Budapesti Műszaki Egyetem
Építőmérnöki Kar
Fotogrammetria Tanszék

DIPLOMATERV

A GIS rendszerekben alkalmazható geometriai adatmodellezési módszerek

Budapest, 1994. május

Terei Gábor
földmérőmérnök hallgató

Tartalomjegyzék

Bevezető

A rendelkezésre álló digitális térbeli adatok gyorsan növekvő mennyisége, és ezek egyre szélesebb körű felhasználása előtérbe helyezte az adattárolás egyre hatékonyabb módszereinek keresését, hiszen nagy mennyiségű adattal csak akkor lehet kényelmesen dolgozni, ha a tárolást és feldolgozást megfelelő sebességgel tudjuk elvégezni.

Az utóbbi években jelentősen megnőtt az integrált tervezési rendszerek iránti igény, melyekhez azonban digitális adatok szükségesek (lehetőleg valamilyen információs rendszerbe integrálva). A napi gyakorlatban alkalmazott térbeli adatfeldolgozó rendszerek általában túlzottan feladat-specifikusak. Vonatkozik ez mind az egyedi rendszerek képessé tételére szélesebb körű alkalmazások befogadására, mind a különböző forrásokból származó eltérő típusú térbeli adatbázisok egybeolvasztására. A nyitottság hiányát és az egységesítés nehézségeit leginkább az a tény okozza, hogy a térbeli adatbázisok komoly nehézségekkel küzdenek az adattárolás és a feldolgozáshoz szükséges idő terén. Gondoljunk csak bele, ha Magyarország 1:10000-es topográfiai térképeit szeretnénk scanner-rel beolvasni, tegyük fel 300 dpi felbontással (ami igazából nem is felelne meg a kartográfiai pontossági céloknak), az adatok kb. 4300*23,6in*15,75in*300*300, azaz $1,44*10^{11}$ pixel-t jelentenének. Hiába a nagyméretű fejlődés a tárolókapacitásban, még mindig hatalmas mennyiségű adathalmazokkal kell dolgozni, így az adatok kezelésének és tárolásának optimalizálása fontos feladat. S noha a GIS rendszerek és a digitális kartográfia hatalmas fejlődésen ment keresztül, viszonylag kevés rendszerben létezik olyan megoldás, mellyel nagy adatsűrűség esetén is áttekinthető, "méretarány-független" megjelenítés hozható automatikusan létre.

Diplomatervemben, a fentiek szem előtt tartásával, megpróbálom összefoglalni a gyakorlatban alkalmazott adatstruktúrákat, felvázolni előnyeiket és hátrányaikat. Az első fejezet az adatmodellezéssel általában és a geometriai adatmodellekkel foglalkozik. Célom a hazai és a nemzetközi szakirodalomban megtalálható módszerek viszonylag széles körű összefoglalása. A második fejezetben részletesebben foglalkozom a topográfiai térképeket elsődlegesen képező vonalas elemek matematikai leírásának módszereivel.

Mindkét fejezetben megemlítem a vázolt adatmodellek segítségével elérhető megjelenítési módszereket, szorosan kapcsolódva a harmadik fejezethez, melyben a vonalas elemek legfontosabb egyszerűsítő módszereit tárgyalom. Végül néhány konkrét példával zárom munkámat.

1. Geometriai adatok modellezése

1.1. Az adatmodellezés szintjei

Az adatmodellt definiálhatjuk adatelemek speciális halmazainak általános leírásaként és ezen halmazok közti kapcsolatokként. Az adatelem olyan valami, ami létezik és megkülönböztethető. Így pl. egy szék, egy személy vagy egy tó mind egy-egy adatelem . Az adatelem halmaz olyan adatelemeket csoportosít, melyek valamely szempontból hasonló tulajdonságokkal rendelkeznek. A kapcsolatok olyan fogamakat takarnak, mint 'balra', 'kisebb', 'tartalmaz',... Mind az adatelemek, mind a kapcsolatok rendelkezhetnek attribútumokkal vagy tulajdonságokkal. Ezekkel egyedi értékeket rendelhetünk az attribútum értékhalmazából minden egyes adatelemhez az adathalmazon belül. Például egy tó attribútumai lehetnek a felülete, maximális mélysége, tengerszint feletti magassága vagy akár az oldott ásványi anyagok mennyisége.

Az adatmodell egy összehasonlítható megfogalmazását vezette be Codd, aki szerint az adatmodell három részből áll; objektum típusokból, operátorok (műveletek) halmazából és általános egységesítő szabályokból. Date kijelentése szerint "Bármely adatmodell célja, hogy formális eszközöket biztosítson az információ ábrázolására és ezen ábrázolás manipulálására".

Miután ezen fenti definíciók emberi megfogalmazások és egy adott alkalmazáshoz szabhatók, különböző felhasználók és alkalmazások különböző adatmodelleket használhatnak ugyanazon jelenség leírására. Mint a 'modell' szó is sugallja, az adatmodell legfőbb tulajdonsága az, hogy a valóság valamilyen szintű absztrakciója jön létre. Ahhoz, hogy egy adathalmazt végül is digitális formában ábrázolhassunk, az adatokat különböző szinteken kell, hogy szemlélhessük. Ezek a szintek a valóságtól, az absztrakt, felhasználó-orientált struktúrán át a konkrét, gép-

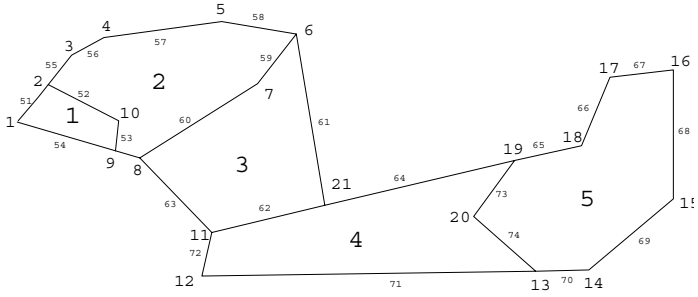
orientált tárolási struktúráig terjednek. A legcélszerűbben a következő négy szintet különböztethetjük meg (1. ábra) :

- *Valóság* — a jelenség, ahogy az éppen létezik, magában foglalva minden szempontot, legyen az akár észrevehetetlen,
- *Adatmodell* — a valós világ egy absztrakciója, mely csak azokat a tulajdonságokat foglalja magában, melyek az adott alkalmazáshoz szükségesek, általában emberi szemszögből,
- *Adatstruktúra* — az adatmodell egy reprezentációja, általában diagramokban, listákban vagy tömbökben kifejezve, melyek az adatok számítógépes kóddal való leírását tükrözik,
- *File-struktúra* — az adatok fizikai tárolásának módja; itt vált át az adatstruktúra a hardware/software specifikus környezetbe.

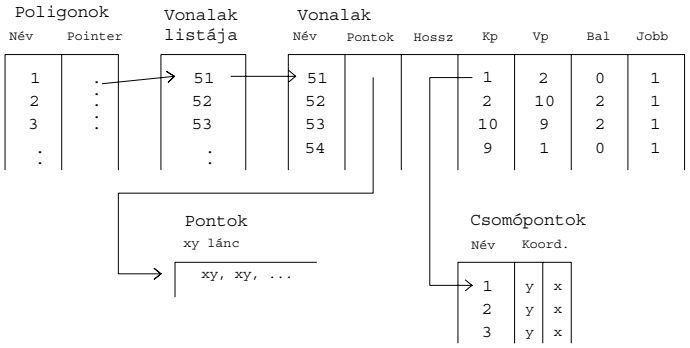
Ez utóbbi három szemlélet megfelel az adatbázis szerkesztés és megvalósítás főbb lépéseinek. A teljes folyamat abból áll, hogy általános állításokat fogalmazunk át meghatározott állításokká.

Miután egyetlen modell vagy absztrakció sem képes a valóságot minden szempontból ábrázolni, lehetetlen olyan általános célú adatmodellt készíteni, mely egyformán hasznos minden alkalmazásban. Ez különösen igaz, ha bonyolult jelenségekkel dolgozunk. Például egyes térbeli adatstruktúrák jók kirajzolásra, de kevéssé hatékonyak analitikus célokra. Más adatstruktúrák kitűnőek lehetnek analitikus feladatokra, de rendkívül lassúak, ha grafikai produktumot kell előállítani. Ezért egy adatmodell tervezésekor vagy kiválasztásakor alapul kell vennünk az adatok által megjeleníteni kívánt jelenség természetét, valamint az alkalmazás által megkívánt sajátos adatműveleteket.

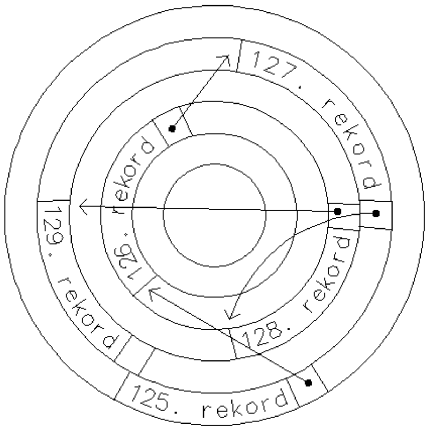
valós világ



adatmodell



adatstruktúra



file-struktúra

1.ábra *Az adatmodellezés szintjei*

1.2. A geometriai adatok típusai

A térbeli vagy geometriai adatok fogalma minden olyan adatra vonatkozik, melyek valamilyen jelenség helyzetét, alakját ábrázolják. Tehát ez a fogalom magában foglalja pl. a gépészeti tervrajzokat is. Jelen dolgozat célja azonban elsősorban a térképi adatok geometriájának vizsgálata, tehát olyan geometriai adatoké, melyek elsősorban a Földre, mint valóságra vonatkoznak.

A geometriai adatok alapvetően három nagy csoportra oszthatók. Az első csoportot a pontszerű adatok képezik, ahol minden adatelemhez egyetlen hely tartozik a két- vagy háromdimenziós térben. Ilyen adatokra példa az európai fővárosok helye.
A második csoportot a vonalas adatok alkotják. Ennél az adattípusnál a helyzetet térbeli koordináták lánca fejezi ki. Ezek a vonalak lehetnek :

- izolált vonalak, ahol az egyes vonalak nem kapcsolódnak egymáshoz (pl. törésvonalak),
- fa struktúrába rendezett vonalak (pl. folyók rendszere),
- hálózati struktúra részei (pl. úthálózat).

A harmadik csoport a poligonokból képezhető, ahol az adatelemet a térbeli koordináták zárt lánca alkotja. A poligon adatok tehát felületeket határolnak le az adott térben vagy síkon. A poligonok csoportját is tovább oszthatjuk :

- izolált poligonok, ahol a poligon körvonala sehol sem érintkezik másik poligonnal,
- szomszédos poligonok, ahol a körvonal minden egyes szakasza érintkezik legalább egy másik poligonnal,
- egymásba ágyazott poligonok, ahol egy vagy több poligon teljes egészében egy másik poligonban helyezkedik el.

Tulajdonképpen egy negyedik kategóriát is képezhetünk a fenti három típus összevonásával. Itt keveredhetnek a különböző vonalstruktúrák egymással, vagy poligon struktúrákkal. Például Magyarország térképén a Duna határa egyes megyéknek is, tehát egyaránt része a poligonstruktúrának és a folyókat képező vonalas fa struktúrának.

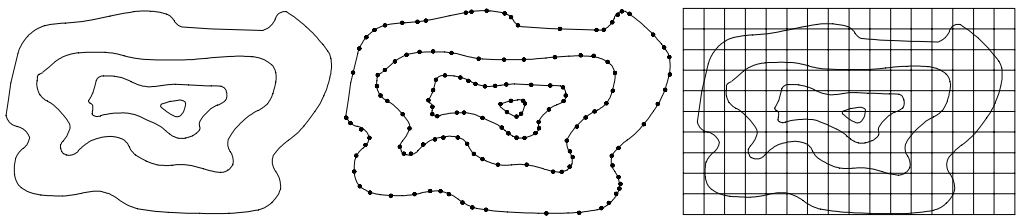
A geometriai adatmodelleknek több olyan tulajdonságuk van, melyek megkülönböztetik az egydimenziós vagy lista típusú adatmodellektől. Először is a térbeli adatelemeknek egyedi jelentésük, hogy tükrözik az adatelem térbeli helyzetét. Ezt a helyzetet általában koordináta rendszerben rögzítjük, mely azonban igen sokféle lehet (földrajzi hosszúság és szélesség, EOV, postacím, ...) és nem mindig létezik matematikailag leírható átszámítási mód a különböző rendszerek között.
A geometriai adatelemek közötti kapcsolatok is rendkívül sokfélék lehetnek és a valóság természete és a modellezési folyamat korlátai miatt gyakorlatilag lehetetlen mindet ábrázolni és tárolni. Ráadásul a kapcsolatok definíciói általában pontatlanok és környezetfüggőek. Ez még a legalapvetőbb térbeli kapcsolatokra is vonatkozik, mint pl. a 'közel', 'távol', 'balra', 'jobbra'. Ezen tulajdonságok miatt a geometriai adatok modellezése rendkívül bonyolult, és ezért a modellek maguk is bonyolultak és a létrejövő adatfile-ok meglehetősen nagyméretűek.

További probléma a megfogalmazott adatmodell adatstruktúrába és file-struktúrába transzformálása a számítógépes reprezentációhoz. A geometriai adatok definíció szerint két- vagy háromdimenziósak. Ezeket kell a számítógép memóriájában ábrázolni, amely általában lineáris, azaz egydimenziós jellegű, miközben meg kell őriznünk a belső kapcsolatokat.

1.3. Geometriai adatmodellek

A geometriai adatmodellek legrégebbi típusa a térkép, mely kétdimenziós analóg adatmodell. A térkép kényelmes módja a térbeli adatok tárolásának, szemléltetésének és folyamatos kézi kiegészítésének, valamint alkalmas mérések végzésére és egyéb feldolgozás céljára. Azonban a térkép felújítása, vagy bármilyen elemzés elvégzése új térkép megrajzolását kívánja, mely rendkívül időigényes folyamat és nagy gyakorlatot és pontosságot igényel.

A geometriai adatok digitális formában való tárolására két alapvető adatmodell alakult ki : a tesszellációs és a vektor modell (2. ábra). A vektoros adatmodell esetében az alapelem a vonal vagy szakasz, melynek kezdő- és végpontjának x-y koordinátáit tároljuk egy adatmezőben. Pontszerű objektumokat nulla hosszúságú szakaszként ábrázolhatunk ilyen adatmodell esetén. A tesszellációs modell alapeleme a háló egy eleme vagy cellája. A közhasználatban ezt a modellt gyakran raszter- vagy rácsmodellként emlegetik, azonban ez az adatmodell nem csak a négyzethálós felosztású modellt tartalmazza.
Létezik még egy harmadik adatmodell is, az ún. hibrid típus. Ez egy viszonylag új fejlesztés, mely mind a vektoros, mind a tesszellációs modell tulajdonságaival rendelkezik.



analóg vektor tesszelláció

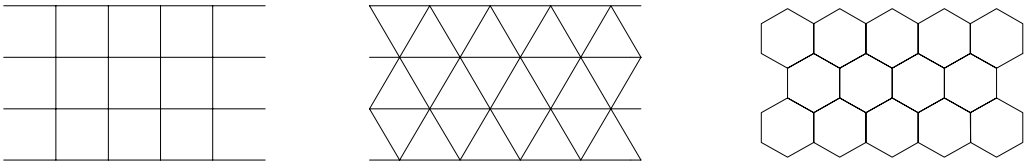
2. ábra *A geometriai adatmodellek alaptípusai*

1.3.1. Tesszellációs adatmodellek

1.3.1.1. Szabályos tesszelláció (fa struktúrák)

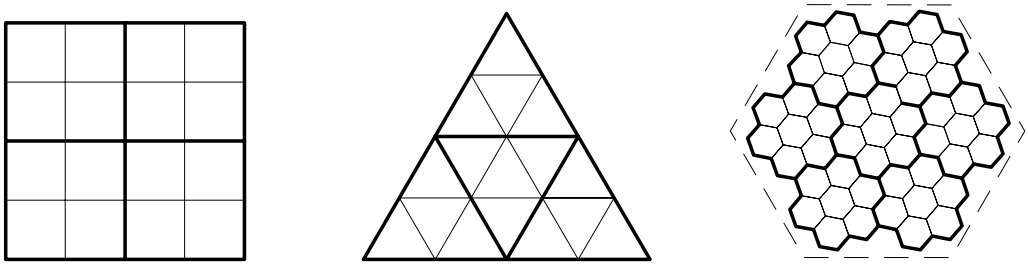
Szabályos tesszellációnak nevezzük, ha a modellt szabályos sokszögekre osztjuk és ez a felosztás bármelyik elemen megismételhető. A szabályos tesszellációs adatmodellek mindhárom lehetséges típusát használják térbeli adatmodellként. Mindegyiknek más - az alkalmazott elemi sokszög geometriáján alapuló - funkcionális tulajdonsága van. Ezek a lehetséges típusok a négyzet-, háromszög- és hatszöghálók (3. ábra). Leginkább a négyzethálós tesszellációs modell terjedt el, mert ez a modell kompatibilis számos térbeli adatnyerő és megjelenítő hardware eszközzel (raszter scanner-ek, színes monitorok, műholdas felvételek, ...). A szabályos hatszögháló legfőbb előnye az, hogy egy elem minden szomszédja egyforma távolságra esik az adott elem középpontjától. Ez a sugárirányú szimmetria rendkívül előnyössé teszi ezt a modellt keresés céljára. A háromszög alapú tesszellációk egyedülálló tulajdonsága, hogy a háromszögek nem azonos irányítottságúak. Ez sok műveletet megnehezít, de ez a tulajdonság teszi lehetővé, hogy a terepfelszínt, vagy más felületet szemléletesen ábrázolhassunk. Ezért használják előszeretettel ezt a módszert digitális terepmodelleknél.

A négyzet és szabályos háromszög alapú hálóknál minden egyes elem tovább osztható ugyanolyan alakú elemekkel. Az alapvető különbség ebből a szempontból a négyzet, a háromszög és a hatszög alapú tesszellációs modellek között az, hogy kizárólag a

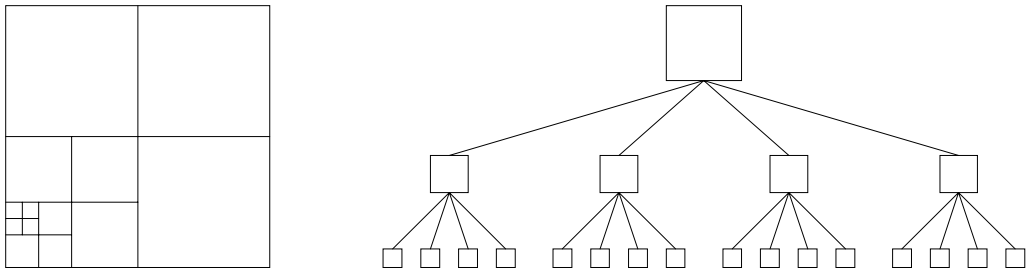


3.ábra A három szabályos tesszelláció

négyzetrács osztható tovább ugyanolyan alakú és irányítottságú elemekkel. A háromszögeket feloszthatjuk ugyan háromszögekkel, de az irányítottság problémája fennmarad. A hatszögeket nem lehet hatszögekre osztani, noha a kiinduló elem alakja közelíthető (4. ábra). A sík ilyetén felosztása rekurzív tesszellációs modellel számos előnnyel jár. Leginkább a rácshálós (négyzethálós) felbontáson alapuló négyesfa (quadtree) modell terjedt el (5. ábra). A négyesfa modell alkalmazása geometriai adatmodellként előnyös, mert a fa struktúrák tárolása és keresési módszerei a számítástechnika legtöbbet kutatott és kidolgozott témái közé tartozik. Jól dokumentált algoritmusok készültek a fák file-struktúrába szervezésére, beleértve a tömörítési technikákat és hatékony címzési módszereket. A rekurzív felosztás megkönnyíti a fizikai tárolás megosztását, így jelentősen leegyszerűsíti a keresési műveleteket. Az ablakolás is nagyon hatékony, ha az ablak egybeesik a négyesfa celláival. Ezek a tulajdonságok rendkívül előnyösek nagyméretű adatbázisok kezelése esetén. További előny, mely főleg kartográfiai alkalmazásoknál jelentkezik,



4. ábra *A szabályos tesszelláció rekurzív felbontás esetén*



5. ábra *A négyesfa felépítése*

hogy ez a struktúra egy, a 2 hatványain alapuló változtatható méretarányú séma. Ez azt jelenti, hogy a méretarány megváltoztatása csak a tárolt adatok különböző mélységű előhívását teszi szükségessé.
A fa fogalma és az erre kidolgozott algoritmusok kiterjeszthetők több dimenzióra is, melyek közül a legismertebb az oct-tree (nyolcas fa), a négyesfa három dimenziós megfelelője.

