Gyors és pontos háromszögelő algoritmus OpenLayers 3-hoz

Farkas G.

Pécsi Tudományegyetem Természettudományi Kar Földtudományok Doktori Iskola

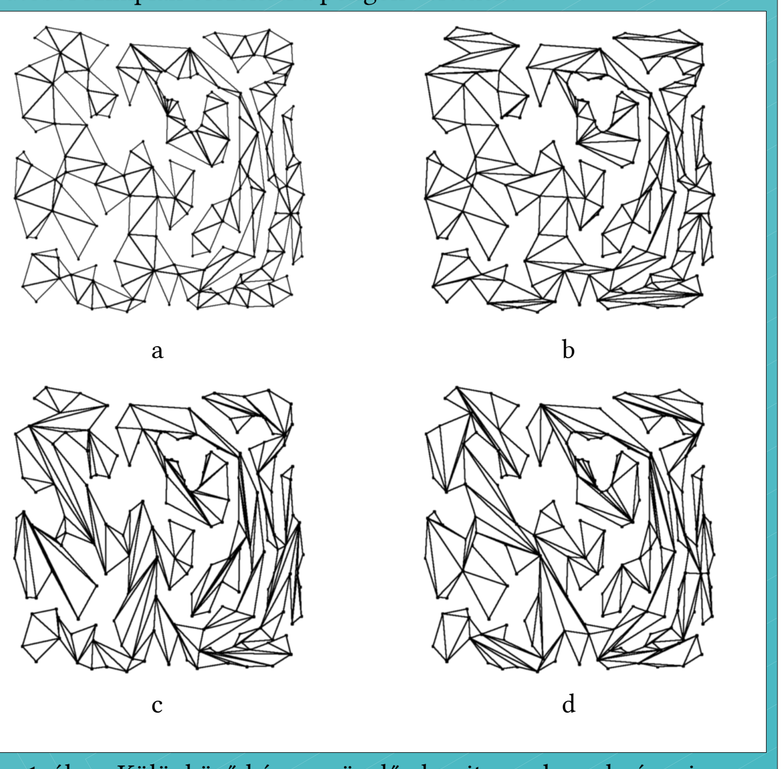
OpenLayers 3

Az OpenLayers 3 egy általános könyvtár, mely interaktív webes térképek készítésére íródott. A könyvtár robusztus, nagyon sok beépített GIS funkcióval rendelkezik, nem korlátozza le magát néhány vetületi rendszerre, könnyen fejleszthető és implementálható, röviden egy nagyszerű alapot biztosít fejlett webes GIS rendszerek fejlesztésére. Nagy hátránya azonban, hogy Canvas technológiával dolgozik, ami szoftveresen gyorsított, így nagyobb adatmennyiségek kirajzolásánál a teljesítménye gyors ütemben romlik. Ez a tulajdonsága már önmagában alkalmatlanná teszi egy általános Web GIS rendszer fejlesztésére, azonban könnyen orvosolható egy teljes értékű vektoros WebGL megjelenítővel. Ennek a megjelenítőnek szerves része egy pontos és egyben gyors háromszögelő algoritmus.

A háromszögelésről általánosságban

A háromszögelés egy igen régre visszanyúló probléma a számítástudományban. Célja, hogy egy n oldalú poligont (n-gont) – ha lehetséges új vertexek (Steiner pontok) bevonása nélkül – háromszögekre bontsunk. Ennek célja a poligonok feldolgozása egy olyan program számára, mely csak háromszögeken tud operálni. A legtipikusabb alkalmazás ilyen tulajdonsággal a grafikus megjelenítés. Videókártyáink bár bonyolult mátrix– és vektorműveleteket képesek páratlan sebességgel végrehajtani, amikor kirajzolásról van szó, csupán három alapvető formát (primitívet) támogatnak. Ezek a pont (valójában négyzet), a vonal (nem vonallánc), és a háromszög.

