# tanulmányok 172/1985

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKAÉDMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

# A SZÁMÍTÓGÉPES GRAFIKA TERÜLETKITÖLTŐ ALGORITMUSAI

Irta:

*REVICZKY JÁNOS*

Tanulmányok 172/1985

A kiadásért felelős:

*DR. VÁMOS TIBOR*

Főosztályvezető:

*NEMES LÁSZLÓ*

ISBN 963 311 193 5

ISSN 0324 - 2951

SZÁMALK Repro 85/179

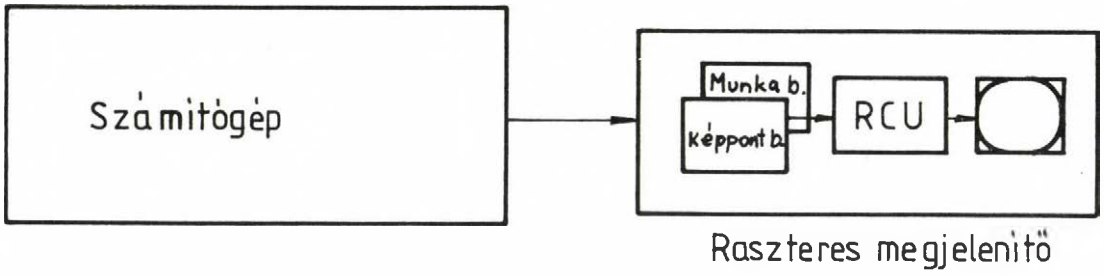
Tartalomjegyzék

1. Bevezetés . . . . . 4
2. Seed Fill algoritmusok . . . . . 7
2.1 Simple 4-Fill . . . . . 9
2.2 Tint Fill . . . . . 10
2.3 Lieberman Fill . . . . . 13
2.4 Graph Fill . . . . . 17
2.5 Contour Fill . . . . . 21
2.6 Distante & Veneziani Fill . . . . . 24
3. Parity Check algoritmusok . . . . . 26
3.1 A1 módszer . . . . . 27
3.2 A2 módszer . . . . . 28
3.3 A3 módszer . . . . . 30
3.4 A4 módszer . . . . . 33
4. Edge Fill algoritmusok . . . . . 34
4.1 Edge Fill . . . . . 35
4.2 Vektor Edge Fill . . . . . 37
4.3 Fence Fill . . . . . 39
4.4 Pairwise Fill . . . . . 40
5. Dekompozíciós algoritmusok . . . . . 41
5.1 Brassel & Fegeas dekompozíció . . . . . 42
5.2 Lane,Magedson & Rarick dekompozíció . . . . . 45
5.3 Schachter dekompozíció . . . . . 47
6. Ordered Edge List . . . . . 50
6.1 OEL . . . . . 51
7. Irodalomjegyzék . . . . . 53

## 1. Bevezetés

Ebben a tanulmányban egy adott zárt poligon belsejének a kitöltésével fogunk foglalkozni. Az algoritmusok többsége alapvetően raszteres megjelenítőt feltételez, de olyan algoritmus is van, amely vektorrajzolós eszközön is alakalmazható.

Az eljárások egyik része a raszteres megjelenítők frame buffer-ének képpontjain (pixeljein) dolgozik, amely már a képernyőre rajzolt kép bit térképe. Az algoritmusok ez esetben gyakran egy hasonló felépítésű munkabuffert ,vagy kitüntetett szinindexet is használnak. Az eljárások másik része még a megjelenítő előtt, a "host" számítógépben tárolt modell adatok alapján dolgozik és állítja elő a poligonkitöltést.




Poligon kitöltésen olyan eljárást értünk amely a poligon belsejét egy szinnel (vagy egyféle tónussal) tölti ki, míg a külsejét változatlanul hagyja. Az előzőt tömör (solid) kitöltésnek is nevezhetjük. Léteznek olyan kitöltések is,amelyek a poligon belsejét megadott mintázat ismétlésével töltik fel.

Az irodalomból ismert kitöltéseknek lehetnek korlátai is. Éppen ezért meg fogjuk vizsgálni minden egyes algoritmusnál hogy milyen poligonokra alkalmazhatók. A területkitöltés szempontjából a poligonoknak néhány osztályát célszerű megkülönböztetni (amelyek nem diszjunktak) :

tetszőleges (akár önmagát metsző poligon)
poligon lyukakkal
4-connented poligon (definiciót lásd később)
8-connected poligon
pozitív irányítással reprezentált poligon (lyukak negatív irányúak)
konvex poligon.

Ebben a dolgozatban három szempont ( használja-e közvetlenül a bit térképet, alkalmas-e tetszőleges mintával való kitöltésre, és végül milyen poligonosztályra alkalmazható a kitöltési eljárás ? ) figyelembe vételével fogjuk megvizsgálni a továbbiakban vázolt algoritmusokat.

A jelenleg előforduló eljárások alapvetően hat csoportba oszthatók:

Seed Fill :

Meg kell adni egy belső pontot, amelynek segítségével meghatározza az összes belső pontot. [3,5,7,10,12,13]

Parity Check:

Mivel belső pontból húzott vizszintes félegyenes páratlan pontban metszi a poligont, ezért egy scan line-on végighaladva minden páratlanadik kontúrpont esetén szint váltunk. [9,10]

Edge Fill :

Félegyeneseket húzunk a határoló szakaszok pontjaiból. A belső ponton páratlan, míg a külső ponton páros egyenes fog áthaladni. [1,4]

Dekompozíciós :

A poligont kisebb, könnyebben kezelhető (pl. konvex) részekre osztja és erre alkalmazza valamelyik másik eljárást. [2,6,10,11]

Ordered Edge List :

Minden egyes scan line-t elmetszünk az összes poligonoldallal. A metszéspontokat rendezzük és az elsőtől a másodikig, a harmadiktól a negyedikig stb. berajzoljuk a vizszintes szakaszokat. [8]

Vegyes :

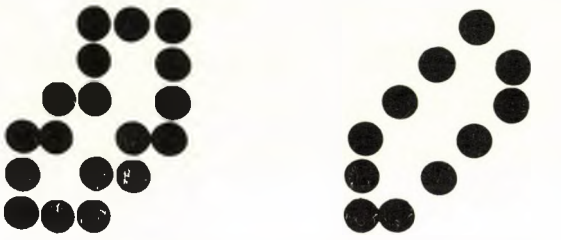
Valamely fentiekhez közel álló, de attól lényegesen eltérő vagy kétfélét egyszerre használó algoritmusok. Az ismertetésben a hozzá legközelebb álló csoportnál ismertetjük, hangsúlyozva az eltéréseket. [3,6,10]

Összesen 18 algoritmust mutatunk be, amelyek áttekintést adnak az irodalomban előforduló fontosabb kitöltési eljárásokról. Nem törekedtünk minden esetben a legapróbb részletek ismertetésére, célunk inkább az általános bemutatás volt.

## 2.Seed Fill

Ez az algoritmus típus kizárólag a bit térképen dolgozik, így már nem is láthatók közvetlenül a poligon oldalai (csak mint határ szinű pixelek). Meg kell adni egy belső pontot és ennek segítségével kitölti a poligont, azaz minden irányban addig terjeszkedik, amíg határponthoz nem ér. Ebből is látható, hogy önmagát metsző alakzat esetén nem működik az eljárás, csak ha két (vagy több) belső pontot adunk meg. Az algoritmus hátránya (némely esetben előnye), hogy egy belső pontot kell megadni a kitöltéshez. Az eljárás némely esetekben igen hasznos (pl. rajzfilm készítés stb), hiszen valamely input eszköz segítségével kijelölt területet önmagában "festi" ki. Az algortimusok egy részében megszorítást kell tenni a határ és a belső rész szinét illetően. Minta kitöltésre sem mindegyik eljárás alkalmas. Mivel a kitöltést nem a modellen végzi, hanem a frame bufferen, ezért ha a poligont más alakzat metszi akkor a kitöltés nem lesz teljes. Ez ellen ideiglenes kitüntetett kontúrszin választásával , vagy munkabufferben végrehajtott kitöltéssel és másolással lehet védekezni. Ezek a problémák viszont már alkalmazásfüggőek. Így például rajzfilmfigura estén nem lépnek fel, mert egy kifestési lépés éppen határtól határig tart (értelemszerűen nincsenek egymást metsző alakzatok). Kétféle típusú alakzat fordulhat elő: 4-connected és 8-connected. Az első típusú alakzatnál bármely belső pontból bármely belső pontba eljuthatunk vizszintes és függőleges irányú mozgások egymásutánjával. A 8-connected alakzatok esetében bármely belső pontból bármely belső pontba való eljutáshoz mind a nyolc irányt fel kell használni (1. ábra). Ugyanakkor a nyolc irány felhasználásával belső pontból külsőbe nem juthatunk el. Az alakzat belső típusa természetesen függ az egyenes reprezentációjától.

Ugyanakkor egy poligon határa is lehet 4 vagy 8-connected. ( A definíció hasonló az előzőkhöz.) Így egy 4-connected határú alakzat belseje 8-connected.




1. ábra

A továbbiakban vázolt algoritmusok 4-connected alakzatokra működnek, de egy részük átalakítható 8-connected alakzatra is. A gyakorlatban elegendő 4-connected alakzatok kitöltése, tekintve hogy az egyenes rajzoló eljárások 8-connected (de nem 4-connected) poligonhatárokat produkálnak. Természetesen a Seed Fill algoritmusok mindegyike csupán raszteres eszközön valósítható meg.

## 2.1 Simple 4-Fill

Egy megadott belső pontból elindulunk mind a négy irányba és bejárjuk a megadott területet. Ezt rekurzív hívásokkal tehetjük meg. A határpontok pixel értékei legyenek "boundary-value", míg a belső pontokat "new-value" értékre akarjuk beállítani. Ekkor az eljárás a következő:

```
procedure BOUNDARY_FILL_4 (x,y)
begin
    if   GET_PIXEL(x,y) <>  boundary_value
       and   GET_PIXEL(x,y) <> new_value
    then
       begin
         SET_PIXEL(x,y,new_value)
         BOUNDARY_FILL_4 (x,y-1)
         BOUNDARY_FILL_4 (x,y+1)
         BOUNDARY_FILL_4 (x-1,y)
         BOUNDARY_FILL_4 (x+1,y)
       end
end
```

A fenti algortimus 4-connected alakzatot tölt ki egy szinnel (new-value). Az algoritmus könnyen átírható 8-connected alakzatokra is,ekkor nem négy, hanem nyolc rekurzív hívás szükséges. Az eljárás egyszínű (solid) kitöltésre alkalmas. Nincs megkötés a határszin és a kitöltési szint illetően, azaz lehet new_value=boundary_value is. Nagyon nagy hátránya hogy nagyobb területű alakzatok esetén rengeteg elem kerül fel a stack-re és így akár a stack is elfogyhat. Másik hátránya, hogy pontonként tölti fel a frame buffert. [5]

## 2.2 Tint Fill

A megadott belső pontból soronként haladunk. A kezdőpontból először jobbra majd balra megyünk, addig amíg határponthoz nem érünk. Ez után megvizsgáljuk a felette és alatta levő sort, pontosabban az előzőleg kitöltött szakasz feletti ill. alatti szakaszt, és balról jobbra a határ utáni helyeket (amelyek még nincsenek kitöltve) feltesszük a stackre. (A legbaloldalibb értéknél a felette levőt.) Ez után a stack-en levő pontokat tekintjük belső pontoknak és ezekre csináljuk meg ugyanazokat a lépéseket. Ezt mindaddig csináljuk, amíg a stack-en van érték. (2. ábra) Mivel egyszerre három sort vizsgálunk new_value és boundary_value -re, ezért jelen esetben a két érték nem lehet egyenlő. Csupán az eljárás befejezése után állítható egyformára a kettő, vagy különböző indexeket használunk ugyanarra a szinre mutatva. Hasonló okok miatt az algoritmus tetszőleges mintával való kitöltésre sem használható. Az alakzatban ugyanakkor tetszőleges számu lyuk is lehet, ami nem zavarja az algoritmus menetét. Ha a képernyő a határ felrajzolása előtt homogén (azaz a határon belüli értékek mind old_value -k lesznek) , akkor az

if GET <> boundary_value then SET utasítás helyett a

if GET == old_value then SET utasítás kerül.

