Vektor geometriai és grafikai megjelenítés a térinformációs rendszerekben

Huszka Csaba Zsolt

Konzulensek:

Dr. Szabó György
Wirth Ervin

Budapest, 2015.11.02.

1 TARTALOMJEGYZÉK

2 BEVEZETÉS

Az analóg korszak geometria kifejezése a tárolást illetve grafikai megjelenítést egyszerre jelentette. A geometriák létrehozásának voltak bizonyos korlátai, mint a pontosság, kezelhetőség, tárolás, munkaidő, ám bizonyos szerkesztői szabványok betartásával e hátrányok java kezelhetővé vált.

A digitális világ megjelenésekor - a korlátok feloldásakor - nyilvánultak meg az újkorban rejlő lehetőségek, amelyek az eddigi határok miatt elérhetetlenek voltak. Ezen lehetőségek munkaidőben, tárolásban és az információk lekérdezésében nyilvánultak meg leginkább.

A tárolással, hordozással kapcsolatos nehézségek megoldódni látszódtak -, ez egyértelműen a digitalizálást dicsérte. Viszont az analóg időkben az információk kiolvasását segítő szabályok továbbra is megmaradtak és átültetésre kerültek a digitális rendszerekbe. Ennek a lépésnek a megtétele bár ésszerűnek tűnik, azonban a későbbiekben ez komoly problémaforrásnak mutatkozik meg. Ami az embernek triviális és könnyen felismerhető, addig az egy algoritmusnak komoly kihívás tud lenni.

A kézi rajzolásnál a grafikai megjelenítés triviálisan történik számos szabály betartása, illetve jelkulcsok használata mellett. Míg csak a rajzeszközt cseréljük le egérre, a gond nem jelentkezik. Az algoritmusoknál - amelyek meglétét a felhasználói igények indokolják -, jelenik meg a probléma, ugyanis itt a tárolt geometriai adat a vektoros grafikai megjelenítéssel nem egyezik meg.

Olyan geometriai alakzatok megjelenítésekor figyelhető ez meg, amelyeknek nincs szélességi paramétere, viszont megjelenítése indokolt, ilyen a pont és a vonal.

A gyakorlatban a közműhálózatok bejegyzésénél pontosan tudnunk kell, hogy hol és milyen típusú közművezeték található, ha fejleszteni, korszerűsíteni vagy ellenőrizni akarunk. A közművezeték típusának megkülönböztetése nagyon fontos, mert a puffer zóna nagysága e szerint tud változni.

A dolgozatban nem csak a probléma feloldása a lényeg, hanem a megoldás egyszerű, viszont biztos eljuttatása minél szélesebb felhasználó csoportnak. Egy olyan web server felállítása a cél, amelyet bárki el tud érni az internet segítségével, egy egyszerű böngésző segítségével is akár. Ezen az oldalon a felhasználó el tudja végezni a digitalizálást és a puffer zóna számítását is egyben.

3 TÖRTÉNETI ÁTTEKINTÉS

3.1 TÉRINFORMATIKA

A térinformatika kezdetei a hatvanas évek elejéig nyúlnak vissza, amikor kezdetét vette a mainframe (több száz vagy ezer felhasználó egyidejű kiszolgálására alkalmas) számítógépek elterjedése. A nagyobb kapacitást egyrészt a kormányzati feladatok másrészt a tudományos programok szolgálatába igyekeztek állítani. Felismerték a számítógépek szerepét a földhasznosításra és a földtulajdonra vonatkozó adatok nyilvántartásában. Például a Bureau of the Census (Népszámlálási iroda, USA) a címekhez való hozzáférést automatizálta.

Technikai problémák legyőzése volt a legnehezebb, grafikus adatok számítógépes módszerekkel történő kezelését kellett megoldani, térképeket kellett számítógépeken tárolni. Megjelenik a SYMAP (Synagraphic Mapping), amit egy általános célú térképészeti programként fejlesztették ki Horward Fisher vezetésével. A program igyekezett maximálisan kihasználni a kor által nyújtott hardware kapacitásokat. Egyszerű sík koordináta-rendszerben dolgozott, különböző nagyítási szinteket engedélyezett, továbbá különböző grafikai tulajdonságokat társított bizonyos adatokhoz. Sornyomtatókra volt tervezve a program, akkor az volt a legelterjedtebb. A nyomtatott térkép karakterekből épült fel, tehát egy „képpont” egy karakternek felelt meg. A fekete fehér térképen az árnyalatok változtatását, a helyes karakterek kiválasztásával érték el.

