Hőeloszlás háromszögelt síkrészen

Miklós Bálint és Zsombori Vilmos
Kolozsvári Műszaki Egyetem, Automatizálás és Számítógépek kar

2002 május

Kivonat. Objektum orientált implentációt és elméleti eszközöket adunk több NURBS görbe segítségével meghatározott síkrészen értelmezett időfüggetlen, hőforrás nélküli hőeloszlás egyenlet numerikus megoldására. Egy Delaunay tulajdonságra alapuló háromszögeléses módszerrel generálunk jó minőségű véges elem felosztást az értelmezési tartományra. A tartományon értelmezett függvények terét lineáris spline-ok segítségével diszkretizáljuk véges vektortérré, aminek dimenzióját a felosztás határozza meg. A megoldandó egyenlet függvényében kiszámoljuk azokat a skalárokat, amelyek meghatározzák a véges tér azon függvényét, amely a legközelebb áll a megoldáshoz.

Kulcsszavak: NURBS görbe, Delaunay háromszögelés, Delaunay finomítási algoritmus, véges elem módszer, lineáris háromszögű merev mátrix

Tartalomjegyzék

1. Bevezető

A fizikai szimulációk során nagyon gyakran szükséges parciális differenciál egyenletek megoldása. Az analitikus módszerekkel értelmezési tartománytól, vagy akár magától az egyenlettől függően nehézkesen, vagy akár egyáltalán nem lehet ezeket megoldani. Mérnöki szempontból gyakran kielégítőek a megközelítő, numerikus módszerek megoldásai, ha lerögzíthető egy olyan hibakorlát, amelyet a módszer hibái nem haladnak meg.

Parciális differenciál egyenletek megoldására alkalmazott numerikus módszerek:

- Véges különbségek módszere (Finite Difference Method) viszonylag egyszerű és gyors módszer, de nem megfelelően flexibilis, ami miatt képtelen bonyolult értelmezési tartományokat kezelni
- Véges elemek módszere (Finite Element Method) A gyakorlati alkalmazásokban a legnagyobb népszerűségnek örvend, mert bonyolult tartományokra is képes konvergens megoldást nyújtani. A módszerhez szükséges az értelmezési tartomány egyszerű elemekre való felosztása (véges elemek).
- Határelem módszere (Boundary Element Method) Ez a módszer is felosztáson alapszik. Ebben az esetben csak a határfelületet szükséges felosztani, s ezért gyorsabb, de csak sajátos esetekben használható.
- Kollokációs módszer (Collocation Method) A többiekkel ellentétben itt nem szükséges az értelmezési tartomány felosztása. Új, még nem teljesen kikristályosodott módszer, mely a gyakorlatban sem terjedt még el.

Egy időfüggetlen, hőforrás nélküli hőeloszlás egyenletet numerikus megoldását tanulmányozzuk dolgozatunk során. Egy objektum orientált implentációt adunk az alábbiakban bemutatott elméleti eszközök felhasználásával. A megoldandó egyenletet egy tetszőleges alakú konex síktartományon értelmezzük.

Ezt a tartományt NURBS (Non Uniform Rational B-Splines) görbék határolják. Azért választottuk ezt a leírási formát, mert egy sor reprezentatív pont segítségével nagy kontrolunk van egy görbe/felület fölött, amellyel minden formát tudunk mímelni. Másik előnye, hogy kiértékelése polinomiális és egyszerűen tudjuk kontrolálható élszámú poligonná alakítani.

Az így kapott poligonális tartomány háromszögelését két lépésben valósítjuk meg. Először a generáljuk a kötött Delaunay háromszögelést, ezután ezt a Delaunay finomítási algoritmus segítségével alakítjuk jó minőségű véges elem felosztássá. Az így kapott háromszögelés garantálja, hogy mindegyik háromszög minimál szöge nagyobb a felhasználó által meghatározott értéknél (ez max. 34-35 fok).

Ezek a feltétel szükséges az általunk használt véges elem módszer működéséhez. A következő lépések a legidőigényesebbek, mert nagyon sok számolással járnak. A generált háromszögeken értelmezünk bázisfüggvényeket, s további feladatként a bázisfüggvényekhez tartozó konstansokat keressünk. Ezek a konstansok egy olyan függvényt határoznak meg ebben a térben, amely a legjobban közelíti meg egyenletünk megoldását. Mi lineáris spline bázisfüggvényekkel dolgozunk, mert az igényelt számolások elfogadható időn belül elvégezhetőek.

A számítógépes fizikai szimulációk a következő folyamatábrát követik:

Dolgozatunk a következő módon valósítja meg a fent említett folyamatot:

- A “CAD system” a mi esetünkben egy olyan grafikus interfész, amely segítségével meg tudunk határozni egy értelmezési tartományt. Ez lesz a rendszer bemenete.
- A “mesh generator” a fent meghatározott tartományt fogja felosztani megfelelő háromszögekre (véges elemekre).
- A probléma természetéből adódóan rendszerünknek. nincs szüksége anyag és adat definíciókra
- A “solver program” a fent említett véges elem módszert alkalmazza.
- A vizualizáció színskála segítségével ábrázolja az eredményt.

2. NURBS görbék

2.1. Miért?

Ennek a fejezetnek az a célja, hogy egy fajta bevezetést nyújtson a NURBS (**n**on**u**niform **r**ational **B**-**s**plines) görbék világába.

Általában minden grafikus rajzoló csomag, a bonyolult idomok megrajzolását egyszerű görbék segítségével (grafikai primitívek), ezek összerakásával teszi lehetővé. Ilyen primitívek közé tartozik például a szakasz, amely a két végpontja által van meghatározva, vagy a háromszög, amelyet a három csúcsa definiál, és így tovább. Ezeknek a felbontása és alakja nem függ a helyzetüktől, ezért végrehajthatunk rajtuk egyszerű grafikai műveleteket: forgatás, sugaras nagyítás, vagy párhuzamos eltolás.

Ezeket a grafikai primitíveket használhatjuk, hogy definiáljunk bonyolultabb objektumokat, mint például egy focilabdát, de természetesen egy sor matematikai tulajdonságnak vagyunk kitéve: ha egy számban konstans szakaszsorozattal közelítünk meg egy kört, forgatás során módosul az alakja, nagyítás során meg módosul a felbontása. Itt jön be a NURBS görbéknek az előnye: lehetővé teszik, hogy ábrázoljunk velük tetszőleges formákat, fenntartva egy matematikai pontosságot és felbontás-független tulajdonságot. A hasznos tulajdonságai közül néhány:

- minden elképzelhető formát tudnak ábrázolni – mímelni: ponttól elkezdve, az egyenes szakaszok és sokszög vonalon keresztül a kúpszeletekig (kör, ellipszis, parabola, hiperbola) illetve a tetszőleges formájú szabad vonalakig
- nagyszerű kezelhetőséget nyújt a görbe változtatásában egy *kontrolpont* vektor és egy ú.n. *knot* vektor által; direkt módon tudjuk változtatni a finomságát ill. görbületét
- képesek ábrázolni nagyon bonyolult formákat, jelentősen kevés adattal; pl. ha szeretnénk egy 30 cm átmérőjű kört rajzolni, amelyet szakaszokkal közelítünk, szükség van több ezer szakaszra, (amelyeknek a kezdőpontjait ill. végpontjait mind el kell tárolnunk) hogy egy körnek nézzen ki, ne egy sokszögnek; ugyanezt a kört NURBS-al definiálva, elég eltárolni hét kontrol pontot és 10 knot-ot

2.2. Görbék

Mint tudjuk, három ismert mód van arra, hogy leírjunk egy görbét függvények segítségével: explicit függvénnyel, implicit függvénnyel ill. parametrikus formában. Azt is tudjuk, hogy ezek közül a parametrikus függvények a legflexibilisebbek és képesek leírni a legtöbb görbét. Általában

$$Q(t) = \{X(t), Y(t)\}$$

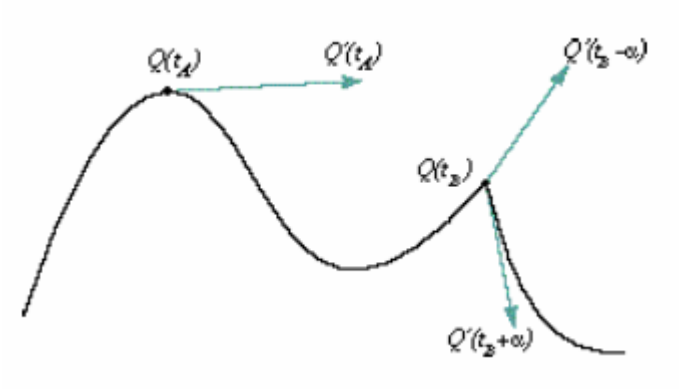
alakúak, ahol X=X(t) és Y=Y(t) t-nek függvényei. *t* egy adott értékére az X(t) megfeleltet egy *x*-et, az Y(t) pedig egy *y*-t, az (*x*,*y*) számpár egy pont a görbén.

2.3. Folytonosság

Tekintsük a görbéhez húzott érintők irányát bármely pontban. Ezeket számolhatjuk a görbét meghatározó függvények elsőrendű deriváltjaiból:

Q'(t)

Például az 1.ábrán a tA-nak megfelelő pont a görbén: Q(tA), az érintő pedig Q'(tA). Ha az érintő nem változtat hirtelen irányt akkor azt mondjuk, hogy a görbe C^1 tulajdonságú. Pontosabban: folytonos, és az elsőrendű deriváltja is folytonos. Az ábrán lévő görbének nincs meg ez a tulajdonsága a tB-nek megfelelő pontban.

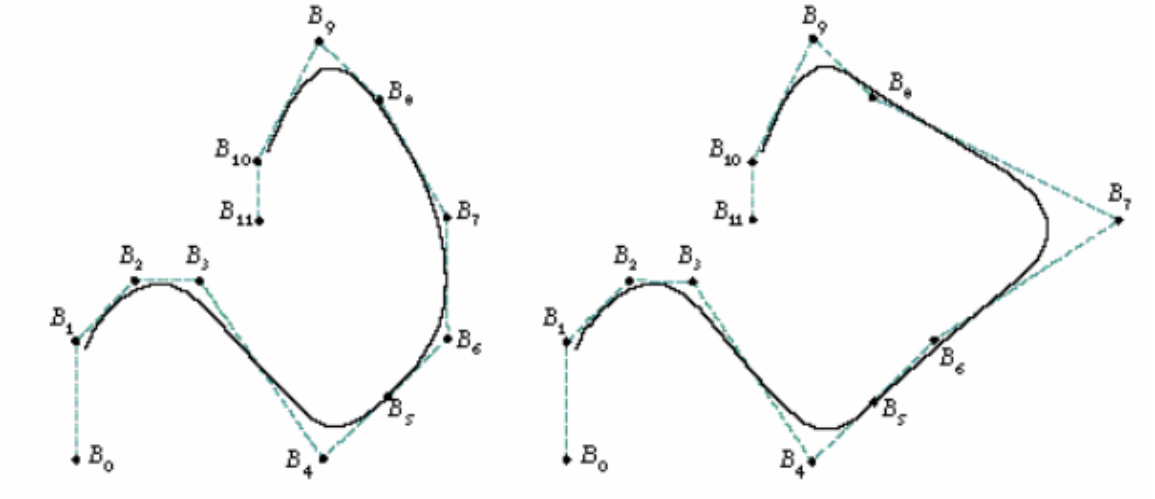


1. ábra

Ha a függvénynek a másodrendű deriváltja is folytonos akkor azt mondjuk, hogy C^2 tulajdonságú. Ez a görbületet határozza meg, de erről majd később.

2.4. Kontrol pontok

Egyik kulcs-tulajdonsága a NURBS görbéknek az, hogy a formájukat (alakjukat) egy véges ponthalmaz elemeinek helyzete határozza meg. Ezek az ú.n. *kontrol pontok*, a 2. ábrán B_i-vel ($i = 0..11$) vannak jelölve:



2. ábra

A kontrol pontok mozgatása az ábrán látható módon befolyásolja a görbét (a B_7 pontot húztuk egy kicsit jobbra). Megjegyezzük, hogy a görbének csak egy nagyon kicsi része változott meg az által, hogy egy kontrol pontot elmozgattunk. Ez egy nagyon fontos tulajdonság (például a Bezier görbékre ez nem jellemző), ugyanis lehetővé teszi, hogy lokalizált változtatásokat végezzünk a görbén, anélkül, hogy befolyásoljuk a görbe többi részét. Intuitíven így tudnánk ezt matematikai formában kifejezni:

$$Q(t) = \sum_{i=0}^{n-1} B_i N_{i,k}(t)$$

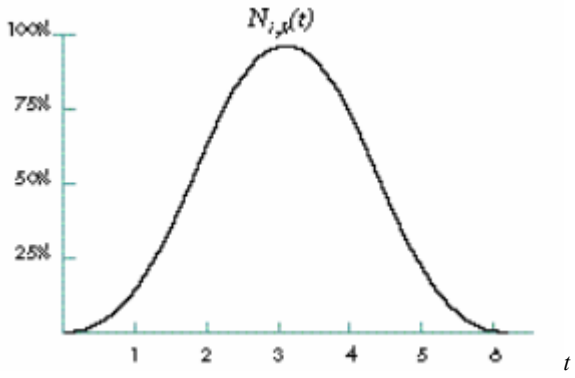
Az $N_{i,k}(t)$ függvények fejezik ki, hogy egy adott B_i kontrol pont milyen mértékben befolyásolja a görbe változását bármely *t*-ben.