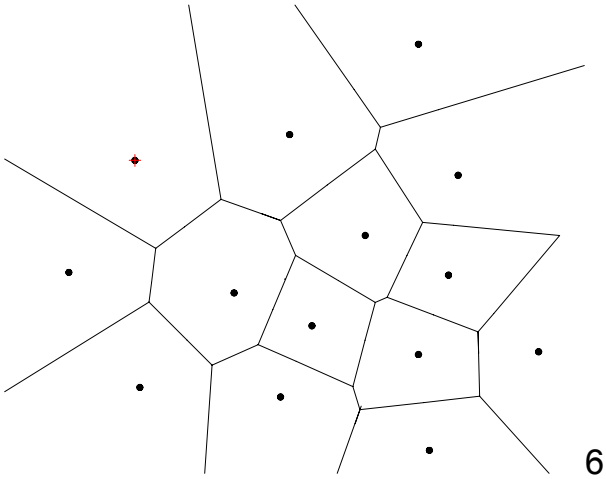
1.3.1.2. Szabálytalan tesszelláció

Vannak esetek, amikor a szabálytalan tesszelláció használata előnyösebb. A leggyakrabban alkalmazott típusok a négyzet, a háromszög és a változó (pl. Thiessen) poligon háló. A szabálytalan háló legfőbb előnye, hogy redundáns adatokat nem szükséges tárolni, és a hálózat szerkezete az adatok elhelyezkedéséhez szabható. Így egy változó felbontású modellhez jutunk abban az értelemben, hogy az alappoligonok mérete és sűrűsége a térben változó. A szabálytalan háló létrehozható úgy, hogy tükrözze az adatsűrűséget, tehát minden elem azonos mennyiségű adatot tartalmazzon. Ennek eredményeképp a hálóelemek mérete megnő, ahol az adatok ritkák, és lecsökken, ahol az adatok sűrűek.

A tény, hogy a hálózat elemeinek mérete, alakja és iránya tükrözi az adatok méretét, alakját és irányát, nagyon hasznos számos feldolgozás vizuális szemléltetésénél. Térbeli adatmodellként talán leggyakrabban használt szabálytalan tesszelláció a szabálytalan háromszöghálózat. A szabálytalan háromszögháló a terepi adatok megjelenítésének általános módja. Előnye, hogy leegyszerűsíti az esésviszonyok és egyéb terepjellemzők számítását, és jobban illeszkedik az adatgyűjtés szisztémájához, ugyanis a terepen mért pontok általában szabálytalan elhelyezkedésűek. Egy felmerülő probléma azonban, hogy a háromszögekre osztás nem egyértelmű, ugyanazon ponthalmazra különböző algoritmusokkal különböző háromszöghálók illeszthetők.

A Thiessen sokszögek, vagy más néven Voronoi diagramok, vagy dirichlet tesszellációk logikus párját képezik a szabálytalan háromszög hálóknak. Úgy készülnek, hogy a háromszögek felezőmerőlegeseit szerkesztjük meg, aminek eredményeképp egy szabálytalan, konvex sokszögekből álló hálózathoz jutunk (6.ábra). A Voronoi diagrammok jól használhatók hatékony számításokhoz a szomszédosság, közelség és elérhetőség elemzésénél, mint például a legközelebbi pont probléma. Az alapelv egyik fejlesztése a pontok súlyozása, melyből a súlyozott Voronoi diagram alakul ki.

Noha látszik, hogy a különböző szabálytalan tesszellációs modellek mind jól megfelelnek egy-egy bizonyos adattípusnak és elemző módszerek egy halmazának, általánosságban rosszul alkalmazhatók egyéb elemzésekhez és feldolgozásokhoz. Például két szabálytalan háló átfedése rendkívül bonyolult feladat. A szabálytalan hálózat generálása is bonyolult és időigényes. Ezért a szabálytalan tesszelláció általános adatmodellként nem megfelelő, kivéve néhány speciális alkalmazást (például digitális terepmodell esetén).

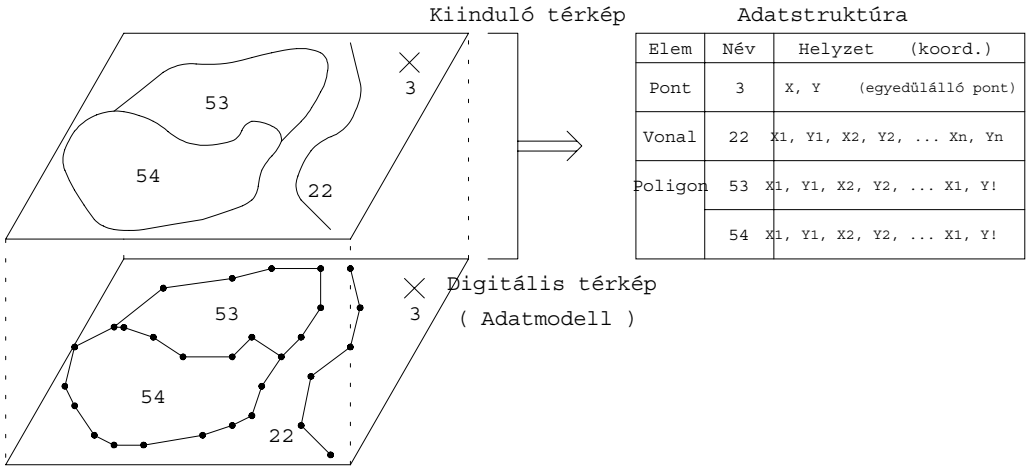


6. ábra *A Voronoi diagram*

1.3.2. Vektoros adatmodellek

1.3.2.1. Spagetti modell

A legegyszerűbb vektoros geometriai adatmodell a térkép közvetlen vonalról vonalra történő leképezése. A térkép minden egyes eleme egy logikai rekord lesz a digitális file-ban, melyet x-y koordináták sorozataként definiálhatunk. Ez egy egyszerű struktúra, melyet értelmezni is egyszerű, mert alapvetően a térkép marad az adatmodell és az x-y koordináták file-ja maga az adatstruktúra. Így a két dimenziós térkép modellt egydimenziós listává képeztük le (7.ábra).
Noha minden elem térbelileg meghatározott, a térbeli kapcsolatokat ez a struktúra nem őrzi meg. Ezért nevezik az így felépített kartográfiai adatfile-t spagetti file-nak ; koordináta sorozatok összessége, mindenféle belső struktúra nélkül. Az ily módon ábrázolt sokszög egy zárt x-y koordinátaláncot képez, mely meghatározza a sokszög körvonalait. A szomszédos sokszög adatainál a közös oldalak töréspontjainak koordinátái ismét rögzítésre kerülnek. A spagetti modell nem elég hatékony a legtöbb térbeli feldolgozási művelethez, miután minden térbeli



7. ábra *A "spagetti" modell*

kapcsolatot számítással kell meghatározni. Ugyanakkor e kapcsolatok, melyek lényegtelenek a kirajzoláshoz, tárolásának hiánya hatékonnyá teszik az eredeti grafika előállítását. Ezért a spagetti modellt gyakran használják a számítógéppel segített kartográfiai termékek előállítására szorítkozó alkalmazásokban.

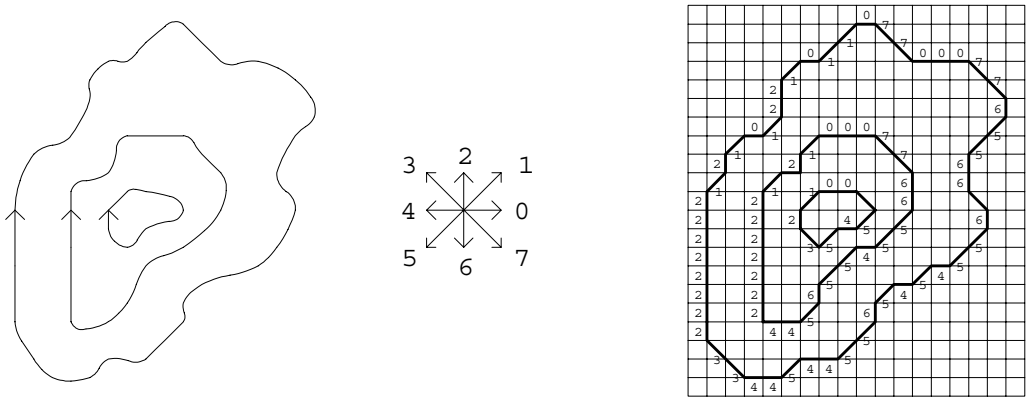
1.3.2.2. Lánckódok

A lánckódos megközelítés tulajdonképpen inkább egy koordináta tömörítési módszer, mint adatmodell. Azért kerül itt mégis említésre, mert ez a módszer jelentős növekedést nyújt a tömörítés és a számítási lehetőségek terén és így gyakran képezik adatmodellek szerves részét. Másodszor a lánckódok olyan hatással voltak a térbeli adatmodellekre és adatfeldolgozásra, hogy gyakran külön adatmodellnek tekintik.
A klasszikus lánckód az ún. Freeman-Hoffman lánckód. Ez abból áll, hogy egy 0 és 7 közötti kódot rendelünk a nyolc főirányú egységvektorhoz. Ez a nyolc főirány megegyezik a koordinátarendszer tengelyeinek irányaival és az átlós irányokkal (8. ábra). Ha ezt a módszert választjuk vonalas adatok kódolására egy adott felbontású hálón, akkor rendkívül tömör digitális kódhoz jutunk. Mint e példából is látható, az x-y koordinátákat csak a vonal kezdőpontjában szükséges tárolni. Az irány a lánckód beépített tulajdonsága, így további tömörítést eredményez, ha irányított adatok ábrázolására használják, mint amilyen például az úthálózat. Speciális kódok alkalmazásával egyes topológiai állapotok, mint a vonalak kereszteződése, megállapíthatók. Egy ilyen speciális kód jelezheti a folyamatos kódolást, mely által hosszú, egyenes vonalak esetén kiszűrhetőek lennének az ismétlődő iránykódok.
Ebből a kódolási sémából több változat is kialakult. Az egyik ilyen, hogy 4, 16 vagy 32 vektorirányt jelölünk ki az egységnégyzeten belül (9. ábra). A 4 irány kódja 3 helyett csupán 2 biten tárolható és általában elég olyan esetekben, ahol az adatok hosszabb egyenes vonalakat képeznek, melyek merőlegesek egymásra,

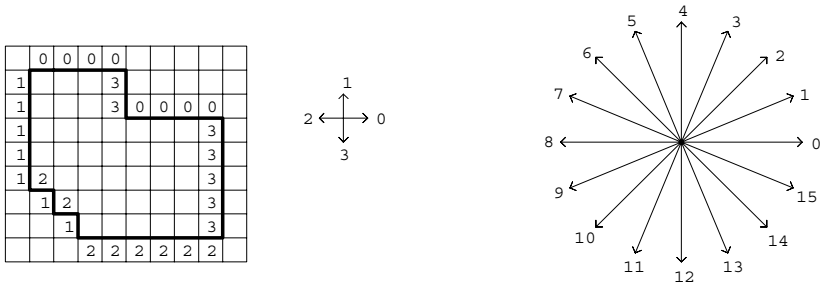
mint egyes mérnöki alkalmazásokban. A 16 vagy 32 irány pedig lehetővé teszi a görbék sokkal finomabb kódolását, így kisimítva a lépcsős hatást. Hasonlóan közvetlen kapcsolat található az irányvektorok száma és az egységvektorok hossza között. Tehát a tömörítés mértéke nagymértékben függ az irányvektorok és így a kódok tárolásához szükséges bitek számától.
A lánckódok egy másik változata az ún. raszter lánckód. Ez a módszer csak a felét használja a 8 alap irányvektornak (10. ábra). Ezt a raszteresen beolvasott (sávonként, balról jobbra és fentről lefelé) vonalas adatokból előállítandó lánckódolt vektoros adatok esetén használják, mivel az ily módon történő feldolgozásnál nem találkozhatunk a feldolgozási irányhoz képest visszafelé irányított vektorokkal. Ez a korlátozott irányítottság azonban azzal jár, hogy megszűnik a vonalak folyamatos irányítottsága. Ha a vektoros adatok irányfolytonosságára szükség van, a raszter lánckód Hoffman lánckóddá történő átalakítása egyszerűen megoldható feladat, csak a kiválasztott szakasz irányát kell megfordítani.

A lánckódok elsődleges hátránya, hogy nem őrzik meg a térbeli kapcsolatokat. Ez tehát csak egy tömörebb spagetti formátumú jelölés. További hátránya, hogy a koordináta transzformációk, főleg az elforgatás meglehetősen bonyolult.

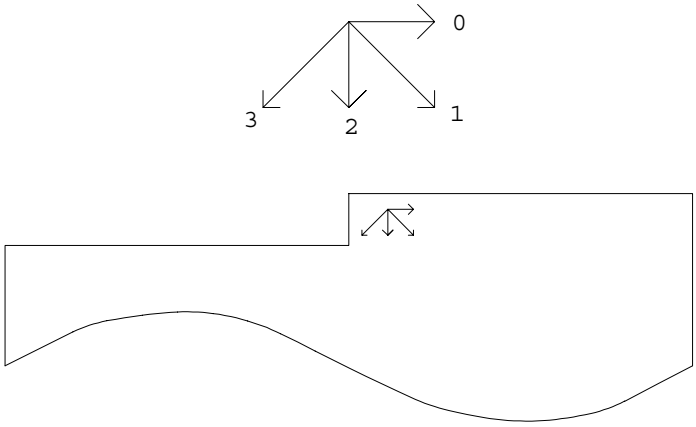
Mint már említettem, a lánckódolás legfőbb előnye a tömörség. Ezért a lánckódot gyakran egybeépítik más módszerekkel, hogy összevonják a tömörség előnyét a másik adatmodell előnyeivel. Az iránykódok alkalmazása derékszögű koordináták helyett jobb teljesítményt nyújt, mint az egyszerű spagetti modell. A vektoros rajzgépek működési rendszere is ezt a rendszert követi, mivel rövid egyenes szakaszokat rajzolnak, általában 8 lehetséges irányban. A grafikus megjelenítés ezeken az eszközökön nem igényel koordináta transzformációt, így ez a folyamat rendkívül hatékony. A lánckódok alkalmazása némely mérési és számítási módszerhez is előnyös, mint pl. a távolságszámítás és az alakelemzés.



8. ábra *A Freeman-Hoffman-féle lánckódolás elve*



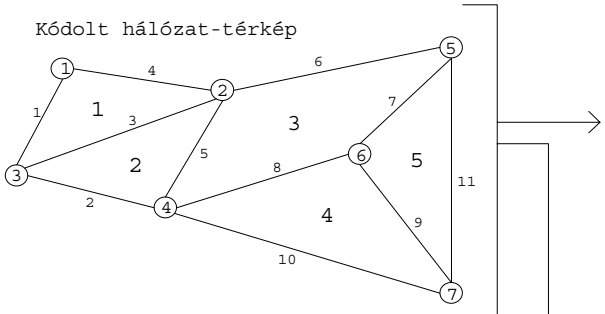
9. ábra *4, illetve 16 irányú lánckódolás*



10. ábra *A raszter lánckódolás elve*

1.3.2.3. Topológiai modell

Az elemek közötti térbeli kapcsolatok megőrzésének leggyakoribb módja a szomszédossági információk explicit tárolása az ún. topológiai adatmodellben. Itt az alap logikai egység az egyenes vonalszakasz, melynek kezdete és vége egy csomópont vagy egy töréspont. Minden szakaszt két végpontjának koordinátáival tárolunk. Ezen felül a szakasz két oldalához tartozó poligonok számát (kódját) is tároljuk (11. ábra). Ily módon a legalapvetőbb térbeli kapcsolatokat rögzítettük és felhasználhatjuk az elemzéskor. Ráadásul így a pont, vonal és poligon típusú egységek redundancia-mentesen tárolhatók. Ez különlegesen előnyös szomszédos poligonok esetén. Itt minden vonalszakasz csak egyszer kerül tárolásra. Az egyes poligonok egyszerűen előállíthatók azokból a szakaszokból, melyeknek az azonos oldalán ugyanaz a poligon található.



