Pécsi Tudományegyetem
Természettudományi Kar
Földtudományok Doktori Iskola

Egy általános célú Web GIS kliens alapjainak megteremtése

PhD értekezés tézisei

Szerző:
Farkas Gábor

Témavezető:
Dr. Bugya Titusz

Pécs, 2020.

A doktori iskola neve és címe:	Pécsi Tudományegyetem Természettudományi Kar Földtudományok Doktori Iskola 7624 Pécs, Ifjúság útja 6.
A doktori iskola vezetője:	Prof. Dr. Geresdi István egyetemi tanár Természettudományi Kar Földrajzi és Földtudományi Intézet Földtani és Meteorológiai Tanszék
A doktori témacsoport címe:	Természeti földrajz és tájértékelés
A doktori témacsoport vezetője:	Prof. Dr. Lóczy Dénes egyetemi tanár Természettudományi Kar Földrajzi és Földtudományi Intézet Természet- és Környezetföldrajzi Tanszék
Témavezető:	Dr. Bugya Titusz adjunktus Természettudományi Kar Földrajzi és Földtudományi Intézet Térképészeti és Geoinformatikai Tanszék

1. Bevezetés

A Web GIS egy relatíve új terület a geoinformatikán belül, mely szorosan követi a Webes technológiák fejlődését. Fő célja az Internet alapú technológiai vívmányok ötvözése a hagyományos digitális térbeli ábrázolással és elemzési eljárásokkal. Egyik látványos kimenete hagyományos GIS rendszerek Webes környezetbe való átültetése, bár fontos megjegyezni, hogy a terület ennél sokkal többet rejt magában. Mivel a környezet alapvetően más, mint az asztali szoftverek esetében, ezek az átültetések új kihívásokkal állítják szembe a fejlesztőket, melyek újfajta lehetőségekkel is kecsegtetnek.

A Web 2.0 megjelenésével (O'Reilly, 2007) a webes térképezés és a Web GIS felváltotta a hagyományos Internet GIS-t. A Google Maps indulásával 2005-ben (Farkas, 2015) egy új trend kezdődött, melyben a hangsúly a térbeli algoritmusok kliens oldalra való átültetésére helyeződött, JavaScript nyelven. A JavaScript-ben írt programok lassabbak a hagyományos fordított kódoknál, így sokkal érzékenyebbek a kódolási stílusra (Gong et al., 2015). Emiatt az automatikus nyelvek közötti konverzió sok esetben előnytelen. Az optimális teljesítmény eléréséhez óvatos, kézi optimalizációk és alternatív eljárások kellenek.

A modern webes megoldások inkább applikációk, mint dokumentumok, és a böngésző egyre inkább egy teljes értékű környezetként realizálódik, mint egy egyszerű dokumentum nézegetőként (Taivalsaari et al., 2011). Ez az irány fektetett hangsúlyt kliens oldali Webes GIS keretrendszerek és könyvtárak fejlesztésére (Ramsey, 2007). Vannak fejlesztők, akik egy teljes GIS rendszer részeit fejlesztik JavaScript nyelven, moduláris, más szoftverek által felhasználható formában. Ezen könyvtárak közül néhányan adatcsere formátumok írására és olvasására szakosodtak (pl. Shapefile, Geopackage, GeoTIFF) és vannak, melyek önkényesen nagy vektoros állományok megjelenítését végzik hardveres gyorsítással (pl. Mapbox GL JS, Tangram, Kepler.gl). Vannak olyan könyvtárak is, melyek vektoros adatokon végeznek különböző elemzéseket (pl. JSTS, Turf). Olyan könyvtár azonban még nincs, mely ezeket a részeket egységesíti, egy keretrendszert nyújtván, amellyel bárki képes teljes kliens oldali Web GIS megoldásokat gyártani.

Egy ilyen alap létrehozásával könnyen skálázható Web GIS rendszerek készíthetők. Minden alkalmazás szabadon dönthet a kliens és a szerver súlyáról, akár kondícionális alapon is. Így például lehetőség nyílik csak a drága számításokat szerveren futtatni. Egy ilyen alappal szerver oldal nélküli Web GIS alkalmazások is gyárthatók. További előnye, hogy a használt eszköz függvényében dönthet a betöltött funkcionalitásokról. Így a terepi felméréshez, adatelemzéshez és vizualizációhoz

elegendő egyetlen kódbázist fenntartani.

Az így gyártott általános Web GIS alkalmazások egyik legnagyobb felhasználói a különböző kézi eszközök lehetnének. Az utóbbi évtizedben ezen eszközök elterjedtek, használatuk általánossá vált. Manapság ezek a kis méterű számítógépek elég erősek, hogy olyan nagy számításigényű alkalmazásokat támogassanak, mint 3D videojátékok, vagy különböző irodai alkalmazások. Sok nagy számításigényű mobilalkalmazás az eszközök operációs rendszereit célozza meg (pl. Android, iOS), így azonban ezeket az alkalmazásokat minden platformon külön fenn kell tartani. Más alkalmazások a Webet célozzák meg, és a böngészők képességeit kihasználva, szolgáltatásként juttatják el funkcionalitásaikat a végfelhasználóknak. Egy ilyen alapon működő általános Web GIS rendszerrel a szakemberek teljesen platformfüggetlenek lehetnek, elegendő lenne egy böngésző a geoinformatikai munka minden fázisához.

Ahhoz hogy egy általános Web GIS rendszer követelményei lefektethetőek legyenek, vissza kell menni a hagyományos GIS rendszerek definícióihoz. Ezek a rendszerek több, korábbi rendszerből alakultak ki. Számos definíciója létezik a GIS-nek a szakirodalomban, melyek közül az egyik négy korábbi rendszer metszeteként jellemzi. Ezek a rendszerek feladata a számítógépes kartográfia, adatbázis-kezelés (DBMS), számítógép-segített tervezés (CAD) és távérzékelés voltak (Maguire, 1991). Amíg egy alapszintű térinformációs rendszer a távérzékeléssel kapcsolatos funkcionalitásokat elhagyhatja, addig a másik három kategória elemei fontosak.

Ezek a nem GIS specifikus képességek alkotják minden GIS rendszer magját. A számítógépes kartográfiából származik a térbeli adatok alapvető struktúrája. Függetlenül az adott rendszer belső felépítésétől, a térbeli adatok rétegeken rendeződnek. Minden réteg a teljes kompozíció részét alkotja, annak egy koherens tulajdonságát foglalva össze (Tomlin, 2017). Az egyik legszámottevőbb számítógépes kartográfiai tulajdonság, melyet a GIS rendszerek megörököltek a reprezentációs modell. Minden GIS rendszer képes többféle térképet készíteni ugyanazon kiinduló adatokból csupán a vizuális változókat változtatva (Roth, 2017).

Az adatbázis-kezelő rendszerek tulajdonságait a GIS rendszerek főleg vektoros adatok attribútumainak kezelésére használják. Mivel a vektoros elemek mellé bármennyi attribútumadatot lehet társítani, a relációs adatbázis-kezelők felépítése nagy segítség a konzisztencia megőrzésében. Amíg a felhasználó a legtöbb esetben csak egy attribútum táblát lát egy mezőkalkulátorral, a relációs adatbázis-kezelő rendszerekhez hasonló felépítés biztosítja a gyors és megbízható kétirányú kapcsolatot geometria és attribútumok között.

A különböző CAD funkcionalitásokat vektoros geometriák kezelésére vették át a GIS rendszerek. Ezek általában olyan interakciók formájában tapasztalhatóak, mint a geometria kiválasztás, affin transzformációk (pl. eltolás, forgatás, skálázás), vagy az attribútumok frissítése geometria alapon. Kevésbé triviális az alacsony szintű megjelenítési képesség, mely eredetileg szintén CAD funkcionalitás volt, mint a geometria stílusozása vagy a hardveresen gyorsított vektoros vizualizáció.

Bár a távérzékeléssel kapcsolatos képességek nem kötelezőek egy alapszintű GIS rendszerhez, mégis vannak olyan tulajdonságok, melyeket érdemes implementálni.

Ezek olyan egyszerű képszerkesztő algoritmusok, amelyek a raszteres elemzést lehetővé teszik. Ha néhány kevésbé bonyolult algoritmus (pl. térképi algebra, klasszifikáció, konvolúció) implementálásra kerül, a felhasználók már csak ezeket felhasználva képesek hasznos és többrétű raszteres elemzéseket végezni.

Ezekre az alapfunkcionalitásokra építkezve számos GIS specifikus tulajdonsággal kell egy általános GIS kliensnek rendelkeznie. Az egyik fő csoport az adatcsere formátumok támogatása. Egy általános GIS kliensnek kezelnie kell különböző formátumú téradatokat mind bemenetként, mind kimenetként. Egy alapszintű GIS rendszernek azonban elég csak a legelterjedtebb formátumokat kezelnie (pl. Shapefile, GeoJSON, KML, GeoTIFF, ArcInfo ASCII Grid) (Orlik & Orlikova, 2014).

