Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vizualizace geografických dat ve webovém prostředí

Bakalářská práce

Autor: Tomáš Kábrt
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Bruno Ježek, Ph.D.

Hradec Králové duben 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

……..

V Hradci Králové dne 21.4.2023 Tomáš Kábrt

Poděkování:

Děkuji mému vedoucímu bakalářské práce Ing. Bruno Ježkovi, Ph.D. za odborné vedení práce a vstřícnost.

Anotace

Bakalářská práce se zabývá vizualizací geografických dat ve webovém prostředí. Cílem práce je popsat a využít dostupné technologie pro vytvoření interaktivních vizualizací geografických dat běžících v reálném čase. V teoretické části je představena problematika geografických dat, tvorba nepravidelných trojúhelníkových sítí, technologie WebGL a vizualizační framework deck.gl. Praktická část je rozdělena na návrh a implementaci. Kapitola návrh popisuje požadavky implementace, důvody výběru daných technologií, použitá data a strukturu aplikace. Následující kapitola rozebírá principy použité při tvorbě vizualizací a dosažené výsledky. V poslední kapitole jsou vytvořené vizualizace otestovány a kriticky zhodnoceny.

Klíčová slova

Geografická data, WebGL, GLSL ES, shader, deck.gl

Annotation

Title: Visualization of geographical data on the web

The bachelor thesis deals with the visualization of geographical data on the web. The main objective is to describe and utilize available technologies for creating real-time interactive visualizations of geographical data. The theoretical part introduces the geographical data, the irregular triangulated networks, WebGL and the framework deck.gl. The practical part is divided into proposal and implementation. The proposal describes the requirements for implementation, the reasons for choosing the given technologies, the data and the application’s structure. The following chapter discusses the principles used for creating the visualizations and the results. In the last chapter, the created visualizations are tested and evaluated.

Keywords

Geographical data, WebGL, GLSL ES, shader, deck.gl

Obsah

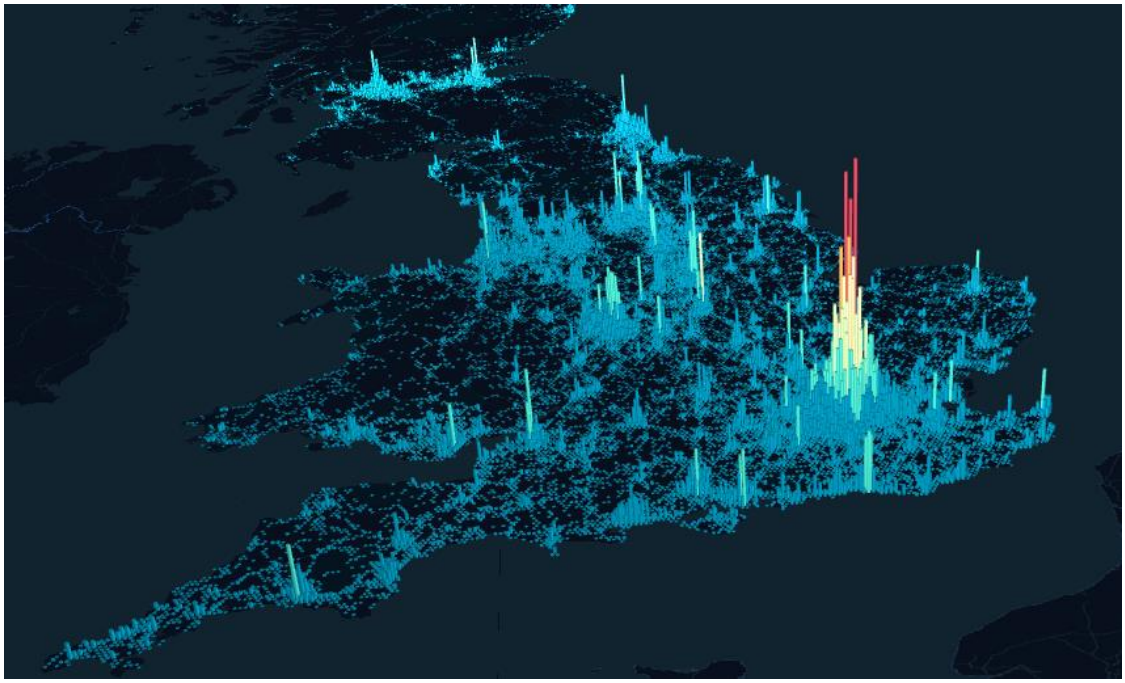
Seznam obrázků

Seznam tabulek

1 Úvod

V posledních letech zaznamenávají webové technologie velký pokrok a s představením grafického rozhraní WebGL je možné vytvářet interaktivní 3D scény, které jsou spustitelné přímo ve webovém prohlížeči. Vizualizace používající webový prohlížeč umožňují uživateli snadný přístup a nekladou specifické nároky na hardware, který používá. Jeden ze způsobů, jak tyto technologie využít je interaktivní vizualizace geografických dat.

Geografická data se vždy vztahují k určité geografické poloze a bývají zobrazovány na 2D nebo 3D podkladových mapách případně na glóbu. Dnes již existuje celá řada softwarových nástrojů, které lze využít pro vizualizaci geografický dat ve webovém prohlížeči. Pro dosažení lepšího výkonu je vhodné, aby daný software uměl pracovat s grafickým hardwarem. Takovým nástrojem je například deck.gl.



Obrázek 1: Ukázka vizualizace ve frameworku deck.gl, počet dopravních nehod se zraněním ve Velké Británii od roku 1979 [1]

Cílem bakalářské práce je prozkoumat způsoby vizualizace geografických dat ve webovém prohlížeči. Popsat zjištěné teoretické poznatky o geografických datech a softwarových knihovnách využívaných k jejich reprezentaci. Navrhnout a implementovat vlastní řešení, které vizualizuje atributy geografických dat v reálném

čase a využívá programovatelné grafické karty. Otestovat a kriticky zhodnotit dosažené výsledky případně navrhnout možná rozšíření.

V teoretické části práce jsou nejprve představena geografická data. V rámci této kapitoly je popsán geografický informační systém společně s hardwarem a softwarem, který ho používá. Dále jsou rozebrány způsoby reprezentace geografických dat především souřadnicové systémy, datové modely a metody modelování terénu.

Pro řešení vizualizačních problémů byly využity algoritmy pracující s nepravidelnými trojúhelníkovými sítěmi, a proto jsou ve druhé kapitole popsány teoretické principy a metody triangulace. Kapitola se zaměřuje zejména na Delaunayho triangulaci a Bowyer-Watsonův algoritmus, který je pro potřeby vizualizací implementován.

Poslední teoretická kapitola je věnována technologiím pro vizualizaci geografických dat ve webovém prostředí. Představeny jsou základní principy WebGL a programovací jazyk GLSL ES. Další představenou technologií je vizualizační framework deck.gl. Knihovna deck.gl je hlavní využívanou technologií pro vytváření vizualizací, z toho důvodu jsou v této podkapitole detailně popsány principy jejího použití, možnosti tvorby vlastní vrstvy a návaznost na související knihovny.

Praktická část začíná kapitolou návrhu řešení, kde jsou popsány použité technologie a data. Následující kapitola rozebírá principy použité při tvorbě vizualizací a dosažené výsledky. V poslední kapitole jsou vytvořené vizualizace otestovány a kriticky zhodnoceny.

2 Geografická data

Geografická data jsou obecná data, která jsou vázána na geografickou polohu na povrchu Země. Poloha je určena souřadnicovým systémem a pro ukládání dat jsou využívány datové modely s definovanými datovými formáty. Geografická data společně s nástroji pro jejich uložení a zpracování tvoří geografický informační systém.

2.1 GIS

Geografický informační systém (GIS) je počítačový systém, který poskytuje nástroje pro práci s prostorovými daty. GIS umožňuje data vkládat, upravovat, spravovat, analyzovat atd. GIS používá celá řada soukromých i veřejných organizací pro zlepšení efektivity jejich služeb. Pro práci s takovým systém je potřeba specifický hardware a software. U hardware jsou kladeny vysoké nároky na výkon, protože často dochází ke zpracování, analýze a případně vizualizaci velkého množství dat ve vysokém rozlišení. GIS software musí nabízet nástroje pro správu, analýzu a efektivní zobrazení prostorových dat. Dnes existuje celá řada veřejných i komerčních GIS softwarů například ArcGIS, QGIS, GeoMedia, MapInfo a další. Standardizací GIS se zabývá organizace OGC (Open Geospatial Consortium), která vydává standarty pro datové formáty, dokumentaci a programy. [2]

Geografické vizualizace bývají složeny ze třech základních částí: podkladové mapy, tematické vrstvy a doplňkových prvků. Podkladová mapa přináší informaci o geografické pozici a je důležité, aby obsahovala přiměřené množství informací o dané oblasti a používala adekvátní mapovou projekci. Aktuálně jsou podkladové mapy dostupné přímo v GIS nebo ve specializovaných mapových softwarech. Tematická vrstva reprezentuje daná data pomocí různých symbolů, vzájemného propojení bodů případně jinými způsoby vizualizace. V některých GIS aplikacích bývají podkladové mapy a tematické vrstvy spojeny do jedné vrstvy. Doplňkové prvky přináší dodatečné informace o vizualizaci například popisek nebo legenda. [3]

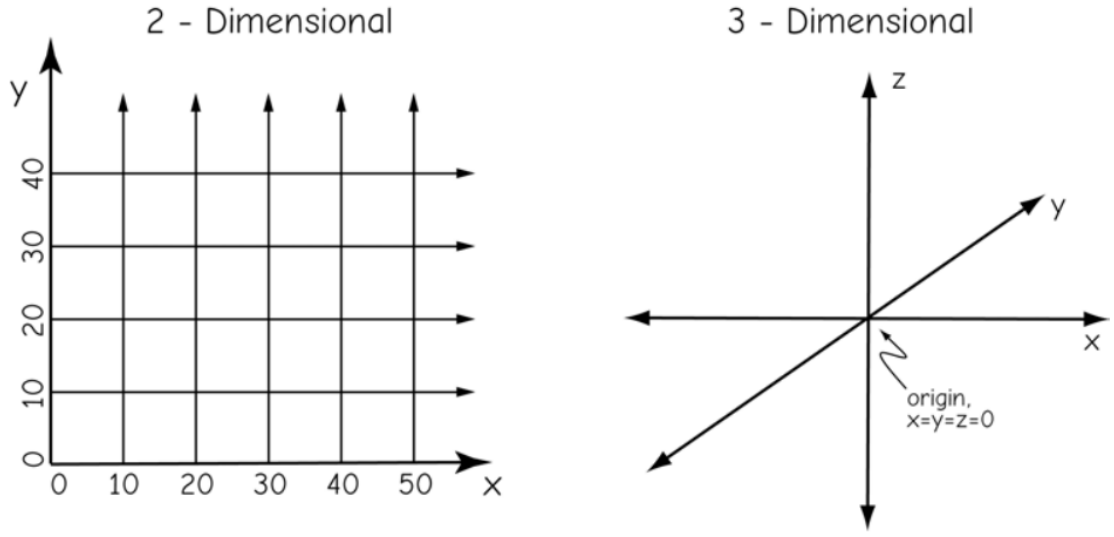
Data v GIS reprezentují objekty reálného světa. Vzhledem k tomu, že každý počítačový systém má své limity, tak při záznamu dat vždy dojde k určitému zjednodušení. Do systému jsou zaznamenány informace o umístění, velikosti a dalších neprostorových vlastnostech daného objektu. Úkolem GIS vývojáře je definovat objekty tak, aby vyhovovaly zamýšlenému účelu použití. [2]

2.2 Souřadnicové systémy

Souřadnice v GIS vyjadřují pozici ve dvourozměrném nebo trojrozměrném prostoru, případně jsou využívány pro definici tvaru prostorových objektů [2]. Souřadnicových systémů existuje celá řada. V kontextu geografických dat bývá nejčastěji používán kartézský souřadnicový systém nebo sférický souřadnicový systém. Většina GIS aplikací nabízí jednoduchý převod mezi těmito systémy [3].