Topológiai kódolás

Kód	Jobb poligon	Bal poligon	Kp	Vp
1	1	0	3	1
2	2	0	4	3
3	2	1	3	2
4	1	0	1	2
5	3	2	4	2
6	3	0	2	5
7	5	3	5	6
8	4	3	6	4
9	5	4	7	6
10	4	0	7	4
11	0	5	5	7

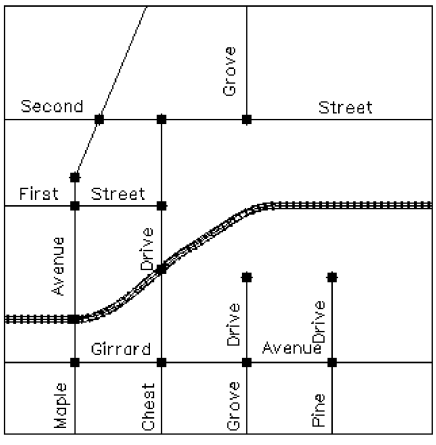
Csomópontok koordinátái

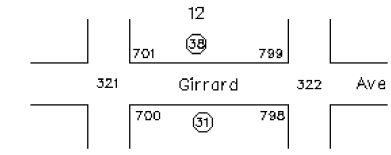
Kód	Y koordináta	X koordináta
1	24,0	111,0
2	23,8	112,4
3	23,0	110,6
4	22,7	111,9
5	24,2	114,4
6	23,2	113,6
7	21,9	114,4

11. ábra *A topológiai adatmodell*

1.3.2.4. GBF/DIME

A GBF/DIME (Geographic Base File / Dual Independent Map Encoding) modell egyértelműen a topológiai struktúrán alapuló, gyakorlatban is használt modell. Az elvet az Amerikai Népszámlálási Hivatal dolgozta ki, hogy a térképek és a hozzájuk kapcsolódó postacím adatok digitális tárolásával megkönnyítse az adatok begyűjtését és feldolgozását. A GBF/DIME file-ban minden utcát, folyót, vasútvonalat, közigazgatási határt, stb. egyenes szakaszok sorozataként ábrázolnak. Egy szakasznak ott van vége, ahol két vonal találkozik, vagy a vonal irányt változtat (töréspont). A szakaszvégpontok csomópontként kerülnek tárolásra (12. ábra). A DIME modell jelentős előnye az egyszerű topológiai modellel szemben, hogy minden egyes szakaszhoz külön irányt rendel egy kezdő- és egy végpont formájában. Az eredmény egy irányított gráf, mely a hiányzó szakaszok automatikus ellenőrzésére is használható, ha követjük a file-ban megtalálható egyes poligonok határoló szakaszait. Ha valahol nem záródnak a határszakaszok, akkor ott vagy egy szakasz hiányzik, vagy valamely csomópont azonosítója hibás. A modell egy másik





Utcanév	Girrard
Utca típus	Avenue
Bal old. házszám	701-799
Jobb old. házszám	700-798
Bal tömb	38
Bal körzet	12
Jobb tömb	31
Jobb körzet	12
Kezdőpont	321
Koordináták	155 000 - 232 000
Végpont	322
Koordináták	156 000 - 234 000

12. ábra *A DIME file grafikai elemei és egy rekordja*

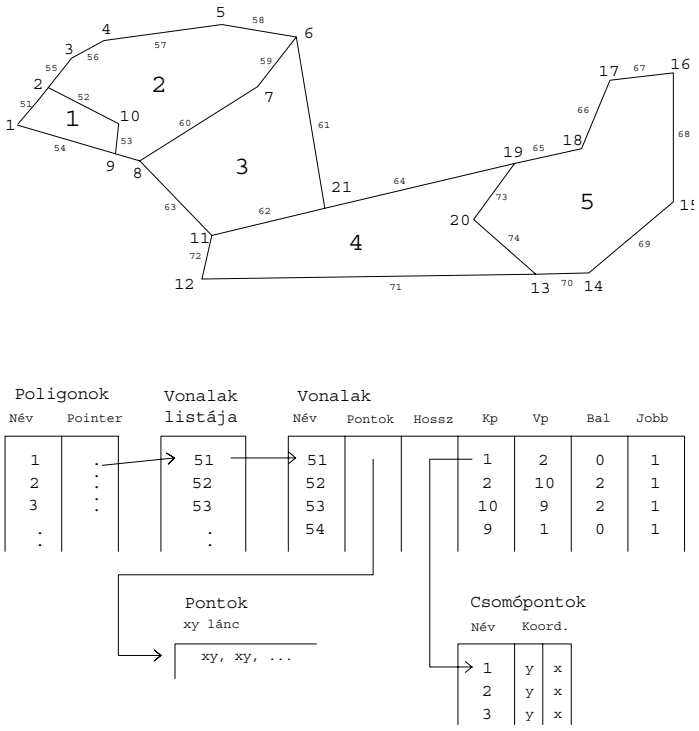
figyelemre méltó vonása, ahonnan a nevét is kapta , hogy minden szakasz duplán definiált a térben, azaz postai címmel és derékszögű (UTM) koordinátákkal is.

A modell legfőbb hátránya az, hogy a szakaszok tárolása semmilyen szempont szerint sem rendezett. Ezért egy-egy szakasz keresésekor szekvenciális keresést kell végezni az egész file-ban. Ha egy tömb határszakaszait kívánjuk megkeresni, akkor annyiszor kell a keresést elvégezni, ahány szakasz határolja a tömböt.

1.3.2.5. POLYVRT

A POLYVRT (POLYgon conVeRTer) modellt Peucker és Chrisman dolgozták ki a 70-es évek végén. Ez a modell kiküszöböli az eddigi modellek alapvető keresési hátrányait azáltal, hogy az egyes adatelemeket elkülönítve tárolja egy hierarchikus adatstruktúrában (13. ábra). Hogy az egyes elemek elkülönítése mind logikailag, mind topológiailag értelmessé váljon, az alapegységnek a vonalat nevezték ki. A vonal egyenes szakaszok sorozatából áll, kezdő- és végpontja egy-egy csomópont. Csomópont két vonal metszésekor keletkezik. A vonalak töréspontjainak koordinátái nem a vonal rekordjában kerülnek tárolásra. Ehelyett egy pointer mutat a külön pont file-ban elhelyezkedő információ elejére. Hasonlóképpen pointerek mutatnak a poligon file-ból a poligonokat határoló vonalakra. Az egyes vonal rekordok megőrzik a GBF/DIME-ban is használt irány és topológiai információkat, azaz a kezdő- és végpontokat, valamint a bal és jobb oldali poligonokat.
Ennek a struktúrának számos előnye van a GBF/DIME-mal szemben. Először is a hierarchikus felépítés lehetővé teszi, hogy egyidejűleg csak bizonyos típusú adatokat hívjunk elő. Egy másik előnye, hogy poligonok szomszédosságát vizsgáló lekérdezések esetén csak a poligonokat és vonalakat tartalmazó file-okban kell keresnünk. A koordinátákkal addig nem is kell törődnünk, amíg

valamilyen művelethez, például kirajzoláshoz, vagy távolságszámításhoz nincsen rájuk szükség.
A vonal rekordok száma a POLYVRT adatbázisában csak a poligonok számától függ, nem pedig a vonalak részletességétől. Ez a fizikai elkülönítés hatékonyabb központi-memória kihasználást és feldolgozási gyorsaságot eredményez számos művelet esetén. Azonban a fizikai elkülönítés szintén szükségessé teszi a pointer struktúrát. Ezek a "nem adatelemek" jelentős túlsúllyal terhelik a modellt, mely túlsúly sok elemet tartalmazó adatbázisban már elfogadhatatlan méretet ölthet. Egy másik hátránya, hogy a hibás pointerek megtalálása és kiszűrése rendkívül bonyolult. A struktúra kezdeti felépítése is nehézkes és időrabló feladat.



13. ábra *A POLYVRT adatmodell és adatstruktúra*

1.3.3. Hibrid adatmodellek

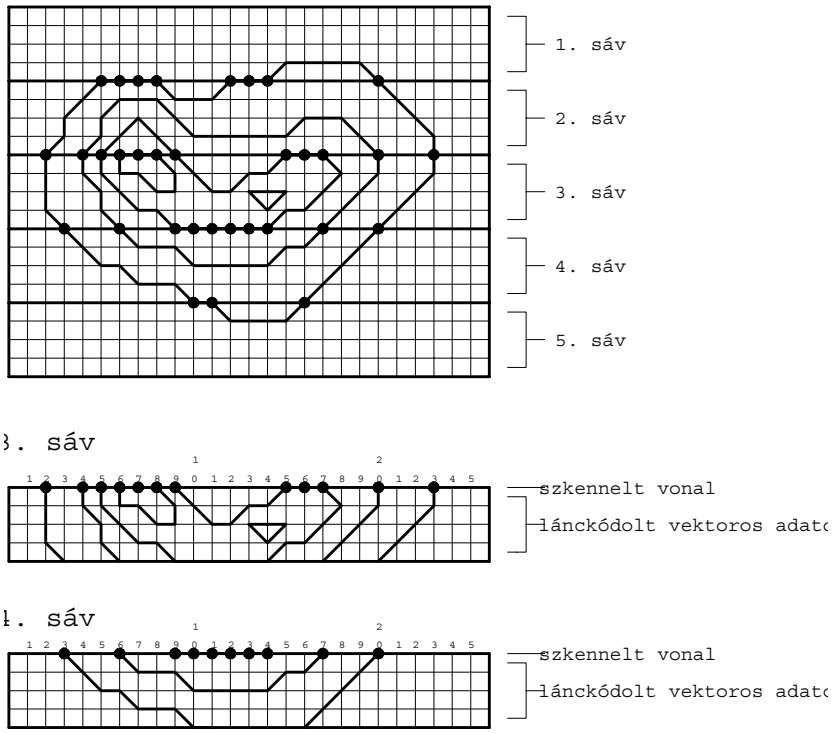
Az adatmodellezés szempontjából a vektoros és a tesszellációs adatmodellek egymás logikus párját képezik. A vektoros modell alap logikai alkotója a térbeli elem, mely azonosítható a terepen, vagy az adott alkalmazásban jön létre. Ez tehát magában foglalhatja a tavakat, folyókat, utakat és más egységeket. Ezen elemek térbeli elrendezése explicit módon, az elemek attribútumaiként kerül tárolásra. Ugyanakkor a tesszellációs modell alap logikai alkotója a térbeli helyzet. Egy adott elem léte az adott helyen szintén explicit módon, helyzeti attribútumként kerül tárolásra. A vektoros adatmodellek a térképi vonalak közvetlen digitális formái. Ez azt jelenti, hogy az algoritmusok is a hagyományos kézi módszerek digitális alkalmazásai felé tendálnak. Ezért a vektoros algoritmusok jobban kidolgozottak és ismertek. Azonban a vektoros adatmodellek elsődleges hátránya, hogy a térbeli kapcsolatokat explicit módon kell tárolni, vagy számítani. Miután gyakorlatilag végtelen sok lehetséges térbeli kapcsolat létezhet, az alkalmazás számára szükséges alapvető kapcsolatokat előre kell megfogalmazni. Ugyanakkor a tesszellációs adatmodelleknél a térbeli kapcsolatok gyakorlatilag beépítettek. A raszteres adatmodellek szintén jól illeszkednek a modern, nagysebességű grafikus input és output eszközökhöz. Elsődleges hátrányuk azonban, hogy általában rendkívül nagy tárigényűek.

Jól látható tehát, hogy egyik fajta adatmodell sem jobb a tér ábrázolására. Az ábrázolás és az algoritmusok előnyei mindkettőnél alkalmazásfüggőek, noha elméletileg mindkettő képes bármilyen típusú adat, vagy eljárás fogadására. Ezen dilemma feloldásának egy módszere a hibrid típusú adatmodellek fejlesztése és alkalmazása, melyek magukba foglalják mindkét struktúra tulajdonságait. Az első és talán legismertebb ilyen adatmodell, a Vaster adatmodell, melyet Peuquet fejlesztett ki.

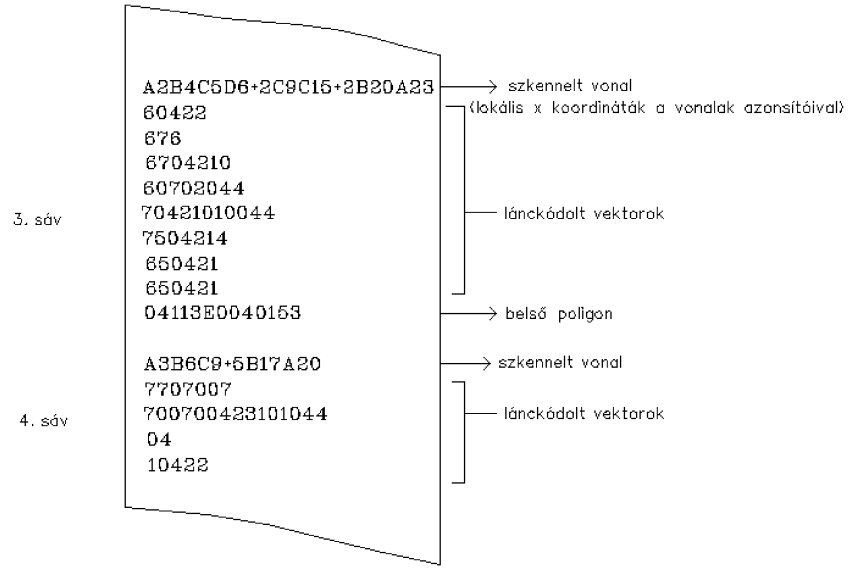
1.3.3.1. A Vaster adatmodell

A Vaster struktúra alap logikai eleme a sáv (Peuquet elnevezésében 'swath'). Minden sáv egy ismert, konstans távolságot fog át az y tengely mentén, melyek a szkennelt vonalak egy csoportját képeznék, ha az adatokat raszter formában tárolnánk (14. ábra). Minden sáv egy raszter komponenst és egy vektor komponenst tartalmaz. Mindkét komponenst ugyanabban a rács felbontásban rögzítjük. Minden sáv kezdő széle (mely a minimum y értékhez tartozik) szkennelt vonalként raszter formában kerül tárolásra és mint a sáv indexe működik a továbbiakban. Ez az index egy azonosítót és x koordinátát tartalmaz minden egyes vonalhoz, melyet metsz, vagy érint. A sáv további adatai vektor formában kerülnek tárolásra. Az egyes sávokban lévő vektorokat a szkennelés sorrendjében rendezi, növekvő x szerint; a belső poligonok külön kerülnek leírásra. Az index rekord minden egyes eleme a sáv vektorainak kezdőpontjait képezi. A vektorok lánckód formájában kerülnek tárolásra (15. ábra).

Miután a Vaster adatstruktúra raszter formátumú adatokat tartalmaz és a többi adat ezekhez a raszteres adatokhoz hely szerint kapcsolódik, a raszter formátumra jellemző térbeli rendezettség és a térbeli kapcsolatok megőrzése jelentős mértékben megmaradt. Ráadásul mindkét adatformátum jelenléte a Vaster struktúrában előnyt jelent a keresésnél és feldolgozásnál a geometriai adatbázist tartalmazó alkalmazásokban.
Funkcionálisan, a Vaster formátum kétszintű hierarchia szerint építkezik. A sávok raszteres része egymagában is használható sok gyakran használt eljáráshoz vagy eljárásrészlethez anélkül, hogy igénybe kellene venni a részletesebb vektoros file-t, ezáltal jelentősen meggyorsítva a feldolgozást. Ez azt igényli, hogy az adatbázis raszteres részét külön file-ban tároljuk. Sok lekérdezés esetén, ahol közelítő eredményt kívánunk, elég, ha csak az adathalmaz durvább, raszteres részét használjuk; például terület-, kerület-, vagy hossz-számítások esetén. Sok térbeli kapcsolat, mint például egy pont tartalmazása, vagy relatív helyzete is



14. ábra *A Vaster elrendezés és a logikai rekord sávok (swath)*



15. ábra *Vaster struktúrájú adatok digitális formában*

megválaszolható raszterformátumú adatok és raszterorientált algoritmusok alkalmazásával, melyek általában hatékonyabbak az ilyen típusú feladatokra. A közelítés miatt létrejövő hiba a sáv szélességétől és a vonalak bonyolultságától függ. Ha ez a hiba elfogadhatatlan mértékű, akkor is a vektoros adatok csak egy kis részét kell igénybe venni.

A Vaster struktúra szintén jól használható teljes részletességű, vektoros, lánckódolt struktúraként. Bármely lánckódot használó algoritmus normál módon használható az egyes sávok láncrészeinek összekötése után, mely az indexrekordok segítségével egyszerűen elvégezhető.

A legnagyobb probléma ennél a struktúránál az adattárolás felbontásának megállapítása. Ez részben a mintavétel, azaz a rácshálózat méretének, másrészt a sávszélesség megválasztásának problémája. Az lenne a jó, ha minden főbb vonalat metszenénk legalább egyszer egy rasztervonallal, így nem kerülnének lényeges elemek egy sáv belsejébe. Ugyanakkor túl keskeny sávok esetén ugyanazt a vonalat túl sokszor metsszük el, így jelentősen megnő az adathalmaz.

2. Vonalas elemek matematikai leírásának módszerei

A geometriai adatok tehát három alapvető adattípusból épülnek fel, pontokból, vonalas elemekből és poligonokból. Miután mindhárom típus felfogható vonalas elemként, hiszen a pont lehet egy nulla hosszúságú szakasz, míg a poligon tulajdonképpen egy zárt vonalat jelent, érdemes kiemelten foglalkozni a vonalas elemek adatmodellekben történő ábrázolási módjaival. A térképek is nagyrészt vonalakból és poligonokból állnak, hiszen a felületek is jól ábrázolhatók körvonalaikkal, melyek természetesen poligonokat képeznek.

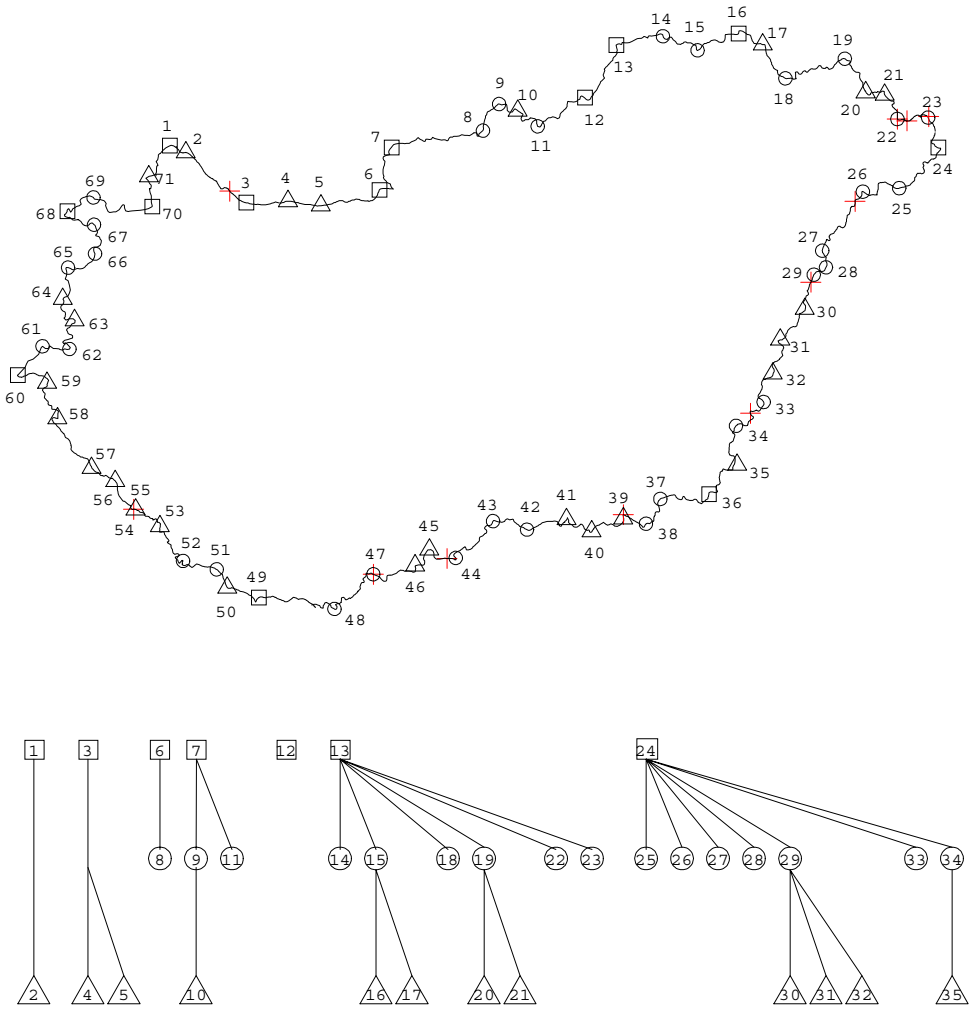
Külön kiemelendő, hogy a vonalas elemek modellezésénél cél kell legyen a méretarányfüggő megjelenítés lehetőségének megteremtése is. Ezért itt elsősorban a hierarchikus adatstruktúrákat említem meg, melyek lehetővé teszik a különböző méretarányú megjelenítést.