Vitatható, hogy a vetületi rendszerek kezelése egy GIS specifikus képesség, vagy a számítógépes kartográfia hozománya. A tisztességes vetületkezelés része különböző vektoros és raszteres rétegek röptében transzformálása egy közös célvetületbe. Mivel ez a művelet vektoros adatok esetében koordináta transzformációt, raszteres adatoknál pedig interpolációt igényel, a vetületkezelést a dolgozat GIS specifikus tulajdonságként kezeli.

Az utolsó GIS specifikus kategória a különböző térelemzések elvégzésének képessége. Amíg egy alapvető GIS rendszernek nincs szüksége túlzott mennyiségű elemző eszközre, egyszerű elemzéseket vektoros és raszteres adatokon egyaránt el kell tudnia végeznie. Egy, a teljesség igényével felállított általános GIS képességeket tartalmazó listáról (Albrecht, 1998) minimum támogatnia kell egy alapszintű kliensnek a méréseket, interpolálást, térbeli (topológiai) lekérdezéseket, és alapvető térbeli műveleteket vektoros rétegeken. Ezek a térbeli műveletek a pufferzóna generálás, összeolvasztás, pontok keresése poligonokban (PIP) (Thrall & Thrall, 1999), térbeli halmazműveletek (metszés, unió, különbség, szimmetrikus különbség). Továbbá, hasznos ha van átalakítási lehetőség vektoros és raszteres típus között (Meaden & Chi, 1996).

Ha egy kliens ezek közül az összes képességgel rendelkezik, azt határozottan lehet általános GIS kliensnek nevezni. Azonban egy általános Web GIS kliensnek ezen felül még Web-specifikus képességekkel is rendelkeznie kell. Ezek a képességek főleg a különböző térszolgáltatásokhoz (pl. WMS, WFS, WCS) való kapcsolódást foglalják magukban. Ezek nélkül, ha egy Web GIS kliens képtelen lenne térszerverekkel kommunikálni sztenderd csatornákon, túlzottan limitált lenne. Továbbá, mivel egy téradatbázisban hatalmas mennyiségű adat tárolható (Agrawal & Gupta, 2014), valamint a böngészők jelenleg még nem képesek közvetlenül RDBMS rendszerekhez csatlakozni, ez az egyetlen módja téradatbázisok használatának Web GIS kliensek esetén.

Ezen tulajdonságok mentén kiindulva, azokat kiegészítve vagy finomítva, ahol szükséges volt, sikerült egy összefoglaló listát készíteni egy alapszintű általános Web GIS rendszer követelményeiről. Mivel a lista készítése – természetéből adódóan – nem zárt ki minden szubjektivitást, fontos megjegyezni, hogy ez nem az egyetlen módja különböző kliensek értékelésének. Ez csak egy lehetőség szakirodalmi megalapozottsággal.

2. Célkitűzés

A dolgozat célja egy olyan kliens oldali webes térképezésre alkalmas könyvtár megtalálása, mely egy Web GIS rendszer stabil alapjává alakítható. A minél objektívebb döntés érdekében ennek első lépése egy összehasonlító elemzés, mely a jelenlegi vezető nyílt forrású technológiákat veszi górcső alá. A legalkalmasabb klienset kiválasztva, annak elemzésére kerül sor, feltárván gyengeségeit egy GIS rendszer nézőpontjából. Ezen gyengeségek szűkítése során kiválasztásra kerül az a néhány igazán súlyos hiányosság, mely megakadályozza, hogy alapjául szolgáljon egy univerzális Web GIS rendszernek. Végső soron, ezek a hiányosságok elemzésre és implementálásra kerülnek, így megalkotva egy alap szintű, de működő Web GIS könyvtárat.

A dolgozat lépései listába szedve a következőképpen néznek ki:

1. Egy alap szintű GIS tulajdonságainak összegzése szakirodalom alapján.
2. Alkalmas kliensek kiválasztása.
3. Összehasonlító elemzés.
4. A legalkalmasabb kliens kiválasztása.
5. A kiválasztott kliens legsúlyosabb hiányosságainak feltárása.
6. A feltárt hiányosságok implementálása.

3. Módszerek

3.1. Összehasonlító elemzés és egyéb metrikák

A disszertáció több, módszertanilag eltérő lépésből áll. Az első lépés a megfelelő alap kiválasztása volt egy általános Web GIS kliens kialakításához. Ehhez minél több adatvizualizációs könyvtár összehasonlítására volt szükség egy elméleten alapuló GIS funkcionalitási lista alapján. Ez a lista (**??**. ábra) került a tézisfüzet első fejezetében összefoglalásra.

Amíg egy effajta összehasonlítás nagyon fontos a megfelelő alap kiválasztásához, önmagában nem elég az összehasonlított könyvtárak mélyebb feltárásához. Vannak egyéb szempontok, melyek befolyásolják a felhasználókat. Ilyen a rendelkezésre álló dokumentáció minősége és mennyisége, a segédletek száma és a fejlesztői aktivitás. Vannak továbbá olyan aspektusok, melyeket kimondottan nehéz számszerűsíteni, mint a könyvtár komplexitása felhasználói szemszögből. A bevett szokás ebben az esetben egy szakértőknek szóló kérdőív megalkotása (Roth et al., 2014). Máshonnan közelítve, vannak statikus szoftvermetrikák, melyek hosszas kérdőívezések nélkül is rávilágíthatnak a problémára.

Az összehasonlító elemzés minden lehetséges könyvtárat pontozott a vizsgált tulajdonság alapján. A legtöbb tulajdonság több szempontból állt, melyeket mind támogatnia kellett az adott könyvtárnak a maximális pontszám eléréséhez. Ha a könyvtár a tulajdonságot teljes mértékben támogatta, 1 pontot kapott. Részpontok két esetben kerültek kiosztásra. Ha a könyvtár csak részben támogatta az adott tulajdonságot, valamint ha a könyvtár támogatta a tulajdonságot, de egy harmadik fél által fejlesztett kiegészítés felhasználásával. Mindkét esetben 0,5 pontot kapott az adott jelölt. Amennyiben a könyvtár egyáltalán nem támogatta a vizsgált tulajdonságot, úgy 0 pontot kapott rá. Több olyan eset volt, amikor egy tulajdonság támogatása jelenlegi technológiával lehetetlen volt (pl. csatlakozás téradatbázishoz). Ezek a tulajdonságok azért kerültek a listába, hogy a vizsgálat megismételhető legyen, amikor a Webes technológia fejlődésével a korlátok oldódnak.

Az összpontszám a részpontszámok átlagolásával került kiosztásra. Így ha egy könyvtár minden vizsgált tulajdonságot támogat, akkor annak 100%-os a lefedettsége. Mivel az összpontszám nem ad teljes képet a könyvtárak erősségeiről, a tulajdonságok kategóriába lettek szervezve (pl. megjelenítés, formátumkezelés, interakció), ahol szintén százalékos eredmények születtek a kategória tagjainak átlagolásával.

A könyvtárak komplexitásának mérésére egy nehezen megfogható tulajdonság került kiválasztásra: a tanulási görbe. Feltételezhető, hogy a megfelelő metrikák ki-

Kategória	Dokumentációs pontszám	Megválaszolt kérdések aránya
Gyenge	0	0 – 0.25
Elfogadható	0 – 0.5	0.25 – 0.5
Jó	0.5 – 1	0.5 – 0.75
Kiváló	1 –	0.75 – 1

1. táblázat. Ordinális értékek kialakításának szabályai dokumentációs pontszámok és támogatottsági pontszámok esetén. A gyenge kategórián kívül minden intervallum balról nyitott és jobbról zárt.

választásával lehetőség nyílik megbecsülni a tanulási görbét. Így egy új metrika, a Hozzávetőleges Tanulási Görbe JavaScript-re (ALC_{JS}) került kialakításra, mely kimondottan JavaScript könyvtárakat céloz meg (Farkas, 2017). A képlet (1. képlet) tartalmazza a logikai sorok számát (LLOC), a ciklomatikus komplexitást ($v(G)$) és a hozzáférhető funkciónkat (EF). Az utóbbit azért, mert egy korábbi tanulmány már bizonyította hatását a tanulási görbére (Fowler et al., 1999).

$$ALC_{JS} = \log_{10} LLOC \times \log_2(\frac{v(G)}{F} \times \frac{LLOC}{EF}) \tag{1}$$

Egy könyvtár használhatóságát több egyéb karakterisztikája is befolyásolja. Ezek közül néhány fontosabb a dokumentáció, a közösség, és a támogatottság (Ramsey, 2007; Steiniger & Hunter, 2013; Poorazizi & Hunter, 2015). Közös tulajdonságaik, hogy sok módszer létezik mérésükre és általában ordinális skálán szokás őket megjeleníteni. A megismételhetőség érdekében egy-egy numerikus módszer került megalkotásra mindhárom karakterisztikára.