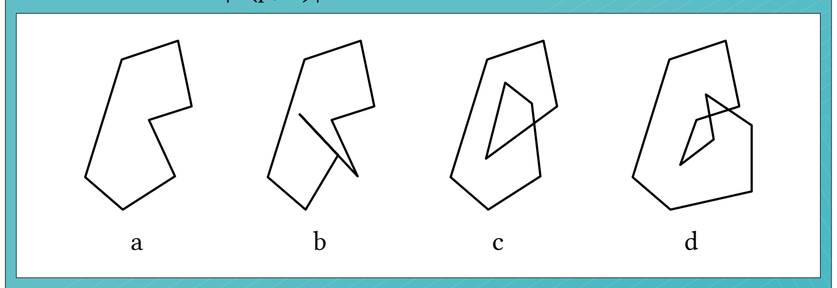
Háromszögelő algoritmusból sok fajta létezik. Geoinformatikusok fülében legismerősebben talán a Delaunay háromszögelés cseng, melynek lényege egy P poligonból egy olyan $H(P)$ háromszögelést készíteni, ahol $H(P)$ minden háromszöge egy Delaunay háromszög. Delaunay háromszög pedig csak olyan $\triangle P_i P_j P_k$ lehet, aminek a köré írt köre nem tartalmazza P más egyéb pontját [4]. Bár ez a háromszögelési forma elegáns, és a legjobb minőségű háromszögelésre felkészíthető (minimum és átlagos belső szögek a legnagyobbak), sokkal inkább alkalmas TIN modell készítésére [6], mintsem valós idejű megjelenítésre. Természetesen ebből a módszerből is született használható, $O(n \log n)$ átlagos futásidejű algoritmus [8], azonban sebessége még így is elmarad a direkt erre a célra fejlesztett algoritmusokkal szemben [2]. A megjelenítési célzattal írt algoritmusok a naiv, $O(n^3)$ futásidőt először $O(n^2)$-re, majd $O(n \log n)$-re, végül Chazelle munkásságának köszönhetően [1] lineáris, $O(n)$ futásidőre szorították vissza. Az utóbbiról érdemes megemlíteni, hogy elméleti jelentősége nagyobb, ugyanis bonyolultsága révén a gyakorlatban nehezen alkalmazható. Így a további kiemelendő – $O(n \log n)$ komplexitású – algoritmusok Seidel trapezoid dekompozíción alapuló algoritmusa [7], és a népszerű „fülvágó” (ear clipping) algoritmus. Míg az előbbi trapezoidokra bontja az adott poligont annak vertexein áthaladó vízszintes seprő vonalak segítségével, és azoknak átlóit használja a háromszögeléshez, az utóbbi szimplán összeköti a poligon vertexeit.



1. ábra: Különböző háromszögelő algoritmusok eredményei. Shewchuk Delaunay háromszögelése (a), Seidel trapezoid dekompozíciója (b), szekvenciális fülvágás (c), véletlenszerű fülvágás (d). Forrás: Held (2016) https://www.cosy.sbg.ac.at/~held/projects/triang/triang.html.

Közös problémája a fent említett algoritmusoknak, hogy bemenetként egyszerű poligonokat (Jordan poligonokat [5]) feltételeznek. Ez a feltétel számítógépes játékoknál egyértelmű és könnyen teljesíthető, azonban a valós adatok, így a CAD/CAM és GIS szoftverek által kirajzolandó poligonok sok esetben nem teljesítik [2]. A kidolgozandó algoritmusnak ezáltal kezelnie kell a nem egyszerű poligonokat is (2. ábra). A poligonok topológiailag kiválóan jellemezhetőek a csavarodási számmal (winding number, ω). A csavarodási szám lényege, hogy megfigyelőt állítunk a sík p pontjára, aki sorban megfigyeli P poligon vertexeit. A végeredmény (p csavarodási száma, azaz $\omega(p, P)$) megmutatja, hogy a megfigyelőnek hányszor kellett 360°-os fordulatot tennie a poligon végigtekintéséhez, amennyiben $P_i \neq p$ [3]. A Jordan poligonok közös jellemzője, hogy a síkot két összefüggő részre particionálják, valamint ha p P-n kívül helyezkedik el, $\omega(p, P) = 0$, míg ha azon belül, $|\omega(p, P)| = 1$ [2].