2.5. Bázis függvények

Az $N_{i,k}(t)$ függvényeket nevezzük bázis függvényeknek. Tulajdonképpen a „B-spline" megnevezésből a B a *basis*-ből (vagyis bázis) jön. Az értéke egy ilyen függvénynek egy pontban egy valós szám, mint például 0,5 így a következő módon definiálódik egy görbén lévő Q(t) pont értéke: mondjuk 25% az egyik kontrol pont helyzetének, plusz 50% egy másik kontrol pont helyzetének, plusz 25% egy harmadikénak. Hogy teljes legyen a NURBS egyenletünk, még meg kell határoznunk a kontrol pontok mindegyikéhez tartozó bázisfüggvényt.

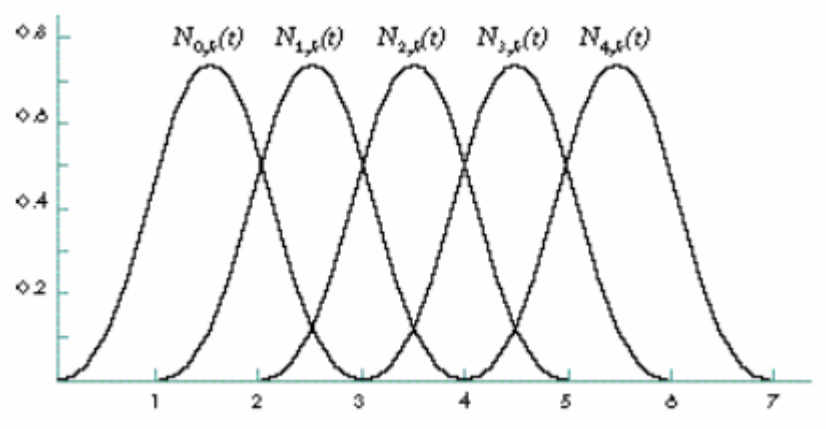
Még emlékszünk, hogy egy cél az, hogy mindegyik kontrol pontnak lokális hatása legyen a görbére. Lévén, hogy a görbénket parametrikusan definiáltuk, amíg *t* változik egy bizonyos [*a*,*b*] intervallumban, ennek a változásnak megfelel a görbének egy része (a Q(t) által). Tegyük fel, hogy a teljes görbe *t=0.0* és *t=10.0* között változik. Így egy görbe-rész lehet például *t=3.3* és *t=7.5* között, amelyen a B_i kontrol pont befolyásolja a görbét és ez a hatás *t=5*-ben központosul.

A 3.ábra egy példát mutat arra, hogy nézhet ki egy bázisfüggvény: a legnagyobb hatása a görbére *t=3.0*-ban van, amikor ez kb. 95%, és az egész görbe változását csak *t=0.0* és *t=6.0* között befolyásolja:



3.ábra

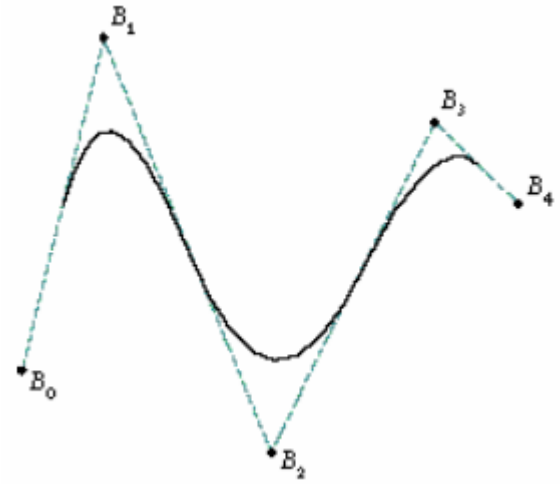
Megjegyezzük, hogy a függvény egy Gauss harang.
Mindegyik kontrol ponthoz tartozik egy bázisfüggvény, így például egy NURBS görbe, amelyet öt kontrol pont határoz meg, van öt bázis függvénye. Mindegyik egy adott *t* intervallumot takar. A 4.ábrán például *t=2.3*-ban a B_0 pont hatása a görbére 0.2, a B_1 ponté 0.7 és a B_2 –é 0.05.



4.ábra Egyenletes bázis függvények

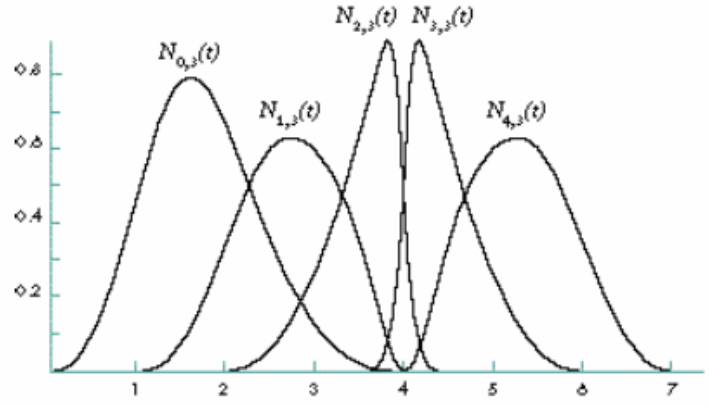
2.6. Knot-ok

Láttuk, hogy mindegyik bázis függvénynek ugyanaz az alakja, és ugyanolyan hosszúságú *t* intervallumot takar. Ha megváltoztatjuk ezeknek az intervallumoknak a hosszát akkor egyes kontrol pontok a görbének nagyobb részét fogják befolyásolni, míg mások kisebb részét. A NU a NURBS rövidítésben a **n**on**u**niform-ból jön – nem egyenletes.
A megoldás az, hogy definiálunk egy pontvektort, amely részekre osztja azt az intervallumot, amelyben *t* változik az egész görbe leírása során. A 4.ábrán látható bázisfüggvények a következő knot-vektoron vannak definiálva: {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0} (ez egy egyenletes knot-vektor). Az 5.ábrán látunk példát egy NURBS görbére, amelyet ilyen knot-vektoron értelmezett bázisfüggvényekkel és a B_i (*i = 0..4*) kontrol pontok segítségével kapunk:

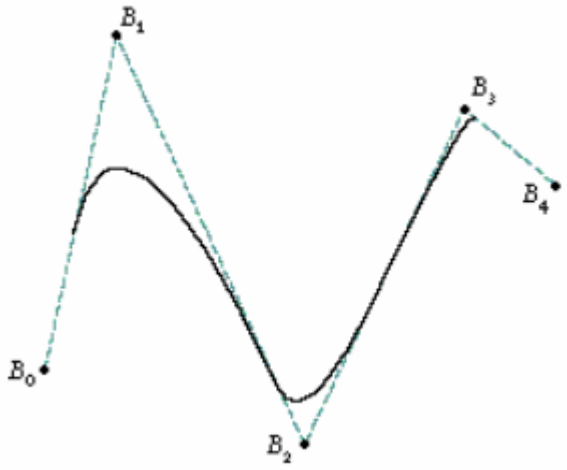


5.ábra NURBS görbe egyenletes knot-vektorral

Ha megváltoztatjuk a knot-vektort {0.0, 1.0, 2.0, 3.75, 4.0, 4.25, 6.0, 7.0}-re, akkor a bázis függvények a 6.ábrán látható módon fognak kinézni és ha ugyanazokat a kontrol pontokat használjuk mint az 5.ábrán akkor a görbe 7.ábrán látható változásokat fogja szenvedni:



6.ábra



7.ábra NURBS görbe egyenletes knot-vektorral

2.7. Bázis függvények definiálása

Elértünk ahhoz a pillanathoz, hogy megadjuk a NURBS görbék teljes definícióját az által, hogy definiáljuk a bázisfüggvényeket. Tulajdonképpen akármilyen lineárisan független függvényhalmazt használhatnánk, de nekünk olyanokra van szükség, amelyek által a fenn kért tulajdonságok kielégülnek. Pluszba még be jön az idő, amely alatt egy függvény evaluálódik. Célszerűek a polinomiális függvények, mert könnyű őket kiértékelni és jóval gyorsabban mennek mint például a trigonometrikus bázis függvények vektorának kiértékelése.
A definíció a következő:

$$N_{i,1}(t) = \begin{cases} 1, \text{ ha } x_i \le t < x_{i+1} \\ 0, \text{ másképp} \end{cases}$$

$$N_{i,k}(t) = \frac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}}$$

ahol x_i a konvencionális jelölés a *i*-dik knot-ra a knot-vektor-ból.

k a bázis függvény *rendje* (*order*), és ez szerint rekurzívan van felépítve. Ha a legnagyobb rendű bázis függvény rendje *n*, a nekik megfelelő NURBS görbét *n*-ed *rendűnek* vagy *n+1*-ed *fokúnak* hívjuk. Természetesen ezek a függvények rendelkeznek egy rakás tulajdonsággal, amelyet önök mind megtalálnak a „*An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*” című könyvben. Itt én most felsorolok néhányat ezek a tulajdonságok közül, amelyek közvetlenül kötődnek ennek a dolgozatnak

a témájához, illetve amelyek figyelembe vétele nélkül képtelenség implementálni egy olyan grafikus interfészt, amely NURBS görbékkel dolgozik:

- bármely *t*-re fennáll: $\sum_{i=0}^{n-1} N_{i,k}(t) = 1$ ha *n* kontrol pont határozza meg a görbét
- ha minden kontrol pont súlya pozitív, akkor a görbe benne van a kontrol pontok által meghatározott ponthalmaz konvex tartományában (konvex kombinációk)
- bármely *t*-re nem több mint *k* bázis függvény befolyásolja a görbét (vagyis értéke zérótól különböző)
- bármely *k*-ad rendű görbe esetében pontosan *k* darab bázis függvény evaluálódik zérótól különbözőre.

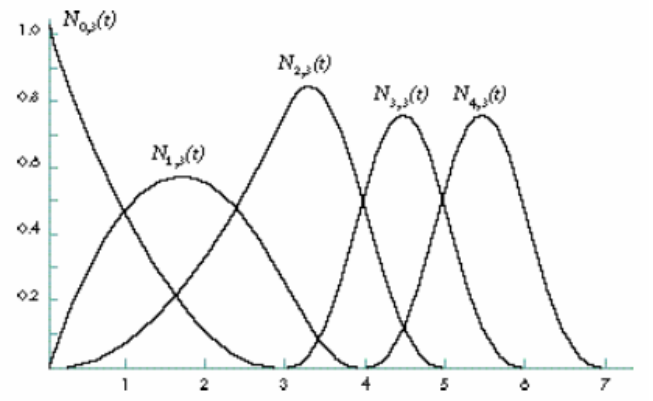
Ez utóbbi tulajdonság elméleti fontosságú, a gyakorlatban többnyire lineáris, kvadrikus, ill. kubikus NURBS-okat használnak (1,2 ill. 3-ad fokúak), ugyanis ezek elégé flexibilisek, majdnem minden folytonos felületet meg lehet velük közelíteni, és megfelelően gyorsan kiértékelhetőek.