A dokumentációs pont (2. képlet) az API dokumentáció létezéséből (A), a segédletek számából (T) és a példák számából (E) tevődik össze. Mivel az API dokumentáció megléte a legfontosabb felhasználói szemszögből, ennek a kritériumnak a teljesítése nem emeli a pontszámot, de hiánya lenullázza azt. Lévén a segédletek sokkal átfogóbbak, lassabban készülnek, és jobban segítik a felhasználót a könyvtár használatában, nagyobb súlyt kaptak. Végül az így kapott pontszám ordinális értékekké lettek alakítva (1. táblázat).

$$Score = A \times (T/10 + E/100) \tag{2}$$

A közösség és a támogatottság puha metrikák. A közösségi kategóriában a kutatás két szempontot vizsgált. Az egyik a fejlesztők és meghatározó fejlesztők száma a projektben. Meghatározó fejlesztőnek az számított, aki legalább 1000 sort adott hozzá a forráskódhoz. A második a kiadások sűrűsége, ami kiszámolható a kiadások számából (n) és az első– (D_{FR}), valamint az utolsó kiadás (D_{LR}) között eltelt napok számából (3. képlet).

$$RF = \frac{D_{LR} - D_{FR}}{n - 1} \tag{3}$$

A támogatottsági metrika két forrást használt. Mivel az összehasonlított projektek nagy része GitHub-ot használt verziókövető rendszerként, annak hibajelentő

rendszerét fel lehetett használni a fejlesztői visszajelzések mérésére. Ez a metrika a nyílt hibajegyek arányát nézi az összes hibajegyhez képest. A felhasználói közösség támogatásának mérését két népszerű fórum (Stack Overflow és GIS Stack Exchange) adatai alapozták meg. A megválaszolt kérdések aránya volt az alapja az adott könyvtár besorolásának (1. táblázat).

3.2. Teljesítménymérés

A dolgozatban bemutatott kiegészítések tesztelése futásidejük és memórialenyomatuk mérésével történt. A hardveres gyorsítás és a raszterkezelés eltérő módszert használt, mivel funkcionalitásuk is eltér. A közös eszköz mindkét esetben egy Dell Inspiron 7567 laptop volt, amin egy 64 bites Chromium böngésző szolgált környezetként Debian 9-es rendszeren. A hardveres gyorsítás mérésére egy második eszköz is jelen volt. Ez egy Lenovo A536 okostelefon volt 32 bites Chrome böngészővel és Android 4.4.2-vel.

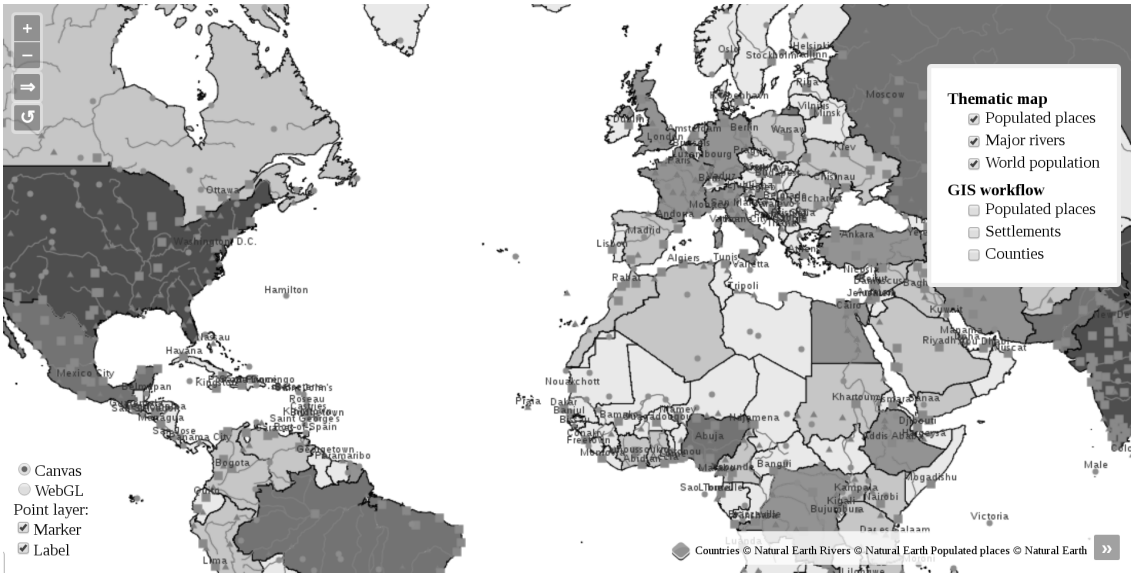
A hardveres gyorsítás mérésére egy Webes applikáció lett fejlesztve, ami a böngészőkbe beépített nagy felbontású időzítő funkciót, a Performance Timeline API-t (Grigorik et al., 2016) használja. Az alkalmazás több egymást követő megjelenítési folyamatot mér, adatsorokat produkálva. Ezen adatsorokból a kiugró értékek kiszűrése után átlagolással kerültek az eredmények kiértékelésre.

Mivel a webes térképezési könyvtárak el tudják tárolni ideiglenesen a megjelenített adatok egy részét, hogy az animációkat és interakciókat (pl. mozgatás, nagyítás) felgyorsítsák, az animáció egy külön kategóriát kapott külön mérésekkel. A teljes kirajzolási folyamat mérésére a térkép középpontja az X tengely mentén került automatikus újrapozicionálásra. Ebből a fajta mérésből minden nagyítási szinten 10 adatpont került feldolgozásra. Mivel az adatpontok száma – a nagy adatmennyiség által okozott lassulás miatt – kevés volt, amennyiben egy mérés kiugró értékeket hozott, ismétlésre került.

Az animációk méréséhez az alkalmazás egy automatizált mozgatást végzett egy előre definiált útvonal mentén a térkép újrapozicionálása helyett. A program az egymást követő képkockák között eltelt időt mérte, így az adatpontok mennyisége a képsebesség függvénye volt. Nagy terhelés alatt a képsebesség annyira le tud csökkenni, hogy a mérések 5 és 100 adatpont között adtak eredményt. Amennyiben egy mérés 10 adatpont alatt produkált, úgy az ismétlésre került, az új adatok pedig hozzá lettek fűzve a régiekhez.

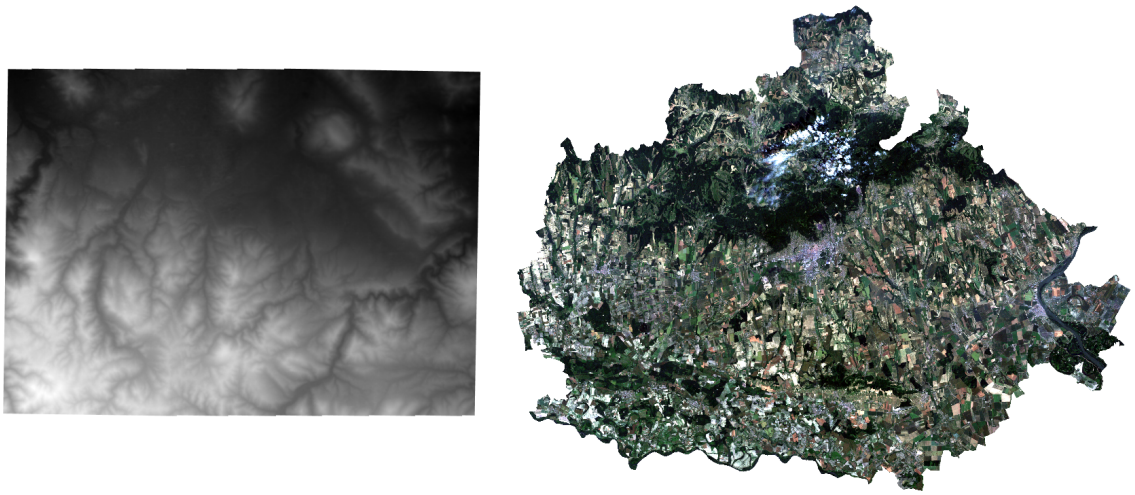
Végezetül a böngésző fejlesztői eszközeivel részletes teljesítménytesztek is készültek. Ezek a tesztek mutatták meg, hogy a futási idő hogy oszlik meg különböző fázisok és függvényhívások között a megjelenítés során. Így rá lehetett világítani a lehetséges szűk keresztmetszetekre, valamint becslést lehetett adni egy optimálisabb teljesítményre azok jelenléte nélkül.