2.2.1 Kartézský souřadnicový systém

Prostorová data v GIS nejčastěji používají kartézský souřadnicový systém. Kartézský dvourozměrný systém bývá používán pro malé oblasti, kde je možné zanedbat zakřivení Země [2]. Pozice v kartézském dvourozměrném systému je vyjádřena pomocí x a y souřadnice, přičemž souřadnice mohou nabývat i záporných hodnot. Většina geografických systémů ovšem nastavuje začátek na jih a západ, tudíž používají pouze kladné hodnoty (pouze první kvadrant viz obrázek 2) [3]. Pro rozsáhlejší oblasti, kde bývá kladen důraz na vyšší přesnost záznamu, je vhodné použít trojrozměrný systém. [2]



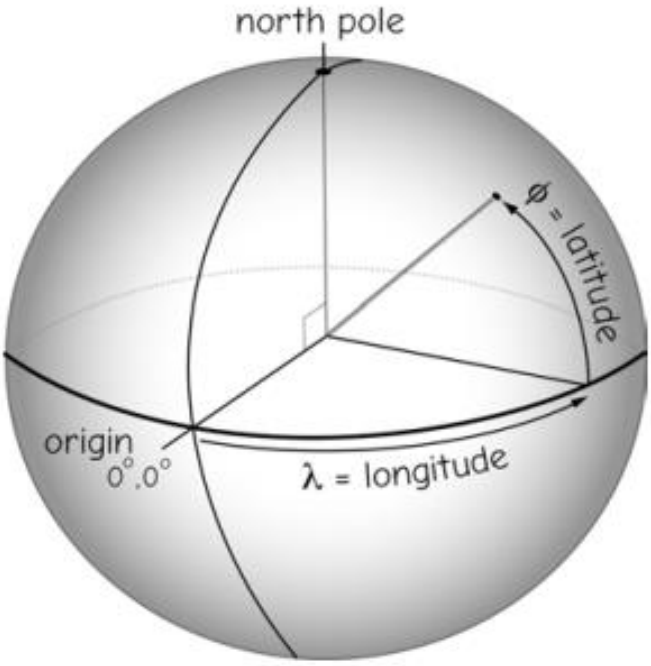
Obrázek 2: Kartézský souřadnicový systém – souřadnicové osy [2]

2.2.2 Sférický souřadnicový systém

Další možností zaznamenání souřadnicových dat je sférický souřadnicový systém (někdy nazýván zeměpisná síť). Geometrie Země je velmi komplexní vzhledem

k jejímu sférickému tvaru. Umístění objektu na model Země může být vyjádřeno pomocí dvou úhlů. Zeměpisné souřadnice v takovém případě používají formát DMS (degrees, minutes, seconds), kdy kruh je rozdělen do 360 stupňů, každý stupeň do 60 minut a každá minuta do 60 sekund. [3]

První úhel longitude (zeměpisná délka) je měřen kolem osy, která prochází přes jižní a severní pól. Nultý (Greenwichský) poledník se nachází v Anglii a úhel je kladný směrem na východ a záporný směrem na západ. Druhý úhel latitude (zeměpisná šířka) svírá rovinu rovníku s přímkou procházející středem Země a daným bodem. Zeměpisná šířka nabývá nulové hodnoty na rovníku a její maximální hodnota je 90 stupňů na severním pólu a minus 90 stupňů na jižním pólu. [2]

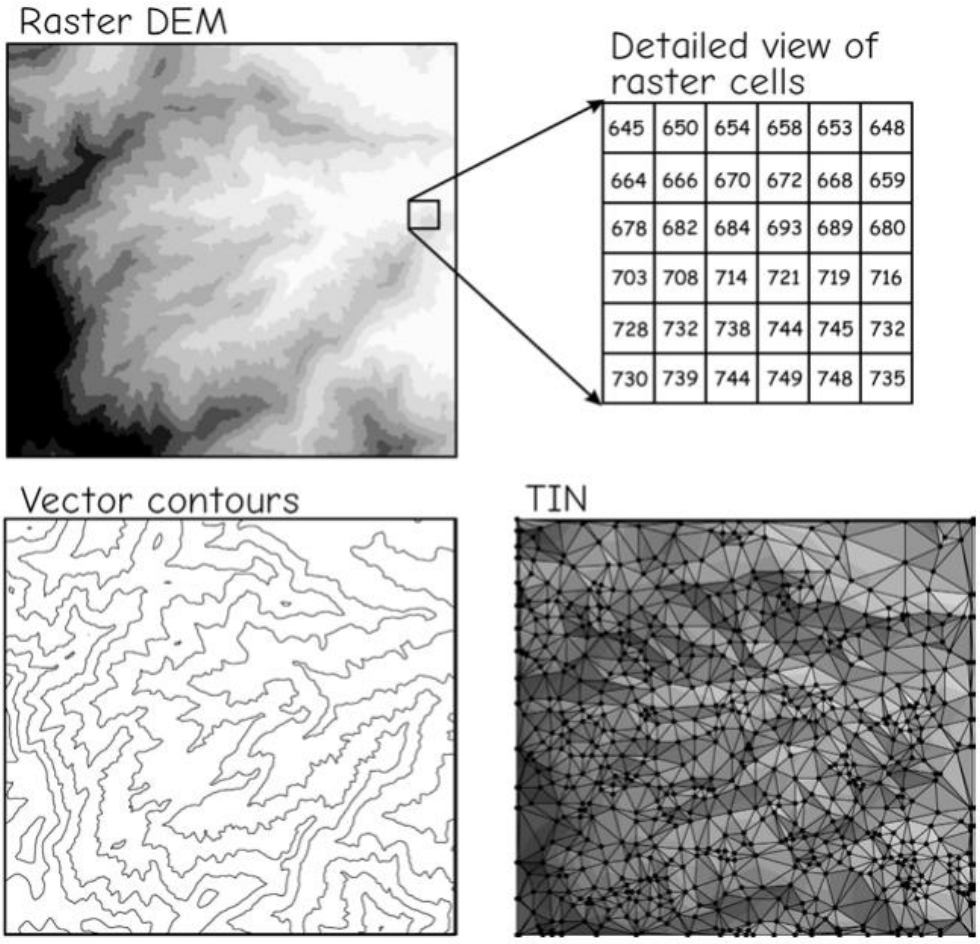


Obrázek 3: Sférický souřadnicový systém – model [2]

2.3 Datové modely a formáty

Mezi nejpoužívanější datové modely řadíme vektorové a rastrové. Vektorové modely jsou založeny na vektorových objektech jako body, linie a mnohoúhelníky. Rastrové modely reprezentují svět jako soubor obrazových elementů uspořádaných do mřížky. Dalším typem vektorových datových modelů je triangulated irregular network, který využívá síť nepravidelných trojúhelníků a je používán k reprezentaci terénů včetně výškové souřadnice. [2]

Po výběru datového modelu je důležité zvolit vhodný formát výsledného souboru. Volba formátu se bude odvíjet od požadavků konkrétní geografické vizualizace (interaktivita, animace, typ mapy) a použité platformy (web, desktopová aplikace).



Obrázek 4: Údaje o nadmořské výšce reprezentované v různých datových modelech [2]

2.3.1 Vektorové formáty

Vektorové formáty bývají používány pro většinu symbolů a dalších elementů tematické vrstvy. Hlavními výhodami vektorových formátů je jednoduchá změna velikosti, možnosti interaktivity a menší velikost oproti rastrovým formátům. [3]

Nejběžněji používaným vektorovým formátem pro geografická data je shapefile. Shapefile se stal standardem a je podporován většinou GIS aplikací. Pro práci s ním je potřeba kompletní sada tří souborů: SHP, SHX a DBF. Soubory s příponou SHP

obsahují geometrii objektu. V souborech typu SHX jsou uloženy indexy tvaru. Databázový soubor DBF uchovává informace o atributech jednotlivých objektů. [4]

Webové aplikace používají převážně formát GeoJSON (Geographic JavaScript Object Notation). GeoJSON vychází z formátu JSON a informace ukládá jako text. Z tohoto důvodu může být upraven jakýmkoliv textovým editorem [4]. GeoJSON umožňuje uchovávat typ geometrie spolu se souřadnicemi a případně dalšími vlastnostmi objektu. Typy geometrie podporované formátem GeoJSON jsou: point, lineString, polygon, multiPoint, multiLineString a multiPolygon [5].

Mezi vektorové formáty řadíme také formáty odvozené od XML, které stejně jako JSON ukládají informace v podobě textu. Příkladem může být GML (Geography Markup Language). GML zaznamenává pro každý objekt seznam vlastností a geometrie. Při porovnání s formátem GeoJSON vytváří pro stejné množství dat větší soubory. Dalším formátem založeným na XML je KML (Keyhole Markup Language). KML bývá používán především pro aplikaci Google Earth. [4]

2.3.2 Rastrové formáty

Rastrové formáty jsou vhodné pro letecké snímky, skeny map a dokumentů, fotografie ad. Mezi jejich hlavní výhody patří rychlost a jednoduchost použití. Většina GIS aplikací umožňuje rychlý export do rastrových formátů a výsledné soubory mohou být jednoduše načteny i na webových stránkách. Hlavním limitem rastrových formátů je fixní rozlišení a menší možnosti interaktivity. [3]

Běžně využívaný rastrový formát pro geografická data je IMG. IMG vyvíjí společnost Hexagon Geospatial a slouží převážně pro ukládání satelitních dat. Zároveň umožňuje zaznamenat informace o vlastnostech souboru, typu snímacího senzoru nebo kontrolních bodech měření. [4]

Další příkladem může být formát GeoTIFF, který se používá pro aplikace satelitního průzkumu Země. Tento rastrový formát bývá doprovázen dalšími typy souborů jako: TFW, XML, AUX nebo OVR. Soubory s příponou TFW poskytují informaci o geolokaci rastru, umístění, natočení a zvětšení daného čtverce vůči definované soustavě souřadnic. V souborech typu XML bývají uložena metadata. AUX soubory ukládají informace o projekci dat. Pyramida rastru bývá uchovávána v souborech s koncovkou OVR. [4]

Pro ukládání geografických dat nemusí být vždy využity specializované datové formáty, ale je běžnou praxí používat i standartní formáty jako je například JPEG nebo PNG. Ve formátu JPEG bývají ukládány různé snímky terénu. Při používání JPEG je potřeba dát si pozor na jeho ztrátovou kompresi, díky které mohou být ztraceny některé informace, které už nelze zpětně zrekonstruovat. Pro zachování všech informací je vhodné použít formát jako PNG, jenž využívá bezeztrátovou kompresi. Nevýhodou PNG může být jeho větší velikosti oproti JPEG. [3]

2.3.3 Další typy formátů

Dalším typem mohou být 3D formáty. 3D formáty ukládají kromě XY souřadnic také informaci o hloubce. Mezi 3D formáty lze zařadit například COLLADA, ve které jsou 3D objekty ukládány pomocí XML. Specifické typy měření vyžadují vlastní formát. Příkladem mohou být lidarové formáty, které ukládají informace jako síť souřadnicových bodů s informacemi o nadmořské výšce. Různé GIS aplikace mají své vlastní specifické formáty. Jako příklad lze uvést formát MXD, který používá aplikace ArcGIS. [4]