SYMAP alapjául szolgált a GRID-nek, ami egy vektor modellt alkalmazó szoftver volt, majd megszületett az első vektoros GIS (Geographical Information System). Pontok és vonalak kezelése azonban nehézkes volt, mivel olyan objektumokat kell megjeleníteni, amelynek térbeli paraméterei nincsenek. Ezzel a problémával foglalkozik a PostScript programnyelv is.

(Kertész), (Chrisman)

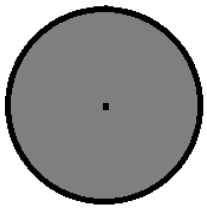
3.2 POSTSCRIPT

A PostScript nyelv a vektor grafika kezelése miatt jött létre 1982-ben. Ez egy úgynevezett oldalleíró nyelv. Koordináta rendszerének egy egysége 1/72 inch, aminek az origója a kép bal alsó sarkában található. Grafikai tipográfiai elemek mindegyikét vektorgrafikus módon, tehát kicsinyíthető, nagyítható, torzítható formában tárolja, tehát eszköz- és felbontás független, mert közvetlenül a nyomtatáskor vagy levilágításkor határozható meg a bittérképpé való átalakítás finomsága, aminek csak a nyomtató vagy a levilágító felbontóképessége szab határt. Sikeres működése miatt nyomdaipari szabvánnyá vált.

(PostScript – Wikipédia, 2015)

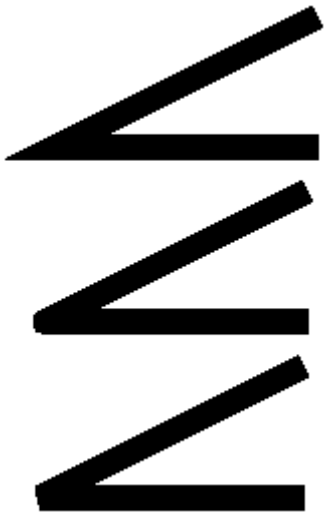
4 GEOMETRIA ÉS A GRAFIKA SZÉTVÁLÁSA

Megfigyelhetjük, hogy egy meghívott pontnak a geometriai adata egy koordináta párból áll, mégis a megjelenítéskor PostScript úgy rendelkezik, hogy bizonyos sugarat ad hozzá. Ez azért szükséges, hogy a kijelzőn meg tudjon jelenni, itt teljesen elkülönül a geometriai adat és a grafikai megjelenítés. Ennek a módszerét felfoghatjuk egy „puffer zónaként” is, ugyanis egy pont köré egy övezetet társítunk.



4.1 ábra: Pont puffer zónája

Egy egyenes megjelenítésekor sem történik másképp a folyamat, ahol a geometriai adat két pont koordinátáiból áll, viszont a megjelenítéshez tudnunk kell a vonal vastagságát, amit szintén puffer zónaként ismét felfoghatunk. A vonalcsatlakozásnál érdekesebb a helyzet, ott több féle forma is meg tud jelenni. Lehet szögletes alakú, félkörökkel bezáródó, vagy háromszögekkel bezáródó alakzat is.



4.2 ábra: Puffer zóna alakzatai vonalcsatlakozásokkal

Geometriai adatok tehát ily módon válnak szét a grafikától. A dolgozatban az elvet más kontextusba helyezzük. Célunk a védősáv szerkesztésének automatizálása, algoritmizálása a jelenlegi szabályok pontos lekövetése. Ehhez viszont ismernünk kell a jelenleg hatályos törvényeket.

5 EMLÍTETT PROBLÉMA MEGJELENÉSE NAPJAINKBAN

5.1 VEZETÉKJOG ADMINISZTRÁLÁSÁNAK AUTOMATIZÁLÁSA, BIZONYOS SZABÁLYOK ISMERTETÉSE.

A vezetékjogról, fő szabályként, a villamosenergiáról szóló törvény rendelkezik. Eszerint közcélú hálózat idegen ingatlanon történő elhelyezésére Hatóság vezetékjogot abban az esetben engedélyezhet, ha az a közcélú hálózat szükséges fejlesztése érdekében indokolt és az ingatlan használatát lényegesen nem akadályozza.