2.8. Knot-ok és sarkak

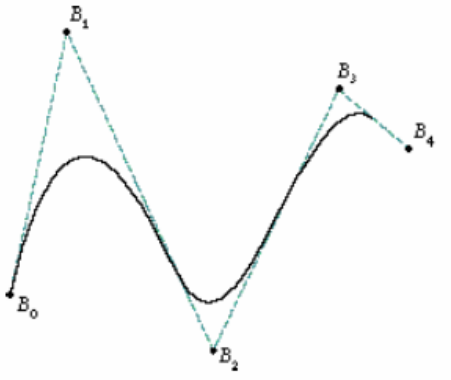
Itt meg kell jegyeznünk, hogy nem egyenletes knot vektor fontossága nem annyira a görbe formájának módosításában van, hanem a következőkben:

- lehetőséget nyújt, hogy a görbét átvezessük bizonyos kontrolpontokon
- tudjuk kontrolálni a görbe folytonosságát -> ki tudjuk ezt csúcsosítani

Mind kettőt az által tudjuk elérni, hogy egy pár pontot egybeesőnek választunk a knot-vektorból. Például a {0.0, 0.0, 0.0, 3.0, 4.0, 5.0, 6.0, 7.0} knot-vektoron értelmezett 3-ad rendű spline bázisfüggvények, és az ezek ill. egy 5 pontból álló kontrol pont vektor vektor által meghatározott görbe a következőképpen néz ki:



8.ábra



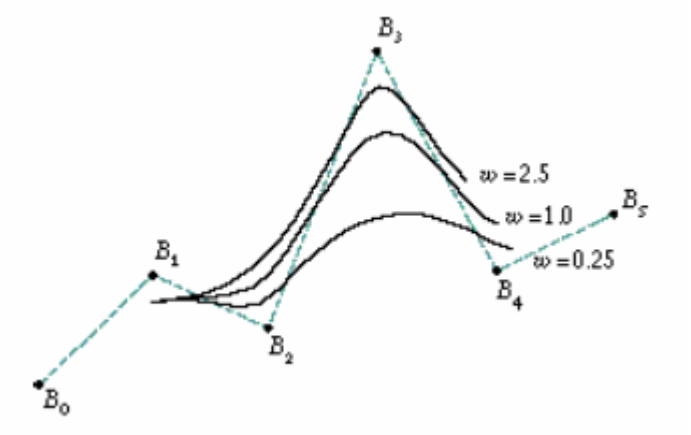
9.ábra

Észrevesszük, hogy a görbe áthalad az első kontrol ponton, mert *t=0*-ban az összes bázis függvény értéke 0 kivétel az $N_{0,3}$-nak, így a B_0 pont az egyetlen amely kontrolálja a görbét, ami által a görbe áthalad ezen a ponton. Hasonló helyzettel állunk szemben,

amikor a knot-vektor közepébe megsokszorozunk egy knot-ot. A görbe csúcsos lesz egy pontban az által, hogy kénytelen áthaladni egy kontrol ponton.

2.9. Racionális görbék

Eddig láttuk hogyan működnek a kontrol pontok, knot-ok és a bázis függvények, tehát láttuk a NUBS (**n**on**u**niform **B**-**s**pline) részét a NURBS rövidítésnek. Még hiányzik az R, amely a **r***ational* (*racionális*) megnevezésből jön.
Az ötlet az, hogy súlyozzuk a kontrol pontokat: mindegyik ponthoz hozzá rendelünk egy *súly*t (***w****eight*), amely egy pozitív valós szám. A görbék, amelyeket eddig láttunk egy olyan sajátos esetet képeztek, amikor mindegyik pont súlya pontosan 1, ami azt jelenti, hogy mindegyik kontrol pont hasonló mértékben befolyásolja a görbe változásának, a rá eső részét.



10.ábra

A 10.ábrán láthatjuk, hogyan befolyásolja a görbe változását egy kontrol pont, ha ennek súlyát 0 és 1 közé esőnek illetve 1 fölöttinek választjuk. Észrevesszük, hogy minél nagyobb egy pont súlya ($w_i>1$) a görbe annál közelebb halad el a kontrol pont mellett, és minél kisebb ez ($0<w_i<1$), a görbe annál távolabb halad el a pont mellett. Matematikailag ez így fejezhető ki:

$$Q(t) = \frac{\sum_{i=0}^{n-1} B_i \cdot w_i \cdot N_{i,k}(t)}{\sum_{i=0}^{n-1} w_i \cdot N_{i,k}(t)}$$, ahol n a kontrol pontok számát, k a görbe rendjét, B_i az i-dik kontrol pontot, w_i az i-dik kontrol pont súlyát és $N_{i,k}(t)$ az i-dik bázis függvényt jelenti. Ha ezt a kifejezést kiértékeljük egy t-re, a $Q(t)=\{X(t),Y(t)\}$ megfeleltet neki egy pontot a görbéről.

2.10. Kiterjesztések

Eddig csak a síkbeli NURBS görbékről beszéltünk, itt pedig fogjuk látni, hogy nagyon egyszerűen át tudunk térni a térbeli NURBS görbékre illetve a NURBS felületekre. Ezek is hasonló tulajdonságokkal rendelkeznek, mint a síkbeli görbék.

- térbeli NURBS *k*-ad rendű görbe *n* darab kontrol ponttal:

$$Q(t) = \{X(t), Y(t), Z(t)\}$$

$$B_i = \{x_i, y_i, z_i, w_i\}, i = \overline{0, n-1}$$

$$Q(t) = \frac{\sum_{i=0}^{n-1} B_i \cdot w_i \cdot N_{i,k}(t)}{\sum_{i=0}^{n-1} w_i \cdot N_{i,k}(t)}$$

- (*p,q*)-ad rendű NURBS felület *m*n* darab kontrol ponttal:

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u) \cdot N_{j,q}(v) \cdot w_{i,j} \cdot B_{i,j}}{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u) \cdot N_{j,q}(v) \cdot w_{i,j}}$$

ahol $N_{i,p}$ és $N_{j,q}$ a bázis függvények, $B_{i,j}$ a kontrol pontok és $w_{i,j}$ ezeknek a súlyai.

2.11. Adatstruktúrák

Ahhoz, hogy könnyebben és áttekinthetőbben tudjunk dolgozni ilyen görbékkel, szükség van egy fajta adatstruktúrára, amelyet láthatunk az alábbiakban – ezek a síkbeli NURBS görbékre vannak definiálva, térbeli görbékre vagy felületekre hasonló adatstruktúrát használunk. A módosításokat a fenti összefüggések értelmében kell tegyük.

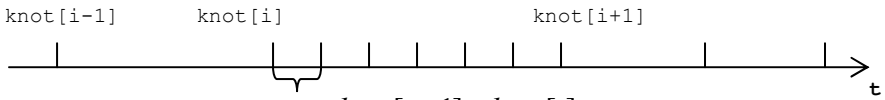
```
class controlPoint {
      double x;     //az x koordináta
      double y;     //az y koordináta
      double w;     //a pont súlya
}

class NURBScurve {
      int order;                        //a görbe rendje
      int numpoints;                    //a kontrol pontok száma
      controlPoint[] controlpoints;     //kontrol pont vektor
      double[] knots;                   //knot vektor
      AtributeSet curveAtributes;       //atributumok (pl. szín)
}
```

Egy pár dolgot nem szabad elfelejtsünk, ami az általunk adott implementációt illeti:

- a görbe rendje 2 és 16 között kell legyen. A 2 rendű görbe tulajdonképpen egy poligonális vonal, amely összeköti a kontrol pontokat. Fölülről azért korlátoztuk a rendet 16-ra, mert azt jelenti, hogy bármely *t*-re 16 darab, *t*-ben 15-öd fokú polinomot kel kiértékelnünk, amely már kezd elég lassan menni
- a *w* súly bármely pont esetén egy pozitív valós szám kell legyen
- a kontrol pontok száma nagyobb vagy egyenlő kell legyen a görbe rendjénél
- a knot-ok száma egyenlő a kontrol pontok száma plusz a görbe rendje
- a knot vektor egy növekvő valós sorozat kell legyen
- ha a görbe rendje *k*, nem szabad a knot vektorban több mint *k-1* egyenlő elem kövesse egymást

2.12. Kirajzolás

Ez nagyon egyszerű: az ötlet az, hogy bejárjuk a knot vektort egy előre meghatározott lépésszámmal, ezekben a pontokban kiértékeljük a görbét megadó függvényeket. Az így kapott pontok biztos a görbén vannak. Ha az egymás utáni pontokat egy-egy szakasszal összekötjük, így egy poligonális vonalat kapunk. Ha megfelelően nagy a lépésszám, vagyis, ha a kiértékelendő pontokat (*t*-ket) nagyon közel vesszük egymáshoz akkor a poligonális vonal fog tartani a görbéhez. Mi úgy oldtuk meg a kirajzolást, hogy van egy állítható változó: *render*, amely egy pozitív egész szám és a görbe egyenletét a knot vektor minden egymás utáni pontja között *render*-nyi egymástól egyenlő távolságra lévő pontban értékeljük ki.



$$\Delta t = \frac{knot[i+1] - knot[i]}{render}$$

Figyelembe kell vennünk azt is, hogy a görbét egy katódcsöves képernyőre rajzoljuk ki amelynek a felbontása limitált, így ennél nagyobb felbontást a görbe esetében képtelenek vagyunk megfigyelni. A görbe, jól meghatározott a kontrol pontok és a knot vektor által, amelyet elég ha eltárolunk, most, hogy mennyire pontosan rajzoljuk ezt ki, az már tőlünk függ. Ha minden *t*-re (*t* valós szám) a knot vektorból kiértékelnénk az egyenletet akkor megkapnánk a pontos görbét. A pontos megrajzolás a jelen esetben elméleti jellegű, ugyanis e dolgozat keretében a NURBS görbék azt a célt szolgálják, hogy flexibilisen definiálhassunk egy értelmezési tartományt, amelyet könnyen tudunk módosítani, egy szóval jó kontrolunk van a görbék fölött. Ezen a tartományon értelmezett kétváltozós függvények közül a keressük azt, amelyik kielégíti egy Laplace egyenletét. Ezt a függvényt véges elem módszerrel keressük, amely értelmében ezt a tartományt fel kell bontanunk kicsi egyszerű elemekre: háromszögekre. A felbontás eredményeként egy poligonális doméniumot kapunk, egy ú.n. *egyenes élű sík gráfot*. Tehát a háromszögelés során a tartomány határvonalai, amelyet a görbék kéne meghatározzanak átalakulnak egy-egy poligonális vonallá. Lévén, hogy az egyenlet megoldása a cél a jelen esetben, így nagyobb hangsúly esik a háromszögelésre, illetve a megoldási módszerre, ezért nincs értelme nagyon pontos felbontásokkal dolgozzunk a görbe esetében.

3. Véges elem módszer

3.1. A Véges elem módszer általános megfogalmazása

Legyen V egy Hilbert tér, $(\,.\,,\,.\,)_v$ skalár szorzattal és $\|\cdot\|_v$ normával. Tekintsük az $a(\,.\,,\,.\,)$ bilineáris formát $\begin{cases} a(\cdot,\cdot): V\times V \to \mathrm{R} \\ (u,v) \to a(u,v) \end{cases}$ és az $\begin{cases} L(\cdot): V \to \mathrm{R} \\ u \to L(u) \end{cases}$ lineáris formát, a következő feltételek mellett:

(i) $a(\cdot,\cdot)$ szimmetrikus

(ii) $a(\cdot,\cdot)$ folytonos mindkét változóban, vagyis létezik $\gamma > 0$ úgy, hogy

$$|a(u,v)| \le \gamma \|u\|_v \|v\|_v, \forall u,v \in V$$

(iii) $a(\cdot,\cdot)$ V - elliptikus, vagyis létezik $\alpha > 0$ úgy, hogy

$$a(u,v) \ge \alpha \|v\|_v^2, \forall v \in V$$

(iv) $a(\cdot,\cdot)$ L folytonos, vagyis $\exists \Lambda > 0$ úgy, hogy

$$|L(v)| \le \Lambda \|v\|_v, \forall v \in V.$$

Tekintjük a következő minimum illetve variacionális problémák megfogalmazását az elliptikus esetre:

(M) Határozzuk meg azt az $u \in V$ - t, amelyre fennáll:

$$F(u) = \min_{v\in V} F(v), \text{ ahol } \quad F(v) = \frac{1}{2}a(u,v) - L(v)$$

és

(V) Határozzuk meg azt az $u \in V$ - t, amelyre fennáll:

$$a(u,v) = L(v), \forall v \in V.$$

Tétel: a két probléma: (M) és (V) ekvivalens, létezik egy és csak egy $u \in V$, amely kielégíti a két problémát, a következő stabilitással:

$$\|u\|_v \le \frac{\Lambda}{\alpha}.$$

Továbbá tekintsük a V-nek egy <u>véges dimenziójú</u> V_h al-terét, és ebben egy bázist: $\{e_1, e_2, e_3, ..., e_M\}, e_i \in V_h$. Így V_h –ból bármely v vektor felírható a következő formában:

$$v = \sum_{i=1}^{M} c_i \cdot e_i, \text{ ahol } c_i \in \mathrm{R}.$$

Ennek értelmében megfogalmazhatjuk az (M) illetve (V) problémákat diszkrét formában:
(M_h) Határozzuk meg azt az $u_h \in V_h$ -t, amelyre fennáll $F(u_h) \le F(v), \forall v \in V_h$;

(V_h) Határozzuk meg azt az $u_h \in V_h$ -t, amelyre fennáll $a(u_h, v) = L(v), \forall v \in V_h$;

Vegyük a diszkrét variacionális problémát $v = e_i$ -re, ahol $i = \overline{1, M}$:
Keressük meg azt az $u_h \in V_h$ -t, amely kielégíti az $a(u_h, e_i) = L(e_i), \mathrm{i} = \overline{1, \mathrm{M}}$ összefüggést.

Használva u_h kifejezését a bázis segítségével

$$u_h = \sum_{i=1}^{M} c_i \cdot e_i, \text{ahol } c_i \in \mathrm{R}$$

a fenti relációból következik:

$$a(u_h, e_i) = L(e_i) \quad \Leftrightarrow \quad a(\sum_{i=1}^{M} c_i \cdot e_i, e_j) = L(e_j) \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \sum_{i=1}^{M} c_i \cdot a(e_i, e_j) = L(e_j), \mathrm{j} = \overline{1, \mathrm{M}} \ ,$$

vagyis

$$A \cdot c = B$$

ahol $c = (c_i) \in \mathrm{R}^M$, $b = (b_i) \in \mathrm{R}^M$, $b_i = L(e_i)$ - vel és $A = (a_{ji})$, $a_{ji} = a(e_i, e_j)$.