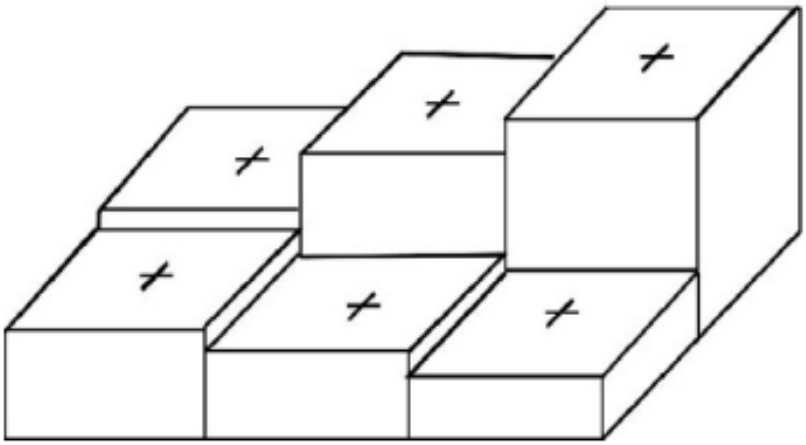
2.4 Modelování terénu

Modelování terénu je jednou z důležitých funkcionalit GIS aplikací. Digitální modely terénu reprezentují zemský povrch jako 3D model získaný interpolací z naměřených dat. Způsob interpolace bude záviset na typu vzorkovací metody, která byla použita při měření vybraných bodů. Obecně platí, že výsledný model by se měl co nejvíce přiblížit reálné podobě daného terénu, ale zároveň by měl minimalizovat nároky na velikost ukládaných dat. [6]

Techniky digitálního modelování terénu mohou být rozděleny podle řady různých kritérií. Podle použitých geometrických útvarů lze techniky rozdělit na: bodové, trojúhelníkové a mřížkové modely, případně různé kombinace těchto variant. Mřížkový model je možné rozložit na trojúhelníkový a zároveň trojúhelníkový model lze pomocí interpolace převést na mřížkový. V praxi jsou nejčastěji používány trojúhelníkové modely. Použití bodových modelů nebývá zpravidla velmi praktické. [7]

2.4.1 Bodové modely

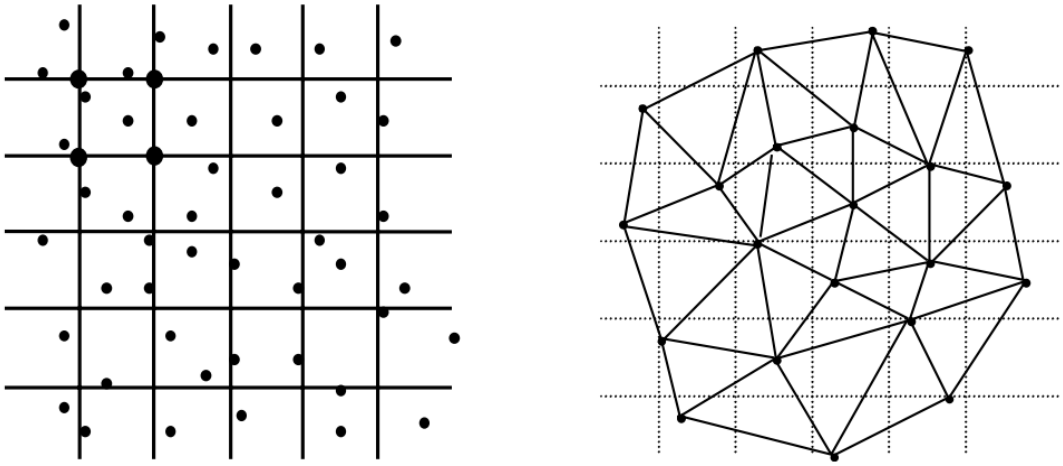
U bodového modelu terénu má každý bod přidělenou oblast v jeho okolí. Každé oblasti je přiřazena výška podle daného bodu. Výsledný model bude reprezentovat terén jako řadu na sebe nenavazujících rovin (viz obrázek 5). Při vytváření modelu je nutné definovat hranice jednotlivých oblastí. K definici hranic lze použít například Voroného diagram. Bodové modely jsou velmi jednoduché a jsou použitelné pro téměř jakákoliv data. Pro modelování terénu tento způsob není vhodný z důvodu nespojitosti modelu. [7]



Obrázek 5: Bodový model terénu [7]

2.4.2 Mřížkové modely

Mřížkové modely zobrazují terén jako sérii na sebe navazujících čtyřúhelníků a jsou vhodné pro práci s daty uloženými v organizované datové struktuře, jako je pole. Pro tvorbu mřížkového modelu lze použít čtyřúhelníky libovolného tvaru, například rovnoběžníky, obdélníky, čtverce atd. Z praktických důvodů bývají nejčastěji používány čtverce [7]. Přesnost mřížkových modelů závisí na intervalu vzorkování. Při vysokém intervalu vzorkování mohou být ztraceny informace o extrémech daného terénu. Příliš malý interval může vést ke zbytečně velkému objemu naměřených dat. Mřížkové modely jsou vhodné pro reprezentaci povrchů s určitou pravidelností [6]. Mřížkové modely lze použít i pro nepravidelně rozmístěná data. V takovém případě je nutné provést interpolaci typu „random-to-grid“ nebo nepřímou interpolaci z trojúhelníkové sítě. [7]



Obrázek 6: Mřížkové sítě: vlevo – random-to-grind interpolace, vpravo – nepřímá interpolace triangulací [7]

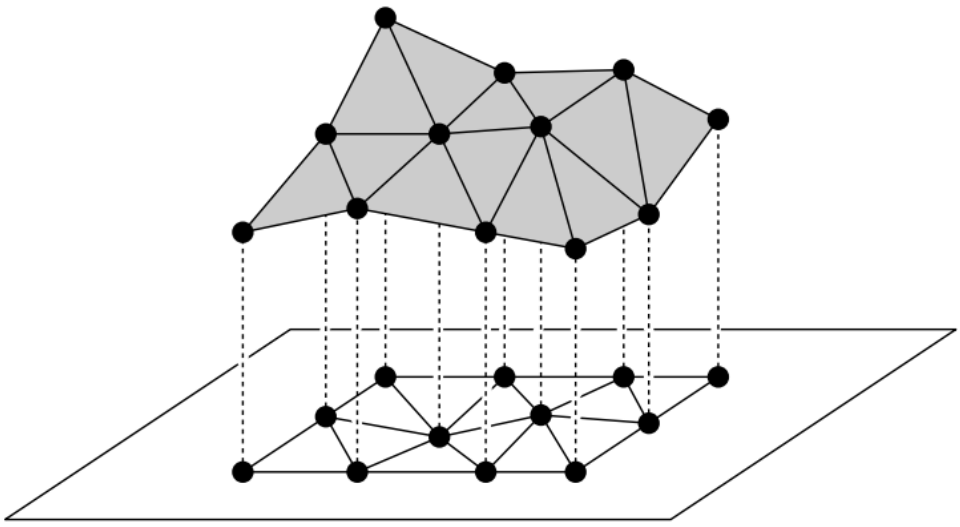
2.4.3 Trojúhelníkové modely

Trojúhelníkové modely reprezentují terén jako sérii na sebe navazujících trojúhelníků. Trojúhelník bývá považován za základní geometrický obrazec pro tvorbu modelů, protože jakýkoliv mnohoúhelník je možné rozdělit na sérii trojúhelníků. Zároveň u trojúhelníků může být jednoduše změněn jejich tvar a velikost [7]. Mezi klíčové výhody trojúhelníkových modelů lze zařadit nezávislost na rozlišení, efektivitu ukládání dat, zahrnutí všech naměřených bodů do modelu a kvalitní zaznamenání extrémů daného terénu. Z těchto důvodů jsou trojúhelníkové modely považovány za hlavní metodu pro modelování terénu [6].

Datová struktura implementující určitý geometrický vzor při modelování terénu je označována jako síť. Pojem triangulace označuje proces tvorby trojúhelníkové sítě. V závislosti na pravidelnosti rozmístění dat může triangulací vzniknout pravidelná trojúhelníková síť (triangulated regular network – TRN) anebo nepravidelná trojúhelníková síť (triangulated irregular network – TIN). Triangulace pravidelné trojúhelníkové sítě je poměrně jednoduchá. V případě, že data tvoří čtvercovou mřížku, tak při triangulaci stačí ke každému čtverci přidat jeho úhlopříčku. Vytvoření nepravidelné trojúhelníkové sítě je výrazně složitější a pravidla pro její tvorbu budou detailně rozebrány v následující kapitole. [7]

3 Nepravidelné trojúhelníkové sítě

Modely založené na nepravidelných trojúhelníkových sítích (TIN) jsou populární metodou reprezentace zemského povrchu. Návaznost trojúhelníků zaručí spojitost modelu a umožní namodelovat i nerovnosti terénu [6]. Při vytváření modelu povrchu Země pomocí nepravidelných trojúhelníkových sítí bývá nejprve zobrazena dvourozměrná trojúhelníková síť v trojrozměrném prostoru a následně ke každému bodu sítě je přiřazena odpovídající výška, přičemž předpokládáme, že výsledná síť pokryje celou naměřenou oblast (viz obrázek 7). Vzhledem k tomu, že Země není plochá, tak je vhodné tuto metodu používat pouze u menších oblastí, kde můžeme zanedbat zaoblení Země. Při měření povrchu zpravidla není možné získat výšku každého bodu na Zemi. Pokud je cílem získat bližší představu o daném terénu, tak je vhodné z výšky naměřených bodů aproximovat výšku dalších bodů [8].

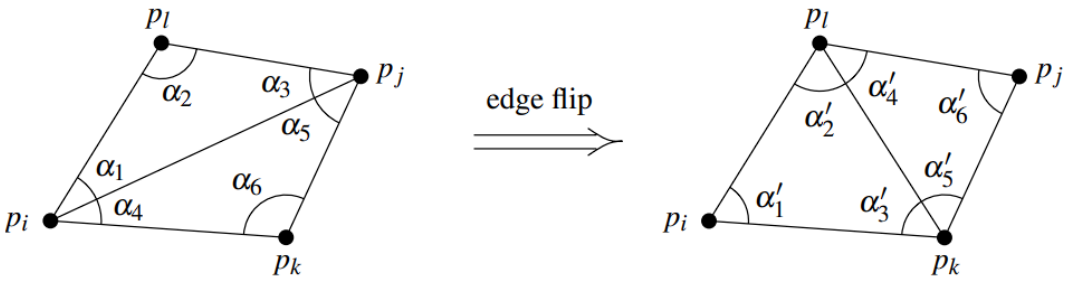


Obrázek 7: Získání TIN z množiny naměřených bodů [8]

3.1 Pravidla pro tvorbu TIN

Pro tvorbu TIN platí tři obecná pravidla. První pravidlo říká, že výstup triangulace by měl být konzistentní. Pro danou množinu bodů by výsledná síť měla být unikátní bez ohledu na to, kterým bodem algoritmus začíná. Podle druhého pravidla by každá triangulace měla maximalizovat nejmenší úhel. Výsledné trojúhelníky budou v optimální situaci rovnostranné. Dle třetího pravidla musí být každý trojúhelník tvořen třemi nejbližšími body. [7]