Három fajta mintaadat adta a mérések alapját. Ezek közül az egyik egy valós példája egy tematikus térképnek, mely több fajta térképi elemet tartalmaz (1. ábra). A másik egy kisebb GIS munkamenet állapotát tükrözi, sokkal nagyobb adatmennyiséggel, de különböző stílusjegyek nélkül.



1. ábra. Tematikus térkép, mely a hardveres gyorsítás méréseinek egy részében szolgált alapul (Farkas, 2019).

A harmadik mintaadat a raszterkezelő méréseiben nyújtott segítséget. Ez két különböző méretű és felépítésű raszteres rétegből állt (2. ábra). Az egyik egy digitális domborzatmodell, mely a GRASS GIS Spearfish60 mintaállományai közül származik. A másik egy Landsat 8 multispektrális műholdfelvétel Baranya megyei kivágata. Ez a réteg négy spektrális csatornát tartalmaz (vörös, zöld, kék, közeli infravörös) külön alrétegeken.



2. ábra. A raszterkezelő tesztelésére használt raszteres mintaállományok. Mindkét raszter Web Mercator (EPSG:3857) vetületben van, röptében történt vetítéssel. A Spearfish60 domborzatmodell (balra) szürkeárnyalatos stílusozással, míg a Baranya műholdfelvétel (jobbra) RGB rétegkompozícióval került megjelenítésre (Farkas, 2020).

4. Eredmények

4.1. A megfelelő jelölt kiválasztása

A legjobb általános Web GIS kliens alap kiválasztásához számos adatvizualizációs könyvtár került összehasonlításra. Az elsődleges szűrőkön azonban csak néhány ment át. Ezek a könyvtárak (Cesium, Leaflet, NASA Web World Wind, OpenLayers 2, OpenLayers) alkották a jelölteket, amik részletes összehasonlításon mentek át. Érdemes megemlíteni, hogy a jelöltek között van két virtuális glóbusz. Ezek a könyvtárak (Cesium, NASA Web World Wind) azonban erős térinformatikai megalapozottsággal rendelkeznek.

Az összehasonlító elemzés eredménye alapján (2. táblázat) nincs túl nagy különbség a könyvtárak lefedettsége között. Az OpenLayers könyvtárak teljesítettek a legjobban, ami azzal magyarázható, hogy ezek a projektek mindvégig GIS szoftverek mintájára épültek. Ezt a belső struktúrájuk támasztja legjobban alá, amivel a felhasználók könnyedén képesek funkcionalitásokban gazdag webes térképeket készíteni.

A statikus metrikák eredményei (3. táblázat) nem hozott váratlan eredményt. Az LLOC metrikák, a funkciónkénti ciklomatikus komplexitás és az elérhető funkciók száma számos aspektusára rávilágítottak a jelölteknek. Továbbá az ALC_{JS} értékek megegyeztek a könyvtárak tapasztalt tanulási görbéivel.

A Leaflet a legkisebb könyvtár, viszont neki van a legtöbb elérhető funkciója is. A komplexitása alacsony, így könnyen tanulható. A NASA Web World Wind a kisebbik a virtuális glóbuszok között. A tanulási görbéje sokkal meredekebb,

Csoport	Cesium	Leaflet	NASA WWW	OpenLayers 2	OpenLayers
Megjelenítés	80%	40%	60%	40%	60%
Formátumok	65%	62%	53%	82%	76%
Adatbázis	0%	8%	0%	17%	0%
Adatkezelés	32%	30%	18%	34%	44%
Vetület	63%	50%	75%	63%	88%
Interakció	33%	50%	33%	83%	72%
Reprezentáció	22%	44%	33%	56%	56%
Átlag	41%	41%	34%	54%	56%

2. táblázat. GIS tulajdonságok lefedettsége a vizsgált könyvtárak esetén (Farkas, 2017).

Könyvtár	Méret (KB)	LLOC	CC/F	EF	ALC_{JS}
Cesium	11 420	292 500	2.08	911	51.27
Leaflet	162	3639	2.00	200	18.47
NASA Web World Wind	1452	13 037	2.50	187	30.64
OpenLayers 2	872	23 702	2.82	207	36.46
OpenLayers	499	21 451	2.36	223	33.90

3. táblázat. A vizsgált könyvtárak statikus metrikái (Farkas, 2017). A CC/F a funkciónkénti ciklomatikus komplexitást jelöli.

Tulajdonság	Cesium	Leaflet	NASA WWW	OpenLayers 2	OpenLayers
Dokumentáció	Jó	Jó	Elfogadható	Kiváló	Kiváló
Támogatottság	Jó	Kiváló	Gyenge	Jó	Jó
Közösség	89 (29)	236 (8)	12 (5)	98 (16)	154 (27)
Hibajegyek	409 (25%)	225 (7%)	40 (56%)	383 (64%)	414 (21%)
RF	28	68	N/A	32	24

4. táblázat. A vizsgált könyvtárak nem kódból származtatható metrikái (Farkas, 2017). Az *RF* a kiadások sűrűségét jelöli, mely két kiadás között átlagosan eltelt napok száma.

ami azzal magyarázható, hogy egy virtuális glóbuszról van szó, a webes térképezési könyvtáraknál bonyolultabb logikával. Az OpenLayers könyvtárak és a Cesium a nagy játékosok a jelöltek között. Nehezebb beletanulni, még nehezebb magas szinten elsajátítani használatukat. Nagy és funkcionalitásokban gazdag könyvtárak. A Cesium a legnagyobb, kódbázisa felér nagyobb asztali szoftverekével. A meredek tanulási görbéje arányban van azzal, ami várható egy olyan virtuális glóbusz esetén, mely képes térben és időben is megjeleníteni 3D adatokat.

Más puha metrikák (4. táblázat) rávilágítottak a könyvtárak használati nehézségeire. A táblázatban a meghatározó fejlesztők zárójelben szerepelnek a fejlesztők mellett. A nyitott hibajegyek száma mellett pedig az összes hibajegyhez mért arányuk szerepel.

A legtöbb jelölt jól dokumentált, így a felhasználók hamar bele tudnak rázódni használatukba. Amíg a Cesium, és az OpenLayers könyvtárak kimagaslóan sok példával rendelkeznek, a Leafletnek nagyon alapos segédletei vannak alapvető esetekre. Az OpenLayers 2 rendelkezik a legtöbb példával (210), míg az OpenLayers-hez adták ki a legtöbb segédletet (23). Sajnos a NASA Web World Wind nagyon kevés példával és segédlettel van ellátva, így ebbe a könyvtárba kimondottan nehéz beletanulni.

A támogatottsági metrikák hasonló képet festettek. Ebben a kategóriában a Leaflet volt a vezető 80%-os megválaszolt kérdésaránnyal és a legtöbb feltett kérdéssel (5447). A NASA Web World Wind itt is a lista alján végzett, mivel nem volt mérhető támogatottsága, valószínűleg mivel a fejlesztés korai stádiumában volt.

A fejlesztői statisztikák azt mutatták, hogy amíg a Leaflet könyvtárhoz tartozott a legtöbb fejlesztő, a legkevesebb meghatározó fejlesztővel szintén ő rendelkezett. A Cesium és az OpenLayers tudta maga mögött a legtöbb meghatározó fejlesztőt,

ami a hosszú távú stabilitásukra utal. A nyílt hibajegyek száma szintén az utóbbi könyvtárakat erősítette. Azonban a Leaflet magasan túlteljesítette a többi könyvtárat a nyílt hibajegyek arányával. A kiadások sűrűségével egy könyvtár esetében volt csak probléma. Mivel a NASA Web World Wind privát környezetet használt a vizsgálat idején, nem volt elérhető adat a kiadások számáról és idejéről.

A könyvtárak lefedettségi és statisztikai mutatói alapján két lehetséges jelöltre szűkíthetőek: a Cesium-ra és az OpenLayers-re. Mivel a két könyvtár között alig mutatkozott különbség, a nem támogatott GIS funkciók közül a legkritikusabbak kerültek összegyűjtésre, amiket mindenképpen orvosolni kell. Három ilyen funkcionalitás volt. A Cesium nem rendelkezett megfelelő vetületkezeléssel, az OpenLayers nem rendelkezett teljes hardveres gyorsítással, és egyik könyvtár sem rendelkezett raszterkezelővel.