A jogszabály a fentiektől eltérő esetként írja le, hogy az 50 MW feletti teljesítményű erőművek és a megújuló energiaforrást hasznosító erőművek termelői vezetékeinek idegen ingatlanon történő elhelyezésére a beruházó vagy a termelői vezeték engedélyese javára a Hatóság vezetékjogot engedélyezhet, ha az ingatlan használatát az lényegesen nem akadályozza.

Látható, hogy elviekben e főszabály kijelöli a vezetékjog korlátait is, amennyiben a szolgáltatónak (pontosabban a hálózati engedélyesnek) meg kell tudnia jelölni, hogy az a hálózat fejlesztése érdekében szükséges. Fontosabb korlát az ingatlantulajdonosok szempontjából, hogy az ingatlan használatát a vezetékjog lényegesen nem akadályozhatja.

5.2 KÖRNYEZET BEMUTATÁSA, NYELV GYORS ISMERTETŐJE.

Elsőnek a QGIS-ben kezdtem el az algoritmus fejlesztését, amely egy több platformon használható nyílt forráskódú térinformatikai rendszer. Ehhez a programhoz egy beépített Python konzol társul, ami nagyban elősegíti a QGIS fejlődését. Ezt nevezzük PyQGIS-nek, ezzel szabadon tudjuk bővíteni az alkalmazástárját a programnak. Írhatunk vele scripteket és beépülő modulokat egyaránt.

A Python egy általános célú, magas szintű programozási nyelv, melyet Guido van Rossum holland programozó kezdett el fejleszteni 1989 végén, majd hozott nyilvánosságra 1991-ben. Jól áttekinthető, objektum orientált és nagyon népszerű nyelv.

(Python (programozási nyelv), 2015)

5.3 PROBLÉMA FELVÁZOLÁSA

A vezetékjog által fogalmazott szabályokat szeretnénk lekövetni egy algoritmussal, amit Python nyelvben készítettem el. Választásom azért a Python-ra esett, mivel a legnagyobb kereskedelmi – ArcGIS -, és a legnagyobb open-source – QGIS – térinformatika szoftver is ezt támogatja. Bármely más nyelvben is előlehet állítani az algoritmust, ennek az az oka, hogy a QGIS támogatja a GeoJson nevű kiterjesztést, ami a lehető legegyszerűbben ír le egy vektor réteget. Ezt a kiterjesztést minden olyan programnyelvvel meg lehet írni, ami szöveges fájlt tud létrehozni.

Általános esetben a közművezetékek pontos helyét egy vektor rétegen tárolják, amely mérések útján nyert pontos koordinátákkal rendelkeznek, esetünkben EOV (Egységes Országos Vetület) koordinátákkal. Védősávot egy puffer zóna modellezi, ez alapértelmezetten a QGIS is tartalmazza. Számunkra ez nem megfelelő, mert a szabványok pontosan rendelkeznek a távolságokról. Például egy sokszögvonal esetében, a puffer zónának pontosan le kell tudnia követni a formát, tehát az éleknek párhuzamosaknak kell lenniük. Egy hagyományos puffer zóna a töréspontoknál körszegmenses lekerekítéseket használ, így a kapott eredmény nagymértékben eltér a szabványtól.

5.4 Probléma megoldása (fényképek, példák)

Itt abból indulunk ki, hogy megtalálható digitális formában a vezetékek pontos helye, akár shapefile-ban is, vagy bármely QGIS által elérhető formátumban. Valóságban amennyiben ez mégsem igaz, akkor a digitalizálást könnyedén el tudjuk végezni, ugyanis számos egyszerű programmal, akár olyan személy is, aki nem feltétlen ismeri ennek hátterét.

Elsőnek meghívjuk a későbbiekben használni kívánt könyvtárakat, illetve függvényeket.