Pokud není znám terén ale pouze souřadnice naměřených bodů, tak může dojít k situaci, kdy body lze spojit více způsoby. Pro dosažení co nejpřirozeněji působící triangulace je používána metoda maximalizace nejmenšího úhlu. Triangulace implementující tuto metodu se označuje jako úhlově optimální triangulace. Obrázek 8 znázorňuje situaci, kdy čtyři body v rovině tvoří konvexní čtyřúhelník. Při triangulaci těchto bodů může být vytvořena hrana p_ip_j nebo p_lp_k. Pro úhlově optimální triangulaci bude vybrána hrana p_lp_k, protože dojde k maximalizaci nejmenšího úhlu. Platí, že nejmenší úhel z α_{1-6} je menší než nejmenší úhel z α'_{1-6}. [8]



Obrázek 8: Prohození hrany pro dosažení optimální triangulace [8]

Pro zjištění, zda je potřeba prohodit hranu, není nutné počítat jednotlivé úhly, ale stačí využít pravidlo, které vyplývá z Thaletovy věty. Pravidlo můžeme aplikovat na obrázek 8. Pokud se bod p_l nachází uvnitř kružnice opsané trojúhelníku $p_ip_jp_k$, tak hrana p_ip_j je označená jako ilegální. Ilegální hrana je hrana jejímž prohozením lze docílit většího nejmenšího úhlu dané triangulace. Triangulaci, jenž neobsahuje žádnou ilegální hranu, říkáme legální triangulace. [8]

Každou oblast v rovině vymezenou mnohoúhelníkem lze triangulovat. Počet trojúhelníků a hran je dán množinou triangulovaných bodů. Pokud počet nekolineárních bodů v rovině označíme jako n a počet bodů ze stejné množiny, které leží na hranici triangulované oblasti, jako k, tak pro počet trojúhelníků v triangulaci platí vztah $2n-2-k$ a pro počet hran $3n-3-k$. [8]

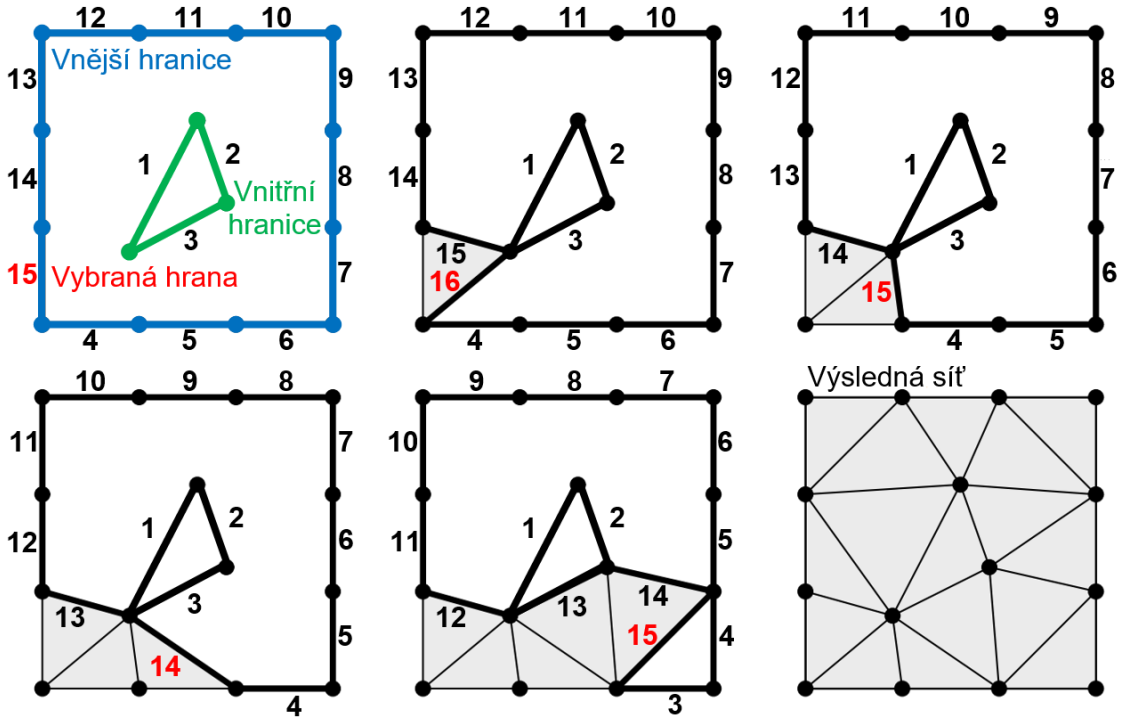
Přidávání bodů do triangulace může probíhat dvěma způsoby. První přístup je označován jako statický, kdy jsou všechna data přidána najednou. Druhý přístup, nazývaný dynamická triangulace, spočívá v přidávání nebo mazání bodů v průběhu procesu triangulace. [7]

Nejpoužívanější techniky pro generování nepravidelných trojúhelníkových sítí jsou AFM (Advancing-front method) a Delaunayho triangulace, případně kombinace

těchto metod. Generování Delaunayho triangulace bývá méně výpočetně náročné než AFM, protože nevyžaduje tak složité výpočty průsečíků a minimálních vzdáleností. [9]

3.2 Advancing-front metoda

Advancing-front metoda vychází z hranice oblasti, přičemž postupně přidává nové buňky, dokud síť nepokryje celou vymezenou oblast. Pokud bude předem známá vnitřní a vnější hranice triangulované oblasti, tak algoritmus lze rozdělit do třech kroků. Nejprve budou hrany podél vnitřní hranice očíslovány ve směru hodinových ručiček. Vnější hrany budou očíslovány v protisměru hodinových ručiček a celé číslování bude uloženo do vektoru *k*. V druhém kroku bude vybrána poslední hrana ve vektoru. Následně bude podle kritéria vybrán jeden z dostupných bodů. Kritérium může být například: „Bod, pro který je součet vzdáleností od bodů poslední hrany nejmenší.“ Vybraný bod společně s body hrany vytvoří trojúhelník. Ve třetím kroku budou hrany trojúhelníku smazány z vektoru *k* a číslování ostatních hran bude upraveno. Hrany trojúhelníku potřebné pro dokončení hranice oblasti budou do vektoru *k* uloženy na konec. Kroky dva a tři budou opakovány, dokud z vektoru *k* nebudou vymazány všechny hrany. [9]



Obrázek 9: Vytvoření nepravidelné trojúhelníkové sítě pomocí advancing-front metody [9]

3.3 Delaunayho triangulace

Delaunayho triangulace je triangulace množiny bodů v rovině, kde žádný bod z dané množiny neleží uvnitř kružnice opsané jakéhokoliv trojúhelníku z vytvořené trojúhelníkové sítě. Každá úhlově optimální triangulace je Delaunayho triangulace. S Delaunayho triangulací je také spojen pojem Delaunayho graf, což je spojitý graf, jehož dva body tvoří hranu pouze pokud existuje kružnice, na které leží tyto dva body a zároveň žádné jiné body tohoto grafu neleží uvnitř dané kružnice. Pokud žádné čtyři body z množiny neleží na stejné kružnici, tak říkáme, že množina je v obecné pozici. V takovém případě se Delaunayho graf shoduje s Delaunayho triangulací. Pakliže množina není v obecné pozici, tak všechny vzniklé triangulace z Delaunayho grafu budou legální. [8]

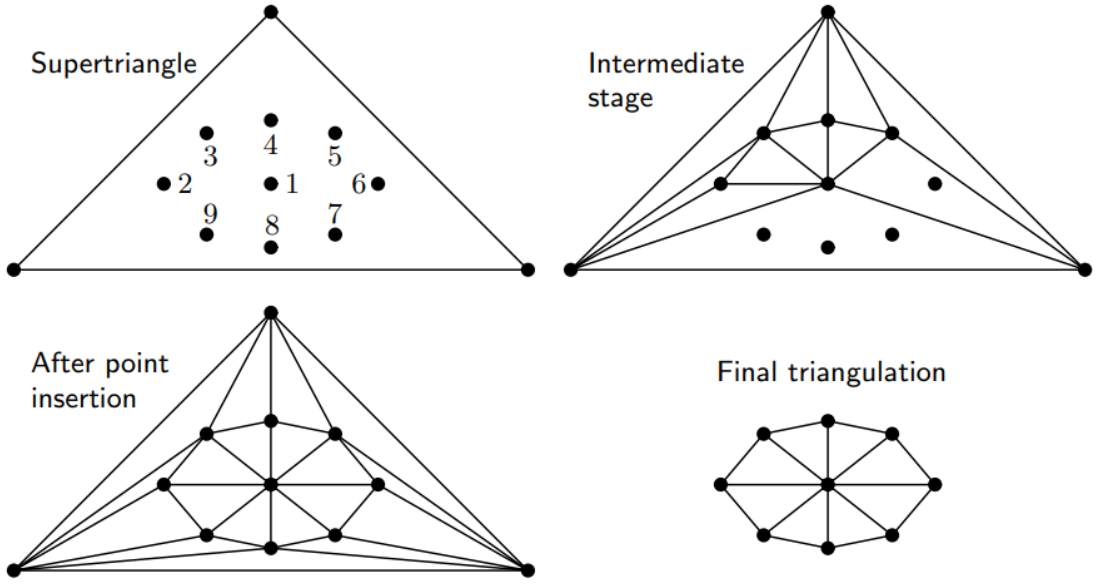
Jednou z výhod Delaunayho triangulace je její nezávislost na zvolení prvního triangulovaného bodu. Volba prvního bodu závisí čistě na potřebách daného algoritmu. Algoritmus může začínat nejbližším bodem od středu dané oblasti, nebo bodem tvořící hranici oblasti, případně může být zvolen jakýkoliv jiný přístup. Delaunayho triangulace je provázána s Voroného diagramem. Delaunayho triangulace může být vytvořena přímo pomocí algoritmu nebo nepřímo z Voroného diagramu. [7]

3.4 Bowyer-Watsonův algoritmus

Bowyer-Watsonův algoritmus je jeden z nejpoužívanějších algoritmů pro výpočet Delaunayho triangulace. Tento inkrementační algoritmus je založený na postupném přidávání bodů do již validní Delaunayho triangulace. Algoritmus lze použít pro libovolnou konečnou množinu bodů v jakékoliv dimenzi. [10]

V první fázi implementace Bowyer-Watsonova algoritmu je nutné vytvořit takzvaný „super trojúhelník“. Super trojúhelníkem může být nazván rovnostranný trojúhelník, který je dostatečně velký na to, aby obsáhl všechny triangulované body [9]. Po přidání všech bodů do triangulace musí být odstraněny všechny trojúhelníky obsahující vrcholy původního rovnostranného trojúhelníku. Rovnostranný trojúhelník lze také nahradit dostatečně velkým čtvercem, který bude rozdělen jeho úhlopříčkou na dva pravoúhlé trojúhelníky. Postup triangulace bude stejný jako při použití super trojúhelníku [11].