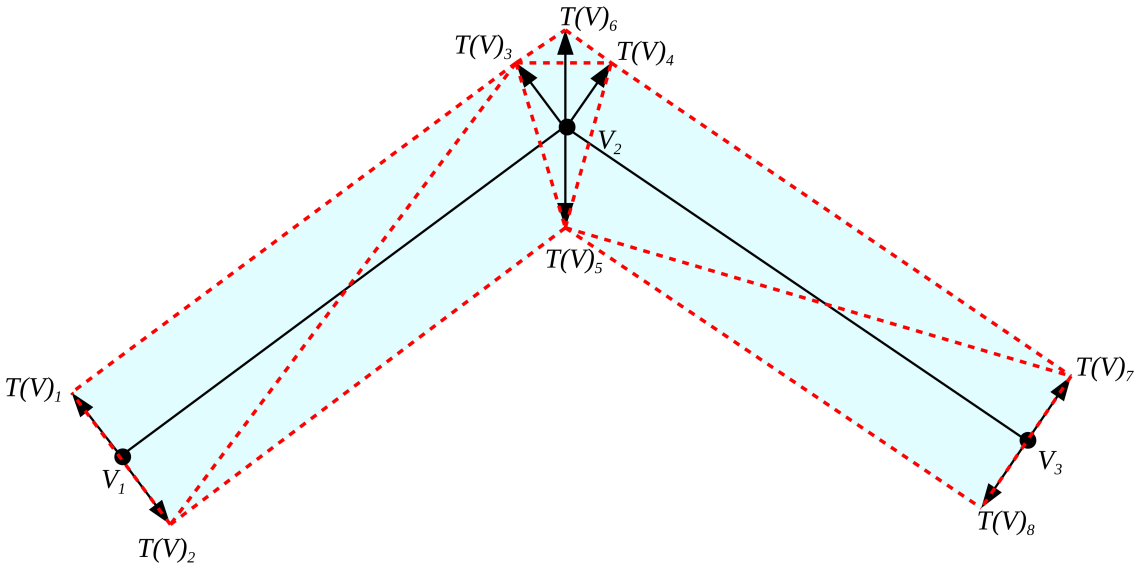
A két egyedi eset implementálására becslések születtek. A Cesium rendelkezett egy üres osztállyal egyedi vetületekre, azonban a vetületkezelésre a megjelenítés túl szorosan épült. Az OpenLayers szintén rendelkezett a hardveres gyorsítás alapjaival egy kezdeti WebGL motor formájában, ami textúrák kirajzolására volt képes. Ezt az alapstruktúrát egyszerűbbnek tűnt kibővíteni, mivel csak a vektoros elemek (vonal, poligon, kör, szöveg) támogatását kellett megírni hozzá. Így az OpenLayers hardveres gyorsításának kibővítésére egy egy éves becslés született, ami el is döntötte, hogy ő lesz a győztes.

4.2. Hardveresen gyorsított vektoros megjelenítés

A kiegészítés előtt az OpenLayers képek és vektoros pontok megjelenítésére volt képes WebGL motorral. Valójában csak textúrákat tudott megjeleníteni, mivel a pontokat először egy belső rétegre rajzolta ki a Canvas API segítségével, ezt eltárolta, majd textúraként jelenítette meg a WebGL rétegsoron. Ahhoz hogy egy teljes értékű hardveresen gyorsított megoldást kapjon, szükség volt vonalláncok, poligonok és címkék kirajzolására is. A címkék kirajzolásához a meglévő textúra megjelenítő felhasználható volt.

A WebGL – mint az OpenGL – nem tud összetett geometriákat kirajzolni önmagától. Egy alacsony szintű API-n keresztül lehet vele dolgozni, ezt megfelelően kihasználva tud 2D és 3D jeleneteket életre kelteni. Limitált számú entitással képes önállóan dolgozni, amiket primitíveknek hív. A vonal primitívek (`gl.LINES`, `gl.LINE_LOOP`, `gl.LINE_STRIP`) eltérő módszerek szegmensek kirajzolására. Amíg a `gl.LINES` változó vastagságú szegmenseket képes megjeleníteni, a vonalcsatlakozásokat nem támogatja. A többi képes vonalakat összefűzni, de a vonalvastagság csak 1 pixel lehet.

Ez a limitáció bonyolultabb technikák alkalmazását teszi szükségessé. Vonalláncok helyes megjelenítéséhez a szegmenseket háromszögelni kell, úgy, hogy a vonalvégződéseket és a vonalcsatlakozásokat egyaránt megoldja a program. Minden szegmens legkevesebb két háromszöggel írható le. Ha vannak vonalvégződések, azokat a hozzá kell adni a szegmenshez szintén két háromszög formájában. Vonalcsatlakozás esetén ki kell számolni az alsó és a felső csatlakozási pontot is. Minden



3. ábra. Háromszögelési séma két szegmensből álló vonalláncra V_1, V_2, V_3 ékes csatlakozással, vonalvégződések nélkül. A nyilak az eltolások irányát és mértékét mutatják a vonal pontjaihoz képest. A szaggatott vonalak a háromszögeket jelölik az eltolt pontok helyes sorrendben történő összekötése után.

csatlakozásnak van alsó pontja, míg az ékes csatlakozáshoz szükség van felső pontra is. A csatlakozások kiszámításához a vonalak irányát is számításba kell venni. Az összes szükséges pont kiszámításával egy két szegmensből álló vonallánc (V_1, V_2, V_3) legkevesebb nyolc ponttal írható le (3. ábra).

A végső program minden eltolt pontot a GPU-n számol ki. Ehhez a módszerhez minden ponthoz szükség van három koordinátapárosra (előző pont, jelenlegi pont, következő pont), és egy kódolt paraméterre. Csupán a vonallánc pontjaira megadni ezeket a paramétereket nem elegendő, ugyanis a WebGL programok nem képesek új sarokpontokat készíteni. Így a háromszögelés összes pontjára szükség van a paraméterekre, ami a bemenő adatok redundanciájával jár. A kódolt paraméter egy i utasítást (pl. merőleges eltolás, ékszámítás, vonalvég számítás), egy d irányt, és egy r kerekítési utasítást tartalmaz egyetlen $p = i \times d \times r$ számként, amit a GPU-n dekódol a program.

Poligonok megjelenítése egyszerűbb a vonalláncokénál. Háromszögelés után a GPU önállóan ki tudja rajzolni a kitöltést, míg a körvonal kirajzolható a vonallánc megjelenítővel. A legnehezebb része a poligonok háromszögelése. A legtöbb grafikus alkalmazásban a háromszögelés nem jelent problémát, mivel számos bevett könyvtár és keretrendszer létezik a probléma megoldására (pl. GLU tessellator, cairo, Qt). Azonban ezek az eszközök asztali alkalmazásokra lettek fejlesztve. A Webes környezetben kevesebb módszer létezik, amik közül néhány az asztali könyvtár egyenes átirata. Ezeknek az átiratoknak az alapjai megfelelő sebességgel működnek asztali környezetben, de maguk az átiratok böngészőben már kevésbé.

Az implementáció egy körkörös duplán láncolt listát alkalmaz fő adatszerkezetként. A láncolt lista elemei szegmensek, mivel metszéseket is kell ellenőrizni

Nagyítási szint	1	2	3	4	5
NVIDIA GPU					
WebGL animáció	56.35	56.82	57.53	56.00	54.19
Canvas animáció	29.24	23.64	22.45	32.59	46.37
WebGL rajzolás	14.75	10.92	10.26	12.18	15.37
Canvas rajzolás	22.94	18.59	14.23	24.33	33.00
Intel GPU					
WebGL animáció	58.99	55.34	52.59	59.01	59.72
Canvas animáció	26.07	20.21	15.17	16.77	22.83
WebGL rajzolás	10.91	8.37	7.17	7.72	8.03
Canvas rajzolás	13.57	12.92	10.28	10.71	14.66
ARM Mali GPU					
WebGL animáció	12.67	12.08	10.66	16.51	21.83
Canvas animáció	1.19	1.22	1.87	3.69	5.84
WebGL rajzolás	1.25	0.94	0.82	1.45	2.27
Canvas rajzolás	1.13	0.96	1.32	2.51	4.05

5. táblázat. A tematikus rétegsor megjelenítési teljesítménye (FPS). Az akadozásokat aláhúzás jelöli, míg a komoly elakadásokat bekeretezés (Farkas, 2019).

önmetsző poligonok esetén. A másodlagos adatstruktúra egy R-fa, ami minden szegmenst térben indexel a gyorsabb keresés érdekében. Annak érdekében, hogy egyszerű és topológiailag helyes poligonoknál a háromszögelés a lehető leggyorsabban lefusson, az algoritmus egy büntető eljárást használ. Minél több hiba van a poligonban, annál precízebb és időigényesebb módszerekhez folyamodik.