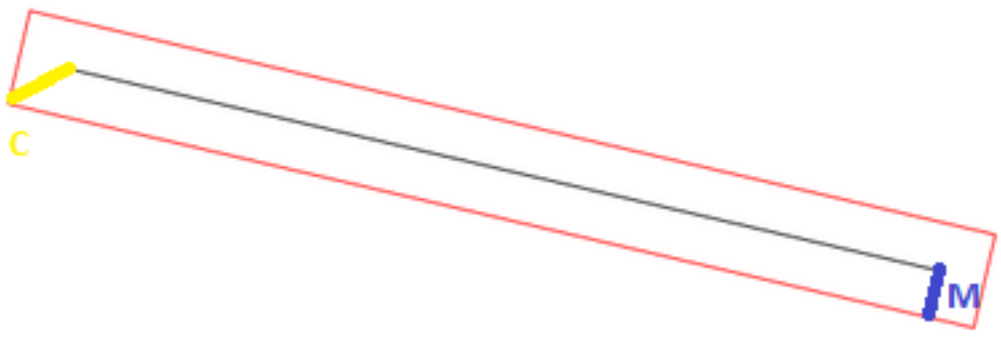
A NumPy egy kiegészítő csomag a Python programozási nyelvhez, bővebb támogatást nyújtva a nagy, többdimenziós tömbök és mátrixok használatára egy nagy magas szintű matematikai függvény könyvtárral. A GDAL (Geospatial Data Abstraction Library) egy könyvtár, ami a raszteres illetve vektoros térbeli adatok olvasására és írására szolgál. Az OGR pedig nyílt forrású könyvtár, ami GIS adatbázisokhoz támogatja a hozzáférést

5.4.1
```
import numpy as np
from math import sin
from math import cos
from math import pi
from math import sqrt
from math import tan
from osgeo import osr
from osgeo import gdal
from osgeo import ogr
```

QGIS-ben a Python konzolban egyszerűen lekérjük az aktív réteg adatait - aktív réteg az, amelyre éppen klickeltünk – majd a réteg adataiból kiválasztjuk a koordinátákat és sokszögvonalként (továbbiakban polyline) értelmezzük őket. A következő módon tudjuk ezt megtenni:

5.4.2
```
get_layer = iface.activeLayer()
if not get_layer.isValid():
print "Hiba a reteg betoltesekor"
iter = get_layer.getFeatures()
for feature in iter:
gLine = feature.geometry()
pontok = gLine.asPolyline()
```

Ezzel megvannak a fő pontok, ami köré kell létrehozni a puffer zónát, ezután definiálunk egy konstans szorzót, amit jelöljünk ’c’-vel, majd tegyük egyenlővé, ’sqrt(2)*m’ – vel. Itt az ’sqrt()’ a gyökvonást jelöli, a zárójelben lévő szám a hatványalap, az ’m’ szintén egy konstans szám, ami a védősáv kiterjedését adja meg, ennek szélessége.



5.1. ábra: C és M paraméterek megjelenése

Következő lépésben üres tömböket hozunk létre a következő módon:

```
x = []
y = []
dx = []
dy = []
angle = []
angle_2pi = []
angle_r = []
qgp = []
p1 = []
p2 = []
p3 = []
p4 = []
qj = []
qj = []
qb = []
qb = []
```

A ’pontok’ tömbben már benne vannak a koordináták, de ez nekünk nem elég. Rendszereznünk kell, hogy tudjunk velük számolni. Ez rendszerezés az ’x’ és ’y’ koordináta párok különválasztásában merül ki. Fontos ezt a lépést megtenni, mert a ’pontok’ tömbben a koordináta párok egymás mellett helyezkednek el, ebben a formában: ’[(0,0),(0,1),(1,1)]’. Folyamatot a következő módon tudjuk leírni:

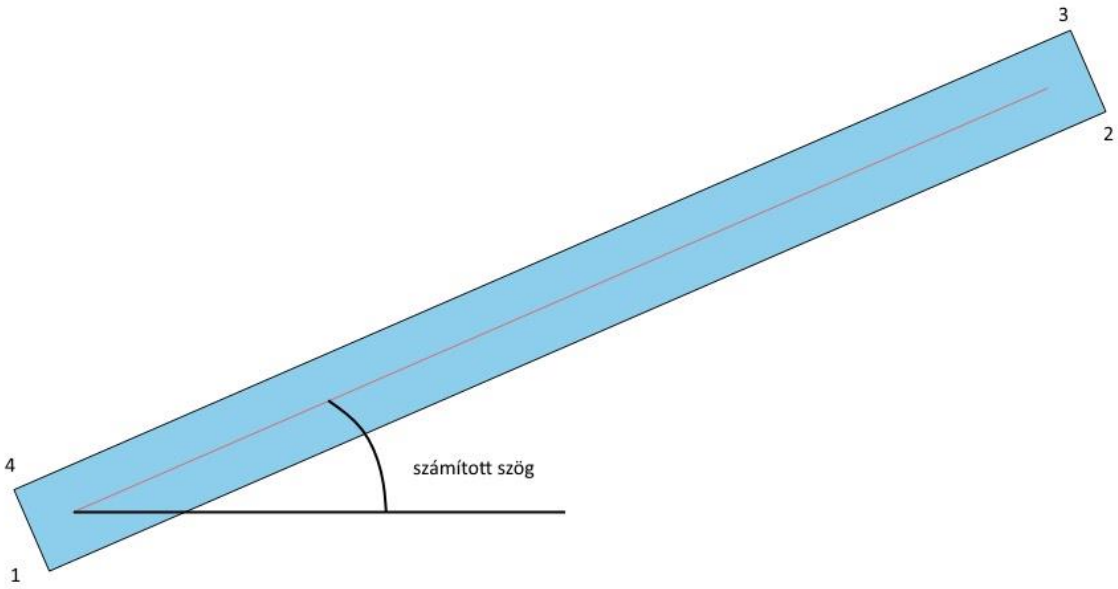
5.4.3
```
for i in range(0,len(pontok)):
        x.append(0)
        y.append(0)
        (x[i],y[i]) = pontok[i]
```

Majd egy ciklusban elhelyezzük az ’y’ és az ’x’ tengelyeken értelmezett különbségek számítását. Elsőnek a nulladik elem és az első elem különbségeit számítjuk, majd hasonlóan folytatjuk, míg végére nem érünk a folyamatnak. A ciklus hosszát a ’pontok’ tömb nagyságából kapjuk meg. Arkusz tangens függvénnyel számítjuk az egyenesek vízszintestől való elfordulását. Amennyiben a kapott szög negatív lesz, pozitívra váltom a későbbi egyszerű kezelhetőség miatt.

5.4.4
```
for i in range(0,len(pontok)-1):
        dx.append(0)
        dy.append(0)
        angle.append(0)
        angle_2pi.append(0)
        dy[i] = y[i+1]-y[i]
        dx[i] = x[i+1]-x[i]
        angle[i] = np.arctan2(dy[i],dx[i])
        if angle[i] <0:
                angle_2pi[i] = angle[i]+2*pi
        else:
                angle_2pi[i] = angle[i]
```

Minden kapott egyenesre számítunk négy ’p’ jelű pontot, helyzetétől függően számozzuk meg őket. Így összegyűjtjük a hasonló pozícióban lévő pontokat, természetesen hozzárendelve a saját egyeneséhez. Sokszögvonalon haladunk végig, de egyenesekként kezeljük őket. Kiszámítjuk minden

egyes egyeneshez a sarokpontokat annak ellenére is, ha akár nem kerülnek felhasználásra, majd tömbökben tároljuk őket. Ezek után kapunk négy tömböt, ami sarokponti koordinátákat tárolnak, indexük pedig az egyenes sorszámára vonatkoznak, számozásuk pedig helyzetükre.



5.2. ábra: Egyenes, puffer zóna hozzáadásával

5.4.5 for i in range(0, len(pontok)-1):.

```
p1.append(0)
p2.append(0)
p3.append(0)
p4.append(0)
p1[i] = x[i]+c*cos(angle[i]+pi/4+pi), y[i]+c*sin(angle[i]+pi/4+pi))
p2[i] = (x[i+1]+c*cos(angle[i]-pi/4), y[i+1]+c*sin(angle[i]-pi/4))
p3[i] = (x[i+1]+c*cos(angle[i]+pi/4), y[i+1]+c*sin(angle[i]+pi/4))
p4[i] = (x[i]+c*cos(angle[i]-pi/4+pi),y[i]+c*sin(angle[i]-pi/4+pi))
```

A p1 és p4 pontokhoz hozzá adunk 180°-ot, mert ők nem a számított irányszögből dolgoznak, hanem az ezzel ellentétesből. A sarokpontok koordinátáit polárisan –szögből és távolságból számítom.