Pro přidání bodu do triangulace jsou vykresleny kružnice opsané ke všem již vytvořeným trojúhelníkům. Pokud některá z kružnic obsahuje právě vkládaný bod, tak je daný trojúhelník označený jako neplatný. V dalším kroku jsou z triangulace odstraněny všechny neplatné trojúhelníky. Po odstraněných neplatných trojúhelnících zůstane v triangulaci „díra“ ve tvaru konvexního polygonu. Pro dokončení triangulace jsou spojeny všechny body vzniklého konvexního polygonu s přidávaným bodem. Časovou náročnost tohoto algoritmu lze vyjádřit $O(n \sqrt{(n)})$. [10]



Obrázek 10: Proces Bowyer-Watsonova algoritmu [9]

Postup Bowyer-Watsonova algoritmu lze shrnout následujícím způsobem [12]:

1. Vytvoř super trojúhelník
2. Pro každý triangulovaný bod
   2.1. Pokud se daný bod nachází uvnitř kružnice opsané nějakého trojúhelníku
      2.1.1. Přidej trojúhelník k neplatným trojúhelníkům
   2.2. Odstraň neplatné trojúhelníky z triangulace
   2.3. Vyber z neplatných trojúhelníků hrany, které patří vždy jen jednomu neplatnému trojúhelníku (vytvoření konvexního polygonu)
   2.4. Spoj přidávaný bod s vrcholy vybraných hran (vrcholy polygonu)
3. Vymaž z triangulace trojúhelníky obsahující alespoň jeden vrchol super trojúhelníku

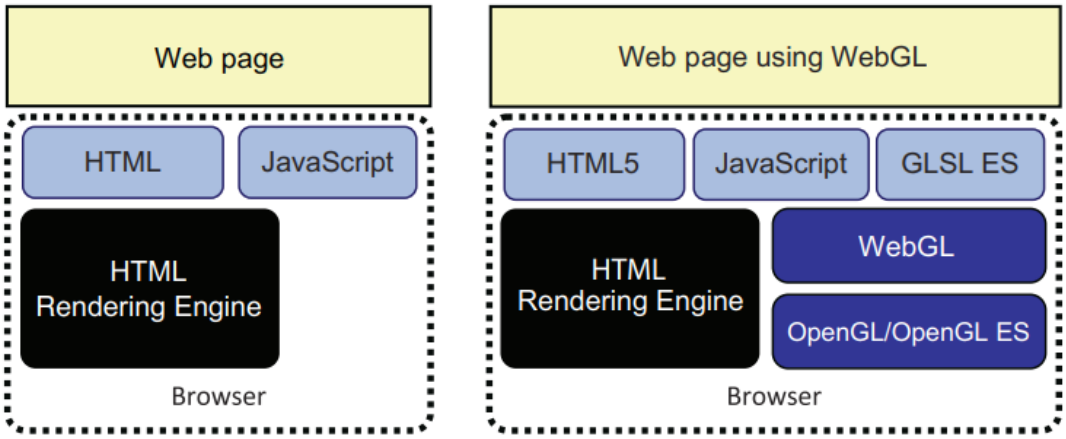
4 Technologie pro vizualizaci geografických dat ve webovém prostředí

V současné době existuje mnoho softwarových nástrojů pro vizualizaci dat ve webovém prohlížeči. Pro dosažení lepšího výkonu při zobrazení složitých scén je důležité, aby daný software uměl pracovat s grafickým hardwarem. Technologie umožňující práci s grafickým hardwarem v rámci webového prohlížeče je rozhraní WebGL.

4.1 WebGL

Web Graphics Library (WebGL) je nízkoúrovňové grafické rozhraní, které umožňuje zobrazovat komplexní 3D scény v prostředí webového prohlížeče. Většina webových prohlížečů (Safari, Chrome, Edge, Firefox) používá WebGL jako výchozí nástroj pro vykreslování 3D grafiky [13]. Před příchodem WebGL bylo nutné pro používání 3D aplikací běžících na webu instalovat různé pluginy nebo jiný externí software. Pro zobrazení webových stránek používajících WebGL stačí uživateli pouze webový prohlížeč a nemusí instalovat žádné další aplikace [14].

Webové stránky používající WebGL jsou vytvořeny pomocí třech jazyků: HTML, JavaScript a GLSL ES. Aplikační rozhraní (API) WebGL je napsáno v programovacím jazyce JavaScript. Pro psaní programů spustitelných na grafické kartě je využíván jazyk GLSL ES. Výsledná grafika je vykreslována do elementu značkovacího jazyka HTML <*canvas*>. [15]



Obrázek 11: Architektura dynamických webových stránek a webových stránek založených na WebGL [15]

4.1.1 Specifika WebGL

Specifika WebGL jsou založena na grafickém rozhraní OpenGL ES 2.0 (OpenGL for Embedded Systems 2.0), které je odvozeno od populární grafické knihovny OpenGL a bylo vytvořeno především pro mobilní zařízení a konzole [13]. Verze OpenGL 2.0 přinesla důležitou funkcionalitu v podobě programování shaderů. Shader je program spustitelný na grafické kartě umožňující vytváření složitých vizuálních efektů v reálném čase. WebGL pro psaní shaderů používá programovací jazyk GLSL ES (OpenGL ES shading language) [15].

V počítačové grafice se pro reprezentaci geometrických objektů běžné používají tři typy primitiv: body, čáry a trojúhelníky. Z těchto tvarů vychází topologie, které jsou dostupné v API WebGL: points, lines, line_strip, line_loop, triangles, triangle_strip a triangle_fan. WebGL na rozdíl od OpenGL nenabízí přímo funkce jako *glTranslate*. Při práci s transformacemi je nutné takovéto funkce vytvořit, nebo využít knihovny třetích stran [15].

Programy založené na WebGL se dají rozdělit do dvou komponent: řídící kód a shader. Řídící kód je napsán v programovacím jazyce JavaScript a běží na procesoru. Cílem řídícího kódu je postarat se o celkový běh aplikace. Řídící kód mění vlastnosti scény a předává potřebná data shaderu. Shader je napsán v programovacím jazyce GLSL ES a je prováděn na grafické kartě. Jeho úkolem je zpracovat jednotlivé geometrické objekty do podoby, jak budou vykresleny na obrazovku. [16]

4.1.2 GLSL ES

GLSL ES je zjednodušená verze programovacího jazyka OpenGL Shading Language (GLSL) používaná rozhraním OpenGL ES. Oproti GLSL jsou některé jeho funkcionality zjednodušené nebo dokonce zredukované. Syntaxe GLSL ES je podobná programovacímu jazyku C. Tento jazyk je case-sensitive (rozlišuje velká a malá písmena), type-sensitive (rozlišuje mezi různými datovými typy) a používá středníky. Každý program napsaný v GLSL ES musí obsahovat bezparametrovou metodu *main()*. [15]. GLSL ES je využíván pro psaní shaderů. Shadery jsou složeny ze dvou programů: vertex shaderu a fragment shaderu. Vertex shader vezme pozici vertexu ve 3D prostoru a transformuje ji do 2D prostoru zobrazovaného okna. Ve fragment shaderu je vypočítána barva výsledného pixelu [17].

4.1.2.1 Vertex shader

Vertex shader je typ programu, který definuje vlastnosti vertexu (bodu ve 3D prostoru). Vlastnosti vertexu mohou být pozice, barva, normála atd. [15]. V rámci vertex shaderu bývají vykonávány grafické operace jako transformace pozice vrcholu pomocí maticových operací, transformace normál, výpočet a transformace souřadnic do textury, animace apod. [16]. Při psaní vertex shaderu by vždy měla být specifikována proměnná *gl_Position*, která uchovává pozici vertexu. Na druhou stranu některé proměnné jako například *gl_PointSize* (velikost bodu) nemusí být definovány, protože mají ve svém výchozím stavu již nastavenou nějakou hodnotu [15].

4.1.2.2 Fragment shader

Po výpočtu atributů vrcholu ve vertex shaderu dojde k ořezání nebo dokonce úplnému odstranění primitiv ležících mimo zobrazovaný 3D region. V další fázi dojde k vykreslení fragmentů (pixelů) na obrazovku. Jednotlivé fragmenty zpracovává program nazývaný fragment shader. Ve fragment shaderu je specifikována barva pixelu. Barva může být přiřazena například pomocí různých barevných transformací (změna sytosti, přidání odstínu atd.), nebo z textury na základě lineární interpolace souřadnic do textury [16]. Barva je ukládána do proměnné *gl_FragColor*, která používá datový typ *vec4*. Daný vektor reprezentuje barvu jako RGBA hodnotu [15].

4.1.2.3 Proměnné

GLSL ES podporuje globální a lokální proměnné. Vstupní data lze do shaderu vkládat dvěma způsoby: attributes a uniforms. Attributes jsou dostupné pouze pro vertex shader a mají odlišné hodnoty pro každý vertex. Pro jejich definici je nutné nejprve uvést klíčové slovo *attribute,* následně datový typ, a nakonec jméno proměnné. Uniforms jsou definována jako read-only, to znamená, že nemohou být změněna v průběhu programu. Pro jejich definici je nejprve potřebné použít klíčové slovo *uniform,* následně datový typ a poté jméno proměnné. K uniforms má přístup vertex i fragment shader. Attributes a uniforms musí být deklarovány jako globální, protože data jsou do nich vkládána mimo shader. Dalším typem proměnné jsou varying. Varyings jsou proměnné, které mohou být definovány ve fragment shaderu a poté předány vertex shaderu. [15]

4.1.2.4 Datové typy

GLSL ES podporuje číselné a booleovské datové typy. U číselných typů rozlišuje mezi hodnotou s desetinou čárkou (float) a celočíselnými hodnotami (int). U booleovského typu používá true a false. Datový typ string není v rámci GLSL ES podporován. GLSL ES také obsahuje datové typy pro vektory a matice. Vektorové typy reprezentují data jako list a jsou užitečné pro reprezentaci souřadnic nebo barvy. Maticové datové typy ukládají data jako pole a jsou používané pro transformační matice. [15]

Kategorie	**Typ**	**Popis**
Vektor	vec2, vec3, vec4	Datové typy pro vektory typu float
	ivec2, ivec3, ivec4	Datové typy pro vektory typu integer
	bvec2, bvec3, bvec4	Datové typy pro vektory typu boolean
Matice	mat2, mat3, mat4	Datové typy pro matice 2×2, 3×3, and 4×4

Tabulka 1: Maticové a vektorové datové typy v GSLS ES [15]

GLSL ES také nabízí možnost vytvoření vlastních typů pomocí structures. Při jejich vytváření je používáno klíčové slovo *struct*. [15]

Vytvoření webové aplikace pro vizualizaci geografických dat využívající WebGL může být časově i energeticky náročné. Pro zlepšení efektivity práce a zajištění stability výsledné aplikace je vhodné využít již existujících nástrojů. Jedním z takových nástrojů je deck.gl.

4.2 deck.gl

JavaScriptový framework deck.gl využívá rozhraní WebGL pro vytváření komplexních interaktivních vizualizací rozsáhlých datových sad v rámci webového prohlížeče. Klade důraz na vysoký výkon a pro důležité výpočty využívá grafickou kartu. Data jsou v deck.gl vizualizována pomocí vrstev. Tvůrci deck.gl nabízí celou řadu již vytvořených vrstev dostupných k okamžitému použití. Zároveň je možné v rámci frameworku vytvářet nové vrstvy včetně implementace vlastních shaderů.

Velkou výhodou deck.gl je možnost propojení s dalšími webovými technologiemi jako React, Google Maps, Mapbox, ArcGIS ad. [1]

4.2.1 Související knihovny

Framework deck.gl je vyvíjen s celou řadou dalších knihoven. Jednou z nich je luma.gl. Luma.gl je toolkit pro vizualizaci dat pomocí WebGL. Nabízí nástroje pro programování GPU přímo s WebGL 2 API, ale zároveň umožňuje zpětnou kompatibilitu s WebGL 1 [18]. Další běžně používanou knihovnou je loaders.gl, která slouží pro načítání a vytváření souborů zaměřených na geografická data. Vzhledem k tomu, že jde o nezávislou knihovnu, tak může být použita s jakýmkoliv jiným frameworkem [19]. Pro použití podkladových map MapBox GL JS je v deck.gl využívána knihovna react-map-gl [20].