Tulajdonság: Az ***A*** *merev mátrix* szimmetrikus és pozitívan meghatározott.

Tétel: Legyen $u \in V$ a (V) probléma megoldása és $u_h \in V_h$ a (V_h) diszkretizált probléma megoldása, ahol $V_h \subset V$. A következő a diszkretizálás miatt felmerülő hiba-esztimáció:

$$\|u - u_h\|_v \leq \frac{\gamma}{\alpha} \|u - v\|_v, \forall v \in V_h.$$

3.2. Véges elem módszer alkalmazása a síkbeli hőeloszlás egyenletére

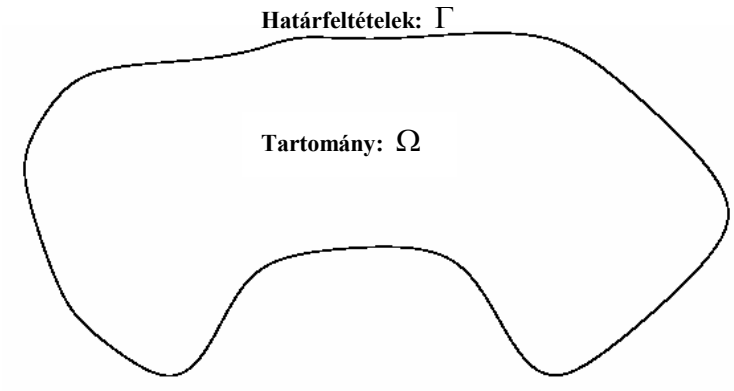
Tekintsük a kétdimenziós, időben invariáns és hőforrás nélküli hőeloszlás egyenletét, amelyet a következő parciális differenciál egyenletet fejez ki:

$$-\frac{\partial}{\partial x}\left(k_x \cdot \frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(k_y \cdot \frac{\partial u}{\partial y}\right) = 0 \qquad (1)$$

ahol k_x és k_y az x illetve y tengely menti hődiffúziós együtthatók. Ha $k_x = k_y = k$, ez a

$$-\nabla(k\cdot\nabla u)=0 \tag{2}$$

alakban írható fel, amelyben, ha k síkban állandó akkor a ***Laplace egyenlet***éhez jutunk: $k\cdot\nabla^2 u=0$. Mi a (2) egyenlet megoldását keressük az Ω tartományon, a Γ-n szabott határ feltételekkel.



Tekintsük a (2) egyenletnek megfelelő súlyozott integrál egyenletet:

$$\int_\Omega -\nabla(k\cdot\nabla u)w d\Omega=0 \tag{3}$$

A Gauss-Green formula értelmében:

$$\int_\Omega -\nabla(k\cdot\nabla u)w d\Omega=\int_\Omega k\nabla u\cdot\nabla w d\Omega-\int_\Gamma k\frac{\partial u}{\partial n}w d\Gamma \tag{4}$$

A (4) relációt behelyettesítve a (3) összefüggésbe, a (2) egyenlet ekvivalens formáját kapjuk:

$$\int_\Omega k\nabla u\cdot\nabla w d\Omega=\int_\Gamma k\frac{\partial u}{\partial n}w d\Gamma \tag{5}$$

A baloldali integrálandót a következőképpen írjuk át:

$$\nabla u\cdot\nabla w=\frac{\partial u}{\partial x_k}\cdot\frac{\partial w}{\partial x_k}=\frac{\partial u}{\partial \xi_i}\frac{\partial \xi_i}{\partial x_k}\cdot\frac{\partial w}{\partial \xi_j}\frac{\partial \xi_j}{\partial x_k} \tag{6}$$

ahol $u=\varphi_n u_n$ és $w=\varphi_m$, ahogy fenn láttuk (φ_n - bázisfüggvények).

A $\frac{\partial \xi_i}{\partial x_k}$ tényezőket a következő módon számoljuk:

$$\begin{bmatrix} \frac{\partial \xi_1}{\partial x} & \frac{\partial \xi_1}{\partial y} \\ \frac{\partial \xi_2}{\partial x} & \frac{\partial \xi_2}{\partial y} \end{bmatrix} = \frac{1}{\frac{\partial x}{\partial \xi_1}\frac{\partial y}{\partial \xi_2} - \frac{\partial x}{\partial \xi_2}\frac{\partial y}{\partial \xi_1}} \begin{bmatrix} \frac{\partial y}{\partial \xi_2} & -\frac{\partial x}{\partial \xi_2} \\ -\frac{\partial y}{\partial \xi_1} & \frac{\partial x}{\partial \xi_1} \end{bmatrix}.$$

3.3. Bázis függvények – diszkretizáció

Legyen $\Omega = \bigcup_{i=1}^{I} \Omega_i$ a tartomány egy háromszögelése.

Vegyünk egy Ω_l háromszöget, amelynek csúcsai: $(x_{vi(l)}, y_{vi(l)}), i = \overline{1,3}$. Tekintsük a d pozitív egész számot. A $P_{ijk}^{[l]} = (x_{ijk}^{[l]}, y_{ijk}^{[l]}), i + j + k = d$ pontokat Bezier doménium pontoknak hívjuk és a következő módon definiáljuk:

$$\begin{cases} x_{ijk}^{[l]} = \frac{ix_{v1(l)} + jx_{v2(l)} + kx_{v3(l)}}{d} \\ y_{ijk}^{[l]} = \frac{iy_{v1(l)} + jy_{v2(l)} + ky_{v3(l)}}{d} \end{cases}$$

Jelöljük $D_d(\Delta) = \{P_{ijk}^{[l]}\}$ az összes doménium pont halmazát, amelynek számossága: $M = |D_d(\Delta)| = V + (d-1)E + \binom{d-1}{2}T$, ahol V a csúcsok számát, E az élek számát és T a háromszögek számát jelöli. Feleltessünk meg minden ilyen doménium pontnak egy valós számot: $c : D_d(\Delta) \to \mathrm{R}, c_{ijk}^{[l]} \leftrightarrow P_{ijk}^{[l]}$.

Definiálunk minden háromszögön egy-egy baricentrikus koordináta rendszert:

$$\begin{cases} 1 = \beta_1 + \beta_2 + \beta_3 \\ x = \beta_1 x_{v1(l)} + \beta_2 x_{v2(l)} + \beta_3 x_{v3(l)} \\ y = \beta_1 y_{v1(l)} + \beta_2 y_{v2(l)} + \beta_3 y_{v3(l)} \end{cases}$$

és egy függvényt:

$$S^{[l]}(x,y) = \sum c_{ijk}^{[l]} \frac{d!}{i!\,j!\,k!} \beta_1^i \beta_2^j \beta_3^k .$$

Így egy M dimenziójú d-ed rendű spline tért kaptunk (ahol M véges) ami segítségével sikerült síkban diszkretizálni a Ω-n értelmezett függvények terét.

Ebben a térben fogjuk keresni a (2) egyenlet megoldását ami egy

$$u(x,y) = S^{[l]}(x,y),\ (x,y) \in \Omega_l \qquad (7)$$

alakú függvény lesz. Meg kell tehát határoznunk a $c_{ijk}^{[l]}$ konstansokat.

3.4. Integrálás

Mi lineáris bázis függvényeket használtunk a diszkretizáláshoz az implementálás során (*d=1*). Így a diszkretizált térnek a dimenziója megegyezik a csúcsok számával. A csúcsok halmaza képezi ebben az esetben a doménium pontok halmazát. Mindegyik csúcshoz hozzá van rendelve egy-egy valós szám: $u_i, i = \overline{1,V}$ (V lévén a csúcsok száma) , amelyeknek a meghatározását fogjuk követni az alábbiakban.

Ebben az esetben *u* a következőképpen néz ki mindegyik háromszögön:

$$u(x,y) = u_k\varphi_1(x,y) + u_p\varphi_2(x,y) + u_q\varphi_3(x,y)$$

ahol u_k , u_p , u_q az Ω_l háromszög *k,p* illetve *q* csúcsainak megfelelő valós számok, $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ ezeknek a koordinátái, és

$$\begin{cases} \varphi_1 = \xi_1 = \beta_1 = \dfrac{1}{\Delta}\left[(y_2-y_3)x + (x_3-x_2)y + x_2y_3 - x_3y_2\right] \\ \varphi_2 = \xi_2 = \beta_2 = \dfrac{1}{\Delta}\left[(y_3-y_1)x + (x_1-x_2)y + x_3y_1 - x_1y_3\right] \\ \varphi_3 = 1-\xi_1-\xi_2 = 1-\beta_1-\beta_2 = \dfrac{1}{\Delta}\left[(y_1-y_2)x + (x_2-x_1)y + x_1y_2 - x_2y_1\right] \\ \Delta = (x_2-x_1)(y_3-y_1) - (x_3-x_1)(y_2-y_1) \end{cases}$$

a fenti jelölések értelmében.

Tehát (5) egyenletből kapjuk a

$$\sum_i u_n \int_\Omega k\left(\frac{\partial\varphi_n}{\partial x}\frac{\partial\varphi_m}{\partial x} + \frac{\partial\varphi_n}{\partial y}\frac{\partial\varphi_m}{\partial y}\right) = \int_\Gamma k\frac{\partial u}{\partial n}\varphi_m d\Gamma$$

relációt, amelyik meghatározza a *merev mátrixot(element stiffness matrix)*: A_{mn}.

Például A_{11} –et így számoljuk: $A_{11} = Terület \cdot \left[\left(\frac{y_2 - y_3}{\Delta} \right)^2 + \left(\frac{x_3 - x_2}{\Delta} \right)^2 \right]$,

ahol $Terület = \frac{1}{2}|\Delta|$, a háromszög területét jelenti.

3.5. A merev mátrixok összevonása egy globális egyenletrendszerré

Eddig, amint láttuk, mindent redukáltunk egy háromszögecskére. Most itt az ideje, hogy ezeket összerakjuk: azok a háromszögek, amelyek szomszédosak (élet osztanak meg egymással) a merev mátrixaiknak a megfelelő elemeit összeadjuk. Így kapunk egy V egyenletből álló egyenletrendszert, amelyben az ismeretlenek $u_i, i = \overline{1,V}$ (V a csúcsok száma). Ennek a rendszernek a mátrixa, mint fenn láttuk szimmetrikus, és ha az elemek száma nagy, akkor ez egy ritka mátrix. A rendszerre alkalmazzuk a határfeltételeket, amely redukálja ennek rendjét. A megoldásból kapott konstansokat ($u_i, i = \overline{1,V}$) visszahelyettesítjük a (7) összefüggésbe és megkaptuk a keresett függvényt, amely kielégíti a kért egyenletet.

3.6. Diszkretizálási hibák

Tekintsük a Ω tartományon értelmezett függvények terén az egyenletes távolságot $dist(f,g) = \max_{(x,y)\in\Omega} |f - g|$. Továbbá tekintsük $\Omega = \bigcup_{i=1}^{I} \Omega_i$ egy háromszögelését és az ez segítségével értelmezett d –ed rendű véges spline teret $S_d(\Delta)$. Igaz a következő összefüggés:

$\exists c \in R$ pozitív valós szám ú.h. $dist(f, S_d(\Delta)) \le c \cdot h^{d+1} \cdot \left\| D^{d+1} f \right\|$, ahol

$h = \max_{1 \le j \le n} (\Omega_j$ háromszög átmérője)

1992-ben Schumaker belátott egy olyan összefüggést, amelyben a közelítés pontosságát a háromszögek minimum szögének függvényében adja meg. Ennek értelmében minél nagyobb a minimum szög, annál pontosabb a közelítés. A dolgozatban szereplő háromszögelési módszer ezt a célt követi, vagyis, hogy maximizálja a minimum szöget.

3.7. Adatstruktúra és implementáció

A következő adatstruktúrákat használtuk a csúcsokra, élekre illetve háromszögekre:

```
class Node {

      Point koordinata;

      /* numerikus információ */
      double aii, bi, xi;
}

class Edge {
```

```
        Node ni, nj;
        Element epos, eneg;

        /* numerikus információ */
        double aij, aji;
}

class Element {

        Node[3] node;
        Edge[3] edge;

        /* numerikus információ */
        double[3][3] stiffnessMatrix;
}
```

A rendszert úgy építjük fel, hogy bejárjuk az éleket, és a numerikus információkat egy új mátrixba írjuk át. A rendszer megoldását három módon is implementáltuk:

- Gauss eliminációval
- Cholesky faktorizációval
- Gauss-Seidel iteratív úton.

A megoldást háromszögenként rajzoljuk ki színekben ábrázolva az értékeket.