2.1. Hagyományos vektoros módszerek

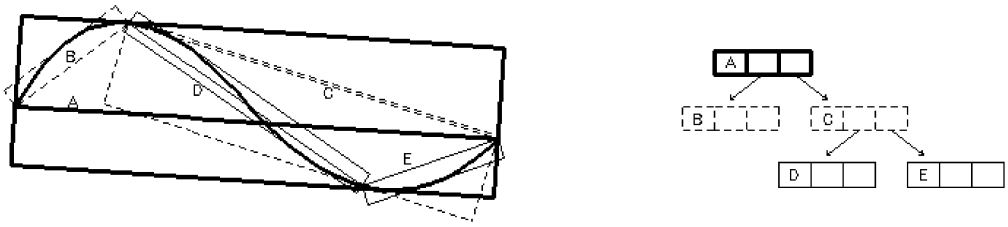
Valószínűleg a vonalak tárolásának legegyszerűbb módja egy szükséges hosszúságú koordinátalánc, mely egy adott objektumot jelképez. Ez a koordinátalánc méretarányfüggő, így a különböző méretarányú megjelenítéshez vagy a legnagyobb felbontásban kell az elemeket tárolni és a megjelenítés előtt egyszerűsíteni, vagy különböző méretarányú verziók tárolása szükséges. Kis méretarányú megjelenítés esetén azonban rendkívül sok adatot kell elérni és feldolgozni egyszerre, mely jelentősen lelassítja a megjelenítést, ha az egyszerűsítő algoritmust használjuk. Ha különböző méretarányokban tároljuk az adatokat, akkor a megjelenítés jelentősen felgyorsul, hiszen köztes méretarányú megjelenítés esetén is csak a legközelebbi nagyobb méretarányú verzió adatait kell figyelembe venni.

A tárolt vonalak adattömege jelentősen csökkenthető relatív koordináták alkalmazásával. Ez lehet relatív egy, a vonal környezetében adott ponthoz viszonyítva, vagy még nagyobb tömörítés esetén minden pont az előző ponthoz képest definiálható (ld. lánckódok).

Egy adatstruktúra, melynek célja a vonalak többszörös tárolásának kiküszöbölése, de mégis több méretarányú tárolása, az úgynevezett vonal-generalizálási fa, vagy vonalfa (line tree). Ebben a struktúrában valamilyen egyszerűsítő algoritmus (pl. Douglas-Peucker algoritmus — ld. 3. fejezet) segítségével osztályozzuk a vonal töréspontjait különböző méretarányfüggő szignifikancia szerint. A legmagasabb szint a legkisebb méretarányú megjelenítéshez szükséges pontokat tartalmazza, míg a lejjebb lévő szintek egyre több részletet tartalmaznak (16. ábra). Tehát egy adott méretarányú megjelenítésnél csak az adott szintig szükséges előhívni az adatokat, így nem kell felesleges adatokkal foglalkozni és az adatok többszörös tárolása is elkerülhető. Ezt a struktúrát kétféleképpen valósították meg. Az egyik módszer minden ponthoz egy bal- és jobboldali azonosító értéket rendel, mely a fa következő szintjén lévő szomszédos pontok számát rögzíti. A fa minden szintje logikailag független rekordokba vagy file-ba kerül. A másik módszer is különválasztja a szinteket, de egy sorszámot tárol minden ponthoz, mely meghatározza a pont helyét az eredeti vonalban. A megjelenítésnél ezután a különböző szintről származó pontokat a sorszám szerint kell összekötni. Ezeknél a módszereknél is alkalmazhatjuk a relatív koordinátákat az adatmennyiség csökkentésére. Ennek egyik módja lehet, hogy minden pontot a felette lévő szinten lévő "szülőjéhez" képest relatívan tárolunk.



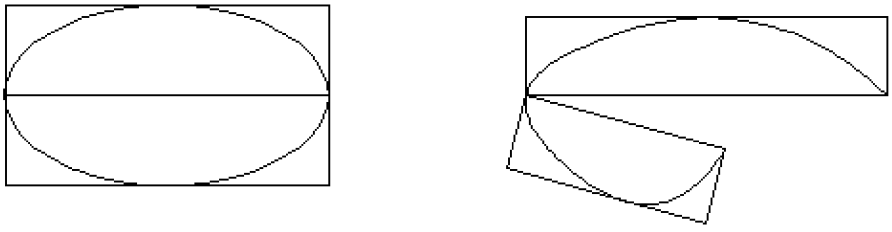
16. ábra *A line tree felépítése. A négyzetek, körök és háromszögek a Douglas-Peucker algoritmus által, különböző toleranciák esetén kiválogatott pontokat képezik, melyekből azután elkészíthető a megfelelő struktúra. Itt a második verzió látható, ahol a pontok sorszámot kapnak.*



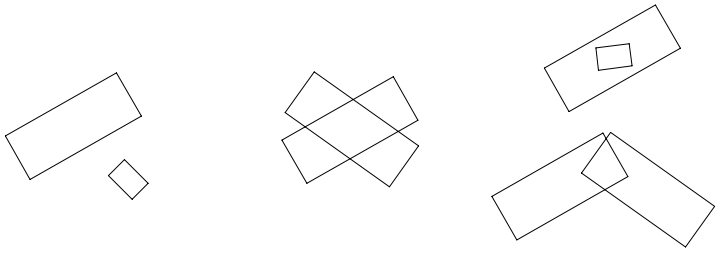
17. ábra *Egy görbe felbontása sávokra és az így kialakuló strip tree*

2.2. Strip tree

A vonalfát tekinthetjük a Ballard által kidolgozott strip tree közeli rokonának. Elsődleges formájában a strip tree egy bináris fa, melynek minden csomópontjában a vonal egy darabjának befoglaló négyszögét tároljuk (17. ábra). Ezen befoglaló négyszög egy téglalap, melynek két oldala párhuzamos a vonaldarab két végpontját összekötő egyenessel. A fa gyökere a teljes görbét magában foglalja,melyet aztán két részre bontunk ott, ahol a görbe érinti a téglalapot. Ilyen érintőpont mindig található. Ha két érintőpont van, akkor a görbe két végpontját összekötő egyenestől távolabb lévő érintőpont lesz az osztópont. Ezeket a sávokat (téglalapokat) ezen módszerrel addig osztjuk, míg a sáv szélessége el nem ér egy adott értéket, mely lehet akár nulla is, ha az eredeti felbontásban kívánjuk ábrázolni a vonalat.
A különböző méretarányú megjelenítéskor a fát különböző mélységig kell bejárni az egyes ágakon, és a sávszélességet minden egyes csomópontban meg kell vizsgálni.
Ahhoz, hogy bonyolultabb görbéket, pontosabban zárt görbéket és végpontjaikon túlnyúló görbéket (18. ábra) is kezelni tudjunk, ezeket két részre kell bontani, mely felbontás után az eredeti elv gond nélkül alkalmazható.
A befoglaló téglalapok tárolásával egyes számítások, például a metszéspont számítása, különböző felbontásban elvégezhetők. A fa többfelbontású voltának köszönhetően ezek a számítások



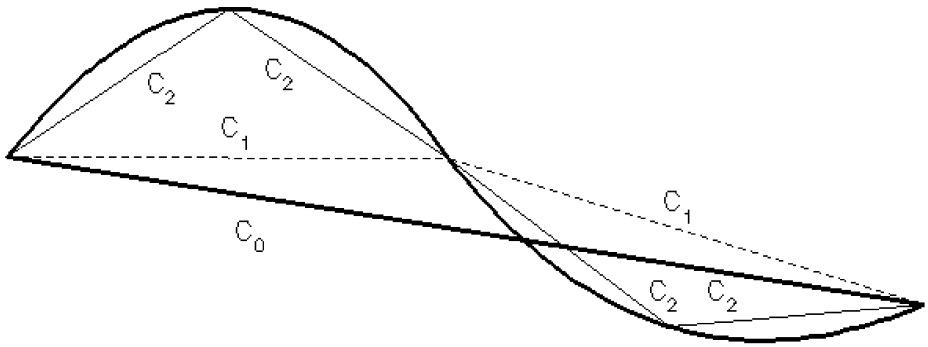
18. ábra *Zárt, és a végpontjaikon túlnyúló görbék felbontása*



19. ábra *Két strip tree metszéspontjának keresése*

nagyon gyorsan elvégezhetők a felsőbb szintű téglalapok segítségével. Ha nagyobb felbontás szükséges, a sávon belül folytatható a számítás, jelentősen csökkent adatmennyiséggel (19. ábra). Így tehát gyorsan eldönthető, hogy a metszéspont létezik-e vagy sem, illetve, hogy tovább szükséges-e haladnunk a fán a kérdés eldöntése érdekében. Más műveletek, melyek hatékonyan elvégezhetőek ezen struktúra segítségével : görbe hosszának számítása, zárt görbék területének meghatározása, pont tartalmazása, stb. Egy, a strip tree segítségével rendkívül elegánsan megoldható probléma annak meghatározása, hogy két görbét tekinthetünk-e ugyanazon görbe kódolt formájának. Erre például digitalizálás után lehet szükség. A megoldást úgy kaphatjuk, ha megvizsgáljuk, hogy a két görbe sávjai milyen mértékben fedik egymást.

A strip tree-nek egy változata a Günther-féle arc tree, mely a görbe hosszának szabályos felosztásán alapul. Pontosabban a görbét egyenes szakaszokkal közelítjük, melyek végpontjai a görbén találhatók. A görbe *k*-ik közelítése 2^k szakaszból áll, melyek $s/2^k$ ívhosszakat helyettesítenek, ahol *s* a görbe teljes hossza (20. ábra). Az arc tree közelítés folyamata hasonlóképp zajlik, mint a strip tree esetén. A folyamat akkor áll le, amikor



20. ábra *Egy görbe és három szintű közelítése arc tree-vel*

minden szakasz megfelelően közelíti a görbét (például a hosszak különbsége egy adott értéken belülre esik). Eredményül szintén bináris fához jutunk. Az arc tree létrehozásához kétszer kell végigjárnunk a görbét. Elsőként meg kell állapítanunk a hosszát, ami meglehetősen nehézkes feladat lehet. Ha ismert a görbe zárt matematikai alakja (pl. *y=f(x)*), akkor ez integrálással kiszámítható. Általában valamilyen simító algoritmust is használnak, hogy kiszűrjék a zajt. A második végigjárás magától értetődő, az arc tree létrehozását takarja.

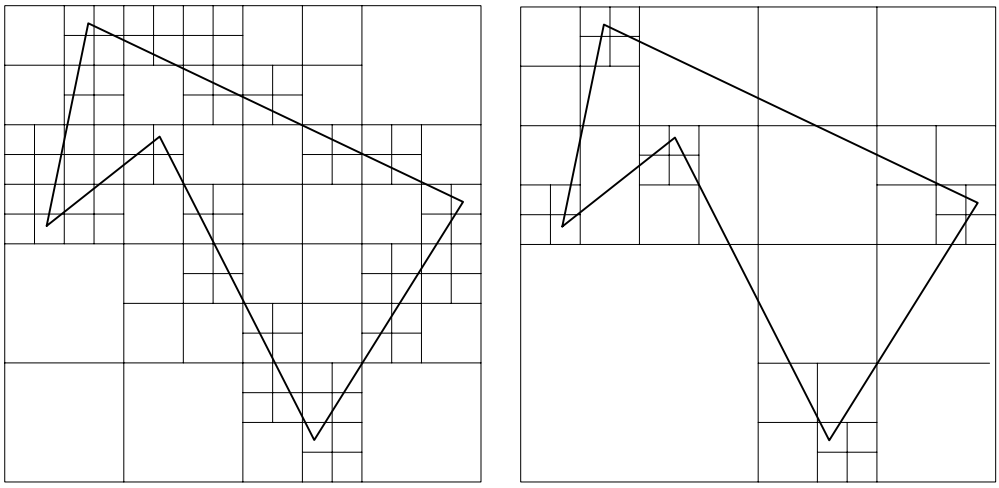
Günther bebizonyította, hogy az arc tree azzal az érdekes tulajdonsággal bír, hogy minden egyes közelítő szakaszhoz és a hozzá tartozó ívdarabhoz számítható egy ellipszis, mely teljes egészében tartalmazza az ívdarabot és a szakaszt. Ezekkel az ellipszisekkel a metszéspont keresése hasonlóan végezhető, mint a strip tree esetén. Ezen ellipszisek előnye a strip tree téglalapjaival szemben, hogy nem szükséges külön foglalkozni a zárt, vagy végpontjaikon túlnyúló görbékkel.

2.3. Négyesfákon alapuló módszerek

A strip tree módszernél a vonalas elemeket téglalapokkal közelítettük. A négyesfa módszerekkel hasonló eredményt érhetünk el különálló, a kettő hatványaival megegyező oldalhosszú négyzetekre való felbontással. A négyesfák számos változatát használják, melyeket aszerint különböztethetünk meg, hogy milyen típusú adatokat ábrázolunk segítségükkel. A Samet és Webber által definiált PM négyesfa és a Tamminen-féle EXCELL módszer kivételével mindegyik pixel-alapú, tehát közelítő módszer, melyek pontosságát az adatok felbontása korlátozza. Ellenkezőleg, a PM négyesfa és az EXCELL módszer pontos ábrázolása a vonalas elemeknek.

2.3.1. Az edge quadtree

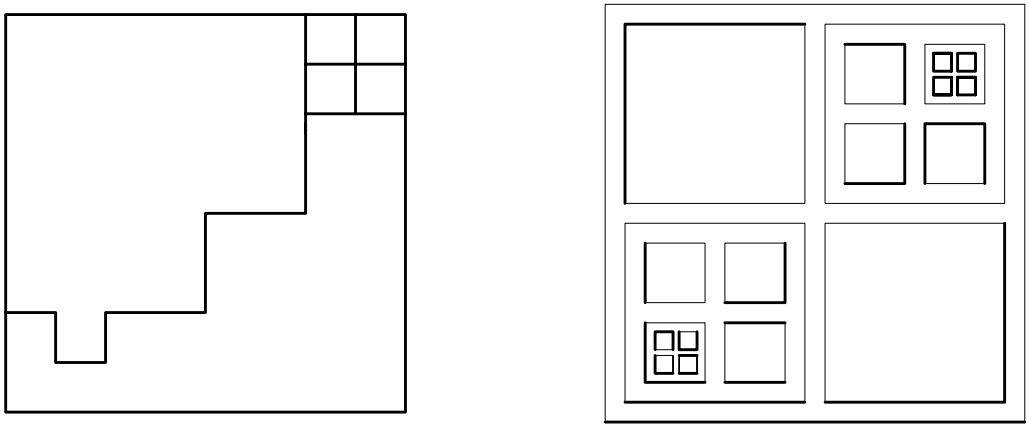
Ezt a módszert Shneier dolgozta ki vonalas elemek ábrázolására a felületi információk tárolásával analóg módszer szerint. Azt a képrészt, mely vonalas elemet vagy annak egy részét tartalmazza addig osztunk ismételten négy részre, míg olyan négyzethez jutunk, mely egyetlen, egyenessel közelíthető görbedarabot tartalmaz. A fa minden levele a következő információkat tartalmazza a rajta áthaladó élről : irány, metszék, esetleg szín, és egy hibaérték, mely az egyenessel való közelítés által bevezetett hibát jellemzi. Ha az él a levélben végződik, azt egy kóddal jelzik és a metszék a végpont helyét jelenti. E módszer alkalmazása olyan négyesfához vezet, melyben a hosszú, egyenes szakaszokat nagyobb négyzetek sorozata ábrázol, míg a sarkok és metszéspontok környezetében kisméretű négyzetekre van szükség. Természetesen lehetnek négyzetek, melyek egyáltalán nem tartalmaznak éleket. Ez a módszer azonban kevesebb négyzetet igényel, mint a szabályos négyzet alapú tesszelláció (21. ábra).



21. ábra *A hagyományos és az edge quadtree*

2.3.2. Line quadtree

A Samet és Webber által kidolgozott line quadtree más oldalról közelíti meg a kérdést. Ezzel a módszerrel mind az egyes felületek területe, mind azok határai hierarchikus módon ábrázolhatók. Ebben is különbözik az egyszerű négyesfától, mely csak a felületeket, valamint a strip tree-től, mely csak a határvonalakat (görbéket) ábrázolja hierarchikusan. A line quadtree esetében a képet rekurzív módszerrel olyan részekre osztjuk, melyek belsejében nincs vonalszakasz. Minden levélhez rendelünk egy kódot, mely azt jelzi, hogy a négyzet oldalai közül melyek képeznek határvonalat. Így a leveleket nem aszerint különböztetjük meg, hogy feketék-e vagy fehérek, hanem a határvonalak képezik az információt. Az alapelv az, hogy ahol egy tovább osztható négyzet oldala nem képez T elágazást a



22. ábra *Egy egyszerű vonalas térkép és line quadtree reprezentációja*

leszármazottainak valamely oldalával, akkor az adott oldalt határvonalként jelöljük (22. ábra). Az, hogy a határvonal információt nem a végpontokban tároljuk, elősegíti a vonalkövető algoritmusok gyors végrehajtását. A line quadtree egyébként ugyanannyi levelet tartalmaz, mint a hagyományos négyesfa.

Az edge quadtree és a line quadtree tehát a vonalas elemek ábrázolásának közelítő módszerei. Ebből adódóan gyakran meglehetősen nagy adattömeget képezhetnek egy bonyolultabb képnél, mert nagyobb pontossági igény esetén a vonalakat esetleg pixel mélységig szükséges felbontani, ami nagyon nagyméretű négyesfákhoz vezethet. Ráadásul a vonalas térképek egyes elemeit nem lehet közvetlenül ábrázolni ezen módszerekkel. Például lehetetlen egy csomópontban metsződő öt vonalat line quadtree segítségével ábrázolni, mert öt négyzet nem találkozhat egy pontban. További probléma az eltolásra és elforgatásra való érzékenység, mely műveletek pontosságcsökkenéshez vezethetnek az eredeti közelítéshez képest.