Ebben az esetben a terület tetszőleges mintával (pattern) is kitölthető, de a mintában nem fordulhat elő old_value érték, tekintettel az előbb felírt utasításváltásra.

Az eljárás "antialiasing" -gal rajzolt kontúr esetén is működik. (Ez azt jelenti hogy némely pixeleket szürkére is állítunk annak érdekében hogy az egyenesnek szebb, simítottabb képe legyen .)
Ez esetben kezdetben a terület fehér, míg a határa szürke ill. fekete. (Pontosabban old_value szinű.) Azt szeretnénk, ha a terület belseje más szinű lenne (new_value), de a határon tartaná a kisimítást (antialiasing) . Ezek után lényegében az előző algoritmust hajtjuk végre kisebb változtatásokkal. Úgy képzeljük el, hogy a kép domborzati viszonyokat reprezentál, ahol a világos területek a magas részeket, míg a sötétebbek az alacsonyabb részeket jelzik. (Minden pixelhez két érték tartozik: az egyik a szinértéke, ez kezdetben old_value, a másik a szürkesége, ami a kitöltés során nem változik.) A kezdőpontból előbb jobbra, majd balra haladunk amig a "magasság" csökken és ezeken a helyeken a szinértékeket new_value-re állítjuk. Ez után az előzőben kitöltött szakasz feletti ill. alatti szakaszt vizsgáljuk. A "legmagasabb" pontokat tesszük a stack-re, pontosabban a következő négy kritériumnak eleget tevő pontokat:

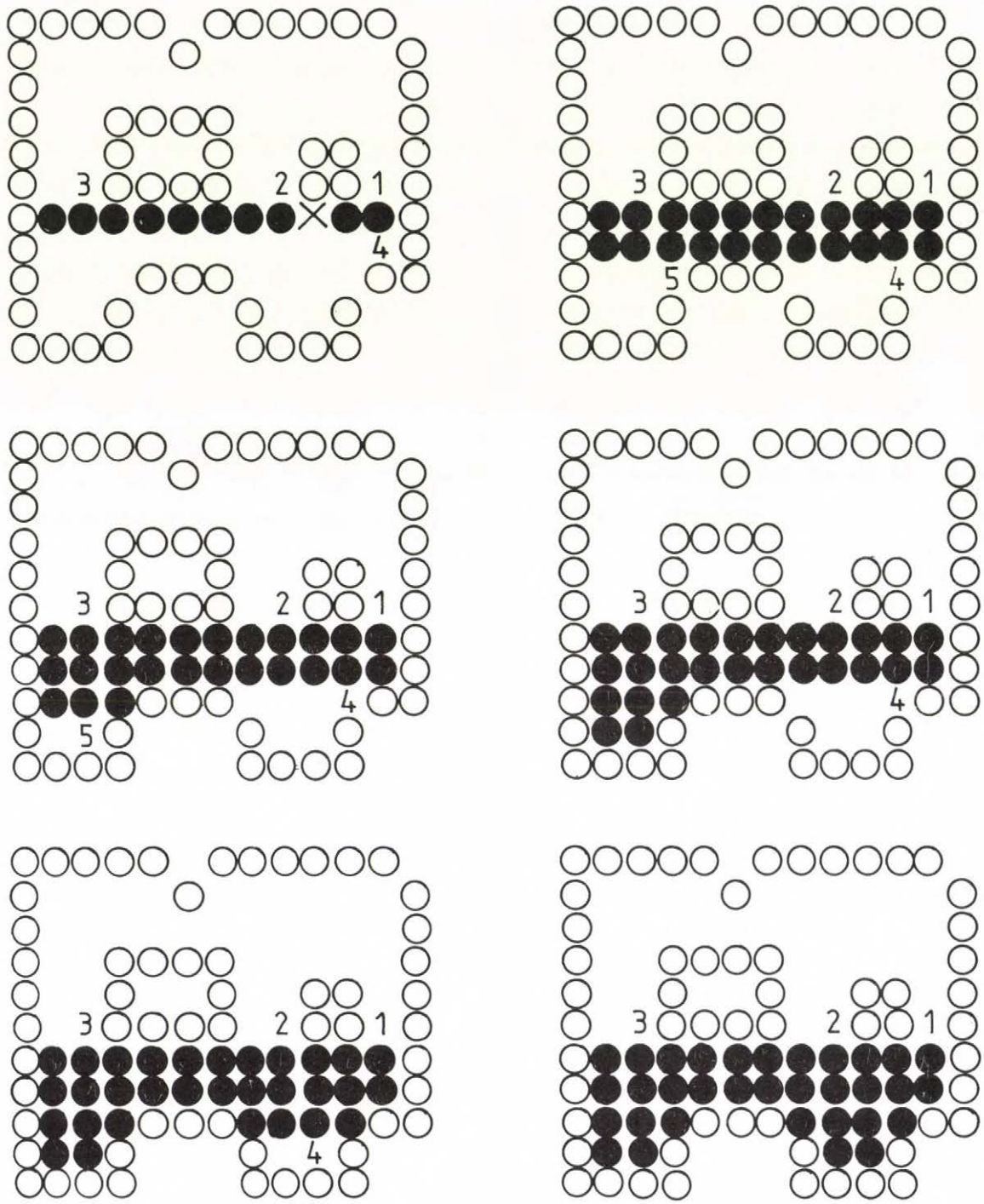
a, a pont szin értéke old_value

b, a pont "magassága" kisebb vagy egyenlő a felette levő pixelénél lefelé történő vizsgálatnál ( ill. az alatta levő pixelnél felfelé történő vizsgálatnál)

c, nincs vele egy sorban olyan pixel jobbra, amely monoton növekedve elérhető a pontból és a,b -t kielégíti

d, nincs vele egy sorban olyan pixel balra, amely monoton növekedve elérhető és már a stack-en van

Ezek után sorbavesszük a stack-en levő elemeket és ezeket tekintjük kezdő pontoknak és az előzőkben leírt eljárást ismételjük,amíg csak lehet (amíg van a stack-en elem). [13]



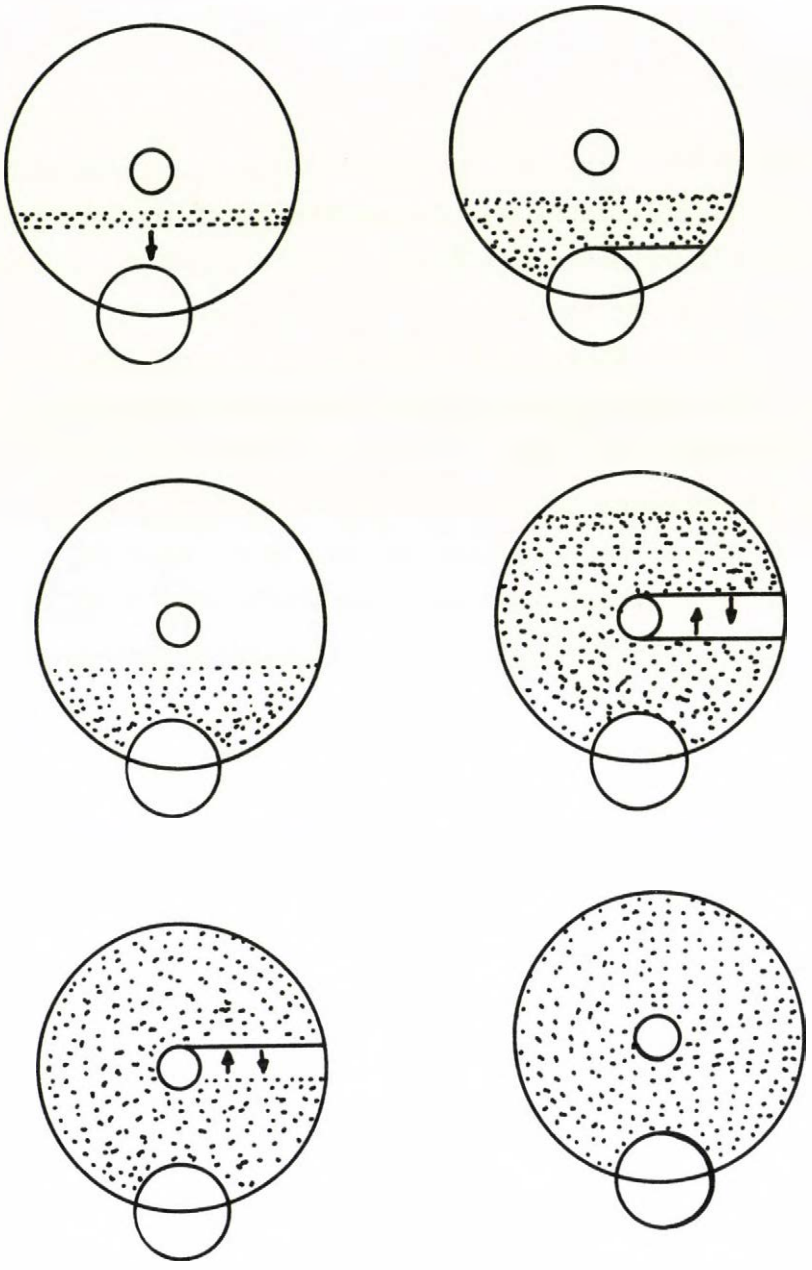

2. ábra

## 2.3 Lieberman Fill

Itt is soronként haladunk, de megpróbáljuk figyelmen kívül hagyni, hogy eddig már mit töltöttünk ki. A megadott belső pontból jobbra és balra haladunk,amíg határhoz nem érünk. Mindig csak egy irányban haladunk ,amíg csak lehet. Ha már egy megadott irányban sehol sem tudunk haladni, akkor irányt váltunk. Minden sor vizsgálatakor megnézzük hogy van-e U (iránytartó szétválás), vagy S (irányváltó szétválás) kanyar. A kanyar ténye úgy állapítható meg, hogy az ujonnan töltött szakasz rövidebb mint az előző és a különbségen nemcsak határpontok vannak. Ekkor ezen a különbségen ideiglenes határvonalat húzunk. Ezzel kerüljük el a végtelen ciklust,amit az okozna,hogy most nem vizsgáljuk,hogy az új területet már kitöltöttük-e. (A végtelen ciklus lyukas alakzatnál fordulna elő. )
Két feljegyzés halmazunk van, külön felfelé és külön a lefelé irányra vonatkozóan. Minden lépésnél U kanyar esetén ugyanazon irányú listára kerül elem,míg S kanyar esetén a másik irányhoz tartozó listára kerül fel elem. (Mindkét esetben ideíglenes határt is húzunk, meghatározott értékkel.)
Az algoritmust mindaddig folytatjuk,amíg az adott irányra az összes szomszéd csupa határpont vagy ideíglenes határpont. (Csupa ideíglenes határpont esetén a hozzá tartozó pontot leszedjük a megfelelő listáról.)
Ebben az esetben tetszőleges mintával is tölthető az alakzat,hiszen sehol sem vizsgáljuk, hogy már kitöltött területre értünk-e. Ugyanakkor két kitüntetett érték kell: a határhoz és az ideíglenes határ húzásához, és ezek nem lehetnek egyenlők és ezen szineket a mintázat sem használhatja.
Hasonlóan az előző algoritmushoz itt is alkalmazható az

```
if  GET == oldvalue  then SET
```

utasítás, amivel elérhető,hogy tetszőleges minta-kitöltés is használható, csupán a mintában old_value nem szerepelhet.[7]