A ’p’ értékei, float típusúak lesznek, ezért vissza kell alakítanunk QGIS-nek megfelelő koordinátákká. QgsPoint() függvénnyel alakítunk egyszerű számokból koordinátákat, mint ahogyan megkaptuk ’pontok’ tömbbe a koordinátákat a rétegről:

5.4.6 for i in range(0, len(pontok)-1):
(local_x,local_y)=p1[i]
p1[i] = QgsPoint(local_x,local_y)
(local_x,local_y)=p2[i]
p2[i] = QgsPoint(local_x,local_y)
(local_x,local_y)=p3[i]
p3[i] = QgsPoint(local_x,local_y)
(local_x,local_y)=p4[i]
p4[i] = QgsPoint(local_x,local_y)

Sokszögvonal nagyságától függetlenül a vonalvégeken mindig ugyanúgy jelennek meg a 'p' pontok, amit előre meg tudunk adni, a 'qgp' tömbbe, amit már korábban hoztunk létre, hogy a poligonunk (végső övezetünk) koordinátáit tárolja.

5.4.7 for i in range(0, (len(pontok)-2)*4+4):
qgp.append(0)

5.4.8 qgp[0] = p1[0]
qgp[len(qgp)/2-1]=p2[len(p2)-1]
qgp[len(qgp)/2]=p3[len(p3)-1]
qgp[len(qgp)-1]=p4[0]

A kitüntetett pontok, mindig hasonló formában veszik fel az indexüket, így ezeket cikluson kívül meg tudjuk adni. Ezek lesznek a puffer zóna sapkái.

Következő lépésben, metszéspontokat állítunk elő, a 'p' tömbök ismeretében. Minden töréspontnál felírunk két egyenletet, a kapott '(x,y)' koordináta pár lesz a metszéspont, ezeket már eleve QGISPoint-ok ként tároljuk, ahogy a réteg is teszi, tehát nem float ként. Gyakorlatilag két egyenes egyenletét írjuk fel, majd metszéspontjukat, tehát az egyenletek megoldásait keressük meg.

5.4.9 for i in range(1, len(pontok)-1):
m1 = tan(angle_2pi[i-1])
m2 = tan(angle_2pi[i])
(p1x,p1y) = p1[i-1]
(p2x,p2y) = p2[i]
c1 = p1y - m1*p1x
c2 = p2y - m2*p2x
qjx = (c1-c2)/(m2-m1)
qjy = m1*qjx+c1
qj[i] = QgsPoint(qjx,qjy)

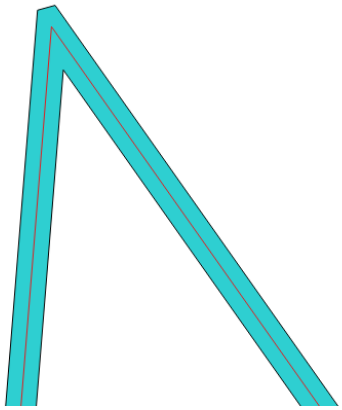
5.4.10
```
for i in range(1, len(pontok)-1):
        m4 = tan(angle_2pi[i-1])
        m3 = tan(angle_2pi[i])
        (p4x,p4y) = p4[i-1]
        (p3x,p3y) = p3[i]
        c4 = p4y - m4*p4x
        c3 = p3y - m3*p3x
        qbx = (c3-c4)/(m4-m3)
        qby = m4*qbx+c4
        qb[i] = QgsPoint(qbx,qby)
```

Ezek után egyszerűen a kapott koordinátákat egy tömbbe helyezzük, természetesen ügyelve a sorrendre. Másik nagyon fontos dolog, hogy egy sarokpontra több koordináta lehetőségünk van, az egyik a metszéspontból számított koordinátapár a másik pedig a 'p' indexel jelölt pontok. Ki kell választanunk a számunkra megfelelő koordináta párokat, hogy helyes poligont kapjunk.

Az első feltétel szerint kiválasztjuk, hogy milyen törésvonallal nézünk szembe. Kezdőponttól nézve és haladva a végpont irányába, azt figyelhetjük meg, hogy a számított szög első tagja akkor nagyobb a másodiknál, ha jobbra fordul a vezeték iránya.