2.3.3. A PM quadtree

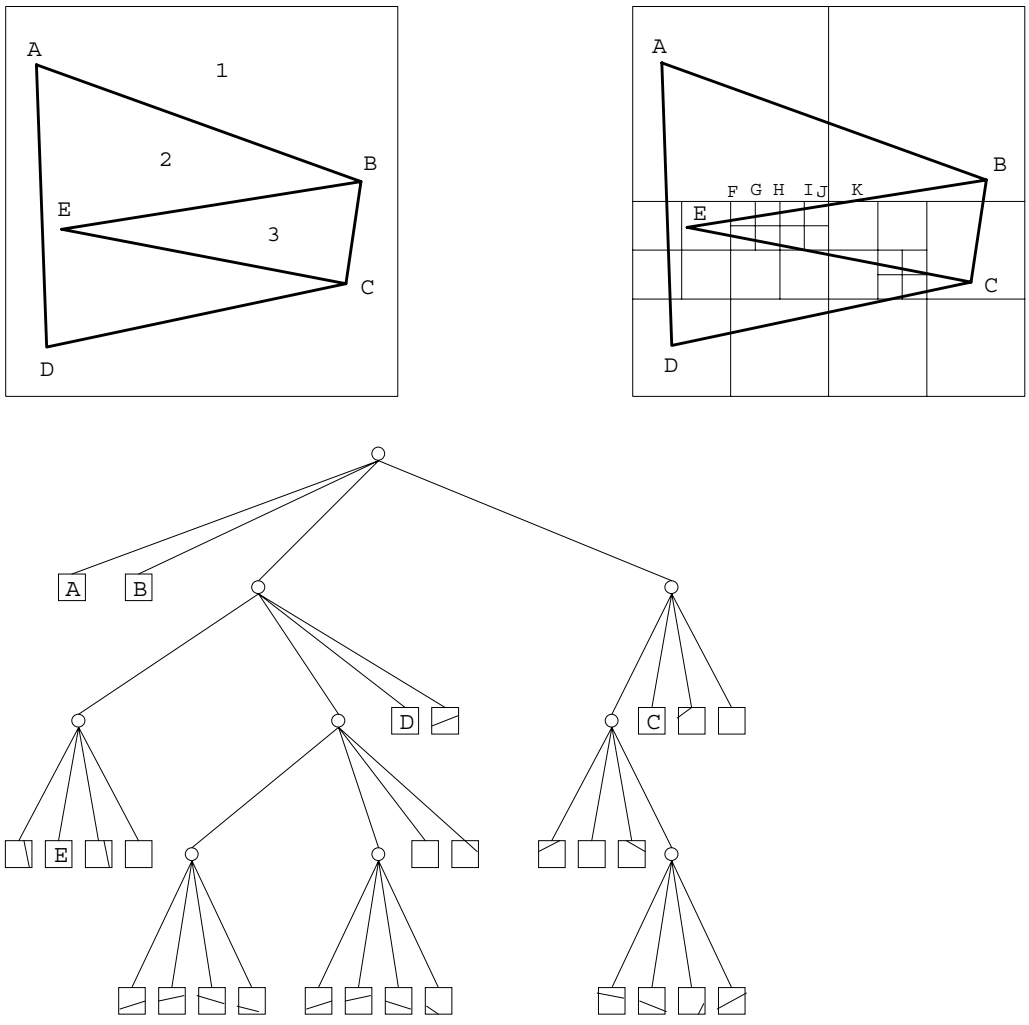
A PM quadtree egy összefoglaló kifejezés, melyet a Samet és Webber által kidolgozott több, szorosan kapcsolódó négyesfa típusú adatstruktúrákra használnak. A legegyszerűbb, a PM1 quadtree alapelve, hogy egy négyzetet addig osztunk négy részre, amíg olyan blokkokhoz jutunk, melyek nem tartalmaznak egynél több vonalat. Hogy egyszerűbben kezelhessük a keresztező vonalakat, ha egy négyzet töréspontot tartalmaz, akkor az magában foglalhat több vonalat is, feltéve, hogy ezek végpontja az adott pont. Egy blokk azonban nem tartalmazhat egynél több töréspontot (23. ábra).

A PM1 quadtree definícióját pontosíthatjuk, ha a vonalas térképet egyenes vonalakból álló síkbeli gráfnak tekintjük, mely csak csúcsokat és éleket tartalmaz. Bevezetjük a q-él fogalmát, ami a gráf éleinek azon szakaszát jelenti, melyet a négyesfa egy négyzete (levele) metsz ki az adott élből. Tehát egy élet q-élek sorozata alkot, melyek csak a végpontjaikban metsződnek. Például a 23. ábrán látható EB élt az EF, FG, GH, HI, IJ, JK, KB q-élek képezik. Most tehát a PM1 quadtree definícióját újrafogalmazhatjuk a következő feltételeknek eleget tevő négyesfaként :

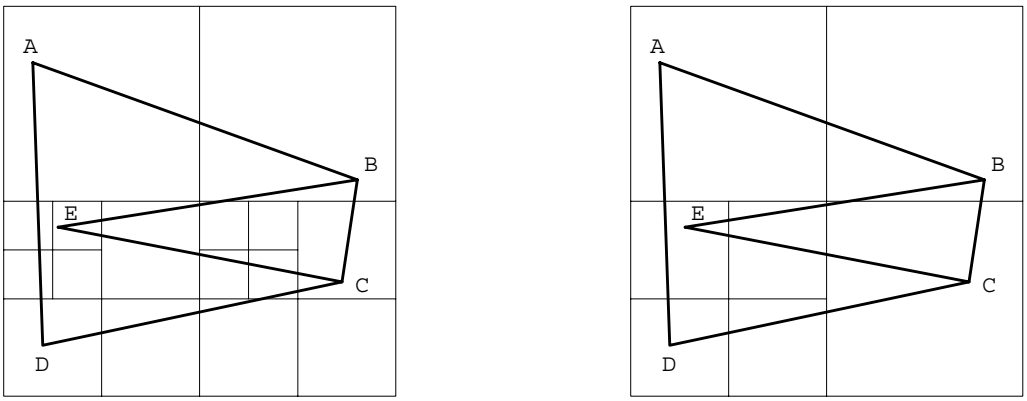
- A négyesfa levelei nem tartalmazhatnak egynél több csúcsot.
- Ha a négyesfa egy levele csúcsot tartalmaz, akkor nem tartalmazhat olyan q-élet, mely nem abban a csúcsban végződik.
- Ha a négyesfa egy levele nem tartalmaz csúcsot, akkor legfeljebb egy q-élet tartalmazhat.
- A négyesfa minden levele a lehető legnagyobb méretű legyen.

A négyesfa minden levele zárt, tehát amennyiben egy csúcs két, három, vagy maximum négy levél határán található, akkor a csúcsot mindegyik levél tartalmazni fogja.
Annak érdekében, hogy kisebb méretű négyesfák jöjjenek létre és ezáltal a tárolókapacitás csökkenjen, továbbfejlesztették a PM1

négyesfát. A PM2 négyesfát a harmadik feltétel feloldásával kapjuk, tehát ha a négyesfa egy levele nem tartalmaz csúcsot, akkor csak olyan q-éleket tartalmazhat, melyek egy csúcsban végződnek. A legfejlettebb változat, a PM3 quadtree a második feltétel feloldásával jön létre, tehát ebben a négyesfában egy csúcsot tartalmazó levél tartalmazhat más q éleket is, ha azok átszelik a négyzetet. Ebben az esetben minden levélben a q-élek hét osztályba sorolhatók. Az első osztályba azok a q-élek tartoznak, melyek az adott blokkban lévő csúcsban végződnek. A további hat osztályt azok a q-élek alkotják, melyek teljes



23. ábra *Egy egyszerű vonalas térkép és a hozzá tartozó PM1 quadtree felbontás és struktúra*



24. ábra A *23. ábra vonalas térképének PM2 és PM3 quadtree felbontása*

egészében átszelik a négyzetet. Ez hatféleképpen történhet : ÉN, ÉD, ÉK, KN, DN és DK, ahol az egyes betűk a négyzet északi, déli, keleti és nyugati oldalát jelentik, tehát például a DK osztályba azok a q-élek tartoznak, melyek a déli oldalon lépnek be és a keleti oldalon távoznak az adott négyzetből (24. ábra).

A PM négyesfák kényelmes, viszonylag hatékony adatstruktúrák számos művelet szempontjából. A pont a poligonban keresés, élek hozzáadása, két térkép átfedése, a clipping és az ablakolás műveletek viszonylag egyszerűen és gyorsan elvégezhetők a Samet és Webber által kidolgozott algoritmusokkal.

2.3.4. Az EXCELL módszer

A Tamminen által kidolgozott EXCELL módszernél csak az adott blokkban lévő szakaszok számától függ, hogy tovább bontjuk-e a négyzetet. Ez is tehát szabályos felbontáson alapul. A tárolóterületet véges kapacitású részekre bontjuk, melyek a

vonalas térkép egy-egy blokkját tartalmazzák. Az adatokat egyenes vonalszakaszok képezik, melyek keresztezik a blokkot. Ha egy rész megtelik, akkor a négyzetet felosztjuk. Azonban ha a térkép olyan csomópontot tartalmaz,melyben több él találkozik, mint amennyi egy tárrész kapacitása, akkor hiába bontjuk egyre kisebb részekre a területet, lehetetlen lesz minden élet egy tárrészben tárolni. Ilyenkor úgynevezett túlcsorduló tárrészeket hozunk létre. Ez azonban hátránya az EXCELL módszernek a PM négyesfákhoz képest.

2.4. Összefoglalás

A lánckód a legtömörebb az ábrázolásmódok között. Azonban ez egy meglehetősen helyhez kötött adatstruktúra és ezért meglehetősen nehézkes különböző műveletek elvégzésekor, mint például két görbe metszéspontjának keresésekor.
A szabályos felbontáson alapuló módszereknek több előnye is van a strip tree-vel szemben. Először is ezzel az adatstruktúrával egynél több görbét is ábrázolhatunk, ami térképek esetén meglehetősen fontos szempont. Egy strip tree-vel egyszerre csak egy görbe ábrázolható. Másodszor pedig ezek az adatstruktúrák egyediek, míg a strip tree esetében egy zárt görbe többféleképpen is ábrázolható. Ugyanakkor a strip tree előnye, hogy érzéketlen az eltolásra és az elforgatásra, mivel nincsen szorosan egy koordinátarendszerhez kötve. A szabályos felbontáson alapuló módszerek vonzóak, mert hatékony számítási lehetőséget nyújtanak számos művelet esetében. Azonban vannak olyan műveletek, melyeket különböző típusú geometriai elemeken kell végezni (pl. görbe metszéspontjai egy területtel). A strip tree egy elegáns adatstruktúra a közelítés szempontjából. Azonban hátránya, hogy a felbontási pontok koordinátarendszertől függetlenek, tehát az adatoktól függenek, nem pedig előre meghatározott helyeken történik a felbontás, mint a szabályos felbontáson alapuló struktúráknál. Tehát például a válasz keresése

egyes kérdésekre - mint például azon búzatermő területek meghatározása, melyek a Dunához képest 50 km-en belül vannak - meglehetősen bonyolult, ha a Dunát egy strip tree-vel, míg a búzatermő vidékeket négyesfával ábrázoltuk. A probléma az, hogy míg a négyesfa módszerek főként pointerkereső műveleteket igényelnek, addig a strip tree módszereknél bonyolult geometriai számítások szükségeltetnek. Tehát igazából egy szabályos felbontáson alapuló strip tree-re lenne szükség.

3. Vonalas elemek méretarányfüggő ábrázolása

Az előző részben tárgyalt adatstruktúrák mind alkalmasak vonalas elemek méretarányfüggő ábrázolására, mert hierarchiájuk részben ezen elv alapján épül fel. A vonalak egyszerűsítésére és generalizálására a legegyszerűbb és leggyorsabb módszerek a vonalak karakterisztikus pontjainak kiválasztásán alapuló algoritmusok, ezért ezekkel foglalkozom a továbbiakban.

A méretarány-független, egységes adatbázis minden GIS rendszer tervezőjének álma. A geometriai adatokat általában a legnagyobb pontossággal tárolják. Elvileg ebből az adatbázisból minden kisebb méretarányú térkép, vagy kisebb pontosságú geometriai adathalmaz előállítható. A legnagyobb akadály azonban a generalizálás problémája. A méretarány megváltoztatása nem csak a koordináták értékeinek egyszerű méretarány-tényezővel történő osztásából áll. A kézi generalizálásnál a kartográfusnak több döntést kell egyidejűleg meghoznia. Kiválasztja a fontosabb jellemzőket, melyek közül néhány akár felnagyítva jelenhet meg, míg más, kevésbé fontos részletek eltűnnek. Ebben a folyamatban a kartográfus tanulmányozza a vonalat, eldönti mely részleteket őriz meg, határoz az eltolás mértékéről és úgy húzza meg a vonalat, hogy az eleget tegyen a hagyományosan kialakult szabályoknak. A végcél az, hogy jól áttekinthető, a tájékozódást elősegítő térképet kapjunk. Tehát a generalizálás egy meglehetősen szubjektív, nagy gyakorlatot igénylő összetett folyamat. Egy automatizált generalizáló algoritmusnak három alapvetően különböző műveletet kellene elvégeznie. A vonalak egyszerűsítése nem azonos a vonalak generalizálásával, mert az egyszerűsítés, tehát a szükségtelen részek kiiktatása csak egy része a generalizálásnak, melynek másik két alapeleme a vonalak simítása és esetleges eltolása.

Az objektív, számítógéppel segített generalizálás első kísérleteiben valamilyen képlettel próbálták kiszámítani a

megőrzendő információ mennyiségét egy adott méretarányban. Ilyen például a Töpfler és Pillewizer által alkotott "gyökszabály", mely szerint

$$nf = na \cdot Cb \cdot Cz \cdot \sqrt{\frac{Ma}{Mf}} \quad ,$$

ahol nf és na a levezetett és az eredeti térkép objektumainak száma, Ma és Mf a méretarány-tényezők, míg Cb és Cz a szimbólumokhoz kapcsolódó konstansok. A vonalakat pontok sorozata határozza meg. A gyökszabály kimondja, hogy a méretarány csökkenésével az információmennyiségnek (a pontok számának) a méretarány-változás gyökével arányosan kell csökkennie. Azonban ez homogén információ-eloszlást feltételez a vonal teljes hosszán, azaz minden koordinátát egyformán fontosnak ítél. Tehát a szabály meghatározza, hogy hány pontot kell megszüntetni, de nem mondja meg, hogy melyeket.
Egy másik lehetőség, hogy a vonal mentén állandó távolságonként pontokat mintavételezünk (Lang). Ezekben az algoritmusokban egy pont megtartásának vagy elhagyásának valószínűsége a pontnak a pontok sorozatában elfoglalt helyétől függ, nem pedig a vonal jellemző vonásaiban játszott szerepétől. Ezt a problémát Boyle úgy próbálta megoldani, hogy a pontokat hierarchikus rendszerbe szervezte. A hierarchiát a pontok koordinátáihoz rendelt súllyal valósította meg.

Egy másfajta geometriai szemszögből a vonalakat csak a görbületük és kígyózásuk különbözteti meg. Maling szerint a kígyózás mérhető, ha a vonalra ráengedünk egy matematikai simító függvényt. Azonban egyes görbék esetén a matematikai leírás túl bonyolulttá válik ahhoz, hogy gyakorlati haszna legyen. A paraméteres simítás egyik módja a spline függvénnyel történő simítás, melyet gyakran használnak CAD programokban. A spline minden szakaszához egy-egy egyenletrendszert kell megoldani, ami meglehetősen nagy számítókapacitást igényel. Léteznek egyéb simító függvények is, melyek mind más-más eredményre vezetnek ugyanabból a vonalból kiindulva. További probléma a

pontosság kérdése, mert a simító függvények többsége nem őrzi meg az eredeti pontokat.

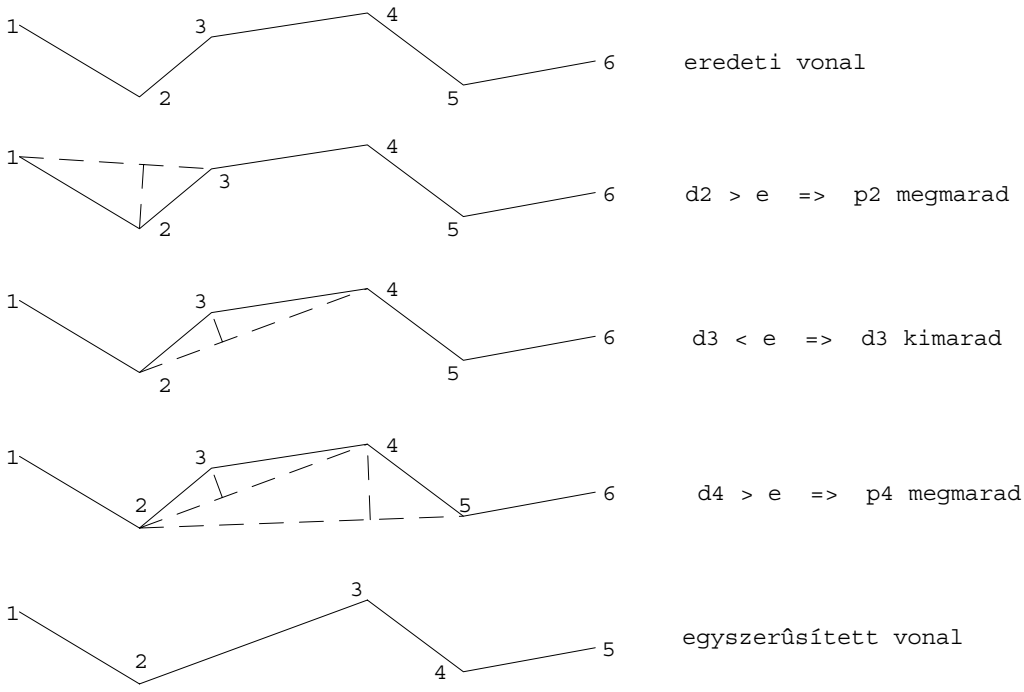
A későbbiekben a generalizáló algoritmusok elsősorban az egyszerűsítést részesítették előnyben a simító függvényekkel szemben. A térképkészítés automatizálásának egyre növekvő szerepével a kartográfusok jobban figyelhettek a megjelenítés pontosságára. Ennek megfelelően a generalizálás fő feladata az információ szűrése lett. A szűrés automatizálása elsősorban határok felállításával történik, melyeken belüli pontok elhagyhatók.

3.1. N-edik pont algoritmus

Ez a legegyszerűbb algoritmus, mely semmilyen módon nem veszi figyelembe a szomszédos pontok közötti kapcsolatokat, távolságokat. Az n-edik pont algoritmusban egyszerűen kiválasztjuk az első, majd minden n-edik pontot az eredeti vonalon, míg a többit elhagyjuk. Noha ez a leggyorsabb módszer, minőségileg meglehetősen halvány eredményt nyújt.

3.2. Merőleges távolság algoritmus

Ebben az algoritmusban egy közbenső pont merőleges távolságát vizsgáljuk a két szomszédos pontját összekötő egyenestől. Ha a távolság nagyobb, mint az előre definiált határérték, akkor a közbenső pontot megtartjuk, mert a vonal jellegének megtartásában lényeges szerepet betöltőnek minősül. Ha a távolság kisebb, mint a határérték, akkor a pontot elhagyjuk és az egyszerűsített vonal csak a két szélső ponton halad keresztül (25. ábra).

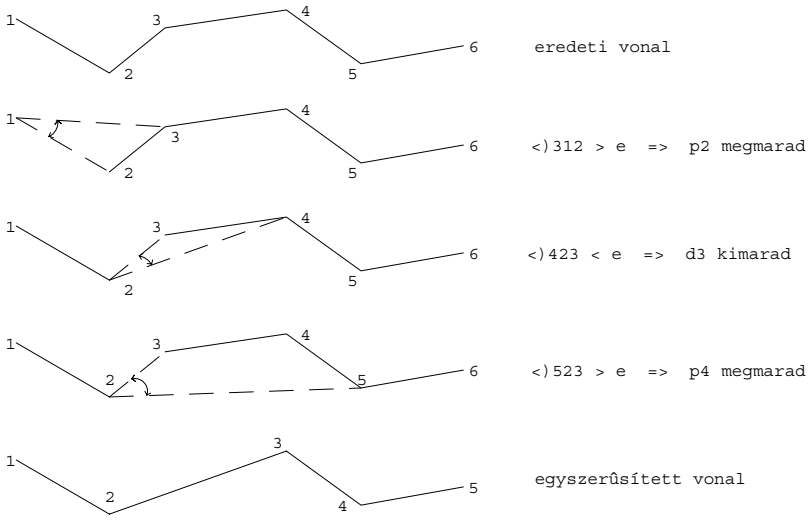


25. ábra *A merőleges távolság algoritmus*

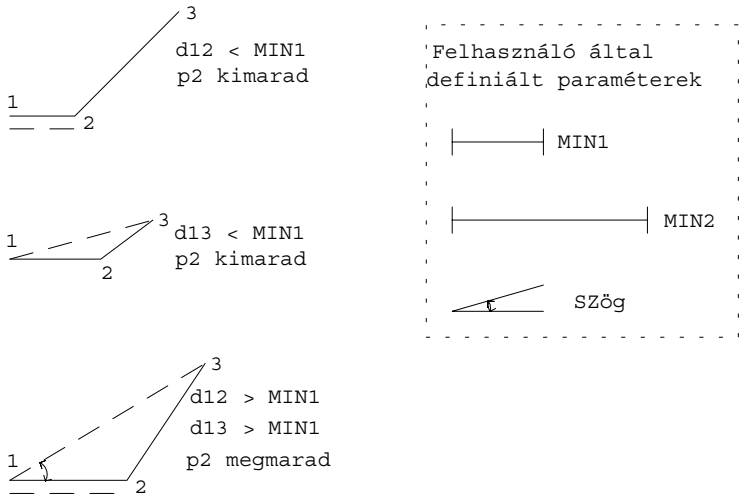
3.3. A szöghatárérték algoritmus, Jenks algoritmus

Ez az algoritmus is mindig három egymást követő pontot vesz figyelembe, és megvizsgálja az első és a második pont, valamint az első és a harmadik pont által meghatározott vektorok szögeltérését. Ha a szögérték kisebb, mint az előre definiált érték, akkor a második pontot elhagyjuk és a következő ponttal folytatjuk a vizsgálatot. Ha a szögérték a határértéknél nagyobb, akkor az első pontot rögzítjük, majd a második pont lesz az első és az eddigi harmadik a második (26. ábra).
Jenks továbbfejlesztette ezt a módszert két távolságellenőrzés hozzáadásával. Itt három előre beállított értéket ellenőriz az algoritmus: a minimum távolságot az első és második pont között, a minimum távolságot az első és a harmadik pont között és a szöghatárértéket. Ha a pontok közötti távolságok kisebbek a

megfelelő minimumértékeknél, akkor a második pontot elhagyjuk. Ha mindkét távolság nagyobb, akkor következik a szög ellenőrzése, a fenti módszerrel (27. ábra). Ez a módszer ugyan számításigényesebb, de előnye, hogy kiküszöböli a digitalizálás során elkövetett hibák egy részét (például a többszörösen digitalizált pontokat).