A feltételeket nem teljesítő poligonok lehetnek degeneráltak, önmetszők, vagy önfedők. Degeneráltnak hívjuk azt a poligont, aminek kettő vagy több éle kollineáris. Az önmetsző poligonok élei legalább egy helyen metszik egymást. Az önfedő poligonoknál az önfedés területén $|\omega(p, P)| \geq 2$.



2. ábra: Poligonok topológiai fajtái. Jordan poligon (a), degenerált poligon (b), önmetsző poligon (c), önfedő poligon (d).

Létező könyvtárak

Annak ellenére, hogy az előbb említett háromszögelő algoritmusok referenciakönyvtárai és alkalmazásai mind alacsony szintű nyelven (C dialektusokban) íródtak, JavaScript háromszögelési könyvtárak is léteznek. Egyik legismertebb könyvtár a libtess, ami az Eric Veach nevéhez kötődő OpenGL háromszögelő könyvtár átirata. Ez a könyvtár egy Seidel variánst használ, azaz seprő vonalakkal monoton poligonokra (trapezoidokra) bontja a bemeneti poligont [9]. Előnye, hogy helyes háromszögelést készít topológiától függetlenül, valamint nagyon jól paraméterezhető. Hátránya, hogy egy algoritmus, ami alacsony szintű nyelven megfelelő sebességgel fut, magasabb szintű nyelven már nem biztos. Így nagyobb adatmennyiség valós idejű háromszögelésére nehezen alkalmazható.

Jelentős könyvtár a 2015-ben létrehozott earcut, mely a Mapbox céghez, azon belül is Vladimir Agafonkin nevéhez köthető. A Mapbox GL megjelenítőhöz íródott, célja a szanitált adat páratanul gyors háromszögelése. Alapját Martin Held FIST (Fast Industrial Strength Triangulation) fülvágó algoritmusa képezi. Előnye a sebessége, és a további vertexek beillesztésének mellőzése. Így egyszerű poliédereket is helyesen háromszögel (a Cesium is ezt a könyvtárat használja). Hátránya, hogy csak Jordan poligonokat, lyukas poligonokat és önfedő poligonokat képes hibátlanul háromszögelni, az önmetszést nem tudja helyesen kezelni.

Az algoritmus követelményrendszere a bevezetésben leírt két kulcsszó mentén fogható meg: pontos és gyors. Mivel a már létező, potenciális könyvtárak ezen feltételek közül egyet-egyet páratlan módon teljesítenek, belátható, hogy kompromisszumokra volt szükség az új algoritmus körvonalazásánál. A gyorsaság biztosításának érdekében alapként az earcut által is használt FIST szolgált.

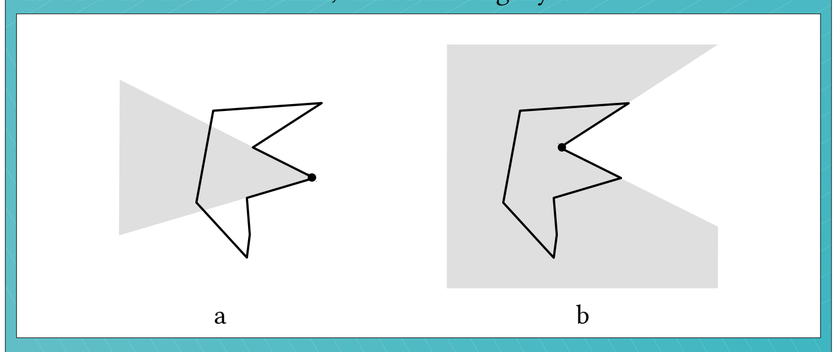
Adatstruktúrák, szabályok

A fülvágó algoritmusok alapvető adatstruktúrája a láncolt lista. Ebből is a kétszeresen láncolt körkörös lista, melynek elemeit a poligon vertexei képezik. Elemenként minimálisan eltárolandó az adott vertex x és y koordinátája. Ezeken felül az algoritmus tárolja az adott vertex indexét (azt a referenciaszámot, amivel a videokártyára megkapja kirajzolásra), valamint a konvexitását. Konvexitás alatt értendő a szomszédos vertexekkel bezárt szöge. Ha $\angle v_{i-1} v_i v_{i+1}$ 180°-nál kisebb, v_i konvex, amennyiben nagyobb, v_i reflex.