5.4.11
```
for i in range(1, len(pontok)-1):
        if angle[i-1]>angle[i]:
                qgp[2*i-1] = qj[i]
                qgp[2*i] = qj[i]
                if angle_r[i-1]<pi/3:
                        qgp[(len(qgp)-1)-(2*i-1)] = p3[i-1]
                        qgp[(len(qgp)-1)-(2*i)] = p4[i]
                else:
                        qgp[(len(qgp)-1)-(2*i-1)] = qb[i]
                        qgp[(len(qgp)-1)-(2*i)] = qb[i]
        else:
                qgp[(len(qgp)-1)-(2*i-1)] = qb[i]
                qgp[(len(qgp)-1)-(2*i)] = qb[i]
                if (2*pi - angle_r[i-1])<pi/3:
                        qgp[2*i-1] = p2[i-1]
                        qgp[2*i] = p1[i]
                else:
                        qgp[2*i-1] = qj[i]
                        qgp[2*i] = qj[i]
```

Amennyiben ez teljesül, ugorhatunk a következő feltételre, ahol a törésszöget vizsgáljuk meg. A törésszög kisebb, mint 60 fok, akkor a 'p' koordinátákat használjuk, ennek esztétikai okai vannak, másrészt, ha törésszög túl kicsi, akkor a puffer zóna ne csúcsosodjon túlságosan ki.

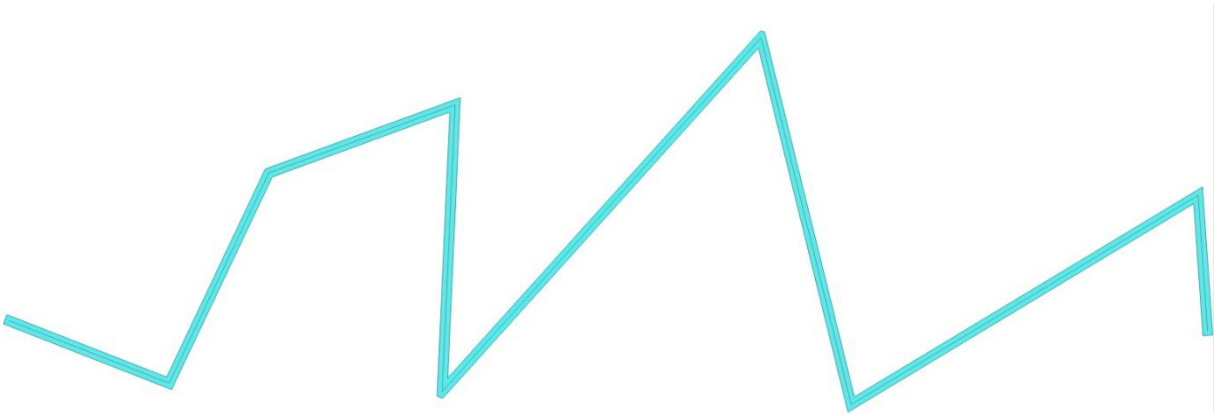


5.3 ábra: Övezet szögletes csúccsal

Metszéspontokat mindkét oldalon, akkor használunk, ha a törésszög 60°-nál nagyobb. Ugyanezt megismételjük a „balrafordulással” is, ha az elsőnek felállított feltétel nem teljesül.

5.4.12
```
negyzet = QgsGeometry.fromPolygon([qgp])
layer = QgsVectorLayer('Polygon','negyzet','memory')
pr = layer.dataProvider()
poly = QgsFeature()
poly.setGeometry(negyzet)
pr.addFeatures([poly])
QgsMapontokayerRegistry.instance()
.addMapLayers([layer])
```

Végül meghívjuk az új vektorréteget, amihez hozzákapcsoljuk a poligon geometriáját, amit az imént említett módon határoztunk meg. A kód teljes lefutása után egy felugró ablak fogad minket, ahol kitudjuk választani a számunkra megfelelő vetületi koordináta rendszert.