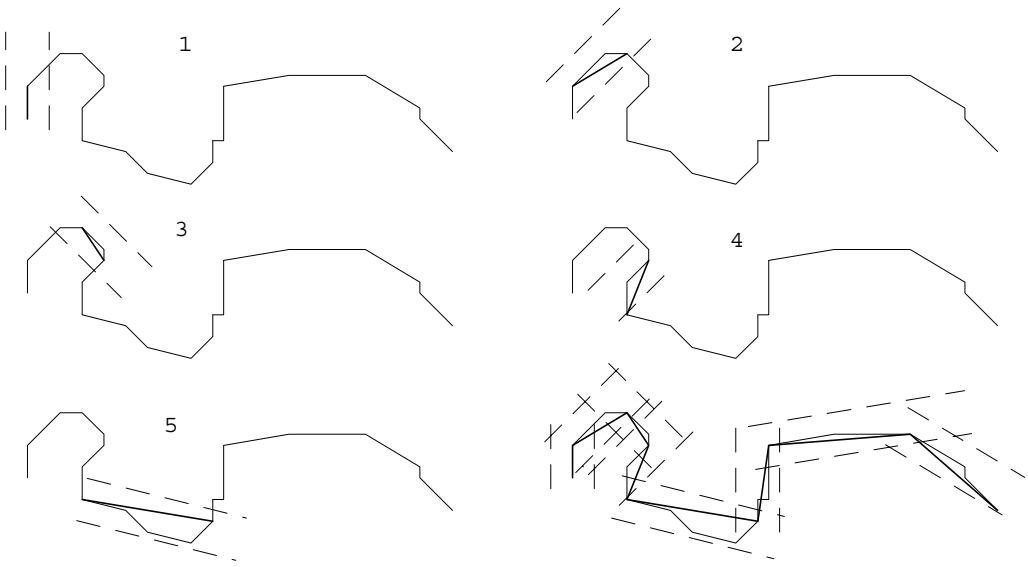
26. ábra *A szöghatárérték algoritmus*



27. ábra *A Jenks algoritmus*

3.4. Párhuzamos távolság vagy sávszélesség algoritmus

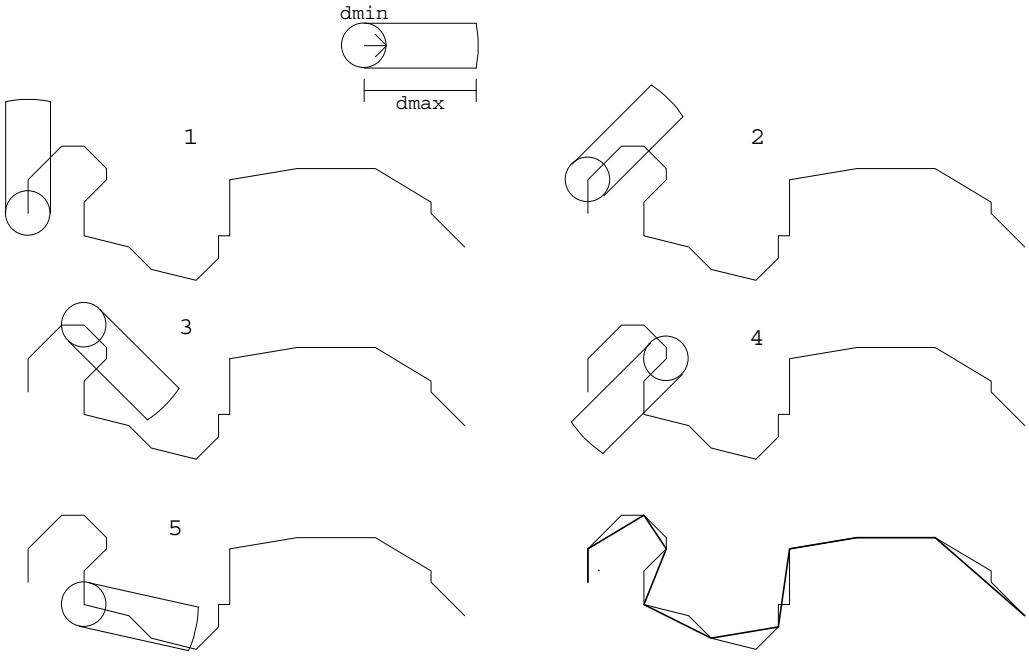
A vonal egyes szakaszait egy előre definiált távolságra lévő párhuzamos sávval látjuk el. Az első szakasztól indulva, mindaddig elhagyjuk a soron következő pontokat, amíg azok a sávon belül találhatók. Ha találtunk egy kívül eső pontot, akkor az azt megelőző pontot jelöljük ki töréspontnak, és innen indítjuk a következő sávot a kívül eső pont felé (28. ábra). Ez a módszer már nem csak a közvetlenül szomszédos pontok között keres, hanem a vonal kisebb-nagyobb részeit vonja vizsgálat alá. A keresés kiterjedése sok kritériumtól függ, többek között a vonal bonyolultságától, a pontok sűrűségétől és a keresés kezdőpontjától. Ez az eljárás jól szűri ki a digitalizálásból adódó esetleges remegéseket.



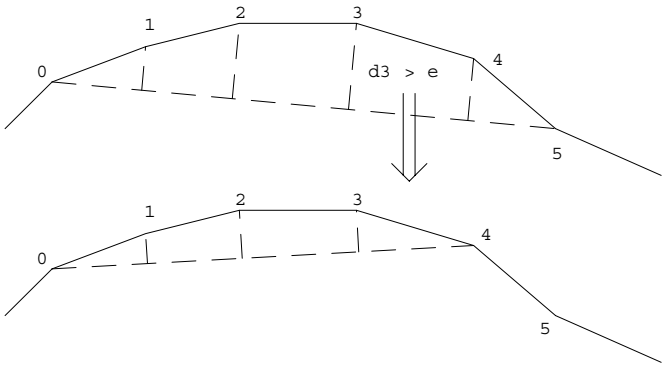
28. ábra *A sávszélesség algoritmus*

3.5. Különböző kötöttségeket hordozó algoritmusok

Ezek a módszerek is a vonal egy-egy szakaszát vetik vizsgálat alá, de a keresésben több kötöttségnek is meg kell felelniük. A kötöttségek általában távolság vagy pontszám paraméterek. Az egyik ilyen algoritmust Opheim dolgozta ki. Ez az eljárás nagyon hasonlóan dolgozik, mint az előbb leírt sávszélesség eljárás, de az algoritmus keresési körét egy-egy minimum és maximum távolságellenőrzés szűkíti. A kezdő keresési sáv ugyanaz, mint az előzőnél, de azokat a pontokat, melyek az első ponttól a minimumtávolságnál közelebb vannak, elhagyjuk. Ha a soron következő pontok közül valamelyik kívül esik a keresési tartományon, beleértve a definiált maximum távolságot is, akkor az utolsó belső pontot tartjuk meg és onnan folytatjuk a keresést (29. ábra).



29. ábra *Az Opheim algoritmus*



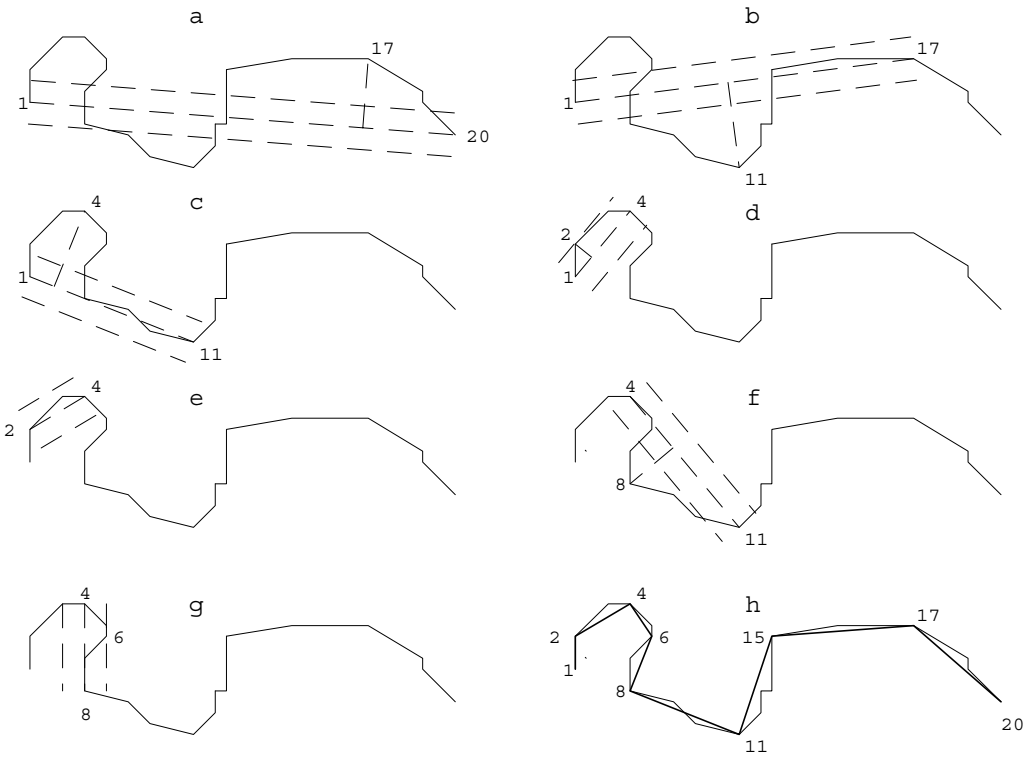
30. ábra *A Lang által javasolt egyszerűsítési módszer*

Egy másik algoritmus, melyet Lang írt le először, szintén merőleges távolságot használ elsődleges feltételként, de az egyszerre figyelembe vehető pontok száma kötött. Az algoritmus a 30. ábrán vázolt módon működik. Ezen a példán egyszerre öt pontot vonunk be a keresésbe. Az első és az ötödik pontra egyenest fektetünk, majd megvizsgáljuk a közbenső pontok távolságát ettől az egyenestől. Ha valamely pont távolsága nagyobb az előre meghatározott határértéknél, akkor az egyenest az első és a negyedik pontokra fektetjük, és ettől számítjuk a távolságokat. Ha minden távolság kisebb a határértéknél, akkor a pontot megőrizzük és a vonal következő öt pontja kerül feldolgozásra.

3.6. A Douglas-Peucker algoritmus

Az eddig bemutatott módszerek a vonalat részenként vizsgálták, az elejétől kezdve folyamatosan haladva a végéig. A Douglas-Peucker algoritmus az egész vonalat globálisan kezeli. Elsőként

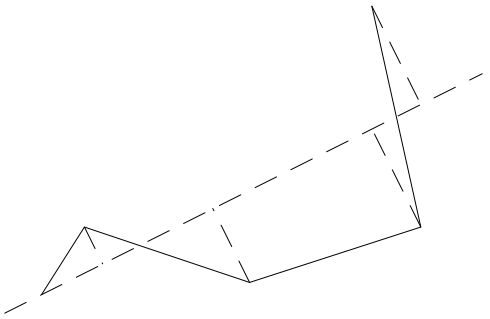
vesszük a vonal kezdő- és végpontját, melyeket horgonynak illetve úszónak nevezünk. Számítjuk a vonal összes pontjának a távolságát a két pontot összekötő egyenestől. Ha minden pont belül van az előre meghatározott sávszélességen, akkor az egyenes megfelelő a vonal ábrázolására, így a közbenső pontokat elhagyjuk. Ha azonban van olyan pont, melynek távolsága nagyobb, mint a határérték, akkor a legtávolabbi pont lesz a következő úszó, míg az előző úszót tároljuk a veremben (stack). Az eljárást megismételjük az új úszóval és a régi horgonnyal, amíg minden közbenső pont a sávon belülre nem kerül. Ha ezt elértük, akkor az aktuális úszó válik horgonnyá, míg az új úszót a verem tetejéről kapjuk (31. ábra).



31. ábra *A Douglas-Peucker algoritmus*
Jól követhető az úszó vándorlása a horgony felé (a - d).
A verem tartalma a d) ábra után : 4, 8, 11, 17, 20

3.7. Főtengely módszer

Ez a módszer is globálisan kezeli a vonalat és alapelvében megegyezik a Douglas-Peucker algoritmussal, azonban a távolságokat nem a két végpontot összekötő egyeneshez viszonyítja, hanem a ponthalmaz tengelyéhez, melynek az a tulajdonsága, hogy a pontok távolságának négyzetösszege minimális (32. ábra). Ez a módszer bonyolult és sok pontot tartalmazó görbék esetén hatékonyabb a Douglas-Peucker algoritmusnál, és kevesebb pontot tart meg.



32. ábra

3.8. Konvolúciós szűrés

A spektrálanalízis és a digitális szűrési technika alkalmazható vonalas elemek generalizálására is. A vonalakat tekinthetjük különböző frekvenciákat tartalmazó periodikus függvényeknek, tehát Fourier tétele alapján előállíthatjuk a következő típusú, frekvenciától függő függvények spektrumaként :

$$A(\omega)\cos[\omega t+\varphi(\omega)],$$

ahol $A(\omega)$ az amplitudó, $\varphi(\omega)$ a fázisszög, $\omega=2\pi u$, és u a frekvencia.

Az adathalmaz spektruma jellemzi a vonal bonyolultságát. A vonalak egyszerűsítése ebben az esetben abból áll, hogy kiszűrjük

a nagyfrekvenciájú komponenseket. A digitális szűrő tulajdonképpen nem más, mint egy súlyfüggvény, a szűrést pedig e súlyfüggvény és az adatrendszer konvolúciójával végezzük :

$$X_{sz}(t)= \int_{-\infty}^{+\infty} S(t)\cdot X(t-\tau)d\tau \quad ,$$

illetve a frekvenciatartományban, Fourier transzformáltak felhasználásával a szűrést az adatrendszer spektrumának és a szűrő átviteli függvényének szorzatával kapjuk :

$$X_{sz}(\omega)= S(\omega)\cdot X(\omega) \quad .$$

A térkép-generalizálásnál a magas frekvenciákat kell tompítani, míg az alacsony frekvenciákat változatlanul kell hagyni. Ez az ún. felülvágó szűrők segítségével történhet. Ilyen például a Gauss-féle felülvágó szűrő, mely irányfüggetlen és szeparálható, így előnyös a térkép-generalizálás szempontjából.

Ahhoz azonban, hogy ezt a módszert alkalmazni tudjuk, a vonalak mintavételezésénél néhány feltételt szem előtt kell tartani. Ezek szerint a mintavételezéskor (digitalizáláskor) a pontok meghatározása egyenlő lépésközönként kell történjen és a lépésköz elég kis érték kell legyen, hogy teljesítse a Nyquist feltételt, azaz a lépésköz kisebb legyen, mint a fél hullámhossz.
A módszer további előnye, hogy ezen digitális szűrők segítségével a térképek bizonyos jellegei tudatosan kiemelhetők (pl. törésvonalak).

3.9. A különböző egyszerűsítő algoritmusok összehasonlítása

Általánosságban kimondhatjuk, hogy a kartográfiában alkalmazott egyszerűsítő algoritmusok elsődleges feladata a térképi vonalak

főbb geomorfológiai jellemzőinek, azaz a karakterisztikus pontoknak kiválasztása. Általános elméleti szabályként előírható, hogy a jobb egyszerűsítő algoritmusok egyenes szakaszokon a pontok nagy részét megszüntetik, míg a görbületekben viszonylag nagy számban megtartják. Ennek az előzőekben felsorolt algoritmusok az n-edik pont algoritmus kivételével többé kevésbé jól eleget tesznek. Többen is vizsgálták már, hogy milyen szempontok alapján lehet kiválasztani ezeket a karakterisztikus pontokat. Megállapították, hogy a szemlélő számára azok a jellegzetes pontok, ahol a görbe vonalában nagymértékű törés jelentkezik. A vizsgálatok kimutatták, hogy a térképet olvasók azokat a pontokat jegyzik meg, melyek leginkább szembetűnőek, azaz ahol a legnagyobb irányváltások vannak.

A kartográfiában a karakterisztikus pontok két típusát különböztethetjük meg : a vonal fizikai jellemzőihez kapcsolódó karakterisztikus pontokat és az emberek által fontosnak tartott jellemző pontokat. Ahogy például egy karikaturista kiemeli az arc kiugró vonásait, a fizikai karakterisztikus pontok ugyanúgy jellemzik a térképi vonalak geomorfológiai vonásait. Ilyenek például egy tengerpart vonalában az öblök, félszigetek, vagy egy folyónál a nagyobb kanyarulatok. A másik csoportba azok a pontok tartoznak, melyek logikai és nem geomorfológiai szempontok miatt szükségesek. Noha például Szeged nem fekszik jellemző kanyarulatban, nem lenne bölcs dolog, ha a Tisza az egyszerűsítés miatt nem folyna keresztül a városon. További ilyen logikai pontok a főbb infrastrukturális hálózatokkal és a politikai, vagy közigazgatási határokkal való metszéspontok.