A fülvágás lényege, hogy olyan vertexeket találjunk, melyek a szomszédos vertexekkel a poligon egy fülét alkotják. Ezeket a vertexeket sorban eltávolítjuk, a háromszögelési információkat pedig eltároljuk. Ahhoz, hogy egy vertex a szomszédaival fület alkothasson, Held a következő két szabályrendszert állította fel [2]:

- CE1: P poligon egymást követő v_{i-1}, v_i, v_{i+1} vertexei fület alkotnak, ha
  - v_i konvex;
  - A $v_{i-1} v_{i+1}$ szegmens nem metszi P-t csak v_{i-1}-ben és v_{i+1}-ben;
  - $v_{i-1} \in K(v_i v_{i+1} v_{i+2})$ és $v_{i+1} \in K(v_{i-2} v_{i-1} v_i)$.
- CE2: P poligon egymást követő v_{i-1}, v_i, v_{i+1} vertexei fület alkotnak, ha
  - v_i konvex;
  - a $\triangle v_{i-1} v_i v_{i+1}$ nem tartalmazza P más egyéb reflex vertexét.

ahol $K(v_{i-1} v_i v_{i+1})$ egy kúp, mely a sík azon pontjait tartalmazza, amelyek $v_i v_{i-1}$ vonaltól szigorúan jobbra vannak, $v_i v_{i+1}$ vonaltól pedig szigorúan balra. Ha v_i reflex, a kúp ennek a komplementere (3. ábra). Ez a szabály azt hivatott ellenőrizni, hogy az átló a poligon belső avagy külső átlója. A CE2-es szabályrendszer csak egyszerű poligonoknál alkalmazható, míg a CE1-es szabályrendszerhez metszéseket kell ellenőrizni, ami számításigényes.

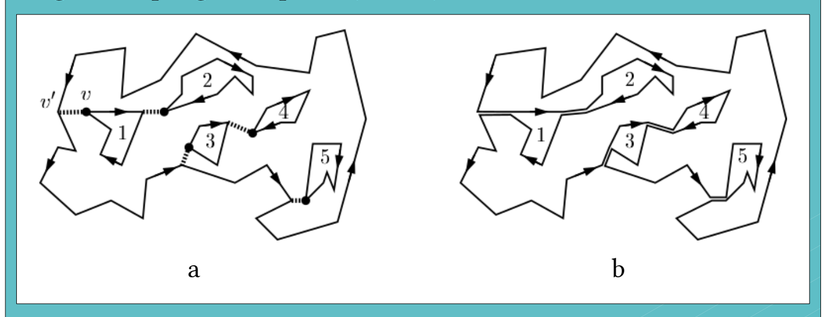


3. ábra: $K(v_{i-1} v_i v_{i+1})$, ahol v_i konvex (a), $K(v_{i-1} v_i v_{i+1})$, ahol v_i reflex (b). A poligonok orientációja jelen esetben az óramutató járásával ellentétes.

A második adatstruktúra egy geometriai index, ami a térbeli lekérdezések felgyorsításához szükséges. Az algoritmus két okból is R-fa adatstruktúrát használ: metszéseket kell ellenőrizni, és az OpenLayers 3 már régóta használja. Az R-fa egy olyan BVH (Bounding Volume Hierarchy), amiben az entitások tengelyorientált határolódoboza kerül hierarchikus eltárolásra. A metszéspontok egyszerű meghatározása végett az R-fa szegmenseket tartalmaz, míg a szinkronban tartás megkönnyítése érdekében a láncolt lista elemei is a poligon egyes szegmenseit reprezentálják.

Az algoritmus

Az algoritmus bemenete egy adott poligon koordinátái, ebből kell háromszögelési információt generálnia. Első lépésként egy láncolt listába helyezi a koordinátákat az óramutató járásával megfelelő irányban. Amennyiben a poligon tartalmaz lyukakat, azokat külön kapja meg, az óramutató járásával ellenkező irányba orientált láncolt listákat készít belőlük, sorba rendezi őket szélső x koordináta szerint, majd a külső gyűrűbe illeszti azokat egy degenerált poligont képezve (4. ábra).