3.8. Kiterjesztések

E dolgozat keretén belül, amint fenn láttuk, lineáris spline bázis függvényekkel diszkretizáltunk, amelyekkel viszonylag könnyű volt dolgozni, lévén, hogy a parciális deriváltak mind konstansnak jöttek ki, így a 3x3-as merev mátrix formáját kézzel ki tudtuk számolni a háromszög csúcsinak koordinátái függvényében. Így a program viszonylag gyorsan fut. Ha áttérnénk másodfokú bázis függvényekre, a merev mátrix 6x6-os lenne, amelyben az elemeket már numerikus kvadratúrákkal kéne számolni. Ez a futási időt jelentősen megnövelné, de viszont a pontosság is nőne.
Be van látva, hogy léteznek olyan egyenletek, amelyekre az itt látott közelítő megoldás nem kielégítő még harmadfokú bázisok választásával sem.
Három dimenziós, térbeli tartományokra való áttérés viszonylag könnyen menne: újra kell definiálnunk a baricentrikus koordináta rendszert, amelyeket már tetraéderekre értelmeznénk, bejönne egy plusz adatstruktúra: a *lap*. A globális egyenletrendszer összerakását ezek alapján tennénk. A többi nagyjából ugyanaz maradna.

4. Felosztás

4.1. Bevezető a véges elem felosztások generálásába

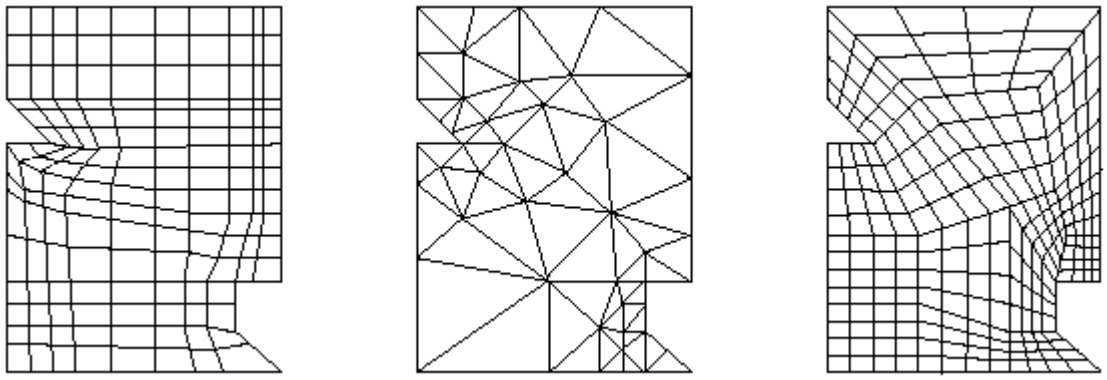
Amint láttuk numerikus megoldások sokszor diszkretizálásra alapoznak, így történik a mi esetünkben is. Egy folytonos tartományt véges számú kis “elem”-re osztjuk fel. Síkban négyszögű vagy háromszögű elemekre, térben hexahedronokra vagy tetraéderekre. Ezt a diszkretizálást nevezzük véges elem felosztásnak.

A véges elem felosztás generálásában három fő irányzat alakult ki: a *strukturált* (structured), a *nem-strukturált* (unstructured) és a hibrid, *blokk-strukturált* (blokk-structured) felosztás. Ez utóbbi az első kettőnek a kombinációja.

A *strukturált* felosztások esetében minden csúcs foka megegyezik, minden csúcs topológikusan identikus. Ezek viszonylag egyszerű felosztások, nem használnak túl sok memóriát, mivel a koordinátákat ki lehet számolni és nem szükséges eltárolni. Hátrányuk a flexibilitás hiánya. Bonyolultabb értelmezési tartományokra nagyon nehéz, vagy akár lehetetlen egy ilyen felosztást generálni. Vannak különböző technikák, ami segítségével mégiscsak kiszámolható egy felosztás, de ezek általában nem kielégítően pontosak főleg a határvonalak mentén.

A *nem-strukturált* felosztások esetében változó a csúcsok foka. Ennek nagy előnye a flexibilitás, aminek köszönhetően tetszőleges tartományokra használható. A nem-strukturált felosztások általában háromszög-elemeket generálnak.

A *blokk-strukturált* felosztás több lokális strukturált felosztás kombinálásából áll. Ezért mindkét előbbi felosztás előnyeit élvezi, ellenben ez a felosztás nem teljesen automatizált, emberi beavatkozás szükséges generálásához és ezért kevésbé népszerű.



Strukturált, nem-strukturált és hibrid felosztás (balról jobbra)

A véges elem módszerek esetében, éppen a flexibilitás miatt, a nem strukturált felosztások a legelterjedtebbek és mi is ezt fogjuk használni. Amint már említettük ez a felosztás háromszögeket generál, tehát háromszögel. A klasszikus értelemben vett háromszögelés nem engedélyezi csak a bemenet csúcsokat, a mi esetünkben azonban szükséges az úgynevezett Steiner pontok használata, vagyis a bemenet csúcsokon kívül

használt csúcsok. Miért szükségesek? A válasz egyszerű: a háromszögek bizonyos, formára és nagyságra vonatkozó feltételeknek kell eleget tegyenek, ami sok esetben nem lehetséges új csúcsok nélkül.

4.2. A feladat megfogalmazása és a megoldás

Mi a két dimenziós *háromszögelés* (triangulation)? A bemenet tartomány olyan felosztása, ahol az elemek csak élekben, csúcsokban találkoznak vagy egyáltalán nem metszik egymást. Az *optimális háromszögelés* (optimal triangulation) pedig nagyság, forma vagy a háromszögek száma, tehát bizonyos szempontok szerinti legjobb háromszögelés.

A mi esetünkben a feltételek a következők:

- ne használjunk kis szögeket, vagyis háromszögeink a lehető legközelebb legyenek az egyenlő oldalú háromszöghöz (konvergencia)
- minél kevesebb háromszöget használjunk (véges elem analízis sebessége)

A feltételek megfogalmazásából már látszik, hogy nem egyértelmű, melyik a jó háromszögelés. Használhatunk nagyon sok kis háromszöget, amelyek nagy mértékben megközelítik az egyenlő oldalú háromszöget, de ebben az esetben nem elégítjük ki a második feltételt. Tehát egy bizonyos kompromisszum egyensúlyt kell találni a két feltétel teljesítésében.

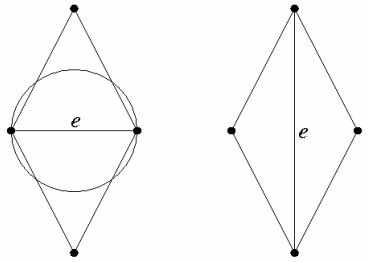
Feladatunk, a fent említett feltételeknek megfelelő háromszögelés generálása tetszőleges formájú, akár lyukakat is tartalmazó síkbeli tartományra. Még pontosabban, tartományunk poligonális tartomány, amit a NURBS görbék tárgyalásánál már bemutattunk. Tehát a bemenet egy poligonális síkbeli tartomány, a kimenet meg ennek a háromszögelése.

Az általunk alkalmazott módszer a Delaunay háromszögelést használja: először is generáljuk a bemenet tartomány kötött Delaunay háromszögelését, majd a Delaunay finomítási algoritmus értelmében elhelyezzük a Steiner pontokat. Ez a módszer biztosítja a kimenet háromszögelés minimum szögét, és elméletileg bizonyíthatóan korlátos számú háromszöget használ. A minimum szöget felhasználó szinten módosíthatjuk. Ellenben határt szab ennek az értéknek, mind az algoritmus működése, mind a számítógép számolási hibája. A gyakorlatban ez akár 35 fok is lehet. Ennek az algoritmusnak Egy másik előnye, hogy változó méretű háromszögeket használ, vagyis ahol nem szükséges, nem osztja tovább a háromszögeket. Az ilyen felosztásokat *nem-egyenletes* (non-uniform) felosztásoknak nevezzük.

4.3. Ponthalmaz Delaunay háromszögelése

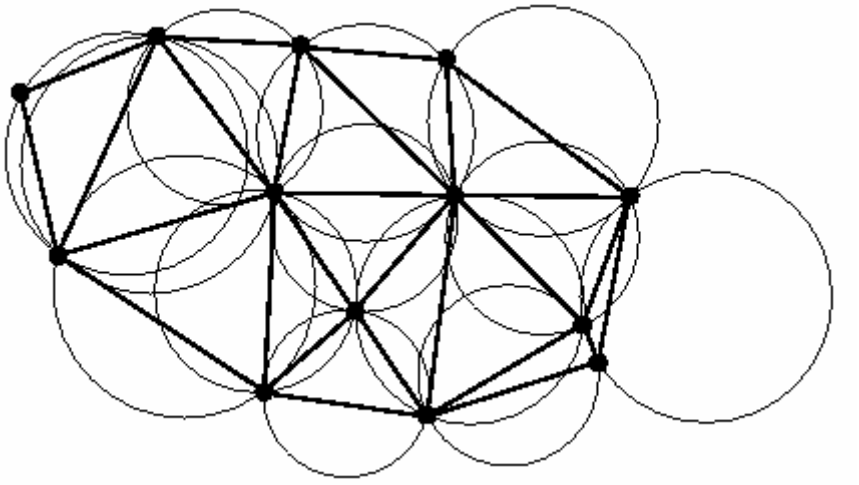
Adott S egy síkbeli ponthalmaz, aminek a Delaunay háromszögelését keressük. S Delaunay háromszögelése a következő tulajdonságú gráf: egy kört *üres*nek nevezünk, ha nem tartalmaz egy pontot sem az S halmazból. *uv* (u,$v \in$ S) *Delaunay él,* ha létezik olyan

üres kör, ami áthalad az *u* és *v* pontokon. Bevezetjük a *Delaunay háromszög* fogalmát is: egy háromszög akkor és csakis akkor Delaunay tulajdonságú, ha a köréje írt kör üres. Könnyen belátható, hogy ha egy háromszög Delaunay tulajdonságú, akkor az élei Delaunay élek és fordítva. Tehát a Delaunay háromszögelés Delaunay éleket tartalmazó maximális sík gráf (graf planar maximal). Ha az S bemenet halmazban a pontok általános helyzetben vannak, vagyis bármely négy pont nincs egy körön, akkor egy és csakis egy Delaunay háromszögelés létezik.



e Delaunay e nem Delaunay

Egyik tulajdonsága a Delaunay háromszögelésnek, hogy tartalmazza a ponthalmaz konvex tartományát (convex hull). Egy másik nagyon fontos (és számunkra érdekesebb) tulajdonsága a Delaunay háromszögelésnek, hogy egy adott S ponthalmaz esetén az összes háromszögelés közül a maximális minimál szöget biztosítja. Mit jelent ez? Definiáljuk egy háromszögelés minimál szögét, mint a háromszögelést alkotó háromszögek szögei halmazából a legkisebb szöget. Világos, hogy egy ponthalmazra nagyon sok háromszögelés létezik, de ezek közül a Delaunay háromszögelés biztosítja a legnagyobb minimál szöget. Egyúttal a Delaunay háromszögelés minimizálja a legnagyobb háromszög köré írt kört.



Delaunay háromszögelés – látható, hogy minden háromszög koré írt kör üres

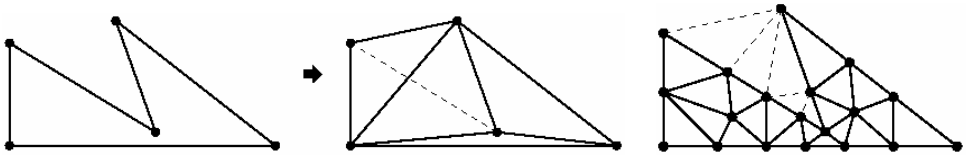
Ezeket a tulajdonságokat a “flip” algoritmus segítségével bizonyíthatjuk. A “flip” algoritmus egy tetszőleges háromszögelésből kiindulva meghatározza a Delaunay háromszögelést, addig “flip”-eli a nem Delaunay éleket, amíg az összes él Delaunay tulajdonságú lesz. A “flip”-elés során az élhez tartózó két háromszög által meghatározott négyszögben a jelenlegi átló helyett a másikat fogjuk használni. Bebizonyítható, hogy ez az algoritmus meghatározza a Delaunay háromszögelést. Ezenkívül minden “flip”

művelet során a fent említett tulajdonságok jó irányba változnak, s ennek köszönhetően az algoritmus végére optimális helyzetbe kerülünk.

Ezek a tulajdonságok a felosztás generálásnak pontosan megfelelnek. A hasonlóan értelmezett nagyobb dimenziójú háromszögelésekben sajnos már nem érvényesek.

