szimulációs modellek a fizikában
18. Szlávi Péter - Zsakó László: Módszeres programozás: Programozási bevezető
19. Szlávi Péter - Zsakó László: Módszeres programozás: Programozási tételek
20. *Hack Frigyes: Számítógéppel támogatott problémamegoldás*
21. Szlávi Péter - Temesvári Tibor - Zsakó László: Módszeres programozás: A programkészítés technológiája
22. Szlávi Péter – Temesvári Tibor – Zsakó László: Programozási nyelvek: Alapfogalmak
23. Illés Zoltán: Programozási nyelvek: C++
24. *Szlávi Péter - Temesvári Tibor - Zsakó László: Programozási nyelvek: ELAN*
25. *Ökrös László: Programozási nyelvek: Az LCN LOGO leírása*
26. *Hack Frigyes: Informatika*
27. Pap Gáborné - Szlávi Péter - Zsakó László: Módszeres programozás: Rekurzív típusok
28. *Hack Frigyes: Programozási nyelvek: BASIC*

---

[2]A *µlógia*-sorozat dőlt betűkkel szedett tagjai már nem kaphatók.