Egy kézzelfogható kutatási eredmény White vizsgálata, melyben összehasonlította néhány egyszerűsítő algoritmus és a kézi generalizálás eredményeit. A vizsgálatban résztvevők három térképi vonal közül kaptak egyet-egyet, melyeken ki kellett választaniuk a karakterisztikus pontokat, míg ugyanezeket a vonalakat különböző egyszerűsítő algoritmusokkal is feldolgozták. Az eredmények összehasonlításához három mércét használt :

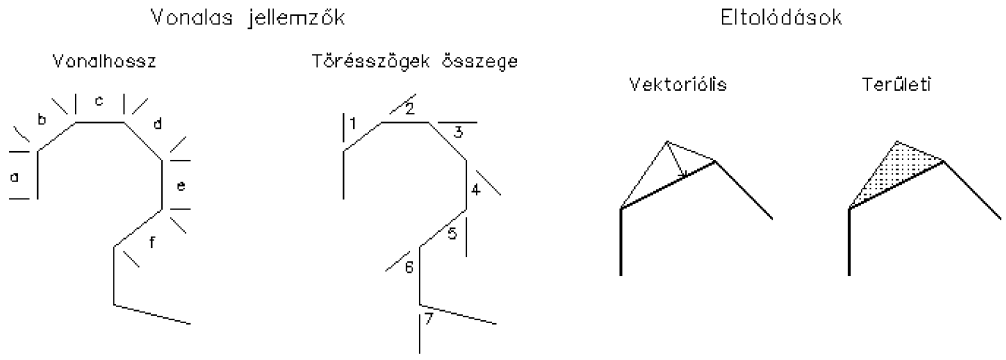
- az egyszerűsített vonalak összehasonlítása szemrevételezéssel,
- az azonos pontok összehasonlítása,
- a vonalak távolságának grafikus összehasonlítása.

Végeredményként azt állapította meg, hogy a Douglas-Peucker algoritmus választotta ki a "legjobb" ponthalmazt.

Más, matematikailag jobban meghatározható módszerek is születtek a vonalegyszerűsítések hatékonyságának vizsgálatára. McMaster harminc matematikai mérőszámot dolgozott ki, melyek két főbb csoportra oszthatók : vonalas jellemzők meghatározására és lineáris eltolódás vizsgálatára. A vonalas jellemzők mérőszámai egy-egy vonalra vonatkoznak ; ilyen például a vonal hossza, vagy törésszögeinek összege. Ugyanakkor az összehasonlító mérőszámok egy vonal és annak egyszerűsített változata közötti különbséget jellemzik (33. ábra).

Ahhoz, hogy az algoritmusok összehasonlítása egyszerűbb és áttekinthetőbb legyen, statisztikai módszerekkel hatra csökkentették a mérőszámok számát. Ez a hat mérőszám :

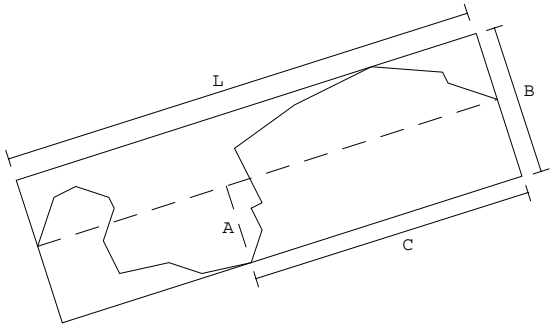
- a pontok számának aránya,
- a törésszögek összegének aránya,
- az egy hüvelykre jutó koordinátaszám szórásának aránya,
- a vektoriális eltolódás hossza / hüvelyk,
- a területi eltolódás / hüvelyk,
- a görbevonalú szakaszok számának aránya.



33. ábra *Vonalas és összehasonlító mérőszámok*

Ezek a mérőszámok megfelelően jellemzik az egyszerűsítő algoritmusokat. A törésszögek összegének aránya például jól tükrözi a kisebb hullámzások kiszűrésének hatékonyságát, míg a területi és vektoriális eltolódások aránya az algoritmus geometriai pontosságára jellemző, minél kisebb az eltolódás, annál jobb.
McMaster kiszámította ezeket a mérőszámokat nyolc különböző algoritmussal egyszerűsített, átlagos bonyolultságú vonalra. Az ő eredményei szerint is a Douglas-Peucker algoritmus hozta a legjobb eredményeket. A további algoritmusok közül a sávszélesség algoritmus és az Opheim algoritmus szolgált megfelelő eredménnyel.

Egy másik megközelítésben Buttenfield a digitális vonalak információtartalmának számszerűsítésére dolgozott ki módszert. Az alapelv a strip tree hierarchikus struktúrájával megegyező. A geometriai mérőszámok, melyeket a felbontási eljárás minden szintjén (a strip tree minden sorára) kiszámítunk és átlagolunk, a következőek (34. ábra) :



Sávhossz	L
Sávszélesség	B
Max. eltérés	M = A / L
Megosztottság	S = C / L

34. ábra *A Buttenfield-féle sávgeometria*

- a sávok hossza,
- a sávok szélessége,
- maximális eltérés,
- megosztottság.

Ezek a mérőszámok jól használhatók digitális vonalak geomorfológiai jellemzőinek automatikus felismerésére, mely segítségével nem csak egyszerűsítés, hanem egyes jellemzők tudatos kiemelése is megfelelően automatizálható.

Egy szintén nem elhanyagolandó tényező a feldolgozáshoz szükséges CPU idő. Egyértelmű, hogy az egyszerűsítő algoritmusok közül az n-edik pont algoritmus a leggyorsabb, de az említett hátrányok miatt csak nagyon szűk körben használható. A Douglas-Peucker algoritmus a leglassúbbak közé tartozik. Jóval hatékonyabb, és a kidolgozója szerint hasonlóan jó eredményeket produkál a főtengely módszer.

3.10. Példa az egyszerűsítő algoritmusok bemutatására

Diplomatervemnek nem célja, hogy az előzőekben ismertetett összehasonlító méréseket megismételje. Azonban azért, hogy szemléletesebbé tegyem a különböző egyszerűsítő algoritmusok különbségeit, rövid programokat írtam a következő eljárásokra :

- n-edik pont algoritmus,
- merőleges távolság algoritmus,
- sávszélesség algoritmus,
- Douglas-Peucker algoritmus.

Mint említettem, e programok célja csupán a szemléltetés, ezért nem biztosítanak teljes körű felhasználást, nincsen bennük tökéletes felhasználói felület.

Az 1. sz. mellékletben található ábrák a Tisza magyarországi szakaszának északi felét ábrázolják. A kiindulási alapot 1:500.000 méretarányú térképről Intergraph Microstation szoftver segítségével digitalizált vonal jelentette. A digitalizált vonalat DXF file-ba kimentve, a fejléc manuális leválasztása után a DXF_KOO.C program segítségével megkaptam a vonal töréspontjainak koordinátáit. Ez a kiinduló vonal 1632 pontból áll. Az egyszerűsítő algoritmusok az így elkészített koordinátafile-t olvassák be és dolgozzák fel, majd a megtartott pontokat egy a Microstation által felismert UCM (felhasználói makró) formátumba öntve mentik el. Ezt a file-t a Microstationbe beolvasva megkapjuk az egyszerűsített vonalat. Az egyes programok kezelése egységes, bemenetként billentyűzetről várják az input és output file nevét, valamint az alkalmazott határértéket (méter mértékegységben). A programok kiírják a megtartott pontok számát, mely első közelítésben segít a megfelelő határérték meghatározásában.

Tájékoztatásul álljon itt a különböző algoritmusok és határértékek segítségével egyszerűsített vonalak megtartott pontjainak száma :

algoritmus / határérték	Douglas-Peucker	Merőleges táv.	Sávszélesség
e = 100 m	402	307	665
e = 200 m	237	123	387
e = 500 m	136	17	204
e = 1000 m	83	-	139

A fenti határértékekhez tartozó egyszerűsített vonalak, a szemléletesség kedvéért az eredeti vonalakkal megegyező méretarányban, az 1. sz. mellékletben megtalálhatók. A programlisták a 2.sz. mellékletet képezik.

Irodalomjegyzék

Ballard, D. H. : Strip Trees : A Hierarhical Representation for Curves
Communications ACM, 1981., v.24., p.310-321.

Boyle, A. R. : The Quantized Line
The Cartographic Journal, 1970., v.7, no.2. p.91-94.

Brassel, K. E. és Weibel, R. : A Review and Conceptual Framework of Automated Map Generalization
International Journal of GIS, 1988., v.2, no.3 p.229-244.

Brassel, K. E. : Strategies and Data Models for Computer-Aided Generalization
International Yearbook of Cartography,1985.,v.25, p.11-28.

Buttenfield, B. : Treatment of the Cartographic Line
Cartographica, 1985., v.22, no.2. p.1-26.

Cederberg, R. : Chain-link Coding and Segmentation for Raster Scan Devices
Computer Graphics and Image Processing, 1979., v.10.
p.224-234.

Chellappa, R. és Bagdazian, R. : Fourier Coding of Image Boundaries
IEEE Transactions on Pattern Analysis and Machine Intelligence, v. PAMI-6, no.1., Jan. 1984. p.102-105.

Cromley, R. G. : Principal Axis Line Simplification
Computers and Geosciences, 1992., v.18., no.8. p.1003-1011.

Dettori, G. és Falcidieno, B. : An Algorithm for Selecting Main Points on a Line
Computers and Geosciences, 1982., v.8., p.3-10.

Douglas, D. H. és Peucker, T. : Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature
The Canadian Cartographer, 1973., v.10, no.2. p.112-122.

Dr. Detrekői Ákos : Kiegyenlítő számítások
Budapest, 1991. Tankönyvkiadó

Dr. Sárközy Ferenc : Geodéziai AMT
Budapest, 1993., kézirat

Freeman, H. : Application of the Generalized Chain Coding Scheme to Map Data Processing
Proceedings of the IEEE, 1978. p.220-226.

Freeman, H. : On the Encoding of Arbitrary Geometric Configurations
IRE Transactions, vol. Ec-10. No.2. June 1961. p.260-268.

Gibson, L. és Lucas, D. : Vectorization of Raster Images Using Hierarchical Methods
Computer Vision, Graphics and Image Processing, 1982., v.20, no.1. p82-89.

Jenks, G. F. : Thoughts on Line Generalization
Proceedings AUTO-CARTO IV, 1979. p.209-220.

Jones, C. : Database Design for Variable-scale, Worldwide Cartography
Computer Graphics '85: Online Publications, Pinner, UK.

Jones, C. és Abraham I. : Line Generalization in a Global Cartographic Database
Cartographica, 1987., v.24, no.3 p.32-45.

Kádár I., Karsay F. : Bevezető tanulmányok a kartográfiai automatizáláshoz
Budapest, 1969., kézirat

Kádár István : A helymeghatározás természetes mértékei
Geodézia és Kartográfia, 1992., 44. évf, 6.sz. p.417-424.

Lang, T : Rules for Robot Draughtsmen
Geographical Magazine, 1969., v.62, no.1. p.50-51.

Marino, J. S. : Identification of Characteristic Points Along Naturally Occurring Lines : An Empirical Study
The Canadian Cartographer, 1979, v.16, no.1. p.70-80.

Mark, D. M. és Abel, D. J. : Linear Quadtrees from Vector Representations of Polygons
IEEE Transactions on Pattern Analysis and Machine Intelligence, v. PAMI-7, no.3., May 1985. p.344-349.

McMaster, R. B. : A Statistical Analysis of Mathematical Measures for Linear Simplification
The American Cartographer, 1986., v13, no.2. p.103-117.

McMaster, R. B. : Automated Line Generalization
The Canadian Cartographer, 1987., v.24, no.2. p.74-111

Meek, D. és Walton, D. : Several Methods for Representing Discrete Data by Line Segments
Cartographica, 1991, v.28, no.2. p.13-20.

Opheim, H. : Fast Data Reduction of a Digitized Curve
Geo-Processing, 1982., v.2. p.33-40.

Peucker, T. : A Theory of the Cartographic Line
International Yearbook of Cartography, 1976.,v.16. p.134-142.

Peucker, T. és Chrisman, N. : Cartographic Data Structures
The American Cartographer, 1975., v.2. p.55-69.

Peuquet, D. J. : A conceptual Framework and Analysis of Spatial Data Models
Cartographica, 1984., v.21, no.4. p.66-113.

Peuquet, D. J. : A Hybrid Data Structure for the Storage and Manipulation of Very Large Data Sets
Computer Vision, Graphics and Image Processing, 1983., v.24, no.1. p14-27.

Peuquet, D. J. : Raster Processing : An Alternative Approach to Automated Cartographic Data Handling
The American Cartographer, 1979., v.6., no.2. p.129-139.

Reumann, K. és Witkam, A. P. M. : Optimizing Curve Segmentation in Computer Graphics
International Computing Symposium, 1973. Amsterdam, North-Holland Publishing Company, p.467-472.

Rhind, D. W. : Generalization and Realism within Automated Cartographic Systems
The Canadian Cartographer, 1973., v.10., p.51-62.

Roberge, J. : A Data Reduction Algorithm for Planar Curves
Computer Vision, Graphics and Image Processing, 1985., v.29. p.244-256.

Samet H. : Applications of Spatial Data Structures : Computer Graphics, Image Processing and GIS
Addison-Wesley, 1990.

Samet H. : The Design and Analysis of Spatial Data Structures
Addison-Wesley, 1990.

Samet, H. : The Quadtree and Related Hierarchical Data Structures
Computing Surveys, v.16., no.2, June 1984. p.187-260.

Samet, H. és Webber, R. E. : On Encoding Boundaries with Quadtrees
IEEE Transactions on Pattern Analysis and Machine Intelligence, v. PAMI-6, no.3., May 1984. p.365-369.

Samet, H. és Webber, R. E. : Using Quadtrees to Represent Polygonal Maps
IEEE Transactions, 1983., vol. CH 83. no.1.

Shneier, M. : Two Hierarchical Linear Feature Representations : Edge Pyramids and Edge Quadtrees
Computer Graphics and Image Processing, 1981., v.17. p.211-224.

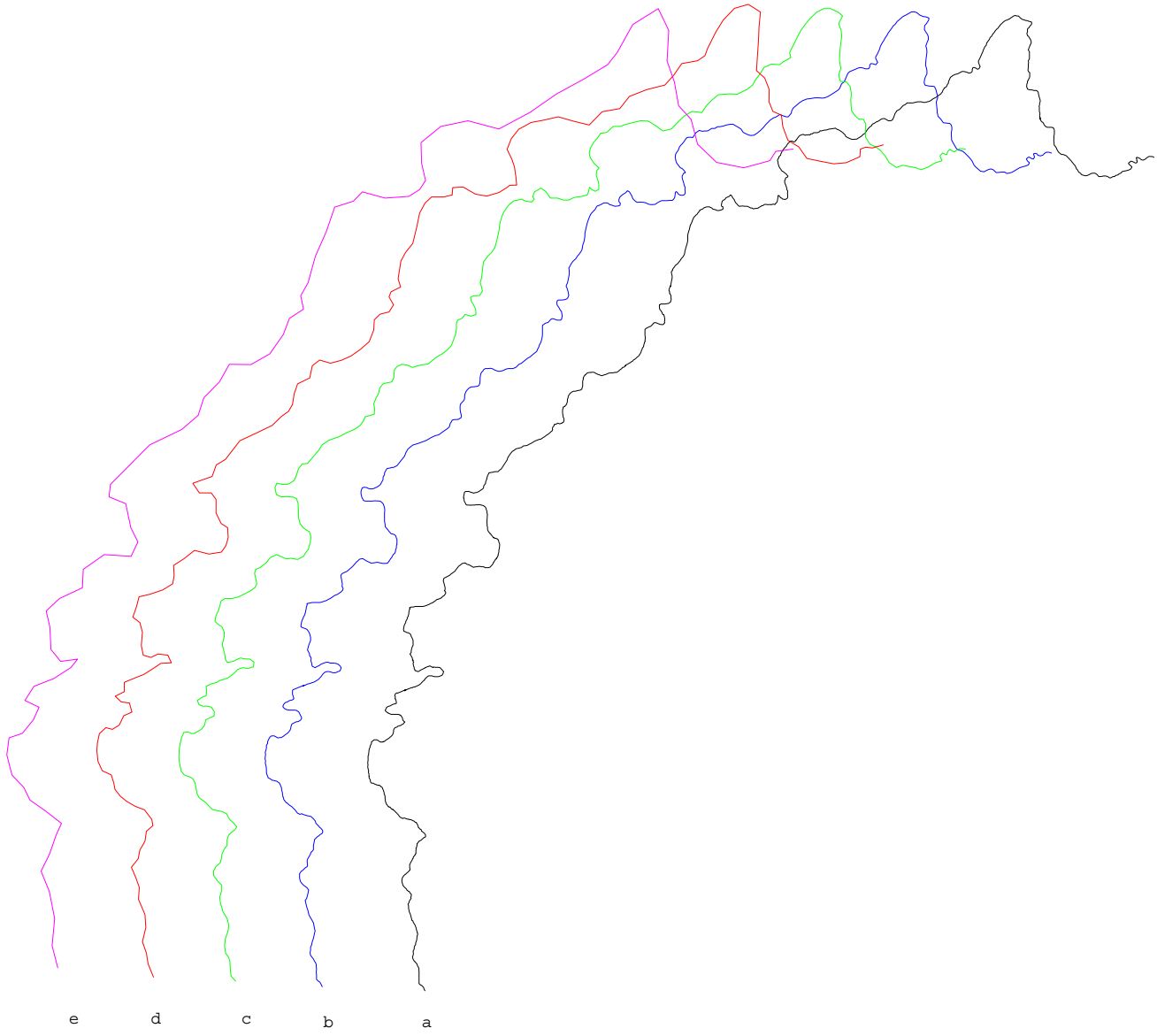
Stegena Lajos : Tools for Automation of Map Generalization : The Filter Theory
Final Report on the ICA Commission III, 1973., Budapest

White E. R. : Assessment of Line-Generalization Algorithms Using Characteristic Points
The American Cartographer, 1985., v12, no.1. p.17-28.

Wiseman, N. E. : Vectors, Rasters and Cartographic Databases
Cartographica, 1982., v.19., p.111-118.

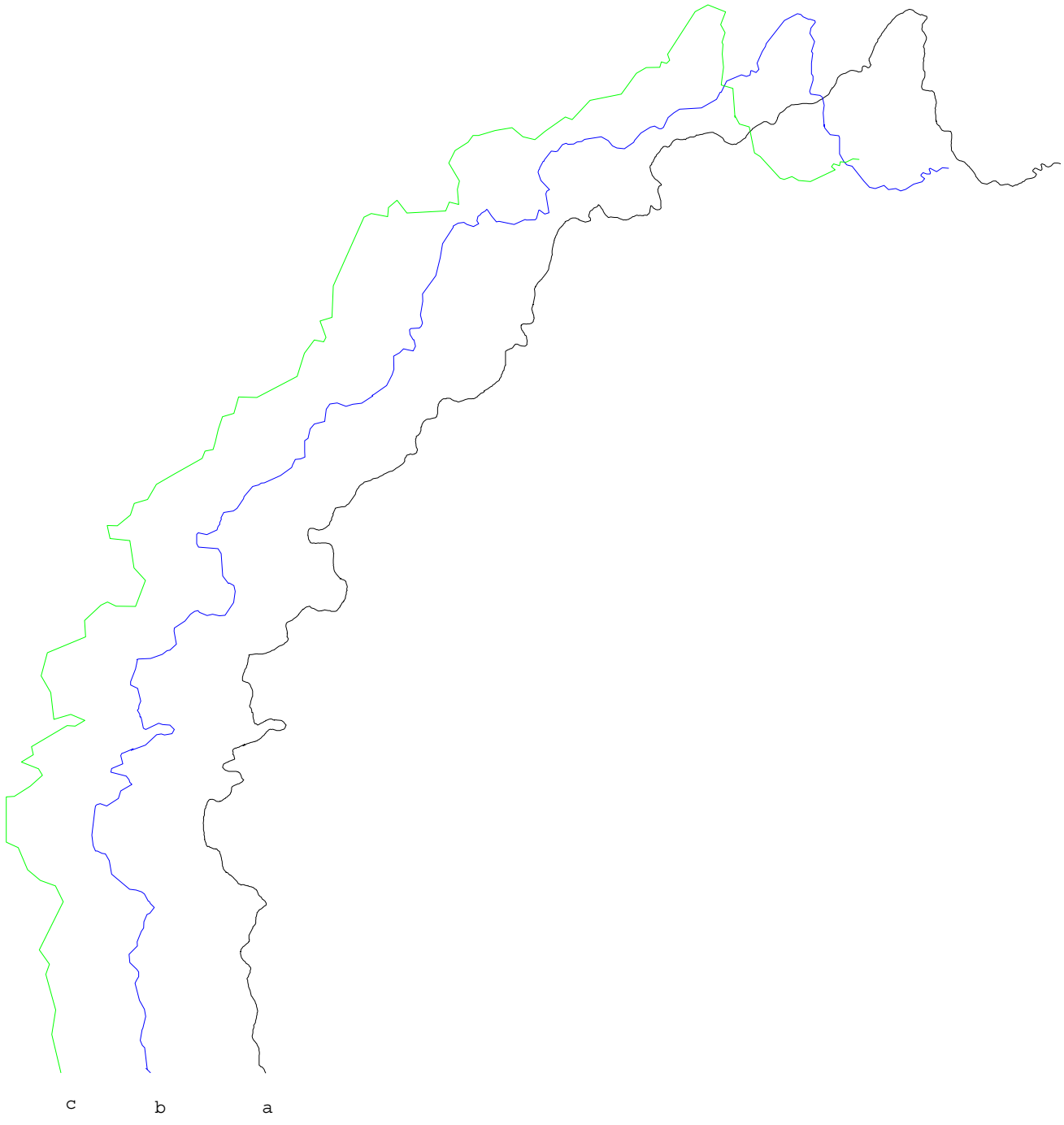
Závoti József : A digitális terepmodell matematikai alapjai és geodéziai alkalmazásai
Kandidátusi értekezés, 1985.