3. ábra

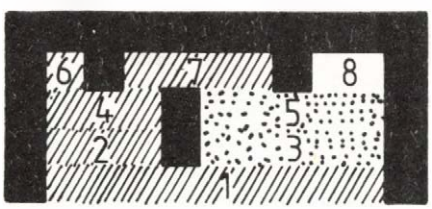
A kitöltés menetét a 3. ábra mutatja. Az algoritmus menetét a következő leírás tartalmazza:

```
FILL()
 { a listák inicializálása }
 repeat  {amíg a feljegyzés nem üres mindkét irányra}
    if  {az adott irányra üres}
    then  { irányváltás }
    {az adott irányú feljegyzésről egy elem választása }
      SHADE_VERTICAL()

SHADE_VERTICAL()
   (argumentumok: irány és kezdőpont)
  repeat  {amíg az előző kitöltött szakasz mellet csak határ vagy
           ideíglenes határ van}
     SHADE_HORIZONT()
     LOOK_E_TURNS()
     { a kezdőpont eggyel felfelé vagy lefelé megy iránytól függően}
```

```
SHADE_HORIZONT()
     {a kezdőpontból jobbra és balra haladunk,amíg a határhoz nem érünk}
     { megjegyezzük a két határpontot }
     { a két határpont közt feltöltjük a megadott mintával }

LOOK_F_TUNRS()
     LOOK_S_TURNS()
     LOOK_U_TURNS()

LOOK_S_TUNRNS()
    if  {a most töltött szakasz rövidebb, mint az előzőleg kitöltött
         és a különbségen nemcsak határpontok vannak }
    then  { ezek ideiglenes határt alkotnak és felkerülnek az azo-
          nos irányú listára}

LOOK_U_TURNS()
    if  {a most töltött szakasz hosszabb, mint az előzőleg kitöltött
         és a különbségen nemcsak határpontok vannak  }
    then  { ezek ideíglenes határt alkotnak és felkerülnek az ellentétes
            irányú listára  }
```

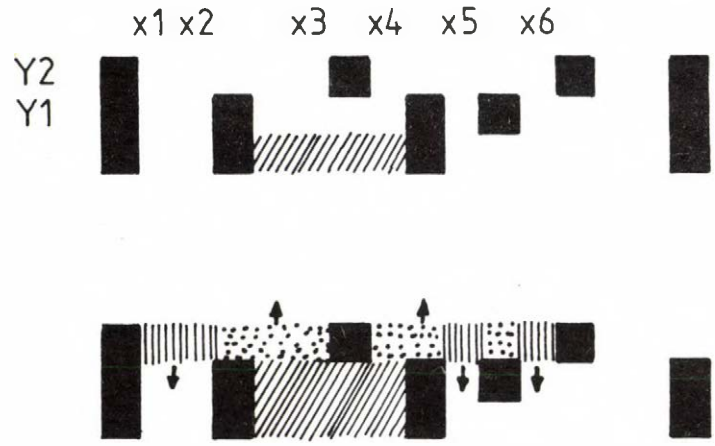
## 2.4 Graph Fill

Egy poligon belsejét többféleképpen is definiálhatjuk. Egy R poligon lehet (x,y) koordinátáju pixelek halmaza, másrészt lehet $(y,x_1,x_2)$ hármasok halmaza is, ahol ez a hármas azon (x',y) koordinátáju pixeleket tartalmazza,amelyre $x_1 < x' < x_2$ . Ezt a hármast scan szegmensnek nevezzük. Így az R poligont scan line darabok halmazára bontottuk szét. Egy R poligont egy síkbeli irányított gráffal is reprezentálhatunk, ahol a csúcsok a scan szegmenseket jelölik és a szomszédos, érintkező részek össze vannak kötve (a nagyobb y értékhez tartozóba mutat az él) . Egy kitöltés azt jelenti,hogy bejárjuk a gráfot, tetszőleges megadott pontból indulva. Az előző algoritmus is ezt a gráfot próbálta bejárni, de túlságosan elágazó és lyukas alakzat esetén hibázhat. (A gyakorlatban előforduló poligonoknál ez csak igen ritkán fordul elő). Nézzük ugyanis a következő példát:




4. ábra

Az első lépésben az 1-gyel jelölt scan szegmenset fogja kitölteni és a 3-t felteszi a stack-re, ami a felfelé irányhoz tartozik. (Ideiglenes határ.) Ez után a 2,4 és 6-os scan szegmenseket festi. Így jut el a 7-es scan szegmensig,amikoris a 5-os scan szegmenset a lefelé irányhoz tartozó stack-re teszi.Amikor a 3-as scan szegmenset leveszi a stack-ről és festi ,akkor eljut az 5-öshöz, amit szintén kitölt és törli a másik stack-ről (mert ideiglenes határhoz jutottunk). Ezzel be is fejezte a kitöltést, így a 8-as scan szegmenset sohasem fogja feltölteni (4. ábra).
Ezt a hibát javítja ki ez az algoritmus, amely kissé hasonlóan működik mind az előző. Bevezetjük a blokkolás fogalmát, amivel területeket zárunk el egymástól, "határszinű" egyeneseket húzva. Egy stack-et használunk amelyre a kurrens irányú elemeket tesszük fel , míg az ellenkező irányú blokkolt elemeket a stack aljára tesszük.Ebből is látható,hogy blokkolást csak ellenkező irány esetén alkalmazunk és csak az érintett pixeleket. Egyébként azokat a lépéseket hajtjuk végre, mint az előző algoritmusnál. Alapvető különbség az előző algoritmushoz képest, hogy csak irányváltó elágazás esetén húzunk ideiglenes határt és akkor is csak az éppen szükséges pixeleknél. [12] Lássunk egy példát:




5. ábra

Ezek után a gráf bejárásának a menete a következő:
(S : stack; p,q,r : gráf csúcsok; dir : irány változó )

```
Block(p);
PushOnBottom(Empty-Stack, p);
PushOnTop(S, p);
WHILE not  Empty(S)   DO
      q :=  Pop(S);
      dir := DirectionOf(q);
      if  q  blocked  then
              PaintArc(q);
              q := NodeOf(q, dir);
              PaintNode(q);
    FOR  all  r  connected to q
             in direction  dir  DO
          PaintNode(r);
          PushOnTop(S,r);
    FOR  all  r  connected to q
             in direction  -dir  DO
           Block(r);
           PushOnBottom(S,r);
    FOR  all blocked arcs o
              leading to q  DO
           Remove(o, S);
           PaintArc(o);
STOP;
```

A meghívott függvények a következőket végzik:

PaintNode(q) a csúcs (scan szegmens) töltését jelenti

Remove(o,S) meghatározott elem levétele a stack-ről

Block(o) blokkolás határszinnel

PaintArc(q) egy blokkolt rész kitöltése

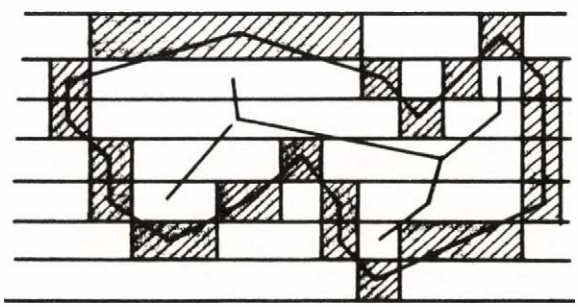
NodeOf(q,dir) a blokkolt q -hoz tartozo csúcs (scan szegmens).

Az algoritmus realizációja esetén a megadott példában (5. ábra) a következő dolgokat tesszük:

```
PushOnTop(y2, x1, x3, UP);
PushOnTop(y2, x4, x6, UP);
PushOnBottom(y2, x1, x2, DOWN, BLOCKED);
PushOnBottom(y2, x5, x5, DOWN, BLOCKED);
PushOnBottom(y2, x6, x6, DOWN, BLOCKED);
```

## 2.5 Contour Fill

Az előbbiekben az alakzat egy scan szegmensekből álló gráfját tekintettük (i-LAG), amelyeknél az összefüggő részek voltak éllel összekötve. Hasonlóan tekinthetjük a kontúr gráfot is (c-LAG), ahol éllel azok lesznek összekötve, amelyeknél a vizszintes kontúrrészek legalább sarokkal érintkeznek (6 ábra). Ez az algoritmus a i-LAG vizsgálata helyett a c-LAG -ot vizsgálja és így valósítja meg a kitöltést. Mivel a szomszédsági viszonyok megállapításához a felette és alatta levő soron kell keresnünk, ezért ez 3C + I pixelvizsgálatot jelent , szemben az előző algoritmussal, ahol ez az érték 3I + C ( C a kontúron levő pixelek száma , I pedig a belső pixelek száma ). Igy, ha C sokkal kisebb, mint I, akkor ez az algoritmus lényegesen gyorsabb is lehet.



6. ábra

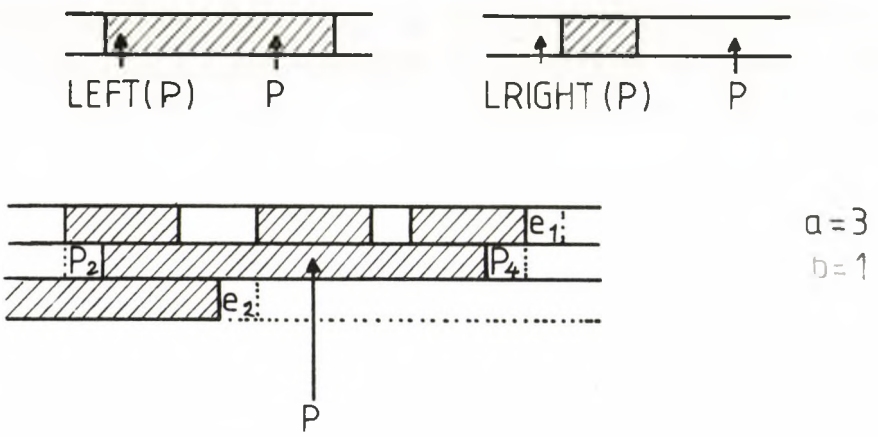
Az eljárás a következő műveleteket használja:

LEFT(p) : a legbaloldalibb, p-vel egy sorban levő, azonos szinű pixel

LRIGHT(p) : a legjobboldalibb, p-vel egyszinű pixel ami p-től balra van és p és e között legalább egy más szinű pixel is van

( = LEFT(LEFT(p) -1)-1 )

LINK(p) : v=(a,b,$p_1$,$e_1$,$e_2$) vektorral tér vissza, ahol a és b a gráf csúcspontja feletti és alatti élek száma, míg a többi értéket a 7. ábra szemlélteti.