4. ábra: Kontúr hidak a külső gyűrű és a lyukak között (a), keletkező degenerált poligon (b). Forrás: Held (2011).

Amint készen vannak a poligon adatstruktúrái, megkezdődik a fülvágás. Az algoritmus jutalmazó rendszert használ, azaz minél kevesebb és kisebb topológiai hiba van a bemeneti adatban, annál gyorsabban fut le. Eltérően Held és Agafonkin algoritmusaitól, amik egyszerre egy szabályrendszert alkalmaznak, itt a CE1 és a CE2 is alkalmazásra kerül, attól függően, hogy tartalmaz-e a poligon önmetszést vagy önfedést. Ennek kiderítésére az algoritmus önmetszéseket keres már a háromszögelés elején, ami költséges művelet, azonban egyszerű poligonoknál megtérül. A fülvágás előtt egy külön rutin a vertexeket osztályozza konvexitásuk szerint és az egy vonalba eső pontokat törli a listából.

A háromszögelő rutin ezután megkezdi a fülvágást. Amennyiben a poligon egyszerű, levág minden fület, majd újraosztályoz és folytatja a vágást. Ha így sem sikerül feldolgozni a poligont, akkor nagy valószínűséggel a poligon egy vagy több pontja érinti a poligon valamely szegmensét, ezeket a pontokat feloldja. Ha a poligon tartalmaz önmetszést vagy önfedést, szintén levágja az érvényes füleket, újraosztályoz, majd feloldja a helyi önmetszéseket. Helyi önmetszésről beszélünk, amikor P poligon $P_i P_{i+1}$ szegmense metszi $P_{i+2} P_{i+3}$ szegmensét. Ha az önmetszések így sem feloldhatóak, elvágja a poligont az első önmetszésnél és az így keletkező két poligont külön háromszögeli.

Ahhoz, hogy az algoritmus helyes háromszögelést adjon, két feltételnek kell teljesülnie. Az új vertexek beillesztése csak akkor lehetséges, ha a megjelenítés két dimenziós. Ez a feltétel az OpenLayers 3 természetéből adódóan teljesül, ugyanis ez egy két dimenziós megjelenítő könyvtár. A másik feltétel az, hogy ha egy poligon önmagába visszametsz, az minden esetben önfedés. Ez a feltétel azzal teljesíthető, hogy az önmetsző és önfedő poligonok a GIS-ben általában topológiailag helytelenek. Kivételt képeznek a három dimenziós poliéderek két dimenziós vetületei, ahol a vetítés eredményének topológiai helyessége nem az alacsony szintű háromszögelő algoritmus felelőssége.

Irodalomjegyzék

[1] Chazelle, B. Triangulating a Simple Polygon in Linear Time. *Discrete and Computational Geometry 6*, (1991), 485 – 524.
[2] Held, M. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica 30*, 4 (2011), 563 – 596.
[3] Hormann, K. és Agathos, A. The point in polygon problem for arbitrary polygons. *Computational Geometry 20*, (2001), 131 – 144.
[4] Lee, D. T. és Schachter, B. J. Two Algorithms for Constructing a Delaunay Triangulation. *International Journal of Computer and Information Sciences 9*, 3 (1980), 219 – 242.
[5] Meisters, G. H. Polygons Have Ears. *The American Mathematical Monthly 82*, 6 (1975), 648 – 651.
[6] Peucker, T. K., Fowler, R. J., Little, J. J. és Mark, D. M. The triangulated irregular network. In *American Society of Photogrammetry Proceedings of the Digital Terrain Models Symposium* (1978), 96 – 103.
[7] Seidel, R. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications 1*, (1991), 51 – 64.
[8] Shewchuk, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First ACM Workshop on Applied Computational Geometry* (1996), 124 – 133.
[9] SGI SI GLU Algorithm Outline. https://cgit.freedesktop.org/mesa/glu/tree/src/libtess/alg-outline (2001).

Készült a FOSS4GHU 2016, 5. nyílt forráskódú térinformatikai munkaértekezletre. 2016. 11. 25. BME Általános- és Felsőgeodézia tanszék, Budapest.