Knihovna deck.gl sice není přímo založená na frameworku React, ale je naprogramována tak, aby umožňovala kompatibilitu s React aplikacemi. Pro použití deck.gl v rámci React aplikace stačí importovat React komponentu DeckGL a vykreslit ji v rámci jiné komponenty. [1]

4.2.2 Vrstvy

Vrstva deck.gl je typ vizualizace. V podstatě jde o kolekci dat, která je spojená s určitou pozicí, barvou, případně dalšími hodnotami. V rámci frameworku deck.gl je možné jednotlivé vrstvy spojovat a vytvářet tak komplexní vizualizace. Každá vrstva má své vlastnosti (properties), které uživatel může specifikovat. Mezi základní vlastnosti patří ID a data. ID slouží pro identifikaci konkrétní vrstvy. Proměnná data specifikuje zdroj dat, která mají být vizualizována. [1]

4.2.3 Souřadnicový systém

Každý objekt v deck.gl má svoji pozici. Tato pozice je reprezentována pomocí 2D ([x, y]), nebo 3D ([x, y, z]) pole. Deck.gl nabízí tři druhy souřadnicových systémů: world space, common space a screen space. World space reprezentuje data tak, jak jsou definovaná. Common space sloučí data do jednoho trojrozměrného kartézského souřadnicového systému. Screen space používá souřadnicový systém obrazovky uživatele. [1]

4.2.4 Výběr objektu

Framework deck.gl pro výběr objektu používá techniku color picking. Každý objekt z každé vrstvy dostane přiřazenou unikátní barvu. Všechny vrstvy, kde je picking aktivován, jsou vykresleny do speciálního neviditelného bufferu. V tomto režimu nejsou vykreslovány skutečné barvy objektů, ale ty, které jim byly přiřazeny v rámci výběru. Pokud uživatel najede myší na nějaký objekt, tak se deck.gl podívá na barvu objektu a podle ní vyhledá jeho index. [1]

4.2.5 Metody vrstvy

Každý potomek třídy Layer může definovat metody, které ovlivňují to, jak bude data vykreslovat. Tyto metody jsou volány v závislosti na tom, v jaké fázi se vrstva nachází. Ve fázi inicializace je použita metoda *initializeState()*, která je pro každou vrstvu zavolána pouze jednou a nastavuje její původní stav. Při aktualizaci vrstvy je volána metoda *updateState()*. Při vykreslování je zavolána metoda *draw()*. Pakliže chce uživatel vybrat nějaký objekt, tak může být použita metoda *getPickingInfo()*, která poskytne informace o objektu. Metoda *finalizeState()* bývá zavolána těsně předtím, než je reference na danou vrstvu vymazána z paměti. [1]

4.2.6 Custom Layer

Deck.gl nabízí možnost implementace vlastní vrstvy. Každá vrstva musí být potomek třídy Layer, být pojmenovaná a mít definované properties. Při vytváření vlastní vrstvy nabízí deck.gl tři možnosti. První možností je vytvořit composite Layer, který se skládá z více již existujících vrstev. Další možnost subclass Layer znamená vytvoření potomka již existující vrstvy a následně přepsání nějaké z metod vrstvy případně i vertex a fragment shaderu. Tento způsob je vhodný, pokud má být přidána nějaká menší funkcionalita do již existující vrstvy. Pokud vývojáři nevyhovuje žádná z předešlých možností, tak může implementovat vlastní vrstvu od nuly (pouze jako potomka třídy Layer). [1]

4.2.6.1 Implementace vlastní vrstvy

Nejprve je potřeba implementovat metodu *initializeState()*, ve které jsou definovány všechny proměnné potřebné pro vykreslení vrstvy. Jednou z důležitých

proměnných je model [1]. Model je třída luma.gl, která uchovává data nezbytná pro vykreslení objektu (shader, uniforms, attributes) [18]. Většina vrstev používá pouze jednu instanci třídy model, která je uložena do *state.model* během inicializace. Dále během inicializace je nutné definovat atributy pomocí AttributeManager. Atributy jsou definovány zavoláním metody *add()* nad instancí *attributeManager*. Metodu *updateState()* je vhodné implementovat, pokud mají být změněny vlastnosti vrstvy. Pokud mají být používány vlastní uniforms nebo upravené nastavení renderování, tak je potřeba implementovat metodu *draw()*. Pro práci se souřadnicovým systémem nabízí luma.gl metodu *project_position*, která je dostupná v JavaScriptu i GLSL. V případě, že je nutné implementovat vlastní výběr objektů, tak musí být ve vertex shaderu nastavena aktuální barva výběru metodou *picking_setPickingColor()* a ve fragment shaderu použita metoda *picking_filterPickingColor()* pro aktualizaci *gl_FragColor*.[1]

5 Návrh

Bakalářská práce byla řešena v rámci navázané spolupráce se soukromou společností, která poskytla data pro vizualizaci. Cílem bylo navrhnout a implementovat interaktivní, graficky atraktivní a jednoduše pochopitelnou 3D vizualizaci terénních deformací, která by pomohla uživateli při vyhodnocování získaných družicových dat. Požadavkem implementace bylo vytvořit vizualizace spustitelné v rámci webového prohlížeče s využitím programovatelných grafických karet. Atributy geografických dat byly měněny v reálném čase. Při vytváření vizualizací byl kladen důraz na zachování plynulosti ovládání při zobrazení mapy.

Ve stejné spolupráci již vznikla bakalářská práce [21], kterou napsal Bc. Ing. Jan Ekr, Ph.D. V rámci jeho práce implementoval metody pro výpočet barvy a transformování dat. Metody pro práci s barvou přiřazují k hodnotám z daného intervalu hodnoty barevného gradientu (minimální hodnota modrou barvou, maximální hodnota červenou barvou). Metody pro transformování dat dopočítávají pomocí lineární interpolace chybějící hodnoty a určují pro daný čas animace čas měření metodou půlení intervalů. Všechny výše zmíněné metody byly využity v rámci implementace této práce.

5.1 Použité technologie

Požadavkem na implementaci bylo využít vizualizační framework deck.gl. Pro zlepšení efektivity práce byly použity další související knihovny. Vizualizace dat pomocí WebGL byla provedena toolkitem luma.gl. Při načtení formátu GeoJSON a textury ve formátu PNG byla použita knihovna loaders.gl. Interaktivní uživatelské rozhrání pro ovládání animace bylo vytvořeno prostřednictvím frameworku React. Pro podkladové mapy byla využita knihovna CARTO basemaps.

5.2 Použitá data

Poskytnutá data vznikla satelitním měřením a mapují terénní deformace. Naměřené hodnoty jsou uchovány ve formátu GeoJSON a zahrnují řadu atributů, které detailně popisují deformaci. Při implementaci byly brány v úvahu pouze atributy vztažené k pozici bodu a velikosti deformace. Použitými atributy byly *vel_cum*, *d_[YYYYMMDD]* a *coordinates*. Hodnota *vel_cum* vyjadřuje v milimetrech celkové kumulativní posunutí bodu za celé sledované období. Atribut *d_[YYYYMMDD]*

uchovává rovněž v milimetrech relativní posun měřeného bodu vzhledem k datu měření, přičemž datum je uvedeno v názvu atributu ve standartním datovém formátu *YYYYMMDD*, který kombinuje zápis čtyřmístného roku (YYYY), dvoumístného měsíce (MM) a dvoumístného dne (DD). V atributu *coordinates* jsou uloženy souřadnice měřeného bodu.

5.3 Struktura aplikace

Výsledné programy byly vytvořeny jako React aplikace složená ze dvou komponent: Controls a DeckGL. Komponenta Controls obsahuje prvky potřebné k ovládání průběhu animace. DeckGL je komponenta z frameworku deck.gl upravená tak, aby fungoval v React aplikacích (viz kapitola 4.2.1). Pro správnou funkci přidaných podkladových map bylo nutné použít knihovnu react-map-gl, která synchronizuje zobrazované okno DeckGL s dalšími React komponentami. Komponentě DeckGL jsou předány ve formě atributů informace o podkladové mapě, výchozím stavu zobrazení a vrstvách.

Vizualizace byly vytvořeny pomocí vlastních vrstev. Tato metoda byla vybrána, protože poskytuje nejvíce možností při tvorbě vizualizace společně s implementací vertex a fragment shaderu. Každá vrstva je potomkem třídy Layer a implementuje její metody *initializeState()*, *updateState()* a *draw()*.

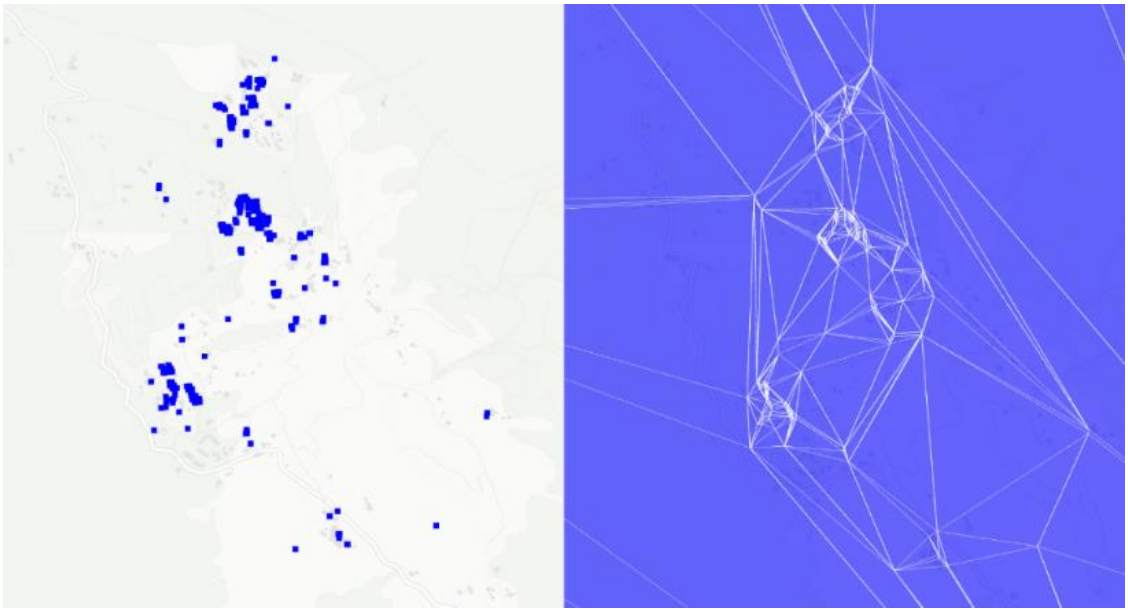
Kód je rozdělen do adresářů data, layers a utils. Ve složce data je uložen soubor typu GeoJSON se zobrazovanými daty. Složka layer obsahuje soubory s vrstvami a shadery. V adresáři utils jsou umístěny soubory s metodami pro výpočet barvy, transformaci dat a triangulaci. Komponenty společně s výchozím HTML souborem se nacházejí v hlavním adresáři.

6 Implementace a výsledky

V rámci bakalářské práce byly vytvořeny čtyři různé metody vizualizace geografických dat. První vizualizace vytváří pomocí Bowyer-Watsonova algoritmu z naměřených bodů statickou nepravidelnou trojúhelníkovou síť. Další vizualizace vycházejí z první a jsou založeny na animaci vzniklé trojúhelníkové sítě. V druhé vizualizaci mění trojúhelníky výšku a barvu podle velikosti posunu daného bodu. Třetí metoda vizualizace mapuje na trojúhelníkovou síť pravidelnou texturu a sleduje její deformaci. Poslední způsob vizualizace vytváří na místech hran trojúhelníků obdélníky, na které je namapována textura. Kompletní zdrojový kód je dostupný v repozitáři na GitLabu (příloha č. 1).