1. sz. melléklet



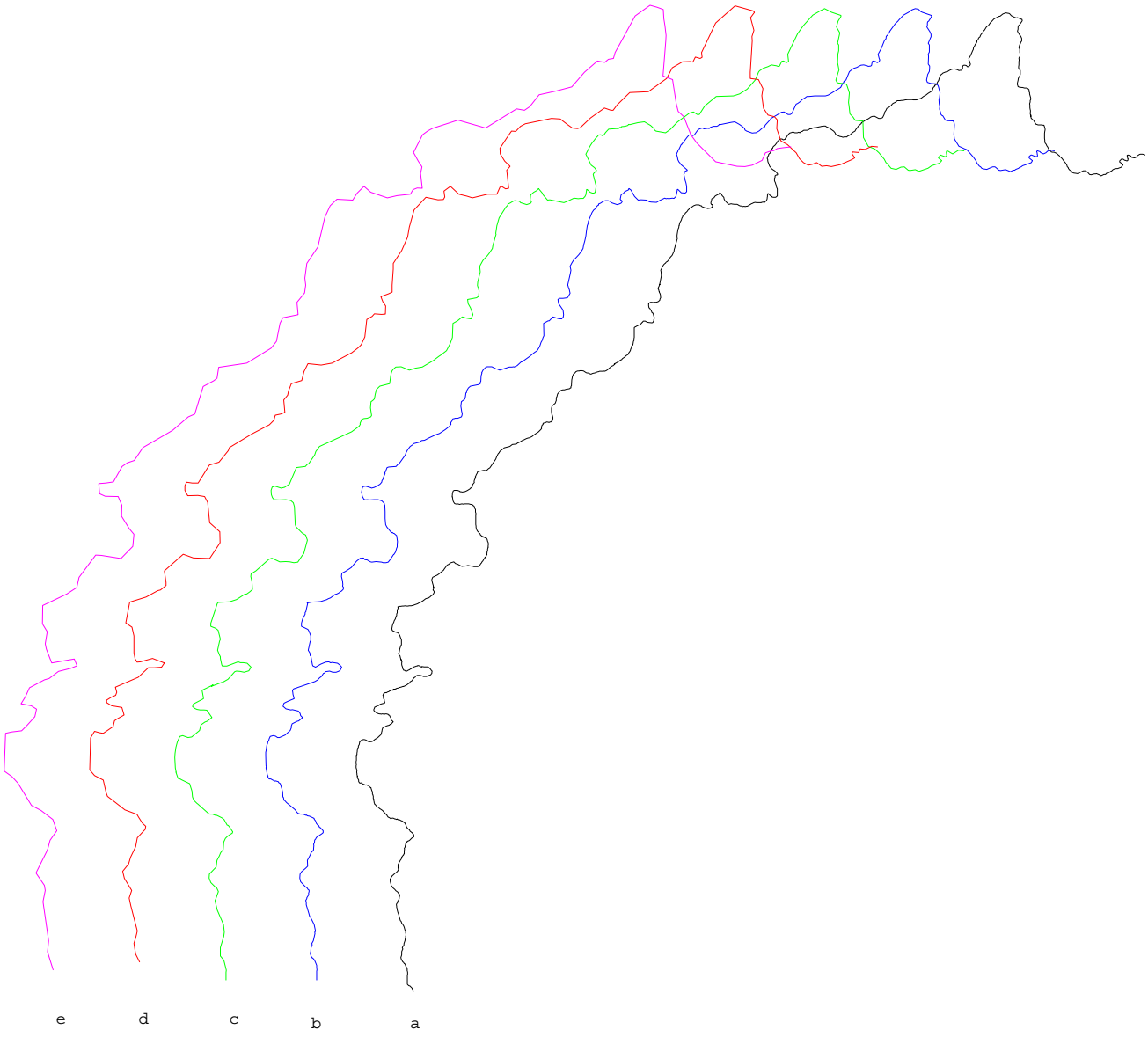
1.) n-edik pont algoritmus

a) eredeti vonal, b) n = 2, c) n = 5, d) n = 10, e) n = 20



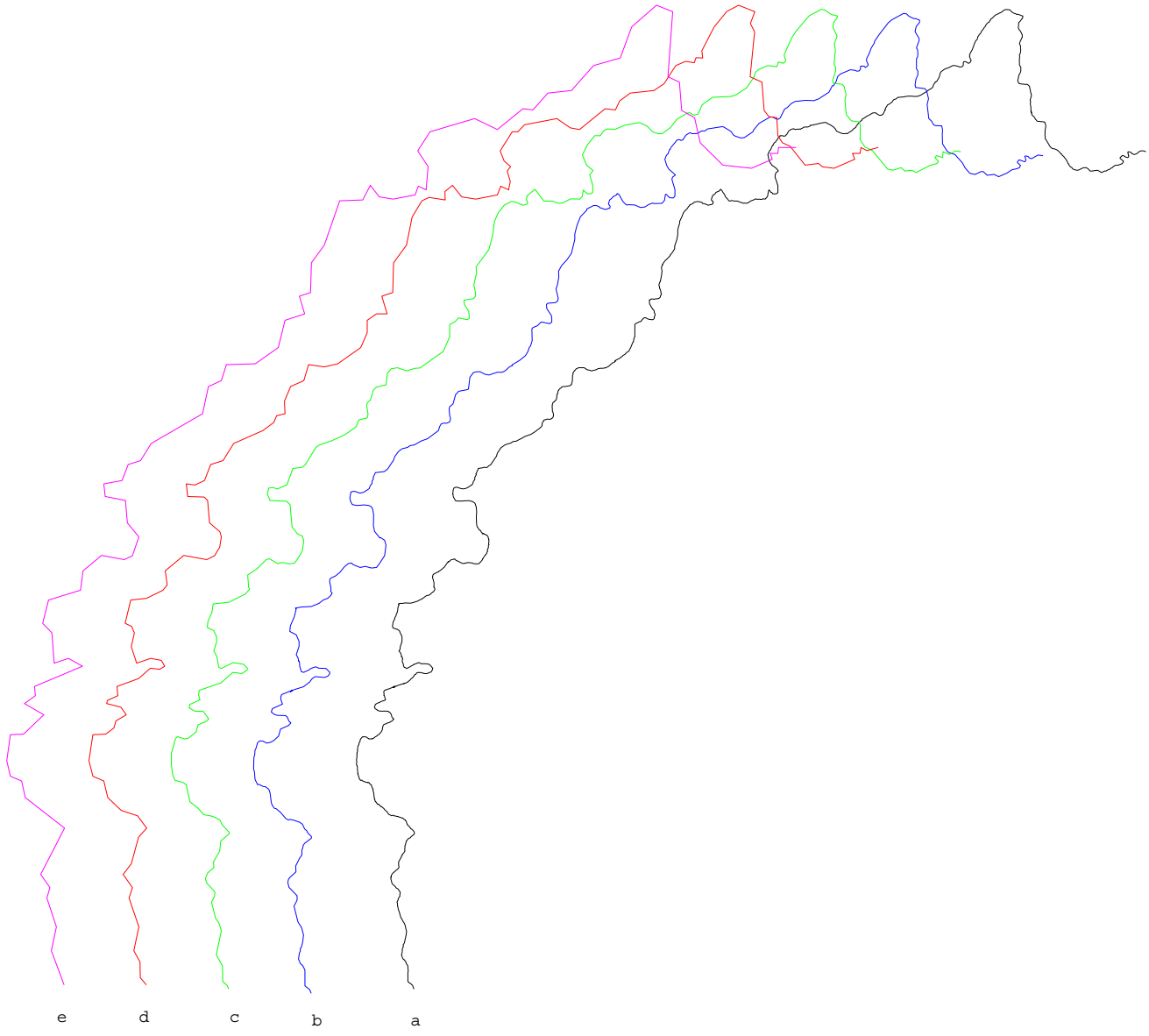
2.) Merőleges távolság algoritmus

a) eredeti vonal, b) e = 100, c) e = 200,



3.) Sávszélesség algoritmus

a) eredeti vonal, b) e = 100, c) e = 200, d) e = 500, e) e = 1000



4.) Douglas-Peucker algoritmus

a) eredeti vonal, b) e = 100, c) e = 200, d) e = 500, e) e = 1000

2. sz. melléklet

1.) DXF_KOO.C , DXF file-ból kiemeli a koordinátákat

```
#include <stdio.h>

int i;
char s[20], nam1[12], nam2[12];

int main(void)
{
   FILE *in, *out;

  clrscr();
  gotoxy( 5,5 ); printf("A dxf file neve : ");
  gets(nam1);
  if ((in = fopen(nam1, "rt")) == NULL)
   {
      fprintf(stderr, "Cannot open input file.\n");
      return 1;
   }
  gotoxy( 5,7 ); printf("A koord. file neve  : ");
  gets(nam2);
  if ((out = fopen(nam2, "wt")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
  fgets(s, 20, in);
  fgets(s, 20, in);
  fputs(fgets(s, 20, in), out);
  fgets(s, 20, in);
  fputs(fgets(s, 20, in), out);
  fgets(s, 20, in);
  fputs(fgets(s, 20, in), out);
  fgets(s, 20, in);
  fputs(fgets(s, 20, in), out);
  fgets(s, 20, in);
  fgets(s, 20, in);
  fgets(s, 20, in);
  fgets(s, 20, in);
  fgets(s, 20, in);
  fgets(s, 20, in);
  fgets(s, 20, in);
  while (!feof(in))
       {
          for (i = 0; i < 6; i++) fgets(s, 20, in);
          fputs(fgets(s, 20, in), out);
          fgets(s, 20, in);
          fputs(fgets(s, 20, in), out);
          for (i = 0; i < 7; i++) fgets(s, 20, in);
       }
  fclose(in);
  fclose(out);
  return 0;
}
```

2.) NPOINT.C , n-edik pont algoritmus

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

int i, n;
char nam1[40], nam2[40];
char xstr[15], ystr[15], xx[15], ns[3];

int main(void)
{
  FILE *inp1, *out1;

  clrscr();
  gotoxy( 5,5 ); printf("A koord. file neve : ");
  gets(nam1);
  if ((inp1 = fopen(nam1, "rt")) == NULL)
   {
      fprintf(stderr, "Cannot open input file.\n");
      return 1;
   }
  gotoxy( 5,7 ); printf("A macro file neve  : ");
  gets(nam2);
  if ((out1 = fopen(nam2, "wt")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
  gotoxy( 5,9 ); printf("N = "); gets(ns);
  if ((n = atoi(ns)) <= 0)
   {
      fprintf(stderr, "N nem lehet negativ.\n");
      return 1;
   }
  fprintf( out1, "%s\n", "KEY 'lv=1'");
  fprintf( out1, "%s\n", "KEY 'PLACE LSTRING POINT'");
  for (i = 0; i < 3420; i++)
      {
       if (!feof(inp1))
        {   fgets (xstr, 11, inp1);
            fgets (xx, 15, inp1);
            fgets (ystr, 11, inp1);
            fgets (xx, 15, inp1);
            if ((i % n) == 0)
             {
              printf("%d %s %s\n", i, ystr, xstr );
              fprintf( out1, "%s%s%s%s%s\n", "KEY 'xy=",
                                  xstr, ",", ystr, "'" );
             }
        }
      }
  fprintf( out1, "%s\n", "KEY 'RESET'");
  fprintf( out1, "%s\n", "END");
  close( inp1 );  close( out1 );
  return 0;
}
```

3.) GENERAL.C , általános eljárások az egyszerűsítő algoritmusokhoz

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include "general.h"

extern struct point line1[3500];/*bemeno koordinatak*/
extern struct point line2[3500];/*kimeno koordinatak*/
extern int np, nn;   /* pontok szama */
extern long int e;   /* tolerancia */

/* pont tavolsaga az egyenestol */

float dist(int p1, int p2, int pp)
{
  double t;
  float dx, dy;
  float ax, ay, bx, by, v;

  dx = line1[p2].px - line1[p1].px;
  dy = line1[p2].py - line1[p1].py;
  t = sqrt(dx * dx + dy * dy);
  if (t < 1e-6) t = 1e-6;
  ax = line1[p1].px - line1[pp].px;
  ay = line1[p1].py - line1[pp].py;
  bx = line1[p2].px - line1[pp].px;
  by = line1[p2].py - line1[pp].py;
  v = ay * bx - by * ax;
  return (fabs(v) / t);
}

/* DXF-bol levalsztott koordinatafile beolvasasa */

int readfile(FILE *fp)
{
  char xx[15], xstr[12], ystr[12];
  int i;

  i = 1;
  do
        {
          fgets (xstr, 11, fp);
          fgets (xx, 15, fp);
          fgets (ystr, 11, fp);
          fgets (xx, 15, fp);
          line1[i].px = atof(xstr);
          line1[i].py = atof(ystr);
          i++;
        }
  while (!feof(fp));
  return (i - 1);
}
```

```
/* kezdoadatok beolvasasa */

int input(void)
{
  FILE *inp1;
  char es[3], nam1[12];
  int i;

  clrscr();
  gotoxy( 5,5 ); printf("A koord. file neve : ");
  gets(nam1);
  if ((inp1 = fopen(nam1, "rt")) == NULL)
   {
      fprintf(stderr, "Cannot open input file.\n");
      return 1;
   }
  np = readfile(inp1);
  fclose (inp1);
  gotoxy( 5,7 ); printf("e = "); gets(es);
  if ((e = atol(es)) <= 0)
   {
      fprintf(stderr, "e nem lehet negativ.\n");
      return 1;
   }
  return 0;
}

/* megtartott koordinatak elmentese UCM formatumba */

int output(void)
{
  FILE *out1;
  char nam2[12];
  int i;

  gotoxy( 5,9 ); printf("A megtartott pontok szama :
                                              %d\n", nn);
  gotoxy( 5,11 ); printf("Az output koord. file neve
:");
  gets(nam2);
  if ((out1 = fopen(nam2, "wt")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
   else
   {
      fprintf( out1, "%s\n", "KEY 'lv=1'");
      fprintf( out1, "%s\n", "KEY 'PLACE LSTRING
                                              POINT'");
      for (i = 1; i <= nn; i++)
        fprintf( out1, "%s%f%s%f%s\n", "KEY 'xy=",
                  line2[i].px, ",", line2[i].py, "'" );
      fprintf( out1, "%s\n", "KEY 'RESET'");
      fprintf( out1, "%s\n", "END");
      fclose(out1);
```

```
   }
  for (i = 1; i <= nn; i++)
      {
       printf ("%d  %f   ", i, line2[i].px);
       printf ("%f\n", line2[i].py);
      }
  return 0;
}
```

4.) PDIST.C , merőleges távolság algoritmus

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include "general.h"

struct point line1[3500];
struct point line2[3500];

int np, nn;
long int e;


int perpdist(void)
{
  int p1, p2, pp, i;

  p1 = 1; p2 = 3; pp = 2;
  i = 1;
  do
      {
        if (dist(p1, p2, pp) > e)
            {
              line2[i++] = line1[p1];
              p1 = pp;
            }
        p2++; pp ++;
      } while (p2 < np);
  return (i - 1);
}

int main(void)
{
  input();
  nn = perpdist();
  output();
  return 0;
}
```

5.) SAV.C , sávszélesség algoritmus

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

struct point {
      float px;
      float py;
      };

struct point line1[3500];
struct point line2[3500];

int np, nn;
long int e;


int pardist(void)
{
  int p1, p2, pp, i;

  p1 = 1; p2 = 2; pp = 3;
  i = 1;
  do
      {
        if (dist(p1, p2, pp) > e)
            {
              line2[i++] = line1[p1];
              p1 = pp - 1;
              p2 = pp;
            }
        pp ++;
      } while (pp <= np);
  return (i - 1);
}

int main(void)
{
  input();
  nn = pardist();
  output();
  return 0;
}
```

6.) DOUGPECK.C , Douglas-Peucker algoritmus

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include "general.h"

struct point line1[3500];   /* bemeno koordinatak */
struct point line2[3500];   /* kimeno koordinatak */
int np, nn;   /* pontok szama */
long int e;   /* tolerncia */
int nst;
int stack[3500];
float dx, dy;
double t;

/* stack muveletek */

int push(int pt)  /* stack push */
{
  nst++; stack[nst] = pt; return 0;
}

int pop(void)    /* stack pop */
{
   nst--;
   return stack[nst + 1];
}

/* egyenes egyenlete */

void eqlin(int p1, int p2)
{
  dx = line1[p2].px - line1[p1].px;
  dy = line1[p2].py - line1[p1].py;
  t = sqrt(dx * dx + dy * dy);
  if (t < 1e-6) t = 1e-6;
}

/* pont tavolsaga az egyenestol */

float dist2(int p1, int p2, int pp)
{
  float ax, ay, bx, by, v;

  ax = line1[p1].px - line1[pp].px;
  ay = line1[p1].py - line1[pp].py;
  bx = line1[p2].px - line1[pp].px;
  by = line1[p2].py - line1[pp].py;
  v = ay * bx - by * ax;
  return (fabs(v) /t);
}
```

```
int douglas_peucker(void)
{
  int flt, anc;
  int k, n, im;
  float dm, d;

  nst = 0;
  anc = 1; flt = np; n = 1; /* anchor, float */
  line2[1] = line1[1];
  push (flt);
  do {
        flt = stack[nst];
        dm = -1.0;
        eqlin(anc, flt);
        for (k = flt - 1; k > anc; k--)
            {
             if ((d = dist2(anc, flt, k)) > dm)
               {
                  im = k;
                  dm = d;
               }
            }
        if (dm < e)
            {
             anc = pop();
             n++;
             line2[n] = line1[anc];
            }
           else
            {
             flt = im;
             push(flt);
            }
       } while (nst > 0);
  return n;
}


int main(void)
{
  input();                  /* koordinatak beolvasasa
*/
  nn = douglas_peucker();
  output();                 /* koordinatak kiirasa */
  return 0;
}
```