Címkék megjelenítéséhez a más meglévő képmegjelenítőhöz hasonlóan az implementáció a HTML5 Canvas API-t használja. A címkéket belső `canvas` elemeken tárolja, majd szükség esetén előhívja. A Canvas API használata nélkül gyorsabb és kevesebb memóriát igénylő megoldások is fejleszthetőek, azonban ezen eljárások esetén minden betűtípust el kéne tárolni a szerveren. Ahhoz, hogy a program kihasználhassa a felhasználó operációs rendszerén lévő betűtípusokat, erre a függőségre szükség van. A hatékonyság növelése érdekében a program egy karakter atlaszt használ, ami címkék helyett karaktereket tárol, és azokat helyezi egymás mellé kirajzoláskor.

A teljesítménymérések első része a tematikus rétegsort célozta meg. Az eredmények alapján (5. táblázat) a Canvas megjelenítő alkalmas egy tipikus, nem optimalizált vektor alapú webes térkép megjelenítésére középkategóriás GPU-val rendelkező számítógépeken. A WebGL motor animációs sebessége kimagasló volt az integrált Intel– és a dedikált NVIDIA GPU-n is. Az értékek ebben az esetben 60 FPS körül mozogtak, ami a 60 Hz-n működő kijelző okozta limitációval magyarázható. Ezen a ponton az adatok kevésbé stabilak. Amíg 20 – 30 FPS körül 1 FPS különbség is jelentős, 50 FPS felett 4 FPS különbséget könnyedén okozhat bármilyen külső zavaró tényező.

A felhasználói élményt főleg az animációs sebesség határozza meg. A térképpel

	Pont		Poligon	
	Canvas	WebGL	Canvas	WebGL-c[a]
Teljes idő (ms)	84.48	36.28	337.79	248.76
Szkriptelés	78.61%	95.78%	28.04%	97.16%
Megjelenítés	0.09%	0.55%	0.05%	0.05%
Festés	0.37%	0.83%	0.15%	0.10%
Egyéb	20.98%	2.78%	71.76%	2.67%

[a]Ebben az esetben a háromszögelés ki volt kapcsolva.

6. táblázat. Különböző böngésző fázisok aránya a GIS rétegsor megjelenítése közben (Farkas, 2019).

folytatott interakciók során (pl. mozgatás, nagyítás, forgatás) a felhasználók az animációs FPS értékeket tapasztalják. A rajzolási sebesség csak akkor tapasztalható, ha a felhasználó megállítja a térképet. Így ha az animációs sebesség elég nagy ahhoz, hogy folyamatos mozgóképet produkáljon, míg a rajzolási folyamat megállítja az alkalmazást fél másodpercre, az jobb felhasználói élményt nyújt, mintha az alkalmazás az animációk során akadozna.

Főleg ez az oka annak, hogy egy naiv, nem tökéletesen optimalizált WebGL motornak is nagy hatása van. A pufferek (háromszögelési eredmények) tárolásával jelentős teljesítménybeli fölényt ér el a Canvas alapú megjelenítővel szemben. Mivel az utóbbi nem tud ilyen alacsony szinten részeredményt tárolni, ott az animációk közbeni sebességnövekedés nem annyira jelentős.

Az egyetlen eset, ahol komoly elakadások mutatkoztak, az okostelefonban lévő ARM Mali GPU volt. Ez egy gyenge GPU egy elavult készülékben, így jól reprezentálja a gyenge számítógépeket. Amíg a Canvas motorral megjelenített térkép ezen a GPU-n animáció közben is erősen akadozott, addig a WebGL motor kisebb elakadásokkal képes volt kezelni a felhasználói interakciókat. Így elmondható, hogy a WebGL technológia használatának fontos szerepe van az alkalmazás eszközfüggetlenségében, mivel így az kevésbé függ a használt eszköz korától és számítási kapacitásától.

A GIS rétegsor mérésével (6. táblázat) részletesebben megmutatkoztak a WebGL motor tulajdonságai. Ebben az esetben a poligon megjelenítés a háromszögelés kikapcsolásával is mérésre került, ugyanis a háromszögelés eredményei eltárolhatóak. Az eredmények szerint a háromszögelés tárolásával a WebGL motor minden tekintetben előnyösebb a Canvas motornál ilyen terhelés alatt. Van egy átfordulási pont, ahol még egy naiv WebGL implementáció is megelőz egy jól optimalizált Canvas implementációt, és a GIS rétegsor már ezen a ponton túl van.

4.3. Raszterkezelés

Az effektív raszterkezelés eléréséhez nem csupán új osztályok létrehozására volt szükség, hanem az egész folyamat újragondolására is. A tradicionális raszteres adatok mátrixok négyzetes rácshálóra képezve. A technológia fejlődésével a raszterek

definíciója nem változott. Még mindig hasznosak, mivel folyamatos térbeli jelenségek hatékony ábrázolására interpretáció szükségessége nélkül alkalmasak (Bugya & Farkas, 2018). Azonban a technológia fejlődésével új igények merültek fel a raszteres adatkezeléssel szemben. Ezek közül néhányat sikeresen meg tudott oldali a hagyományos rasztermodell, mint a téglalap alakú cellák engedélyezése, vagy a 3D raszterek (voxelek) támogatása. Azonban a rácsháló helyett más mintázatokat megjeleníteni még mindig nem lehet, míg hexagonális fedvényeket már régóta aktívan használnak számos területen.

Megfigyelhető, hogy a rasztermodell előnyei az adatmodellből származnak, míg a korlátai a reprezentációs modellből (Bugya & Farkas, 2018). Egy kevésbé korlátozott reprezentációs modellel a korlátok nagy részét ki lehetne váltani, míg az adatmodellből származó előnyök változatlanul megmaradnának. Mivel a Weben még nincs bevett módja a geoinformatikai raszterek kezelésének, az új raszterkezelő jó alapot biztosított egy ilyen módszer implementálására. Az új modell a fedvénymodell nevet kapta az OGC WCS (Web Coverage Service) szolgáltatása nyomán, mely hasonló megfontolások alapján szolgáltat raszteres adatot.

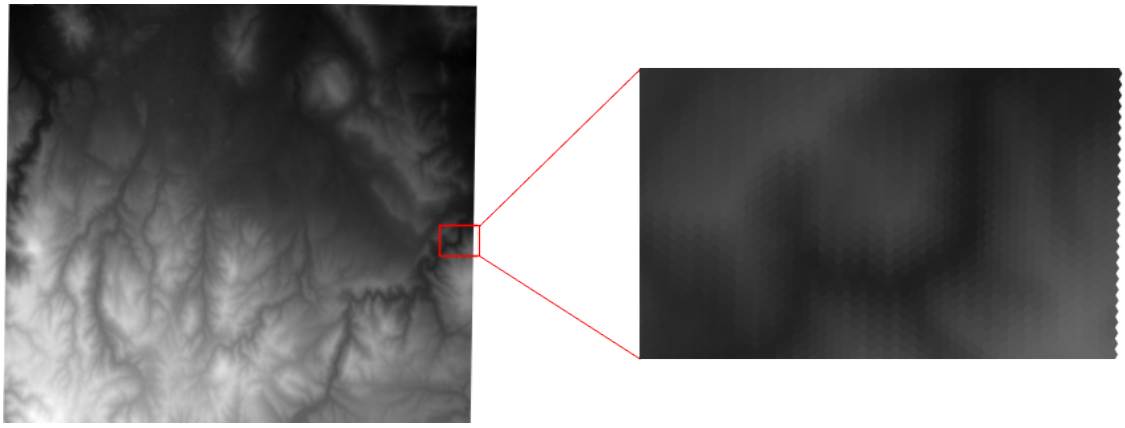
A fedvénymodell megtartja a raszterek adatmodelljét, tehát mátrix alapú, azonban vektorosan jeleníti meg a cellákat. Minden cellát egy különálló poligonként kezel körvonal nélkül. Ezzel a módszerrel a megjelenítés lassabb, de minden cella transzformálható és vetíthető. Továbbá lehetőséget biztosít különböző mintázatok használatára. Az egyetlen követelménye, hogy legyen egy egyértelmű leképezés a mátrix tagjai és a fedvény cellái között. Ez a leképezés a mintázat nevet kapta, ami leírja az egymást követő cellák eltolását és forgatását.

A fedvénymodell segítségével a raszterek természetesebben vannak kezelve; a vektormodell szélső eseteiként. A különböző esetek a mintázatuk szabályossága fényében optimalizálhatóak. Amíg a mintázat négyzetes, a hagyományos rasztermodell összes előnyét birtokolja, a hátrányai egy részével együtt. Hexagonális fedvények esetében a hátrányok közül kevesebb jelenik meg, de még mindig jól optimalizálhatóak és gyorsan feldolgozhatóak (Her, 1995). Ahogy a szabályosság csökken, úgy realizálódik egyre több korlát és egyre kevesebb előny, míg végül egy olyan vektoros réteg marad, amiben csak a cellák folytonos térkitöltése garantált.