6.1 Triangulace bodů

Body byly triangulovány na základně jejich zeměpisných souřadnic. Triangulace byla provedena pomocí Bowyer-Watsonova algoritmu, který byl podle teorie popsané v kapitole 3.4 implementován následujícím způsobem. Nejprve bylo potřeba pro výpočet vrcholů super trojúhelníku najít maximální a minimální hodnoty na každé souřadnicové ose a na základě těchto hodnot vypočítat dané vrcholy. Následně bylo vytvořeno prázdné pole, do kterého byl přidán super trojúhelník. Dále byly postupně přidávány všechny body do triangulace. V rámci každého přidání bodu bylo procházeno pole trojúhelníků a kontrolována podmínka, že se přidávaný bod nesmí nacházet v kružnici opsané jakéhokoliv trojúhelníku. Trojúhelníky nesplňující tuto podmínku byly přidány do pole ilegálních trojúhelníků. Po kontrole všech trojúhelníků byly ilegální trojúhelníky odstraněny z pole validních trojúhelníků. V dalším kroku byly nalezeny hranice polygonu. Pro nalezení hranic polygonu bylo potřeba projít pole ilegálních trojúhelníků a najít hrany, které patří vždy jen jednomu trojúhelníku. Nalezené hrany byly uloženy do proměnné *polygon*. Posledním krokem při přidávání bodu bylo spojit daný bod se všemi vrcholy hran uložených v proměnné *polygon*. Po přidání všech bodů byly odstraněny z pole trojúhelníků všechny trojúhelníky, které obsahovaly alespoň jeden vrchol super trojúhelníku. Kód metody *triangulate(points)* je uveden jako příloha č. 2. Obrázek 12 zobrazuje vlevo původní množinu bodů a vpravo triangulaci dané množiny.

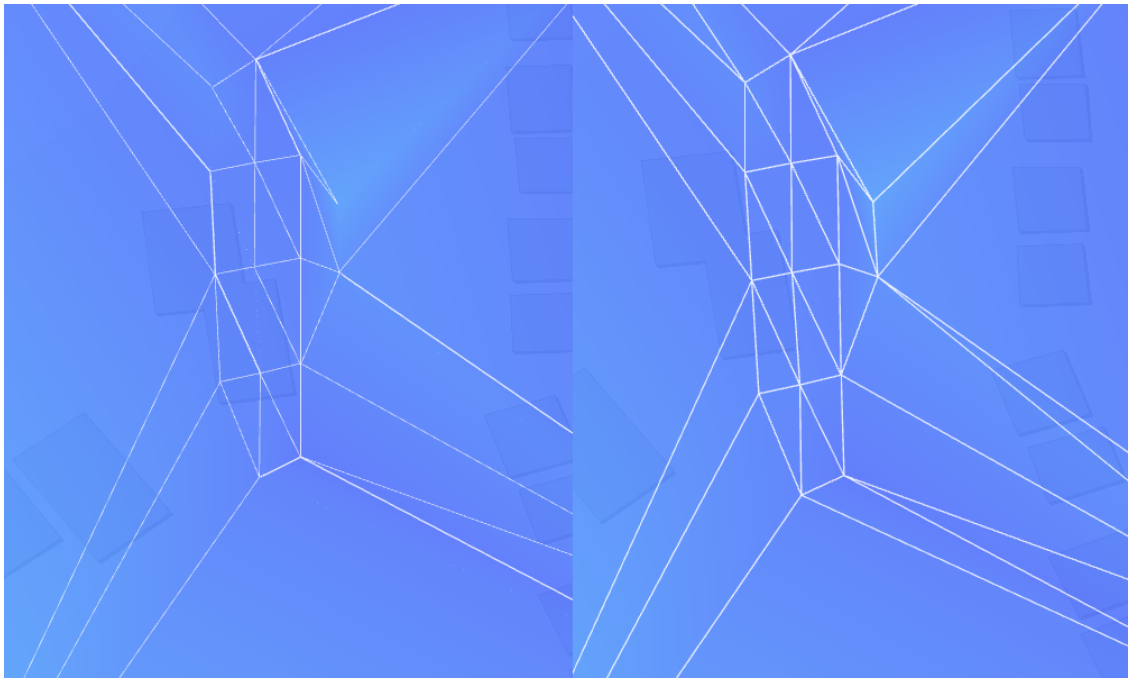


Obrázek 12: Vytvoření trojúhelníkové sítě z množiny bodů

Vlastní vizualizace je složena ze dvou vrstev: vrstvy trojúhelníků a vrstvy linií (hran trojúhelníků). Vzhledem k tomu, že obě vrstvy používají stejná data, tak se při pozorování vizualizace projevil v počítačové grafice známý problém Z-fighting. Z-figting se objevuje v situaci, kdy dvě nebo více primitiv mají shodnou, nebo velmi podobnou vzdálenost od kamery. V této situaci není jednoznačné, jak má být pixel vykreslen a při změně kamery nebo scény se mohou primitiva různě navzájem překrývat. Příčinou je použití algoritmu Z-buffer pro řešení viditelnosti s omezenou přesností reprezentace hodnoty hloubky, tj. vzdálenosti od pozorovatele. Řešením tohoto problému bylo využít jednu z vlastností (properties) vrstvy deck.gl *getPolygonOffset*.

Vlastnost *getPolygonOffset* třídy Layer umožňuje vrstvě použít WebGL metodu *gl.polygonOffset* pro přidání malé hodnoty (offsetu) k její vzdálenosti od kamery. Při přidání záporného čísla je vrstva posunuta blíže ke kameře a při přidání kladného čísla dále od kamery. Ve výchozím nastavení je v deck.gl ke každé vrstvě přiřazen velmi malý offset podle jejího indexu. [1]

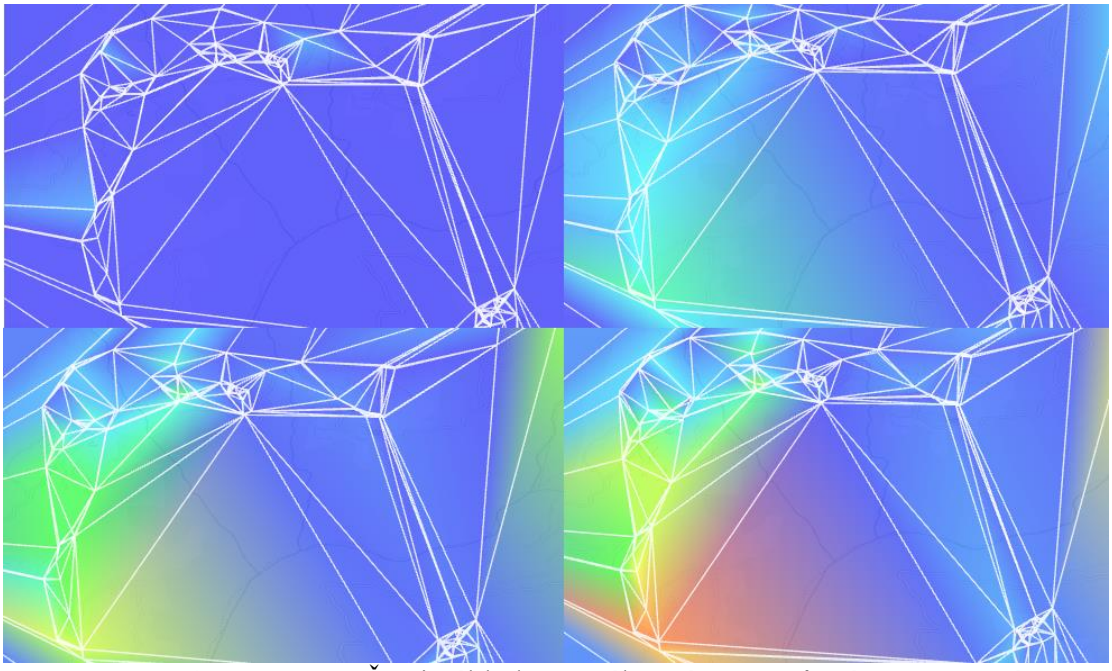
Vrstvě trojúhelníků byl přidán kladný offset, díky kterému byla posunuta dál od kamery a přestalo tak docházet k překrývání s vrstvou hran. Obrázek 13 zobrazuje jednu z problémových oblastí před a po přidání offsetu.



Obrázek 13: Triangulace před a po přidání offsetu

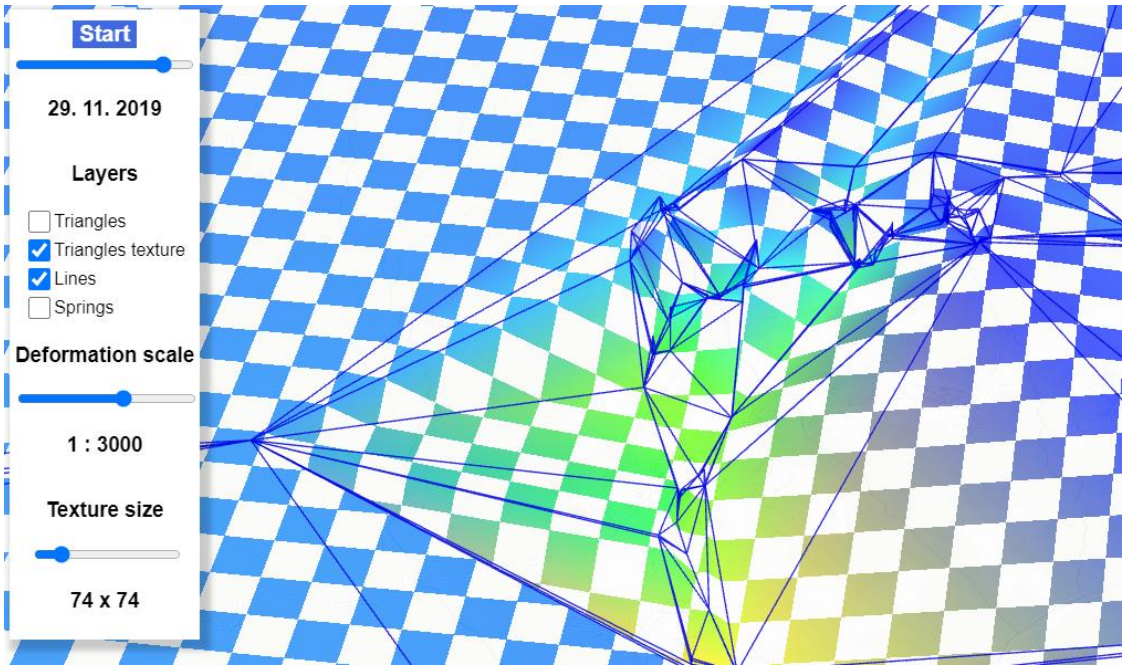
6.2 Animace triangulace

Vstupní data obsahují informace o relativním posunu každého bodu vzhledem k času měření viz kapitola 5.2. Tato data byla použita pro vytvoření interaktivní animace, která ukazuje velikost deformace v daných bodech vzhledem danému datumu. Pro vytvoření animace bylo potřeba při každém vykreslení měnit data posílaná vertex shaderu prostřednicím WebGL objektu *Buffer*. Změněná data byla předána pomocí metody z luma.gl *setBuffer()* atributu modelu *vertexArray*. Data bylo potřeba předat jako JavaScriptové typové pole *Float32Array* společně s lokací daného atributu nalezené prostřednictvím metody *gl.getAttribLocation()*. Pro zjištění hodnoty v daném čase měření podle času animace byly využity metody zmíněné v kapitole 5. V průběhu programu byly měněny atributy *color* a *position*. Výše popisovaný kód je uveden jako příloha č. 3. Obrázek 14 ukazuje stejnou oblast v různých časech animace.