7. ábra

Balról jobbra fogunk tölteni kontúrponttól kontúrpontig. (Először LEFT(seed) -del keressük meg a bal kontúr utáni aktuális értéket: legyen ez p. ( Mielőtt kitöltenénk a sort , LINK(p-1) -gyel vizsgáljuk meg a c-LAG -ot. Ha a=b=1 akkor nincs probléma (a poligon egyetlen oldaláról van szó). Jelölje a következő scan line induló elemét $p_{next}$ ami $e_1$ vagy $e_2$ -gyel egyenlő, attól függően, hogy az irány felfelé vagy lefelé mutat. A vizszintes kitöltés ezek után p-től kontúrpontig történik. Legyen az utolsó szinezett pixel $p_{right}$ . Ekkor LINK($p_{right}$ +1) -gyel a c-LAG -ot vizsgáljuk a másik oldalon. Ha itt is a=b=1, akkor semmi dolgunk sincs. Ez esetekben a kitöltés igen egyszerű. Ha a és b közül valamelyik 0, akkor szélsőértékhez értünk, ami egy pixel stack-re tevését eredményezi, vagy az adott irányban befejeződött a kitöltés. Ez a módszer már eléggé hasonlít lépéseiben a Parity Check algoritmusokhoz, hiszen ott is a kontúr gráfját vizsgáljuk (lásd 25. oldal). Felmerül a kérdés hogy ezt a két algoritmust nem lehetne-e egyszerre alkalmazni. A Seed Fill eljárások kényelmetlensége, hogy ismerni kell egy belső pontot. A belső pontot megkereshetjük a LINK eljárás segítségével. Addig vizsgáljuk a sorokat, amíg az első kontúrponthoz érve LINK(p) a=b=1 -t nem ad. Ekkor a kontúrpont utáni pont lesz a kitöltési eljárás kezdeti pontja, és így már tudjuk alkalmazni a fent leírt algoritmust.

## 2.6 Distante & Veneziani Fill

Ez az algoritmus már nem tartozik szigorúan véve a Seed Fill csoportba , inkább a vegyes-be, mégis azért soroltuk ide, mert ehhez áll a legközelebb. Ebben az esetben ugyanis speciális igényeink vannak. Adva vannnak ugyanis a képernyőn vonalakkal határolt összefüggő területek (pl. egy kontúrokkal megrajzolt térkép, azaz néhány ország határokkal feltüntetve ). A könnyebb áttekinthetőség kedvéért azt szeretnénk,ha végigjárva a frame buffert, minden összefüggő terület más szinű lenne, vagyis minden területet más szinnel töltenénk ki. Ez az eljárás két menetből áll.
Az első lépésben a pontokat osztályokba soroljuk , majd a másodikban az egy osztályban levő pontokat azonos szinűekre festjük. Az első menetben mindig két scan line-t vizsgálunk egyszerre és a következő szabályokat tartjuk szem előtt:

(1), nem kontúr utáni pixel azonos osztályban van az előzővel, vagyis bal szomszédjával (8. a,h ábra )

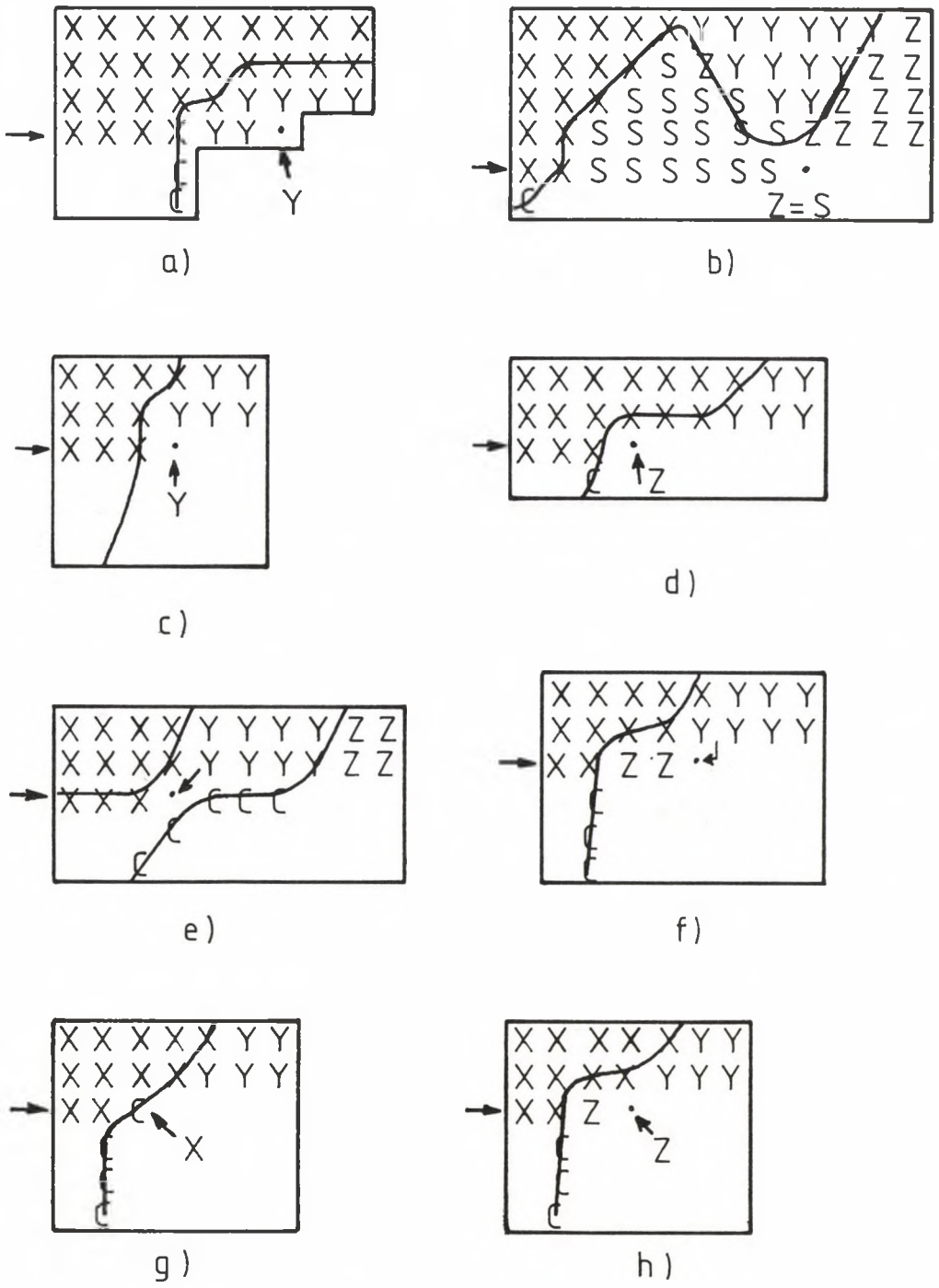
(2), ha a felette levő pixel más osztályban van mint az éppen kitöltött, akkor a két osztály ekvivalens (8. b,f ábra )

(3), kontúr utáni pixel megegyezik a felette levővel, ha az utóbbi nem kontúr (8. c ábra)

(4), kontúr utáni és alatti pixel új osztályt jelent (8. d ábra )

(5), ha (4) esetben az utána következő pixel is kontúr, akkor a jobb felső pixel értékkel egyezik meg (8. e ábra )

Mivel mindig két sort vizsgálunk, ezért a két sorban levő kontúrpontokat ismernünk kell a kitöltés után is. (A kitöltés ugyanis már eltünteti a kontúrértékeket.) A második lépésben meghatározzuk az ekvivalenciaosztályokat, mert az első pass után csak ekvivalencia párok állnak rendelkezésre. Ekkor ha sok bit per pixelünk van, akkor az ekvivalenciaosztályba tartozó indexek ugyanarra a szinre (vagy szürkeségre) mutatnak, vagy még egyszer végigmegyünk a frame bufferen és az ekvivalenseket egységesen jelöljük. [3]




8. ábra

## 3. Parity Check

Az algoritmusok ezen csoportja azt használja ki , hogy a belső pontból húzott félegyenes páratlan pontban, míg a külső pontból húzott félegyenes páros pontban metszi az alakzatot. Kizárólag bit térképen dolgozik (a nem bit térképen dolgozó hasonló változat az Ordered Edge List típushoz tartozik). Scan line-onként haladunk és minden páratlan kontúrpont esetén töltjük a megadott pixeleket a következő kontúrpontig. Egyes algoritmusoknál megszorítást kell tenni a poligon alakjára, ezenkívül a kitöltés eredményessége erősen függ az egyenes reprezentációjától is. Ha a poligont más alakzat metszi, akkor hasonló problémák jönnek elő, mint a Seed Fill csoportnál. Ez ellen itt is munkabuffer használatos, vagy kitüntetett szin választása segíthet. Természetesen a Seed Fill -hez hasonlóan ez az algoritmus típus is kizárólag raszteres eszközön valósítható meg. Elvi probléma itt nincs, akár önmagát metsző alakzat kitöltése esetén sem. Algoritmustól függően a kontúr szin és a kitöltési szin lehet egyenlő.

## 3.1 Al algoritmus

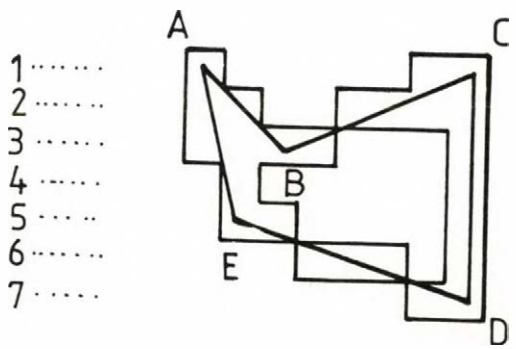
Soronként haladunk. Kontúr ponthoz érve növeljük a sor elején nullázott számláló értékét. Ha a számláló páratlan , ettől kezdve minden képpontot töltünk a kontúr (vagy más) értékkel. Ha a számláló páros, akkor abbahagyjuk ezt a töltést. Ez az eljárás csak akkor működik helyesen ha :

a, maximális és minimális értéknél (a felső és alsó csúcsban) páros pixel van egymás után (a 9. ábran C és D ilyen, de A nem.)

b, összelógó szakaszoknál (egy scan line-on levő kontúrszakaszok összemosódnak) az egymásutáni kontúrpixelek paritása megegyezik az összelógó szakaszok számának paritásával. (a 9. ábrán a második sorban ilyen van, de a harmadikban nem.)

c, nem összelógó szakaszoknál egy oldalnak a scan line-on elhelyezkedő kontúpontjai páros pixelből állnak. (ellenpélda a 9. ábrán a hatos sor.)