4.4. Kötött Delaunay háromszögelés

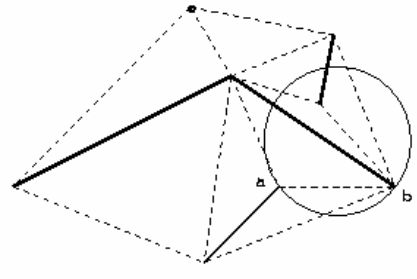
A ponthalmaz Delaunay háromszögelése nem megfelelő a véges elem felosztás esetében, amint az alábbi ábrán is látható. Ez a háromszögelés tartalmazhat olyan éleket, amelyek nem érvényesek, vagy olyan háromszögeket, amelyek rossz formájúak.



a tartomány a csúcsok Delaunay háromszögelése egy érvényes felosztás

Világos, hogy a minket érdeklő tartomány határéleit is mindenképp kell tartalmazza felosztásunk. Ezért bevezetjük a *kötött Delaunay háromszögelést* (constrained Delaunay triangulation). A ponthalmaz hasonló Delaunay háromszögeléséhez, de itt a bemenet egy egyenes élű sík gráf. Az egyenes élű sík gráf olyan gráf, ami két feltételnek tesz eleget: tartalmazza minden él végpontjait, illetve az élek csak végpontokban vagy egyáltalán nem metszhetik egymást. A kötött Delaunay háromszögelés abban különbözik a ponthalmazétól, hogy minden bemenet él jelen kell legyen a háromszögelésben.

Ahhoz, hogy definiálhassuk a kötött Delaunay háromszögelést szükségünk van még egy fogalomra. Egy *u* csúcs *látható v* csúcsból, ha az *uv* szakasz nem metszi a gráf egyetlen élét sem. Hasonlóan a *uv* él látható a *w* csúcsból, ha *uv* él bármely pontja látható *w*-ből. A kötött Delaunay háromszögelés akkor és csakis akkor tartalmazza az *uv* élet, ha *u* látható *v*-ből és létezik egy kör *u* és *v* csúcsokon keresztül, amely nem tartalmaz olyan bemenet-csúcsot, amelyből látható az *uv* szakasz. Természetesen, amint már említettük a bemenet-élek részei a háromszögelésnek.



Kötött Delaunay háromszögelés

A kötött Delaunay háromszögelésre is érvényesek azok a tulajdonságok amiket a ponthalmaz Delaunay háromszögelésére kijelentettünk, s ezért használhatóak véges elem felosztás generálására.

De még ez sem biztosít egy jó minőségű felosztást. Hiszen láttuk, hogy a ponthalmaz Delaunay háromszögelése tartalmazhatott olyan háromszögeket, amelyek nagyon laposak, vagy nagyon hegyesek voltak, s ezt a hátrányt nem távolította el a kötött Delaunay háromszögelés sem. Ha új csúcsokat is használunk a háromszögelésünkben, akkor amint az ábra mutatja, érvényes, jó minőségű felosztáshoz juthatunk. Ezeket a csúcsokat Steiner pontoknak nevezzük.

Természetesen ezeket a Steiner pontokat csak tartományunk belsejébe, vagy esetleg határvonalaira vehetjük fel. A feladat az, hogy miként válasszuk meg ezeket az új csúcsokat, hogy megfelelő felosztás eredményezzenek. Amint később látni fogjuk, a Delaunay finomítási algoritmus választ ad erre a kérdésre.

Tehát körvonalazódtak feladatunk megoldásához szükséges lépések:

- generáljuk a poligonális tartomány kötött Delaunay háromszögelését, és ebből csak a tartomány belsejét vegyük figyelembe (a tartományon kívül eső részek nem érdekesek a mi szempontunkból)
- ezt a háromszögelést addig javítjuk a Delaunay finomítási algoritmus alapján, amíg megfelel a felhasználó minimum szög szempontjának. Ez a finomítás megfelelő Steiner pontok hozzáadásával történik.

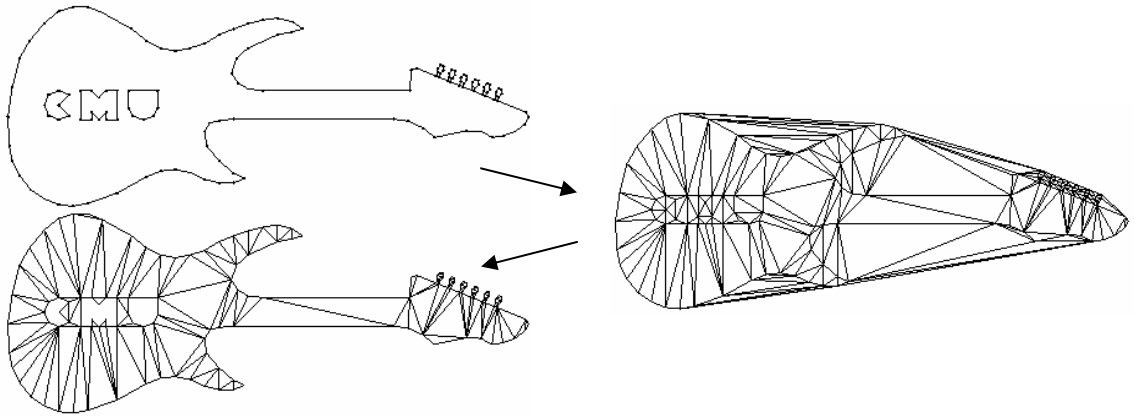
4.5. Poligonális tartomány Delaunay háromszögelése

Mit jelent a *poligonális síkbeli tartomány*?

Egy $B = \{b_0, b_1, \ldots, b_n\}$ határpoligon halmaz által meghatározott tartomány, ami eleget tesz a következő feltételeknek:

- minden határpoligon olyan zárt poligon, amely véges számú irányított szakaszból áll (ezek a szakaszok nem metszik egymást csak végpontokban) és minden szakasz bal oldalán található a meghatározott tartomány. Egy ilyen poligont a csúcsok bizonyos sorrendben megadott sorozata is meghatároz.
- az első határpoligon a külső, ez mindenképp jelen kell legyen.
- a belső határpoligonok a lyukakat határozzák meg, ezek legrosszabb esetben is csak egy csúcsban metszhetik egymást. A belső határpoligonok száma tetszőleges, akár hiányozhatnak is.

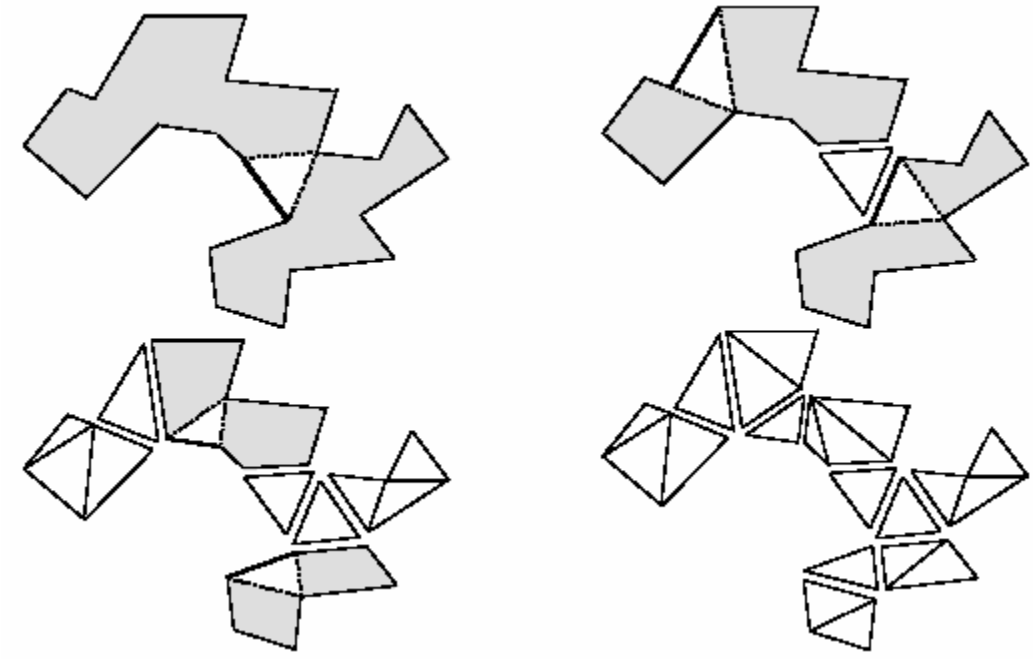
Ez a meghatározás eleget tesz az egyenes élű sík gráf meghatározásának is, de sajátos eset. Sok szerző a gráf kötött Delaunay háromszögelését generálja, majd a külső éleket kitörli az eredményből. Hatékonyabb megoldás a háromszögelés azon éleit generálni, amelyek a tartomány belsejében vannak. Ezt a módszert használja a mi programunk is.



Hagyományos kötött Delaunay háromszögelés egy síktartomány esetén – a mi módszerünk kiugorja a második lépést.

Természetesen az alkalmazott algoritmus veszít általánosságából, s tetszőleges egyenes élű sík gráf bementre nem működik. De ez, a mi szempontunkból egyáltalán nem fontos.

Ezt az algoritmust 1996-ban Reinhard Klein javasolta. Divide-et-impera típusú algoritmus. A legfontosabb megfigyelés az, hogy minden bemenet irányított élnek megfelel egy és csakis egy csúcs, amivel az él Delaunay háromszöget alkot. Egy élnek megkeressük a megfelelő csúcsát, és létrehozzuk a háromszöget, majd rekurzívan más élek háromszögeit keressük. Egy poligonális láncot egy háromszög két részre oszt. Most ezeknek a láncoknak a háromszögelése a feladatunk, egészen addig amíg csak háromszögeink maradnak. Egy fontos feladat a poligonális láncon az elválasztó él megválasztása úgy, hogy egyenletesen darabolja fel a láncot, különben logaritmikus komplexitás helyett lineáris lesz a domináns. A megválasztásról bővebben a későbbiekben lesz szó.



Az algoritmus működése egy szimpla poligon esetén (nincsenek lyukak)

Melyik a keresett csúcs és hogyan találjuk meg?

Ez a csúcs a következő tulajdonságoknak kell eleget tegyen:

- ebből a csúcsból látható kell legyen az él, és az alkotott háromszög a tartomány belsejében kell legyen.
- a három pont által meghatározott kör üres kell legyen.

Természetesen ez a csúcs akkor egyedi, ha a bemenet csúcsok általános helyzetben vannak, de az algoritmusunk szempontjából elégséges, ha megtalálunk egy érvényes csúcsot. Természetesen fontos, hogy egy adott élre hatásos algoritmussal kapjuk meg ezt a hozzátartozó csúcsot. Először lefordítjuk a feltételeket a programozás szempontjából elérhetőbb kijelentésekké:

1. a keresett csúcs az irányított él balfelén helyezkedik el
2. az él végpontjai láthatóak kell legyenek ebből a pontból
3. a három pont által meghatározott kör üres kell legyen.

A 2. feltétel nem födi azt a tényt, hogy az él látható a csúcsból, hiszen lehetséges, hogy a végpontoktól különböző pont az érül nem látható a mi csúcsunkból. De összességében a 3 feltétel nem veszít általánosságából, mert 3. feltétel kiszűri az előbb átengedett pontokat.

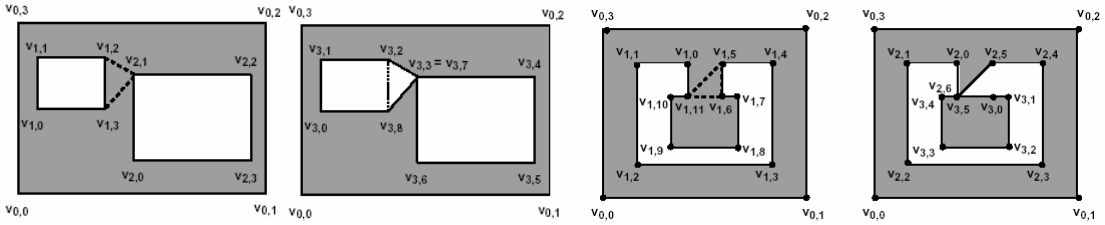
Keresésünk a lehetséges csúcsokat teszteli. Megjegyzendő, hogy ha a feltételek egyike nincs kielégítve, akkor a pont biztos nem jó, és ilyen meggondolásból fölösleges a többi feltételt ellenőrizni. Ha feltételeket ilyen sorrendben vizsgáljuk, akkor kevés pontra kell mind a három feltételt leellenőriznünk. Ez fontos a hatékonyság szempontjából.

Egy másik fontos kérdés az, hogy válasszuk meg e lehetséges csúcsokat, hogy minél hamarabb megtaláljuk a jó csúcsot. Világos, hogy a keresett csúcs nincs nagy távolságra az éltől és ezért a következő módszert alkalmazzuk: az él körül meghatározunk egy környezetet, amelyben keressük a csúcsot, és ezt a környezetet addig növeljük, amíg megkapjuk a csúcsot.