Az implementáció első lépése az ősosztályok és alap funkcionalitások megírása volt a raszter– és fedvénymodellekre egyaránt. Ezek az osztályok főleg a hagyományos raszterkezeléshez tartoznak, mivel a fedvénymodell csak megjelenítésben tér el a rasztermodelltől. Ezek az osztályok az adatmodellhez tartozó tárolók, stílusozáshoz tartozó osztályok, megjelenítők, valamint réteg és forrás osztályok az OpenLayers-be történő integrációhoz.

Mivel az OpenLayers egy teljesen funkcionális képi megjelenítővel rendelkezett, és mivel a hagyományos rasztermodell textúrákat használ a cellaadatok megjelenítéséhez, a hagyományos raszterekhez nem volt szükség egyedi megjelenítőre. A raszter réteg csupán a képi réteg leszármazottja, így a kettő ugyanazt a megjelenítőt használja. Hagyományos raszterekhez két formátumkezelő osztály került implementálásra: a GeoTIFF és az ArcGrid.

A fedvénymegjelenítő sokkal több új logikát tartalmaz a rasztermegjelenítőnél.



4. ábra. A Spearfish60 domborzatmodell HexASCII fedvényként megjelenítve (Farkas, 2020).

Ugyanazokkal az ősosztályokkal dolgozik, azonban jelentős mennyiségű saját kódot is tartalmaz. Az egyik ezek közül egy hexagonális formátumkezelő osztály. A HexASCII (de Sousa & Leitão, 2017) egy ArcInfo ASCII Grid alapú formátum hexagonális fedvények (4. ábra) tárolására. Az adatmátrixot ASCII formátumban tárolja, a metaadatokat pedig a fájl elején fejlécként. A fejléc tartalmazza a szükséges információkat a cellák kirajzolásához.

A megjelenítési folyamat gyorsításához piramisok és R-fák egyaránt felhasználásra kerültek. Amikor egy réteg betöltődik, a megjelenítő minden celláját indexeli minden piramisszinten. Alapértelmezetten maximum 10 szintet generál le a program. Így az eljárás végére minden szint tartalmaz egy R-fát, melyből gyorsan le tudja kérdezni a megjelenítő az éppen látható cellákat.

A térindex használata gyakorolja a legnagyobb hatást a sebességre. Enélkül a lépés nélkül mindkét motor (Canvas, WebGL) használhatatlan volt, míg térbeli indexeléssel a WebGL motor használhatóvá vált (7. táblázat).

A WebGL megjelenítőben az első optimalizáció a színek átadásának módja volt. Amíg uniform tulajdonságként kerültek a GPU-ba, nem volt szükség annyi memóriára, de a kirajzolási sebesség a színek gyakori változtatása miatt nagyon lelassult. Vertex tulajdonságként négyszeres memóriamennyiségre van szükség, de az animációs sebesség egy nagyságrenddel javult. A végső optimalizáció a piramisok implementálása volt, mely kisebb méretarányok esetében gyorsította fel a megjelenítést. Ez főleg a rajzolási sebességre volt hatással, mivel animációk közben tárolásra kerülnek a megjelenített vertex pufferek.

Az előkészítési fázis mindkét rétegfajta esetében a rétegek stílusozásával és gyorsítótárazással kapcsolatos eljárásokat foglalja magában. Hagyományos raszterekénél az előkészítési idő kicsi (8. táblázat). A megjelenítés még ennél is gyorsabb, és a réteg memórialábnyoma minimális. A két fázis kombinációja azonban már túl lassú folyamatos animációk készítéséhez a réteg képkockánkénti átstílusozásával. Kisebb rétegek esetén, mint a Spearfish60, azonban ez is lehetséges.

Fedvények esetében csupán a Spearfish60 domborzatmodell került mérésre, négyzetes és hexagonális rétegként egyaránt. Sajnos az eljárás optimalizálatlansága mi-

	Rajzolás	Animáció
Canvas motor		
Idő	2202.1 ms	1678.6 ms
Teljesítmény	0.5 fps	1.7 fps
Memória	112.5 MiB	
WebGL motor (színek uniform-ként)		
Idő	3009.6 ms	410.2 ms
Teljesítmény	0.3 fps	2.4 fps
Memória	30.8 MiB	
WebGL motor (színek vertex tulajdonságként)		
Idő	2299.1 ms	48.1 ms
Teljesítmény	0.4 fps	20.8 fps
Memória	112.0 MiB	

7. táblázat. Teljesítmény és memória adatok két fedvény megjelenítőre a Spearfish60 domborzat-modellel négyzetes fedvényként megjelenítve 12-es nagyítási szinten. Minden eset használ R-fát, míg a WebGL motor két eltérő megoldást használ a cellák színeinek paraméterezésére (Farkas, 2018).

	Előkészítési idő	Rajzolási idő	Memória
Raszter réteg			
Spearfish60	37 ms	3 ms	87 KiB
Baranya műholdkép	343 ms	8 ms	77.5 KiB
Fedvény réteg			
Spearfish60	2579 ms	1 – 1032 ms	152.9 MiB
Spearfish60 (hexagonális)	3001 ms	1 – 1747 ms	170.4 MiB

8. táblázat. Raszter és fedvényrétegek megjelenítési metrikái. A fedvényrétegek esetén a rajzolási idő a piramisszint és a látható cellák függvénye. A tartomány határai tapasztalati legjobb és legrosszabb értékek (Farkas, 2020).

Szint	Celák száma	Idő	Memória	Halom memória
1	292 220	1032 ms	111.8 MiB	365 (+172) MiB
2	73 181	265 ms	31.0 MiB	193 (+17) MiB
3	18 230	96 ms	7.6 MiB	176 (+5) MiB
4	4 468	16 ms	1.9 MiB	171 (+6) MiB
5	1 026	18 ms	478.2 KiB	165 (+1) MiB
6	169	5 ms	114.1 KiB	164 (+0) MiB
7	63	3 ms	27.4 KiB	164 (+0) MiB
8	12	1 ms	5.7 KiB	164 (+0) MiB
9	2	1 ms	1.2 KiB	164 (+0) MiB

9. táblázat. Különböző piramisszintek megjelenítési metrikái a Spearfish60 négyzetes fedvénymodell esetén (Farkas, 2020). A halom által elfoglalt memória azért került mérésre, mert nagyobb nagyítási szinteken a böngésző kifogyott a memóriából pontos mérések esetén.

att van még néhány memóriaigényes lépés, melyek miatt a Baranya megyei műholdfelvétel feldolgozása nem volt lehetséges. Ez az eljárás skálázhatatlanságára utal, és a hosszú útra ami még előtte el mire több lesz egy prototípusnál.

A piramisszintek mérési eredményei (9. táblázat) alapján a részletesebb képeket nem csak sokkal több idő kirajzolni, de exponenciálisan több memóriát is igényelnek. Ez azért problémás, mert a mért réteg nagyon kicsi tipikus valós adatokhoz képest. A halom teljes memóriaigénye mellett zárójelben található az alkalmazás által megjelenítés közben lefoglalt memóriamennyiség. Mivel ez a memóriamennyiség az OpenLayers belső megjelenítési kialakításából származik, nehezebb skálázhatóbbá tenni.

A tanulságokat levonva, több mód van a fedvénymodell skálázhatóbbá tételére. A térindex használata elkerülendő, amennyiben lehetőség van rá. Négyzetes és hexagonális rácshálók esetében térképi koordináták könnyedén átalakíthatóan mátrixon belüli sor– és oszlopszámmá. Egyedi mintázatok esetén azonban félő, nincs könnyű módja a térindex elkerülésének.

Alternatív megoldásként a térindex foglalhatna kevesebb memóriát. Ha minden bejegyzés csupán a cellák középpontját és színét tartalmazná, a négyzetes cellák memóriaigénye 70%-al csökkenne. Ebben az esetben a több sarokszámmal rendelkező cellák még inkább kamatoznának a memórialábnyom-csökkentésből. Azonban ehhez a módszerhez a GPU-nak pontosan kell tudnia, hogyan kell kirajzolnia a cellát annak középpontja ismeretében.

5. Összefoglalás

A dolgozat felmérte egy általános Web GIS kliens megépítésének lehetőségeit már létező technológiákat felhasználva. Mivel erre a célra nem volt jó megoldás, a legfontosabb hiányosságok implementálásra kerültek a legalkalmasabb könyvtárba. Ezek a hiányosságok, melyek mindenképp szükségesek egy GIS programhoz és az OpenLayers-ből hiányoztak a teljes hardveres gyorsítás és a raszterkezelés. Ezzel a két kiegészítéssel már létezik egy olyan alap, mely felhasználható általános Web GIS megoldások készítésére.