[9]




9. ábra

## 3.2 A2 algoritmus

Ez az eljárás megpróbálja kivédeni az előzőnek néhány hibáját. Most egyszerre három scan line-t vizsgálunk. Ha kontúrhoz érünk, megnézzük a felette és alatta levő sorban a most megtalált kontúrrészhez csatlakozó kontúrintervallumok számát. Csak ha mindkettő egy, akkor növeljük a számláló értékét. Ebben az esetben a belső pontokat más szinnel kell jelölni, hiszen három sort vizsgálunk. (A kitöltés befejezése után már átírhatjuk kontúr értékekre a kitöltött területet is.) Ez a módszer a fent említett a, és c, hibát kiküszöböli (felső és alsó csúcspontnál a felette ill. alatta levő kontúrintervallumok száma 0, míg nem összelógó szakasznál a felette és alatta levő kontúrintervallumok száma egyaránt 1), viszont a b, hibát nem javítja ki. Az eljárás menetét a következő programmséma mutatja ( L0 a határ értéke, L1 a kitöltésé, b(x,y) a pixel értéke, X,Y az x és y irányú határértékek ):

```
FOR y=0 to Y DO
 BEGIN
  count = 0
  x=0
  WHILE (x<X) DO
    BEGIN
      IF (b(x,y) ≠ L0)  THEN DO
          BEGIN
```

```
        IF  (count = odd) THEN  b(x,y)=L1
        INCREMENT(x)
      END
    ELSE DO
      BEGIN
        above=below=0
        IF (b(x-1,y-1) = L0)   INCREMENT(above)
        IF (b(x-1,y+1) = L0)   INCREMENT(below)
        WHILE (b(x,y) = L0) DO
        BEGIN
          IF ((b(x,y-1)=L0) & (b(x-1,y-1)≠L0))  INCREMENT(above)
          IF ((b(x,y+1)=L0) & (b(x-1,y+1)≠L0)) INCREMENT(below)
          INCREMENT(x)
        END
        IF ((b(x-1,y-1)≠L0) & (b(x,y-1)=L0))  INCREMENT(above)
        IF ((b(x-1,y+1)≠L0) & (b(x,y+1)=L0))  INCREMENT(below)
        IF ((above = 1) & (below = 1))  INCREMENT(count)
        IF ((above + below > 2) | (above + below = 1))
            print error message
      END
    END
END
```

## 3.3 A3 algoritmus

Az előző eljárás nem működött helyesen ha a b, feltétel nem teljesül. A gyakorlatban viszont a b, eset gyakran előfordul, így például ha két oldalegyenes hegyes szögben metszi egymást, az A2 algoritmussal könnyen lehet probléma. (Ilyen esetben az A2 módszer jelez. )
Itt is három scan line-t fogunk egyszerre vizsgálni, de jelen esetben minden kontúrintervallumot tipusától függően egy betűvel jelölünk meg l,c vagy r -rel. Csak l és r érték esetén növeljük a számlálót, c előfordulása esetén nem. Ezen értékeket a felette és alatta levő kontúrintervallumok számából kaphatjuk meg egy táblázat segítségével. ( I. táblázat) Igy például egy poligon legfelső és legalsó csúcspontjai c kontúrintervallumot alkotnak.
Ez az algoritmus azon b, esetben is helyes eredményt ad, ha két poligonoldal összelóg, de három oldal összelógása esetén nem mindig ad jó megoldást (ez a gyakorlatban már ritkább).

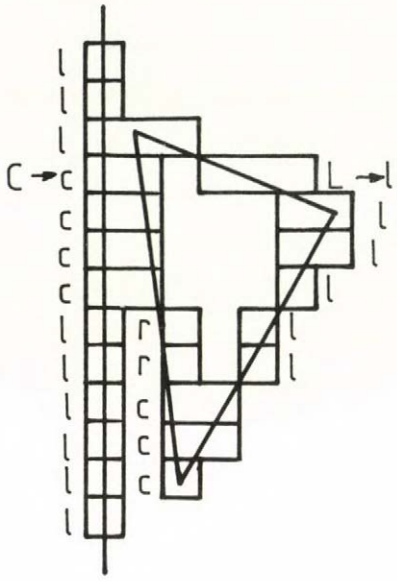
I TÁBLÁZAT

| ABOVE | BELOW | oldmark | newmarks |
|---|---|---|---|
| 0 | 1 | none | c |
| 0 | 2 | none | l,r |
| >0 | 1 | any | same as old |
| >0 | 2 | c | l,r |
| >0 | 2 | l | C,L |
| >0 | 2 | r | R,C |
| >0 | 3 | l | l,r,l |
| >0 | 3 | r | r,l,r |
| >0 | 3 | c | use previous marks |
| >0 | >3 | Print error message stop. | |

Az algoritmus menete a következő: a kontúrintervallumokat l,L,c,C,r vagy R -rel jelöljük. A számlálót csak akkor növeljük ha l, vagy r intervallumhoz értünk. Van egy kurrens scan line,ahol a kontúrintervallumok típusa már ismert és a következőn fogjuk ezeket meghatározni. Ezt az I.táblázat segítségével tesszük. Ha egy kontúrintervallum a következő sorban két típusjelet is kapna, akkor a II. Táblázatot használjuk. (Itt még kisebb módosítások is előfordulhatnak, de ezt most nem részletezzük.) Befejezve egy sor vizsgálatát a következő sor lesz a kurrens sor, és az L,C,R értékeket rendre l,c és r értékekkel helyettesítjük.
A kitöltés menetét a 10. ábra mutatja.

II. TÁBLÁZAT

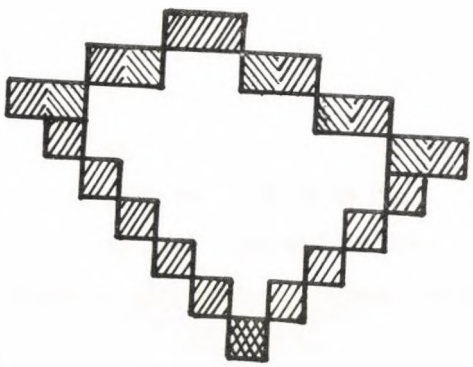
| Firstmark | Secondmark | | | | | |
|---|---|---|---|---|---|---|
| | l | L | c | C | r | R |
| l | C | - | l | l | c | l |
| L | l | - | l | r | l | c |
| c | l | - | c | c | l | c |
| C | l | - | c | c | r | c |
| r | c | - | r | r | C | r |
| R | c | - | r | r | c | C |



10. ábra

## 3.4 A4 algoritmus

Ez az eljárás azt célozza meg, hogy a kontúron úgy haladjon végig, hogy ez által az A1 eljárás helyesen fusson le. A kontúron haladunk pozitív körüljárási irányban (ezt úgy tesszük,hogy mindig figyeljük a kapcsolódó kontúrintervallumokat) és a lefelé haladó szakaszoknál a baloldali, míg a felfelé haladó részeknél a kontúrintervallum jobboldali pixelét jelöljük meg. Azt a pixelt amit kétfélén jelöltünk meg az külön jelzést kap (11. ábra). Ezek után az egyszer megjelölt pixeleket vesszük mint kontúrértékeket,és erre fogjuk végrehajtani az A1 algoritmust. Az egyenes előbb leírt átalakításához természetesen két újabb foglalt pixel értéket kell használnunk.



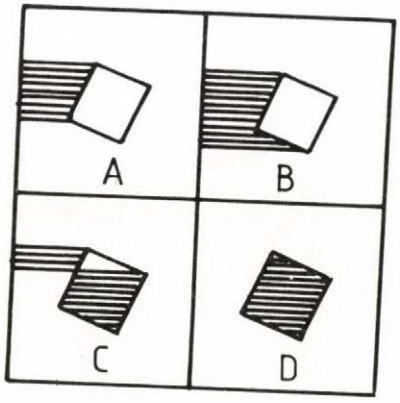
11. ábra

## 4. Edge Fill

Ha a poligon határoló pontjaiból vizszintes félegyeneseket húzunk, akkor belső ponton páratlan , míg külsőn páros számú félegyenes megy keresztül. Ezért ha a kitöltésnél XOR aritmetikát használunk , ennek segítségével kitölthetünk egy tetszőleges poligont. Az algoritmus a poligon modellt használja és nem közvetlenül a frame buffert, ahol már ez közvetlenül nem érhető el. Előnye hogy az alakzat tetszőleges poligon lehet akár lyukakkal is. Ekkor először az alakzatot töltjük ki, majd sorba megyünk a lyukak határán is.Hátránya hogy speciális egyenesreprezentációt használ. A XOR aritmetika következménye, hogy a poligonon kívüli pontok eredeti értéküket kapják meg az eljárás befejeztével,viszont ha az alakzat belsejében volt valami más, akkor a kitöltés végén ez "negatív"-ban fog megjelenni. Ez ellen munkabufferrel és az eredmény bemásolásával védekezhetünk. Ugyanakkor, ha például megfelelő hidden line algoritmussal együtt használjuk az eljárást (ami biztosítja hogy az alakzat belsejében nincs semmi), akkor az előző probléma nem jelentkezik. Az eljárás alapvetően raszteres megjelenítőre készült és igen nagy előnye még, hogy tetszőleges mintával való töltésre egyaránt használható. Önmagát metsző alakzatra is működik. Az egyes eljárásoknál még a következő problémák léphetnek fel: a határpixeleket hol tölti, hol nem ,végül némely algoritmusnál kis koordinátamódósítás is szükséges a helyes működéshez.

## 4.1 Edge Fill

A poligon határát úgy tekintjük,mint egy mozgássorozatot, ahol egy helyről a nyolc szomszéd valamelyikébe léphetünk. Az elmozdulást a (dx,dy) párral jellemezhetjük. Ha dy=1, akkor azt mondjuk hogy NORTH mozgás van, míg ha dy=-1, akkor SOUTH mozgásról beszélünk (függetlenül dx -től). A WEST és EAST mozgásokat a (0,-1) ill. a (0,1) párok jellemeznek. Ezek után olvassuk a poligon határának előbb leírt sorozatát és NORTH vagy SOUTH mozgásnál invertáljuk a 0-tól addig a pixelig tartó félegyenest. Ez az eljárás a poligon felső és alsó extremális pontjain fog csak hibázni,ezért ha a koordinátarendszert 1/2- vel eltoljuk (azaz a megadott értékek nem a pixelközéppontokat hanem a pixelhatárrács pontjait jelzik), akkor az algoritmus helyesen fog működni (12. ábra ).
[1,4]




12. ábra

Az algoritmus menete a következő:

```
PROCEDURE edge_fill;
   VAR x,y,dx,dy,i:INTEGER;
   PROCEDURE invert_scan(x1,x2,y:INTEGER);  BEGIN
      IF  x1<x2  THEN
         FOR  i:= x1 + 1  TO  x2  DO  invert_pixel(i,y);
      ELSE
         FOR  i:= X2 + 1  TO  x1  DO  invert_pixel(i,y);
      END;

   PROCEDURE clear(x,y:INTEGER);  BEGIN
      invert_scan(0,x,y);
   END;

   BEGIN
      read(x,y);
      WHILE  not end_of_file  DO  BEGIN
        read(dx,dy);
          CASE  Class(dx,dy)  OF
                    east:  NOP;
                    north:  clear(x + dx/2,y);
                    west:  NOP;
                    south:  clear(x + dx/2,y-1);
          END
        x := x + dx;
        y := y + dy;
      END
    END
```