Most, hogy megfelelő módon meg tudjuk határozni a Delaunay csúcsot, vizsgáljuk meg, hogy a poligonális láncról milyen szempont szerint válasszuk ki a feldolgozandó élet. A feladat a láncról egy olyan élet kiválasztani, ami a legegyenletesebben választja szét a láncot. Ennek érdekében értelmezhetünk egy funkcionálist, ami megfeleltet egy értéket minden élnek, és ennek a maximuma szerint választjuk meg az élet. Az algoritmus eredeti szerzője öt különböző ilyen függvényt tesztelt különböző bemenetekre és így választotta ki a legmegfelelőbbet: az élhez tartozó két szög összege. Tehát mielőtt egy poligonális láncot feldolgoznánk végigmegyünk a láncon és kiválasztjuk azt az élet, amely maximizálja a függvényt.

Az eddigi tanulmányunk nem tért ki, arra hogyan tud az algoritmus elbánni a lyukakkal. A lyukaknak megfelelő lánc feldolgozásakor két eset lehetséges:

- két lyuk egyesül
- a tartomány két részre szakad



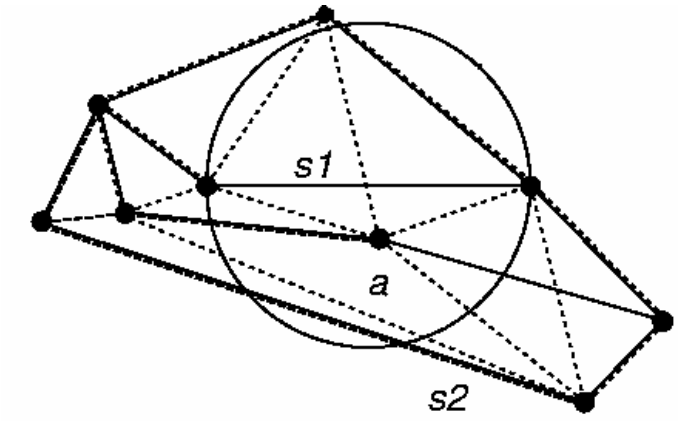
a két lyuk egyesül a tartomány két részre szakad

Az algoritmusnak nem jelent nehézséget egyik eset sem, könnyen belátható, hogy mivel a láncok feldolgozása teljesen független, ezért tökéletesen fog működni lyukak esetén is.

4.6. Delaunay finomítási algoritmus

Adott a tartomány belsejének kötött Delaunay háromszögelése és keressük azokat a pontokat, amelyeket ha hozzáadjuk a háromszögelésünkhez, akkor növelik a minimál szöget. Ezt az algoritmust Jim Ruppert fejlesztette ki és 1994-ben kerül kiadásra.

Két alapműveletet használunk az algoritmus során: az élosztás és a háromszögosztás. Értelmezzük egy él *átmérős kör*ének azt a kört melyet úgy szerkesztünk, hogy az él legyen a kör átmérője. Egy bemenet élet egy csúcs által *birtokolt*nak nevezünk, hogy ha az él átmérős köre tartalmazza azt a csúcsot. Most meg értelmezzük a két műveletet. Az *élosztás* során bevezetjük az él felezőpontját, mint a háromszögelés Steiner pontját. A *háromszögosztás* során pedig hasonlóan a háromszög köré írt kör középpontját vezetjük be mint Steiner pontot. Az algoritmus szempontjából az élosztás nagyobb prioritású, tehát, amíg léteznek birtokolt élek, addig nem oszthatunk háromszöget, illetve, ha egy háromszögosztás során valamely bemeneti él birtokolttá válhat, akkor, a háromszögosztás helyett azokat az éleket osztjuk. *Kisszögű* háromszögnek azt a háromszöget nevezzük, amelynek minimum szöge kisebb, mint a mi korlátunk.

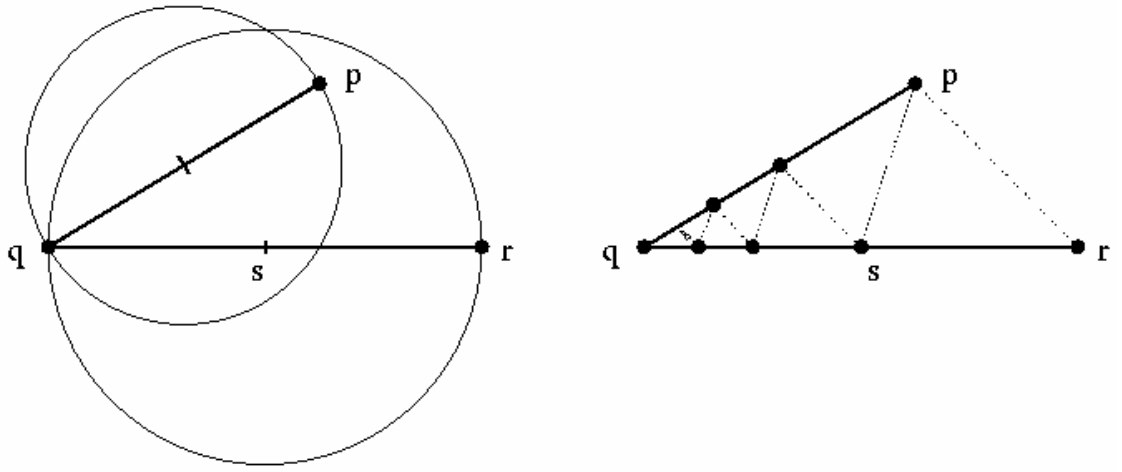


A mellékelt ábrán mind az s_1, mind az s_2 az a pont által birtokolt

Az algoritmus a következő módon működik:

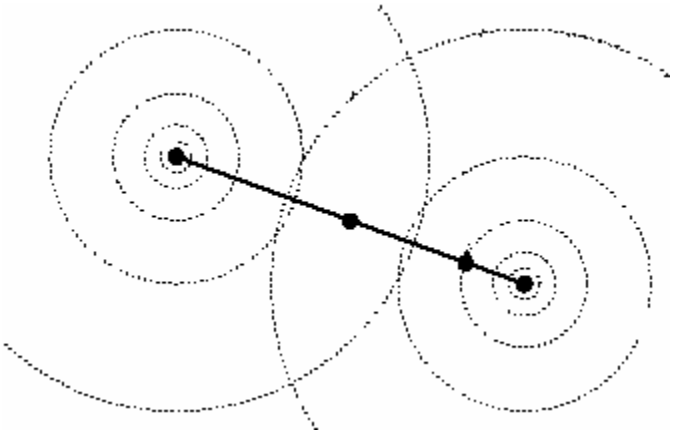
- inicializálunk egy FIFO listát a birtokolt éleknek és a kisszögű háromszögeknek
- amíg nem üres az élek listája osztjuk a soron következő élet
- ha létezik kisszögű háromszög, akkor azt osztjuk. Ha ezáltal élek birtokolttá válnának, akkor nem osztjuk a háromszöget, hanem a megfelelő éleket betesszük a listába és folytatjuk az előbbi lépéstől.
- addig ismételjük az utolsó két lépést, míg elfogynak a kisszögű háromszögek.

Ruppert bebizonyította, hogy az fent vázolt algoritmus befejeződik bármely 90 foknál nagyobb bemenet szögű és 20,7 foknál kisebb korlátra. Természetesen a gyakorlatban a bemenet szögek nagyon gyakran kisebbek mint 90 fok. Az algoritmusnak gondot okozhatnak tehát a kis szögek. Erre a problémára ugyancsak Ruppert hozott elméleti megoldást sarok levágásos pajzs módszerrel.



A pqr szög mértéke kisebb mint 45 fok és ezért birtokoltság egyik élről a másikra ugrik az élosztás során

A fenti ábrán tisztán látható miért is okoz gondot a kis bemeneti szög. A gyakorlati implementációkban egy sokkal egyszerűbb módszert alkalmaznak, nem pedig a Ruppert sarok levágásos módszerét, ami nagy implementációs nehézségeket okozna és sokszor fölöslegesen sok háromszöget generálna. Ez az egyszerűbb módszer végül is szimulálja az elméletileg javasoltat. Egy módosított élosztást alkalmazunk.



Azt az élet, amelynek egyik és csakis egyik csúcsa bemenet csúcs nem a középpontban, hanem a felezőponthoz legközelebb eső imaginárius kör metszeténél fogjuk osztani

Amint a fenti ábrán is látható a kritikus helyzetekben a körök fogják meghatározni az elosztási pontot. S mivel ezt a pontot diszkréten ugyanannyinak választjuk meg, ezért nem fog a végtelenségig ismétlődni a birtokolás, hiszen egy adott pillanatban a keletkezett háromszög akár egyenlő szárú is lehet. A képlet ami megadja a kör sugarát feltételezi egy konstans *D* rögzítését. A sugár: $d' = D \cdot 2^{k'}$, ahol; ahol *d* az él hossza és *k'* a $k = \log_2 \frac{d}{2 \cdot D}$ lekerekített egész értéke. A *D* értékének 0,01-et javasolt Ruppert.

Még mindig kérdés, hogy ha már tudjuk, hogy melyik Steiner pontot fogjuk betenni háromszögelésünkbe, akkor hogyan fogjuk beépíteni. Természetesen Delaunay tulajdonságoknak megfelelően. Erre egy megfelelő algoritmus lenne az inkrementális Delaunay építési algoritmus. Ez az algoritmus lokalizálja azt a háromszöget, amelyben található az új pont és újraköti az éleket, megváltoztatva a háromszögeket. Két módszer is ismert, a Lawson, illetve a Bowyer-Watson. Az első, amelyet mi is használunk, a lokalizált háromszöget felosztja az új pont szerint három új háromszögre, majd rekurzívan az összes módosított háromszög oldalait leellenőrzi, hogy Delaunay tulajdonságúak-e, és ha nem akkor “flip”-pel, és újból ellenőrzi az új háromszögeket, és így tovább ... Természetesen a bemenet éleket nem “flip”-pelhetjük. Ha pedig élosztás útján pont bemenet élre kerül a Steiner pont, akkor csak két új háromszöget szerkesszünk, s csak ilyen irányba kell ellenőrizzük a Delaunay tulajdonságot. A Delaunay tulajdonságot a háromszög köré írt körrel ellenőrizzük. Általában egy ilyen Steiner pont bekötése esetén a háromszög lokalizációja emészti fel a legtöbb időt. Az implementáció során egy gyorsító hálós adatstrukturát használunk ennek a felgyorsítására.

Ha egy háromszögosztás birtokolna éleket, akkor az algoritmus során az éleket kell osszuk. Ez a művelet feltételezi, hogy tudjunk kitörölni is a háromszögelésből, ami a következő módon történik: kitöröljük az összes élet, ami a csúcsba érkezik/indul, s az így keletkezett lyukat újraháromszögeljük (az eddig is használt poligon-lánc Delaunay háromszögeléssel).

4.7. Implementáció és adat-struktúrák

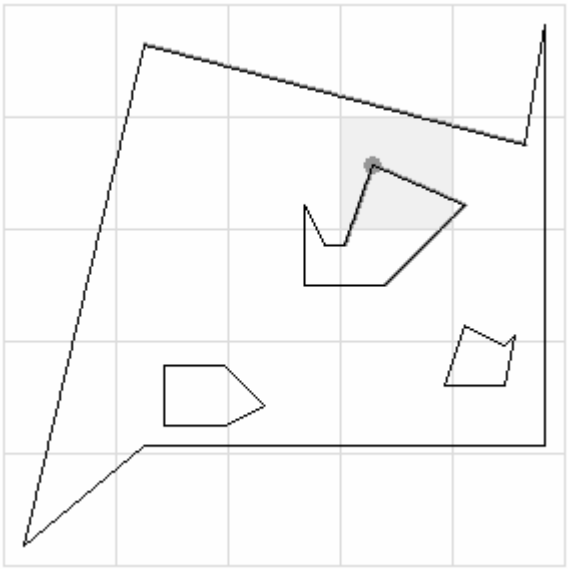
Miután mérlegeltük a megoldás módszereket egyik legfontosabb kérdés, az adatstruktúrák helyes megválasztása. A programunk a ”duplán láncolt él lista” (doubly connected edge list) módosított változatát és egy gyorsító egyenletes négyzetháló (uniform grid) struktúrát használ.

Az első három listát tartalmaz: az élekkel, a csúcsokkal és a háromszögekkel. Ezek közül az élek a legfontosabbak, pontosabban az irányított félélek (halfedge), melyek segítségével elérhetőek a végpont csúcsok, az érintő háromszögek, illetve ezek élei. Természetesen az irányítás a topológia egyértelműsége céljából szükséges.

A csúcsok tartalmazzák a koordinátákat, illetve egy csúcsból kiindulva is elérhetünk egy élet, ami ebből indul ki, s innen, már mindent elérünk a fent említettek alapján.

A háromszögek is egy élre mutatnak, ahonnan akármit lekérdezhetünk, de tartalmaznak olyan információkat, amire többször is szükség lesz: a háromszög köré írt kör középpontja, sugara, illetve a legkisebb szög.