A legjobb alap keresése közben több összehasonlítási szempont is vizsgálat alá került. Mivel cél volt a kérdőívezés elkerülése, szoftvermetrikák képezték a vizsgálat alapját. Az összehasonlítás alapján elmondható, hogy jól kiválasztott statikus szoftvermetrikákkal, valamint olyan puha tulajdonságok kiértékelésével, mint a dokumentáció vagy a közösség, sikeresen össze lehet hasonlítani különböző JavaScript könyvtárakat. Ezen felül egy új metrika is született, a Hozzávetőleges Tanulási Görbe JavaScript-re. Az ALC_{JS} durva becslést tud adni a különböző könyvtárak komplexitására, mellyel kizárhatóak a kiugróan komplex vagy kiugróan egyszerű könyvtárak.

Egy jól működő hardveresen gyorsított motor megírása nem triviális feladat, mivel a megoldásnak gyorsnak és elég általánosnak is kell lennie. Feltételezhető, hogy számos olyan akadály létezik, mely csak a későbbi, optimalizációs fázisban fog felbukkanni. Ennek ellenére egy alap szintű, de működőképes megjelenítőt sikerült készíteni, mely kartográfiai célokra kevésbé alkalmas, mint a Canvas motor, de nagyobb adatmennyiségeknél teljesítményben megelőzi azt.

Kiderült, hogy a Canvas megjelenítő elégséges olyan webes térképekhez, melyeknél a szerver oldali adatmennyiség kontrollálható. Így a megfelelően generalizált, esetleg vektor csempéket használó térképek megfelelően használhatóak vele. Körülbelül 2000 elemig vagy 60 000 csúcspontig jobb teljesítményt nyújt, mint a WebGL megjelenítő, valamit több stílusozási opcióval is rendelkezik. Amikor a Canvas megjelenítő használatához már túl sok adatot kell megjeleníteni, akkor a WebGL motor egy hasznos alternatívaként jelenik meg.

Feltehetően a dolgozat legjelentősebb része a raszterkezelés újragondolása. Az asztali megoldások jól bevált, hosszú ideig fejlesztett könyvtárakon alapulnak, melyek jól optimalizáltak, kevés hiba van bennük. Ezek a könyvtárak (pl. GDAL) olyan jó munkát végeznek, hogy kiváltani őket egy új rendszerre a felmerült új igények miatt nem logikus döntés. A Webes környezet viszont még fiatal, nem rendelkezik hasonlóan bevett megoldásokkal, és mivel más fajta limitációi vannak, sok

hagyományos probléma újfajta megoldást kíván. Ez teszi a Webet egy nagyon jó alappá új koncepciók kikísérletezésére. Ha egy technika, megközelítés, vagy alkalmazás a Weben beválik, talán nagyobb lesz az érdeklődés a más környezetekbe való átültetés iránt.

Egy ilyen koncepció példája a fedvénymodell. Míg a rasztermodell népszerűsége az előnyei fényében érthető, nagyon sok limitációval rendelkezik. Az igények alternatív megoldásokra nem voltak eddig elég erősek a modell újragondolásához, azonban mostanában egyre nagyobb hangsúly tevődik a hexagonális raszterek bevezetésére és használatára. A meglévő rasztermodell megfelelő generalizálásával az új modell stabilabb lesz, annak későbbi bővítésére már kevesebb eséllyel lesz igény. A fedvénymodell egy ilyen újragondolást mutat be a rasztermodell vektoros alapon történő generalizálásával.

Elsődleges eredmények azt mutatták, hogy bár a jelenlegi fedvénymodell implementáció még nem használható valós adatokkal, megfelelő szintű optimalizálással az lesz. A prototípus demonstrálta a hexagonális fedvények használhatóságát nagy méretű és topológiailag problémás vektoros adatstruktúrák nélkül. Mivel a fedvények soha nem lesznek olyan gyorsak, mint a textúra alapú raszterek, a fedvénymodellnek nem célja lecserélni a hagyományos rasztermodellt. Ehelyett inkább kiegészíti azt, egy hibrid megoldást nyújtva azokra a helyzetekre, amikor a négyzetes rácshálónál bonyolultabb mintázatra van szükség. Mobilkészülékek és beépített eszközök esetén, amikor a számítási kapacitás és a memória limitált, esetleg az akkumulátor merülése számottevő tényező, a textúra alapú raszterek jobb megoldást fognak nyújtani, mint a fedvények.

Irodalomjegyzék

Agrawal, S., & Gupta, R. D. (2014). Development and Comparison of Open Source Based Web GIS Frameworks on WAMP and Apache Tomcat Web Servers. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL*(4), 1–5.

Albrecht, J. (1998). Universal analytical GIS operations – a task-oriented systematization of data structure-independent GIS functionality. *Geographic information research: Transatlantic perspectives*, 577–591.

Bugya, T., & Farkas, G. (2018). An Alternative Raster Display Model. In C. Grueau, R. Laurini, & L. Ragia (Eds.), *Proceedings of the 4th international conference on geographical information systems theory, applications and management (gistam 2018)* (pp. 262–268).

de Sousa, L. M., & Leitão, J. P. (2017). HexASCII: A file format for cartographical hexagonal rasters. *Transactions in GIS*, *22*, 217–232.

Farkas, G. (2015). *Comparison of Web Mapping Libraries for Building WebGIS Clients* (Unpublished master's thesis). Pécsi Tudományegyetem, Pécs.

Farkas, G. (2017). Applicability of open-source web mapping libraries for building massive Web GIS clients. *Journal of Geographical Systems*, *19*(3), 273–295.

Farkas, G. (2018). Towards visualizing coverage data on the Web. In *Az elmélet és a gyakorlat találkozása a térinformatikában ix.: Theory meets practice in gis.* (pp. 107–113).

Farkas, G. (2019). Hardware-Accelerating 2D Web Maps: A Case Study. *Cartographica*, *54*(4), 245–260.

Farkas, G. (2020). Possibilities of using raster data in client side Web maps. *Transactions in GIS*, *24*(1), 72–84.

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional.

Gong, L., Pradel, M., & Sen, K. (2015). JITProf: pinpointing JIT-unfriendly JavaScript code. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 357–368).

Grigorik, I., Mann, J., & Wang, Z. (2016). *Performance timeline level 2* (Candidate Recommendation). W3C.

Her, I. (1995). Geometric transformations on the hexagonal grid. *IEEE Transactions on Image Processing*, *4*(9), 1213–1222.

Maguire, D. J. (1991). An overview and definition of GIS. *Geographical information systems: Principles and applications*, *1*, 9–20.

Meaden, G. J., & Chi, T. D. (1996). *Geographical information systems Applications to marine fisheries*. Egyesült Nemzetek Szervezetének Élelmezésügyi és Mezőgazdasági Szervezete, Róma.

O'Reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, *65*(1), 17–37.

Orlik, A., & Orlikova, L. (2014). Current Trends in Formats and Coordinate Transformations of Geospatial Data – Based on MyGeoData Converter. *Central European Journal of Geosciences*, *6*(3), 354–362.

Poorazizi, M. E., & Hunter, A. J. (2015). Evaluation of Web Processing Service Frameworks. *OSGEO Journal*, *14*, 29–42.

Ramsey, P. (2007). *The State of Open Source GIS* (Tech. Rep.). Refractions Research Inc.

Roth, R. E. (2017). Visual variables. In D. Richardson, N. Castree, M. F. Goodchild, A. Kobayashi, W. Liu, & R. A. Marston (Eds.), *International encyclopedia of geography: People, the earth, environment and technology* (pp. 1–11).

Roth, R. E., Donohue, R., Sack, C., Wallace, T., & Buckingham, T. (2014). A Process for Keeping Pace with Evolving Web Mapping Technologies. *Cartographic Perspectives*, *0*(78), 25–52.

Steiniger, S., & Hunter, A. J. (2013). The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, *39*, 136–150.

Taivalsaari, A., Mikkonen, T., Anttonen, M., & Salminen, A. (2011). The death of binary software: End user software moves to the web. In *Creating, connecting and collaborating through computing (c5)* (pp. 17–23).

Thrall, S. E., & Thrall, G. I. (1999). Desktop GIS software. In P. A. Longley, M. F. Goodchild, D. J. Maguire, & D. W. Rhind (Eds.), *Geographical Information Systems Abridged* (pp. 331–345). John Wiley & Sons, Inc.

Tomlin, C. D. (2017). Cartographic modeling. In D. Richardson, N. Castree, M. F. Goodchild, A. Kobayashi, W. Liu, & R. A. Marston (Eds.), *International encyclopedia of geography: People, the earth, environment and technology* (pp. 1–6).