## 4.2 Vektor Edge fill

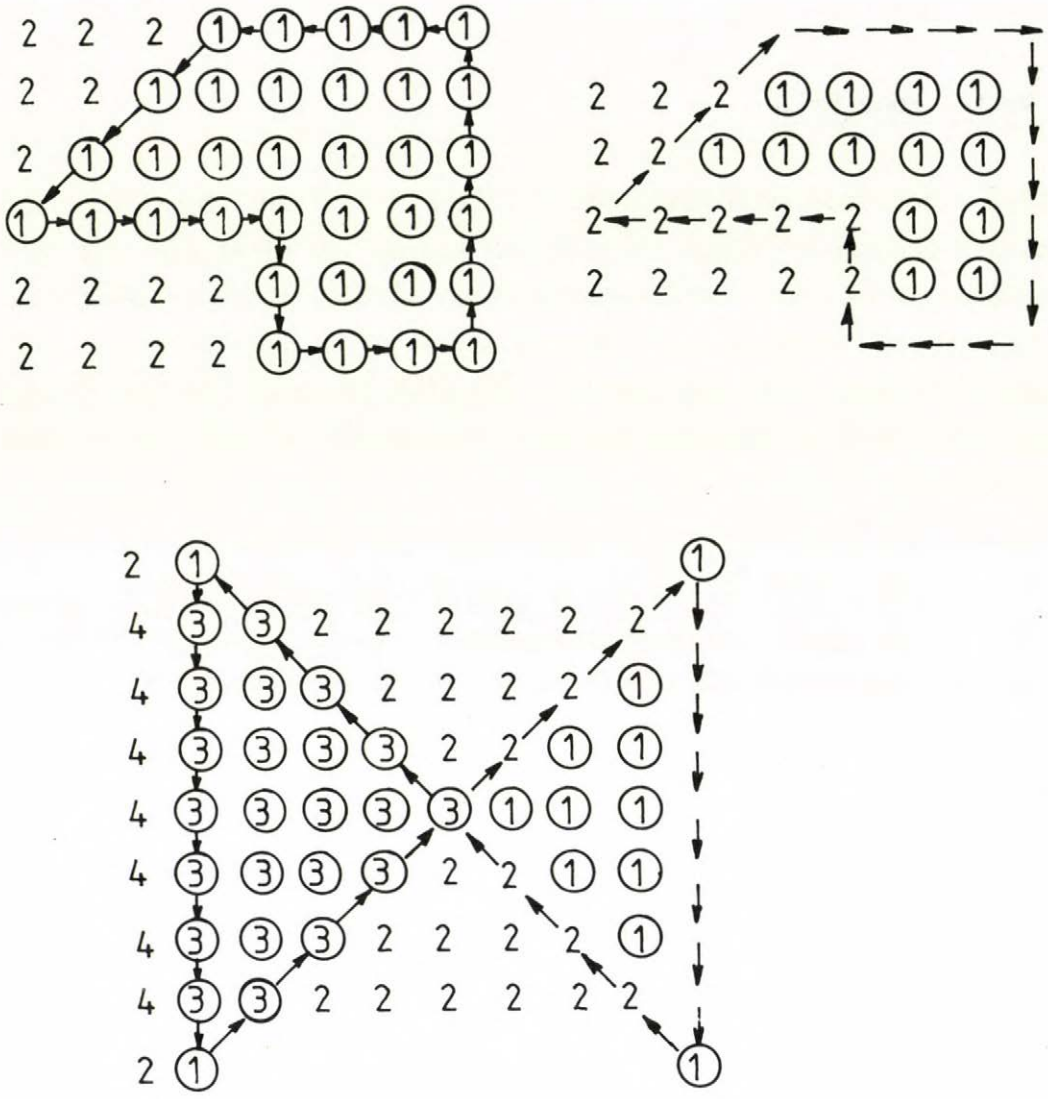
Az előző algoritmus azon hátrányát próbálja kijavítani, hogy nem rendes koordináta rendszert használtunk, hanem 1/2-del eltóltat (ez még kis pontatlanságot is okozhat a kényelmetlenségen kívül). Ebben az esetben a poligont pozitív körüljárási irányban rajzoljuk és egyszerre két lépést vizsgálva döntünk, hogy mit teszünk. (Az első lépésnél még nem teszünk semmit, csak ha újra visszaértünk erre a helyre.) Négyféle akció közül választhatunk:

a, C az y scan line 0-tól x-ig történő invertálása
b, Cl az y scan line 0-tól x-1 ig történő invertálása
c, I az (x,y) pixel invertálása
d, N nem csinálunk semmit

[4]
A táblázat a következő:

| /OLDCLASS/ | North | West | South | East | /NEWCLASS/ |
|---|---|---|---|---|---|
| East | C | C | N | N | |
| North | C | C | I | N | |
| West | N | N | Cl | Cl | |
| South | I | N | Cl | Cl | |

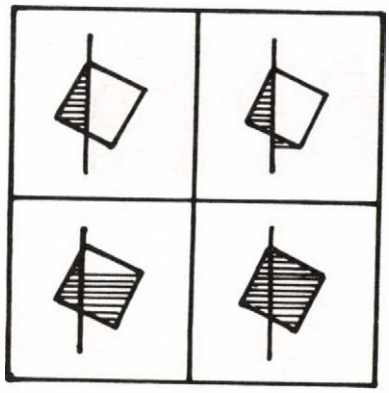
Ezt felhasználva elvégezhetjük a kitöltést. Pozitív irányításu alakzatoknál a határ is töltve lesz, míg negatív irányítás esetén a határ nem lesz töltve, csupán a két extremális pont. Ezért önmagát metsző alakzat esetén az alakzat egyik részén töltve lesz a kontúr,míg másik részén nem. A kitöltést a 13.ábra mutatja, a számok az invertálások számát jelzik.



13. ábra

## 4.3 Fence fill

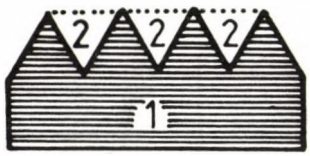
Ha egy scan line több oldalt metsz, akkor egyes pixeleket sokszor kell invertálni, valamint ha a poligon a képernyő méretéhez képest kicsi, akkor sok pixelt invertálunk a területen kívül fölöslegesen. Ez a felismerés vezet el ahhoz,hogy az invertálást ne 0-tól végezzük, hanem egy előre megválasztott értéktől. Így például, ha ez a megválasztott érték min x, akkor jelentősen csökkenthetjük a fölöslegesen invertált pixelek számát. Ezt az értéket a poligonon belül is megválaszthatjuk (14. ábra). [4]



14. ábra

## 4.4 Pairwise fill

Ennél az algoritmusnál azokat a fölösleges invertálásokat szeretnénk kiküszöbölni, amikor egy scan line sok poligonoldalt metsz. Ehhez szükségünk van egy Q[y] tömbre, amely minden sorra tartalmazza a kontúr utolsó NORTH-SOUTH -vagy SOUTH-NORTH irányú áthaladásának helyét , és így scan line-onként tartalmazni fog egy-egy x értéket. Abban az esetben ha egy NORTH vagy SOUTH lépéssel metszünk egy scan line-t, akkor az így fellépő x értéket betesszük a Q[y] tömb megfelelő elemébe. Újabb metszés esetén kivesszük a tömbben levő értéket, és csak ettől az értéktől fogunk invertálni. A 15. ábra ennek az eljárásnak a realizációját szemlélteti, ahol a számok az invertálások számát jelzik.[4]



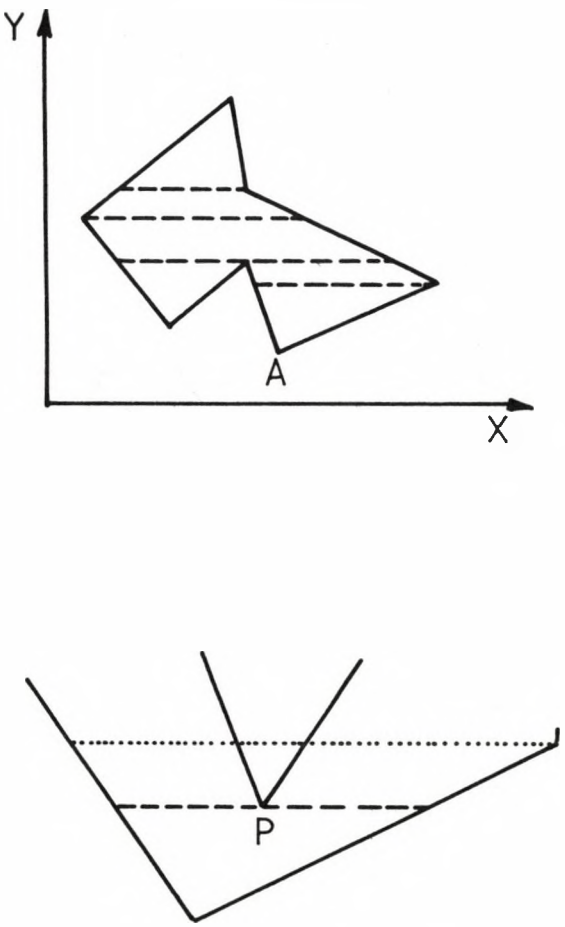

15. ábra

## 5. Dekompoziciós eljárások

Az algoritmusok ezen típusa a poligont megpróbálja több kisebb részre osztani, amelyek már egyszerűbben tölthetők. A dekompozíció éppen azért szükséges, mert a poligon túlságosan bonyolult, például nem konvex. vagy több lyuk is van benne. Igy az ilyen algoritmusok többsége nem tesz megszorítást a poligon alakjára, de az önmagát metsző alakzatot általában nem engedi meg. Az algoritmusok host számítógépen futnak, hiszen még ismerni kell a poligonoldalakat. Abban az esetben ha valamilyen mintával töltjük a poligont, akkor az ilyen típusú eljárásoknál ügyelni kell arra, hogy a levágott részeknél a minta jól illeszkedjen. A dekompoziciós algoritmusokat bármilyen típusu output eszközön használhatjuk. Az eljárás előnye, hogy a részekre bontás során a kitöltendő részek már egyszerűen tölthetők fel, viszont hátránya, hogy amíg addig eljutunk, addig sok más vizsgálatot, rendezést és számolást kell végrehajtani, és csak ezek után kezdhetjük a kisebb alakzatok feltöltését. Erre már valamilyen más típusú kitöltést fogunk használni. A dekompozició során általában konvex alakzatokra törekszünk, amely leggyakrabban háromszögek, vagy trapézek összességét szokta jelenteni.