A másik fontos struktúránk a négyzetháló. Ezáltal felosztjuk a tartomány által lefödött téglalap alakú területet kis téglalapokra, amelyeket voxelnek nevezünk. Ezek a voxelek az összes olyan élre illetve csúcsra fognak mutatni, amely metszi az illető voxelt. Nagyon hasznosak lesznek a láthatóság, Delaunay csúcs, illetve a lokalizálás műveleteknél. Bármely pontnak, ami a mi tartományunkban, vagy környékén van biztosítanak egy megközelítést. Például a láthatóság tesztelése során nem szükséges csak bizonyos voxel-ekben található élekkel való metszetet ellenőrizni, ami nagyon leszűkíti a kört. Igy tudjuk szabályozni a Delaunay csúcs meghatározásánál, hogy milyen zónából választunk lehetséges csúcsokat.



A négyzetháló adatstruktúra – a sötét voxelhez a vastag élek tartóznak, illetve a megerősített csúcs

A háromszögelésben való lokalizálás a következő képen működik. Egy kiinduló élből fordulunk, amíg a pont irányába álltunk és akkor kezdünk közelíteni egészen addig,

míg elérjük a pont háromszögét. Ezeknek a műveleteknek a száma nagyon lecsökkenhet, ha eredetileg egy közeli élből indulunk, amit egy voxel segítségével megkapunk.

Ahhoz, hogy ezeket a listákat felépítsük szükséges egy megfelelő módszer a voxelek meghatározására, amelyekben egy szakasz található. Az Amatides és Woo által publikált algoritmus grafikában alkalmazott. Ott is nagyon fontos a sebesség, mi is ezt használtuk.

Tehát a lehető legmegfelelőbb az objektum orientált implementálás volt, mert minden egységet (csúcs, él, háromszög) jellemzett bizonyos adat, és leírt bizonyos viselkedés. Például az élek esetén a skalárszorzat, vagy hogy egy csúcs tőle balra, vagy jobbra fekszik ...

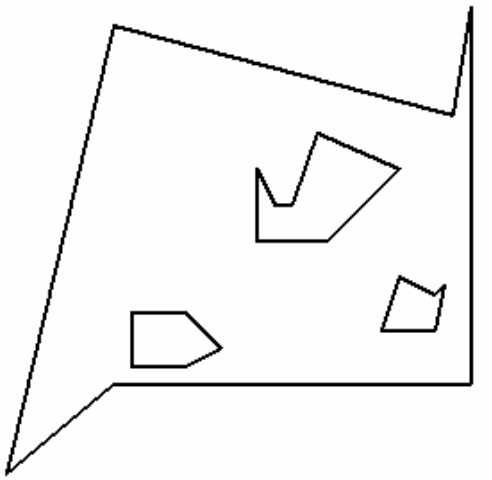
Ugyancsak említésre érdemes, hogy sajnos a valóságban nagyon gyakran degenerált helyzetekkel is találkozunk. Minden algoritmus, módszer, amit a fentiekben tárgyaltunk, kisebb-nagyobb módosításokkal működik degenerált esetekre is. Implementációnkba a lehető legtöbb ilyen esetet próbáltuk lefedni. Egy másik hibaforrást jelenthetnek a számítási hibák. Például egy számítás eredményeként egy csúcs egy éltől balra kerülhet, noha a másik számítás, ami az irányított háromszög területét adná, az pontosan az ellenkezőjét tanúsítaná. Ezek kikerülésére jobban kifejlesztett algoritmusok adnak részleges megoldást, illetve az “exact arithmetic computing”, ami egy bizonyos pontosságot biztosít, de sokkal időigényesebb. Mi “epszilonos környezetes” technikát használunk, ahol az egyenlőség nem pontos egyenlőséget, hanem epszilonos környezetet jelent.

4.8. Fejlesztési lehetőségek, módosítások

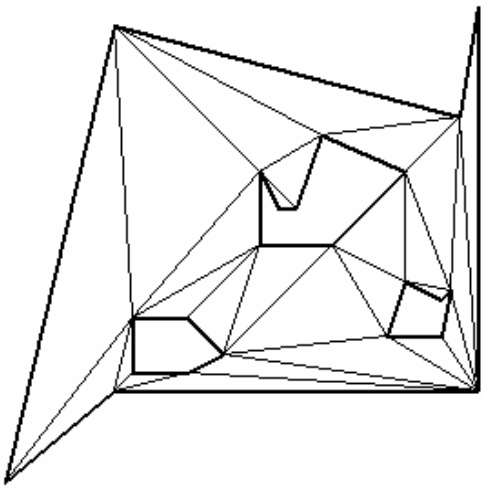
Ezzel az algoritmus egy nagyon nagy kontrollt ad a minimum szög felett. Ez nagyon fontos a konvergencia szempontjából, de a hiba szempontjából a háromszög méret is számottevő. Sokszor az alkalmazás során a tanulmányozott tartomány egy bizonyos területe kielemelten fontos ezért jó lenne arra a területre kisebb háromszögeket generálni, finomabb felosztást létrehozni. Lehetne módosítani az osztási feltételeket a háromszög köré írt kör sugara szempontjából is (ami a háromszög nagyságával arányos, ha már megfelelő a minimum szög).

Egy méret- és szöggarantált algoritmust Chew fejlesztett ki. Ez az algoritmus minden szöget garantáltan 30 és 120 között tart és a háromszögek élei is korlátoltak, mindegyik háromszög köré írt kör sugara egy bemenet argumentum *h* és duplája között lévén. Ennek a módszernek nagy hátránya az, hogy egyenletes mértű háromszögeket generál a tartomány teljes részén, ami nagyon sok számoláshoz vezet a véges elem analízis további lépései során.

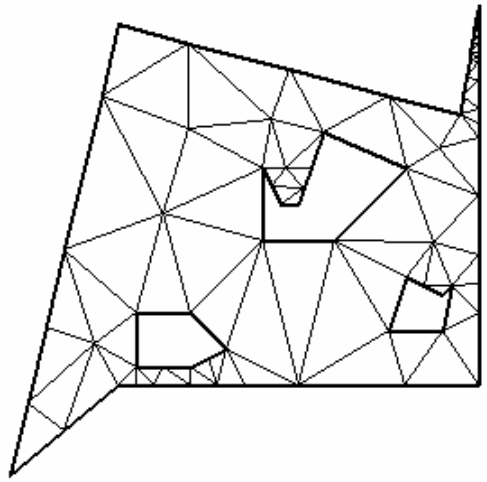
Három dimenzióban mivel nem érvényesek a minimál szög maximizációs tulajdonságok a Deaunay háromszögelés esetén, más megközelítés szükséges, s egyik nagyon eltejedt módszer az ”advancing front approach”, ami során a Delaunay módszerrel feloszott felületre fokozatosan szerkesztük a tetraédereket, míg megtöltjük a teljes tartományt.



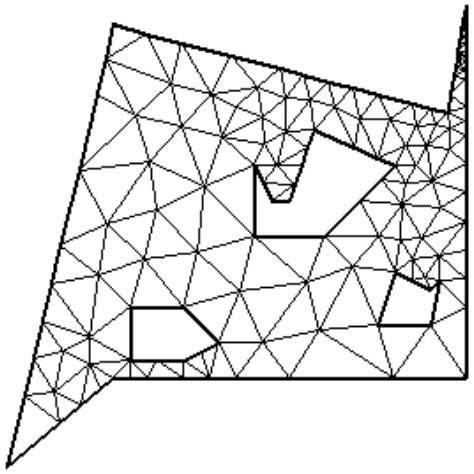
Bemenet



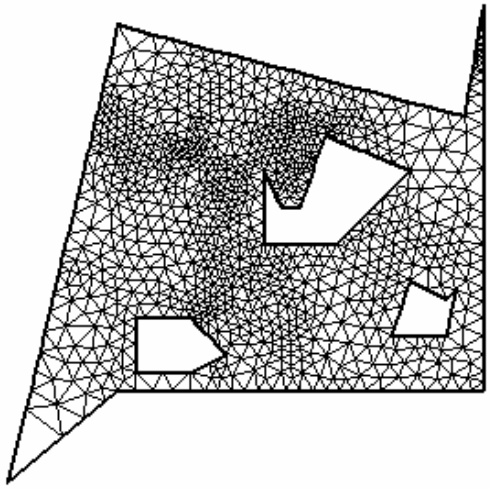
Kötött Delaunay háromszögelés (27 háromszög)



Finomítás után – min. 25 fok (91 háromszög)



Finomítás után – min. 32 fok (206 háromszög)



Finomítás után – min. 34 fok (1770 háromszög)

5. Következtetések

Régebb az ilyen célú alkalmazásokat Fortran-ba írták. Ez célszerű volt, mert az akkori számítógépeken performensen futott. Az ilyen programoknak átláthatósága minimális és továbbfejlesztése nehézkes. Ezért manapság erősödik az a tendencia, hogy ezt a problémát objektum orientált módon közelítsék meg, ami által kiküszöbölhetőek a fent említett hátrányok. Természetesen nőnek az erőforrás igények, de a mai számítógépek mellett ez megengedhető.

Mi is hasonlóan gondolkoztunk és az objektum orientált fejlesztői környezetek közül a Java mellé tettük le voksunkat. A Javanak, mint tudjuk, a legnagyobb előnye a portabilitás, és számunkra adatstruktúrai vizibilitást is nyújtott. A hátránya a sebességben nyilvánul meg, valamint abban, hogy nem nyújt teljes kontrollt a program fölött. Tehát célszerű lenne egy másik gyorsabb, de szintén objektum orientált környezetre áttérni.

A programunk keretében használt felosztás generátornak több felhasználói paramétere van. Az első, a négyzetháló méretének megválasztása kizárólag a felosztás generálásának sebességét befolyásolja. Javasolt legalább 15x15-ot használni, de fontos tudni, hogy a méret növelése nem föltétlenül növeli a sebességet, egy adott határ fölött csak a memóriaigényt. A másik paraméter, a minimál szög, sokkal fontosabb. Ez direkt befolyásolja a háromszögek számát és ezáltal a további számolások idejét. Tesztelés során arra a következtetésre jutottunk, hogy a 30 foknál nagyobb minimál szögeknél rohamosan nő a háromszögek száma ezért általában nem célszerű ilyen felosztást használni.

A három dimenzióba való áttérés elsősorban egy új felosztás generálót igényelne. De az adatok mennyisége nagyon megnőne, ezért megoldást kellene találni a sebesség növelésére. Jó megoldás a párhuzamosítás, amely a probléma természetéből adódóan lehetséges és célszerű.

Könyvészet

[1] – John Amanatides, Andrew Wo: A fast voxel traversal algorithm for ray tracing; Eurographics ’87, pages 3-10, North-Holand, August 1987

[2] – Mark de Berg, Marc van Kreveld, Mark Overmars, and Ottfried Schwarzkopf: Computational Geometry: Algorithms and Applications. Second edition, Springer-Verlag, 2000

[3] – Marshall Bern and David Eppstein: Mesh Generation and Optimal Triangulation. Computing in Euclidean Geometry, Lecture Notes Series on Computing, volume 1; World Scientific, Singapore, 1992

[4] – Marshall Bern, Paul Plassmann: Mesh generation, 2000

[5] – Paul Blaga: Geometrie computațională. Notițe de curs; Cluj-Napoca, 2002

[6] – Glenn Eguchi, Massachusetts Institute of Technology: 6.838J/4.214J - Geometric Computation, Lecture Notes

[7] – Paul Heckbert: Quad-Edge Data Structure and Library; Carnegie Mellon University, Revision 1: 14 Feb. 2001

[8] – Professor Peter Hunter: FEM/BEM Notes; Department of Engineering Science; The University of Auckland New Zealand February 19, 2002

[9] – Reinhard Klein: Construction of the constrained Delaunay triangulation of a polygonal domain, 1996

[10] – Fumei Lam, Massachusetts Institute of Technology: Representig Polyhedra, 6.838J/4.214J - Geometric Computation

[11] – Dani Lischinski, Cornell University, Ithaca: Incremental Delaunay Triangulation; New York, 1993 Academic Press

[12] – masterand Tania Angelica Lungu: Lucrare de Disertație. Metoda elementului finit pentru ecuații cu derivate parțiale de tip parabolic; Facultatea de Matematică şi Informatică Universitatea „Babeş-Bolyai” Cluj-Napoca, 2000

[13] – Gheorghe Micula, Sanda Micula: Handbook of Splines; Dordrecht, Netherlands: Kluwer, 1999

[14] – Steven Owen: An introduction to unstructured mesh generation, Part I: Meshing algorithms; Sandia National Laboratories, 2001

[15] – Jim Ruppert: A Delaunay refinement algorithm for quality 2-dimensional mesh generation; Journal of Algorithms 18(3):545-585, May 1995

[16] – Jonathan Richard Shewchuk: Lecture Notes on Delaunay Mesh; Department of Electrical Engeneering and Computer Science, University of California at Berkley, September 1999

[17] – Jonathan Richard Shewchuk: Triangle: Engeneering a 2D Quality Mesh Generator and Delaunay Triangulator; University of California at Berkley, May 1996

[18] – NURB Curves: A Guide for the Uninitiated – The Apple Technical Journal