5.4. ábra: Sokszögvonal puffer zóna hozzáadásával

6 A KIDOLGOZOTT ELJÁRÁS HASZNOSÍTÁSA SZABAD ELÉRÉSŰ TÉRINFORMÁCIÓS RENDSZER SZOLGÁLTATÁSAKÉNT

6.1 LOGIKA ÁTÜLTETÉSE

Mivel nem szükséges QGIS vagy egyéb térinformatikai program ahhoz, hogy egy GeoJson állományt tudjunk készíteni, ezért a legcélszerűbb az, ha egy internetes oldalon, online szolgáltatásként tesszük. Felállítunk egy olyan weboldalt, ami egyszerre kezel online térképet és vektoros állományt. Kérdés csak az, hogy melyik szerver típus a legelőnyösebb számunkra.

6.2 SZERVEREK

6.2.1 Webserver

A webserver egy információs technológia, ami számítási kéréseket küld HTTP-n keresztül. Az elsődleges funkciója a webservernek a tárolás, számítás és az üzenet átadása a kliens és a weboldal között, ez lehet bármilyen jellegű adat.

(Web server - Wikipedia, the free encyclopedia, 2015)

6.2.2 WMS - Web Map Service

A WMS egy standard protokoll a georeferált térképek kiszolgálására az interneten. Ezt egy GIS adatbázis biztosítja. WMS server nyújt egy bitmap formátumú térképet, ami PNG, JPEG vagy GIF formátumban tárolt. Ehhez hozzá tudunk adni vektoros alakzatokat, akár szöveget is. Több szabad forrás kódú software tálalható az interneten, ilyen például a GeoServer.

(Web Map Service - Wikipedia, the free encyclopedia, dátum nélk.)

6.2.3 WFS - Web Feature Service

WFS biztosít egy felületet, hogy a térképeket ne csak képként tudjuk használni, hanem adatbázis ként is. A képet a felhasználó nem tudja szerkeszteni, de ezzel a szolgáltatással lehetséges további értékek csatolása a képfájlhoz ez shapefile formában történik. Amennyiben az adott szerver engedélyezi az adott értékek, tulajdonságok létrehozását, törlését vagy frissítését, akkor WFS – T szervernek nevezzük.

(Web server - Wikipedia, the free encyclopedia, 2015)

6.2.4 WPS – Web Processing Service

WPS feladata a bemenő adatok regisztrálása, számítási folyamat ismerete és lebonyolítása, majd az eredmény visszaküldésének kezelése. Lényege, hogy a számítás a szerveren történik meg. Eredetileg térbeli adatok kezelésére lett tervezve, de minden más számítást képes elvégezni.

7 WEBSERVER

Végül az webserverre esett a választás, mert az a legkönnyebben elérhető és legnépszerűbb is egyben. Általános esetben html kódból épül fel egy internetes honlap, ezt kiegészítheti javascript, ami egy objektum alapú script nyelv. Javascript építi az oldal dinamikus részét, ami képes kezelni vektor illetve térkép réteget egyaránt.

Open street map teljesen megfelelő a térkép réteg meghívására, segítségünkre az openlayers.org-nak egy állománya. *(http://www.openlayers.org/api/OpenLayers.js).* Ezek után létre enged hozni a Javascript egy vektor réteget, amit számunkra megfelelő koordinátarendszerbe tudunk állítani. Esetünkben ez az EOV koordináta rendszer lesz. Az algoritmus átültetésekor a logika nem változik, meg csak a változókat más úton hívjuk meg.

Ezáltal könnyen elérhető lesz, bárki hozzáférhet, nem szükséges hozzá semmilyen program letöltése esetleg vásárlása viszont kompatibilitás az megmarad.

8 HIVATKOZÁSOK

(2015). Forrás: PostScript – Wikipédia: https://hu.wikipedia.org/wiki/PostScript

(2015). Forrás: Python (programozási nyelv): https://hu.wikipedia.org/wiki/Python_(programoz%C3%A1si_nyelv)

Chrisman, N. (dátum nélk.). *Charting the Unknown.*

Kertész, Á. (dátum nélk.). *Térinformatika és alkalmazásai.*

Web Map Service - Wikipedia, the free encyclopedia. (dátum nélk.). Forrás: https://en.wikipedia.org/wiki/Web_Map_Service

Web server - Wikipedia, the free encyclopedia. (2015). Forrás: https://en.wikipedia.org/wiki/Web_server