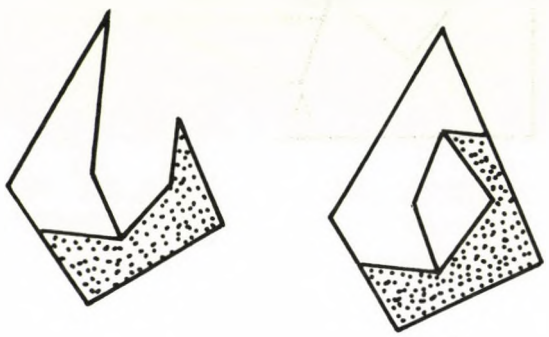
## 5.1 Brassel & Fegeas dekompozíció

Ez az eljárás lényegében vonalrajzolós eszközre készült, de alkalmazható raszteres megjelenítőre is. Egy megadott poligont szeretnénk kitölteni vizszintes, függőleges, vagy tetszőleges $\alpha$ szögű egyenesdarabokkal, azaz "besraffozzuk" a poligont (hatch). Ezt a feladatot végezhetnénk OEL algoritmussal is, de sok szögpontú poligon esetén túl sok időt venne el a rendezés és keresés a poligonoldalak között. Az eljárás bármely szögű vonalkázást visszavezet a vizszintes vonalkázásra, az alakzat $\alpha$ szögű elforgatásával. Az alakzatot háromszögek és az x tengellyel párhuzamos oldalú trapézokra bontja az eljárás. Egy ilyen trapéz (háromszög) kitöltése nem túlságosan bonyolult feladat ,hiszen a két trapézoldalon a metszéspontok egyenlő x és y távolságban vannak egymástól (ez az eredeti - nem $\alpha$ szöggel elforgatott - rajzon is igaz), és így csak egy vonalkázó egyenes metszéspontjait kell kiszámolni. Ezek után a vizszintes vonalkázást megvalósító algoritmus menete a következő : vesszük a legkisebb y értékhez tartozó pontot (16. ábrán A) és elindulunk két irányban a poligonon. A két szomszédos csúcs közül a kisebb y értékhez tartozón át vizszintes egyenest húzunk és ez határozza meg az első háromszöget. Abban az esetben ha ebbe a háromszögbe bemetsz valamely oldal, vagyis a vizszintes egyenes magasságánál kisebb y értékű P pont van a poligonon, akkor ezen a minimális P-n át húzzuk a vizszintes egyenest és így kapjuk az első háromszöget. Ekkor a P-n átmenő vizszintes egyenes két részre vágja az alakzatot és az egyiken fogunk tovább haladni, míg a másikra utaló értékeket stack-re tesszük. Hasonlóan haladunk tovább a trapézok létrehozásával, amíg a max y értéket el nem érjük és amíg van a stack-en elem (16 ábra). Ugyancsak stack-re kerül elem, ha az éppen vizsgált két él különböző irányban halad.




16. ábra

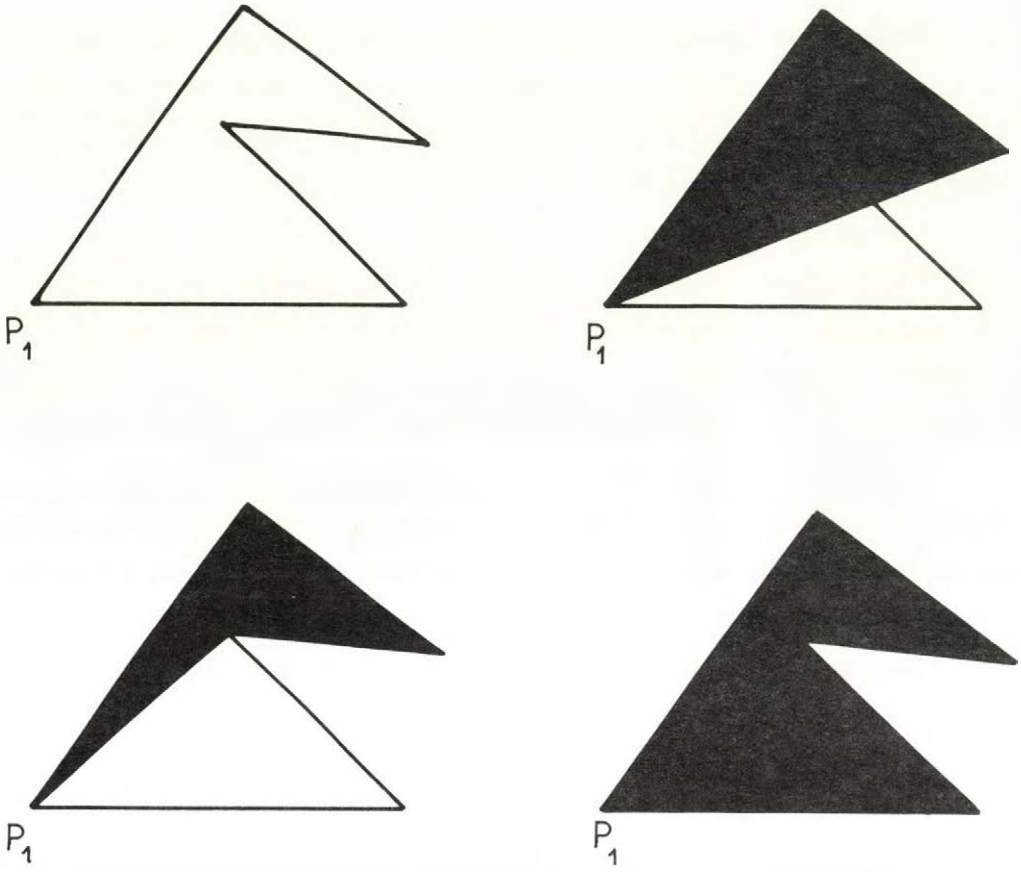
Az eljárás lyukas alakzatokra is általánosítható. Ha egy trapéz létrehozásakor a trapézba benyúló csúcs a lyuk minimális y pontja, akkor az egyik oldali folytatást csak a lyuk maximális értékéig végezzük, és a másik ágon a módosított határok közt történik a sraffozás (17.ábra).



17. ábra

## 5.2 Lane,Magedson & Rarick dekompozíció

Ez az eljárás a poligont mint háromszögek összességét tekinti, pontosabban mint ezek összegét és különbségét. A poligont pozitív irányításúnak veszi. Ez után kiválasztja a poligon egy csúcsát, pl. $P_1$-t, és sorba veszi a $P_1 P_i P_{i+1}$ háromszögeket. Ha ez a háromszög pozitív irányítású, akkor a háromszög kitöltést hozzáadja a képhez, míg ha negatív irányítású, akkor levonja a képből. Az algoritmus feltételez egy meglévő (szoftver vagy hardver) háromszög kitöltő rutint ami hívható az eljárásból. Ha a poligon konvex, akkor a beállított pixelek számát illetőleg az eljárás optimális, hiszen minden pixel értékkel csak egyszer foglalkozik. Az algoritmus önmagát metsző alakzatnál nem működik jól. Ha a poligonban lyukak vannak, akkor a lyukakat mint negatív irányítású poligonokat tekintjük, és ugyanúgy vesszük a kiválasztott $P_1 Q_i Q_{i+1}$ háromszögeket, mint az előbb. Az algortimus mintázattal való kitöltésre is alkalmas, ha a háromszögkitöltő képes erre. Egy probléma fordulhat elő, ha az intenzitásértékeket sokszor kell hozzáadni és csak utána levonni, akkor elképzelhető,hogy az intenzitásértékek túlcsordulhatnak. (Ez lyukas alakzat esetén fordulhat elő.) Ezt az eljárást hidden-line algoritmussal együtt használták, így a poligon belsejében már nem volt semmilyen más alakzat. Az algortimust úgy is lehet interpretálni, hogy a háromszög feltöltést XOR -ral végezzük. Ekkor a poligon irányítását sem kell használni, és önmagát metsző alakzatra is jól működhet a kitöltés. Így az értékek sem fognak túlcsordulni. Természetesen a szokásos problémék itt is fennálnak, amelyek a XOR -os aritmetika felhasználásánál előfordulnak. (Az alakzaton és lyukakon belüli bármely egyéb ábrarész negatívban fog megjelenni, azaz a poligon nem fogja ezt takarni.) [6]



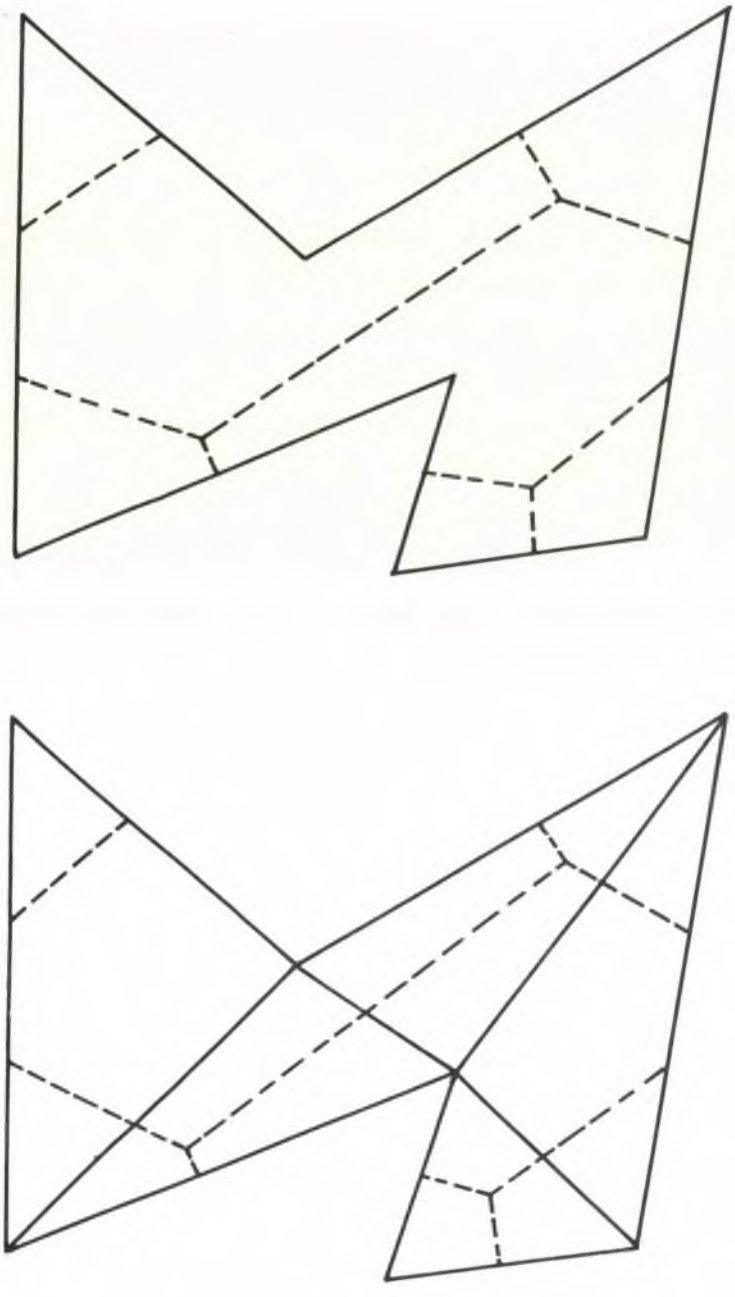

18. ábra

## 5.3 Schachter dekompozíció

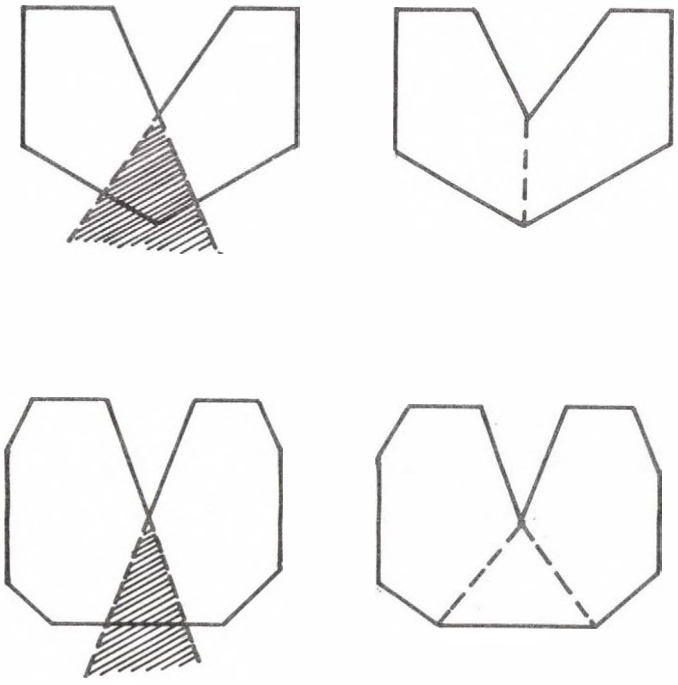
Ebben az esetben a poligont kizárólag diszjunkt konvex poligonok összegére fogjuk bontani. A sokszögek csúcsai a poligon csúcsaiból kerülnek ki. Próbáljuk először a poligont cellák összegére bontani. Minden poligon csúcshoz tartozzon egy olyan cella, amely a poligon azon pontjait tartalmazza, amelyek ehhez a csúcshoz vannak a legközelebb. Ez a poligon egy "Voronoi" felbontásához vezet (19. ábra ). A Voronoi cellák határai egyenes szakaszok. A szomszédos Voronoi cellák csúcspontjainak összekötésével jutunk el a "Delaunay" felbontáshoz, amivel a poligont háromszögek összegére bontottuk (19. ábra ). Jelen dekompozíciós eljárás a Delaunay felbontáson alapszik, de annak csak egy részét használja, mert nem háromszögekre, hanem konvex részekre akarja bontani az alakzatot. Így sorban haladunk a poligon csúcsain,és csak konkáv csúcs esetén húzzuk meg a csúcson átmenő Delaunay cella oldalt.Ezzel két kisebb poligonra osztottuk az eredeti poligont, így a részekre ismét tudjuk alkalmazni az eljárást. Ennek megfelelően az eljárás menete a következő:

a, legyen V a poligon egy konkáv csúcsa
b, legyen a V-hez legközelebbi poligoncsúcs V'
c, ha a V-hez tartozó két él által meghatározott szögtartományban van V' , akkor V-t összekötjük V' -vel és így megkaptuk a két részpoligont, most már a részpoligonokkal fogunk foglalkozni
d, határozzuk meg V és V' felezőpontját, ami legyen m
e, az m kezdőpontból meghatározzuk a V-hez tartozó Voronoi cellát (félsíkok metszésvonalaiként) , amit elég a két él által meghatározott szögtartományban megtenni. Ha a szögtartományban találtunk Delaunay élt, akkor ezt meghúzva készen vagyunk. Ellenkező esetben a szögtartományhoz jobbra és balra levő Delauney éleket húzzuk meg (20. ábra ).

[11]



12. ábra



20. ábra

## 6. Ordered Edge List (OEL)

Az algoritmus típus lényegében a Parity Check módszer Host számítógépen megvalósított változata. Scan line-onként elmetszük az összes poligonoldallal (itt ismerjük pontosan a csúcsokat) , majd a metszéspontokat x szerint rendezzük. Az elsőt a másodikkal,a harmadikat a negyedikkel , és így tovább , kötjük össze, és ez fogja adni a kitöltést. Előnye, hogy jelen esetben a kitöltés nem függ az egyenes reprezentációjától, és így tetszőleges alakzat is tölthető, akár önmagát metsző, vagy tetszőleges számu lyukkal ellátott is a poligon. Előnye még,hogy az eljárás csak a megadott poligont vizsgálja, tehát érdektelen, hogy még mi minden van a képernyőn. Érdektelen még az is, hogy milyen szinnel töltjük a poligont és milyen szinű a kontúr. Hátránya hogy egy scan line-on levő szakaszok meghatározásához sokmindent kell elvégezni: metszéspontok meghatározása, a metszéspontok rendezése. Ennek megfelelően az eljárás végrehajtási ideje erősen függ a poligon oldalainak számától. A végrehajtás során ügyelni kell a csúcspontokon áthaladó scan line-ok esetében, mert itt lehet páratlan metszéspontja a scan line-nak a poligonnal , ha nem vigyázunk eléggé.

## 6.1 OEL

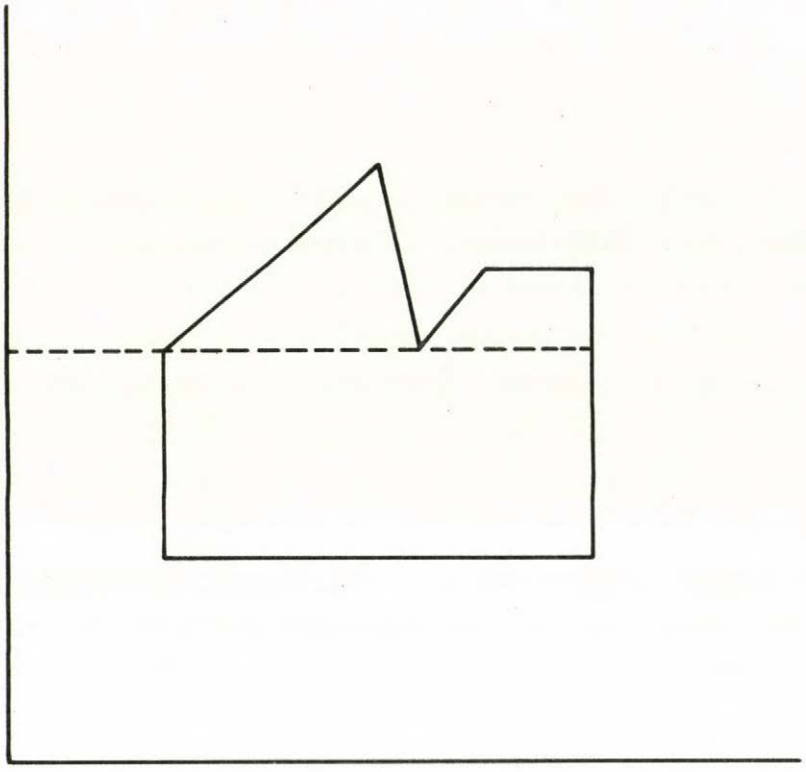
Meghatározzuk a poligonhoz tartozó minimális és maximális y értékeket, majd egyenként (scan line-onként) haladunk a minimumtól a maximumig. Minden egyes esetben az adott vizszintes egyenest elmetszük a poligon oldalaival, majd a metszéspontok rendezésével megrajzoljuk a poligon belsejéhez tartozó szakaszokat. Probléma a csúcsponton átmenő vizszintes egyenessel lehet (21. ábra). A megadott példán ugyanis az egyenes három pontban metszi az alakzatot. A probléma megoldására három lehetőség is kinálkozik:

a, a már egyszer alkalmazott 1/2 -del eltolt koordináta rendszert tekintjük. Ekkor nem lesz egy csúcspont sem scan line-on (csak 0.5-des értékeken). Hátránya hogy kismértékben hibát vihetünk a rajzba.

b, irányváltásnál (előző él fölfelé, utóbbi lefelé, vagy előző él lefelé,utóbbi meg felfelé halad) a csúcspontot kettözöttnek tekintjük , igy az adott példában a csúcsban való metszés kétszer fog szerepelni.

c, a poligonoldalakat alulról nyílt és felülről zárt szakaszoknak tekintjük ($P_i P_{i+1}$, $y_i < y_{i+1}$ esetén $P_i$ nem tartozik a szakaszhoz, de $P_{i+1}$ igen) . Ez a megoldás a vizszintes poligonoldalnál is jól működik, mert ekkor nem kell elmetszeni a scan line-nal (a többi esetben a vizszintes oldalakat külön kell vizsgálni, mert ekkor a metszéspont határozatlan).

[8]



21. ábra

## IRODALOMJEGYZÉK

1, Ackland, B.D., Weste, N.H. :" The egde flag algorithm - A fill method for raster scan displays."
IEEE Trans. Comput. C-30,1. (Jan. 1981), 41-48.

2, Brassel, K.E., Fegeas, R. :" An algorithm for shading regions on vector display devices."
SIGGRAPH '79 Proceedings, published as Computer Graphics 13,2. (August 1979), 126-133.

3, Distante, A., Veneziani, N. :"A two-pass filling algorithm for raster graphics."
Comput. Graphics and Image Proc. 20,3. (Nov. 1982), 288-295.

4, Dunlavey, M.R. :"Efficient polygon-filling algorithms for raster displays."
ACM Trans. Graphics 2,4. (Okt. 1983), 264-273.

5, Foley, J.D., Van Dam, A. :"Fundamentals of interactive Computer Graphics."
Addison-Wesley,Reading,Mass., 1982, 446-450.

6, Lane, J.M., Magedson, R., Rarick M. :"An algorithm for filling regions on graphics display devices."
ACM Trans. Graphics 2,3. (July 1983), 192-196.

7, Lieberman, H.: "How to color in a coloring book."
SIGGRAPH '78 Proceedings, published as Computer Graphics 12,3. (August 1978), 111-116.

8, Newman, W.M., Sproull, R.F. :"Principles of interactive Computer graphics."
McGraw-Hill,New York, 1979, 230-236.

9, Pavlidis, T.: "Filling algorithms for raster graphics."
Comput. Graphics and Image Proc. 10,2. (Jun. 1979), 126-141.

10,Pavlidis, T.: "Contour filling in raster graphics."
SIGGRAPH '81 Proceedings, published as Computer Graphics
15,3. (August 1981), 29-36.

11,Schachter, B.: "Decomposition of polygons into convex sets."
IEEE Trans. Comput. C-27,11 (Nov. 1978), 1078-1082.

12,Shani, U.: "Filling regions in binary raster images - a graph-theoretic approach."
Proc. SIGGRAPH '80, published as Computer Graphics
(1980), 321-327.

13,Smith, A.R.: "Tint fill."
SIGGRAPH '79 Proceedings, published as Computer Graphics
13,2. (August 1979), 276-283.

1984-BEN JELENTEK MEG:

155/1984 Deák, Hoffer, Mayer, Németh, Potecz, Prékopa, Straziczky: Termikus erőműveken alapuló villamos-energiarendszerek rövidtávu, optimális, erőművi menetrendjének meghatározása hálózati feltételek figyelembevételével.

156/1984 Radó Péter: Relációs adatbáziskezelő rendszerek összehasonlitó vizsgálata

157/1984 Ho Ngoc Luat: A geometriai programozás fejlődései és megoldási módszerei

158/1984 PROCEEDINGS of the 3rd International Meeting of Young Computer Scientists.
Edited by: J. Demetrovics and J. Kelemen

159/1984 Bertók Péter: A system for monitoring the machining operation in automatic manufacturing systems

160/1984 Ratkó István: Válogatott számitástechnikai és matematikai módszerek orvosi alkalmazása

161/1984 Hannák László: Többértékü logikák szerkezetéről.

162/1984 Kocsis J. - Fetviszov V.: Rugalas automatizált rendszerek: megbizhatóság és irányitási problémák

163/1984 Kalavszky Dezső: Meleghengermüvi villamos hurokemelő hajtás vizsgálata

164/1984 Knuth Előd: Specifikációs adatbázis modellek

165/1984 Petróczy Judit: Publikációk 1983

1985-BEN EDDIG MEGJELENTEK:

166/1985 Radó Péter: Információs rendszerek számitógépes tervezése

167/1985 Studies in Applied Stochastic Programming I. Szerkesztette: Prékopa András

168/1985 Böszörményi László - Kovács László - Martos Balázs Szabó Miklós: LILIPUTH

169/1985 Horváth Mátyás: Alkatrészgyártási folyamatok automatizált tervezése

170/1985 Márkus Gábor: Algoritmus mátrix alapu logaritmus kiszámitására kriptográfiai alkalmazásokkal

171/1985 Tamás Várady: Integration of free-form surfaces into a volumetric modeller