Obrázek 14: Čtyři náhledy stejného místa pro různé časy měření

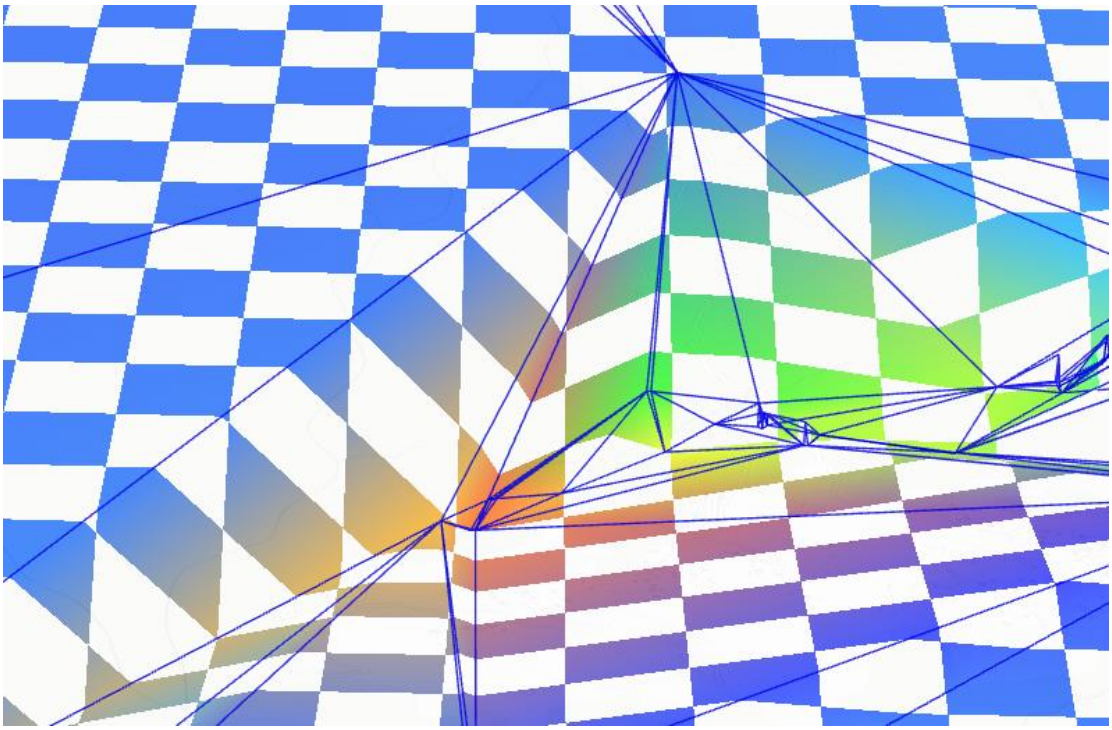
Pro zlepšení interaktivity bylo do aplikace přidáno ovládání parametrů animace. Uživatel může libovolně posouvat animaci v čase, případně ji úplně zastavit. Dále může vypínat a zapínat jednotlivé vrstvy, měnit počet namapování textury a měřítko deformace, tedy koeficientu zvětšení hodnoty určující posun terénu v daném bodě. Samotné ovládání bylo vytvořeno jako React komponenta. Proměnné *time*, *textureSize* a *scale* jsou poté předávány jednotlivým vrstvám jako properties.



Obrázek 15: Ovládání animace

6.3 Animace s deformací textury

Následující vizualizace využívá pravidelnou texturu pro ukázání deformace povrchu. Na trojúhelníkovou síť v rovině byla namapována textura v podobě šachovnice a během animace je měněna výška bodů a barva textury podle velikosti deformace v daném bodě. Díky tomu, že souřadnice do textury zůstávají stejné po celou dobu animace, tak dochází k deformaci textury.



Obrázek 16: Triangulace s deformací textury

Pro namapování textury bylo potřeba pro každý bod triangulace vypočítat souřadnice do textury. Pomocí metody *attributeManager.add()* byl popsán nový atribut *textureCoord*. Následně byla parametru atributu *update* předána vytvořená funkce *getTextureCoordinates(attribute)*, v rámci, které byly vypočítány souřadnice do textury. Souřadnice byly uloženy pomocí struktury *Float32Array* a přiřazeny jako hodnota atributu *attribute.value*. (příloha č. 4)

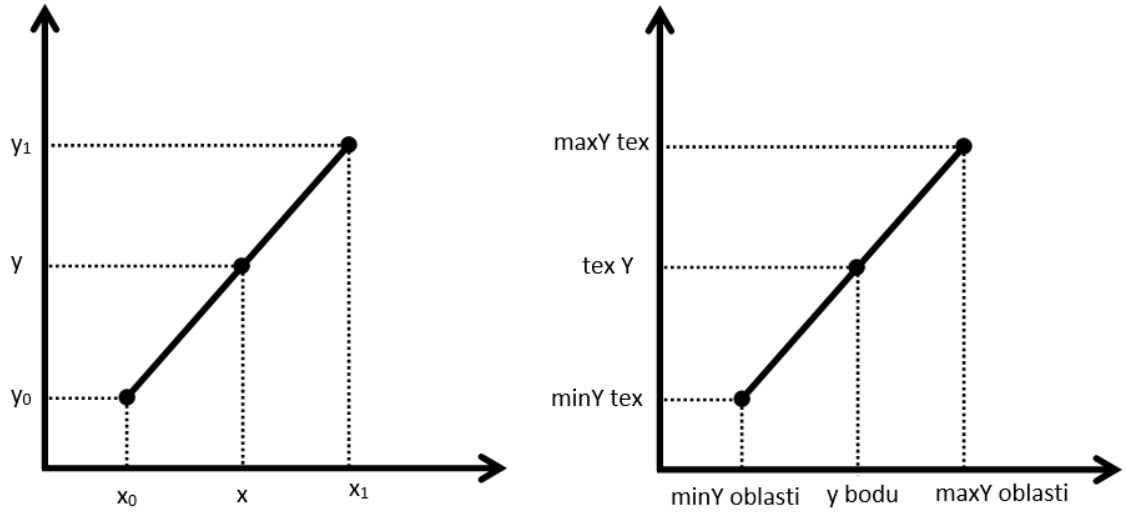
Výpočet souřadnic do textury bylo nutné rozdělit do několika kroků. Nejprve byly zjištěny maximální a minimální geografické souřadnice z množiny naměřených bodů. Na základě zjištěných geografických souřadnic byl vypočítán čtverec velký tak, aby obsáhl celou trojúhelníkovou síť. V dalším kroku byly vrcholům čtverce přiřazeny souřadnice do textury podle frekvence mapování dané textury. V posledním kroku byly pomocí lineární interpolace vypočítány souřadnice do textury pro každý vrchol

trojúhelníkové sítě. Pro výpočet byly použity souřadnice do textury *[texX; texY]* protilehlých vrcholů vytvořeného čtverce, souřadnice daného bodu a velikost triangulované geografické oblasti. Vzhledem k tomu, že souřadnice textury začínají v bodě [0; 0], tak bylo možné rovnici pro výpočet souřadnic do textury pomocí lineární interpolace zjednodušit následujícím způsobem:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$$

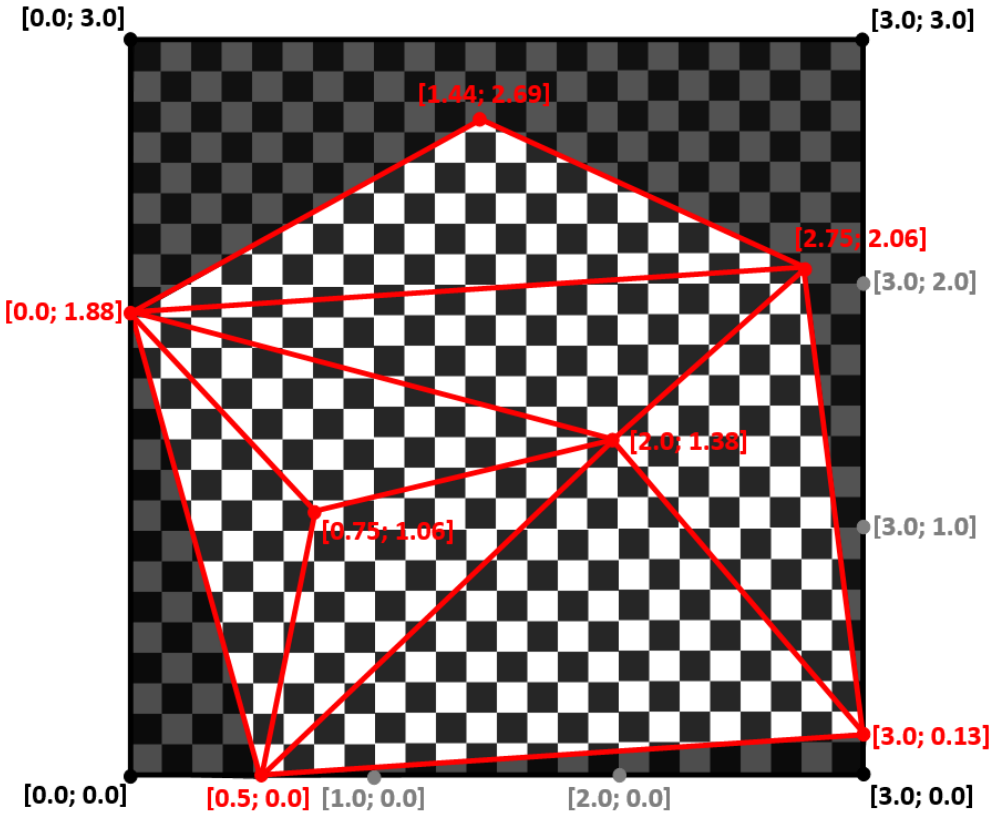
$$texY = minY\ tex + (y\ bodu - minY\ dané\ oblasti)\frac{maxY\ tex - minY\ tex}{maxY\ oblasti - minY\ oblasti}$$

$$texY\ = (y\ bodu - minY\ dané\ oblasti)\ \frac{počet\ opakování\ textury}{velikost\ dané\ oblasti}$$



Obrázek 17: Dosazení jednotek pro výpočet souřadnic textury do rovnice lineární interpolace

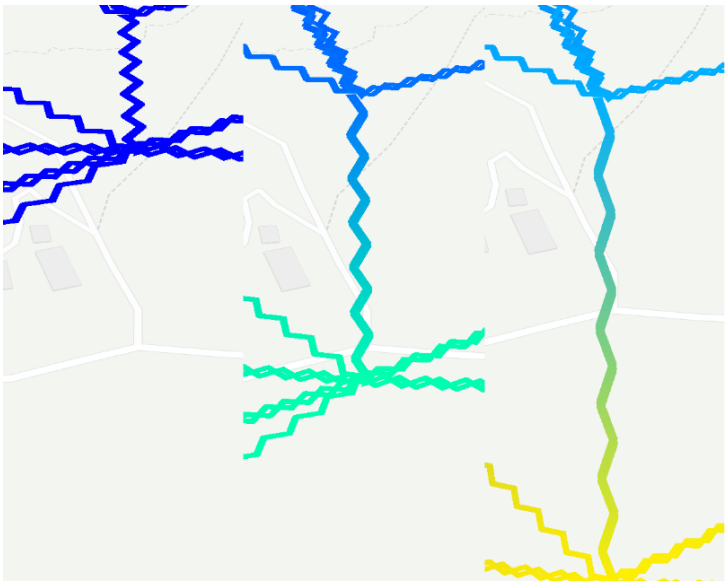
Souřadnice texX byla vypočítána stejným způsobem, pouze byly vyměněny y souřadnice za x. Obrázek 18 ukazuje, jakým způsobem jsou vypočítané souřadnice použity při mapování textury. Kód pro výpočet souřadnic je uveden jako příloha č. 5.



Obrázek 18: Namapování textury na trojúhelníkovou síť podle vypočítaných souřadnic do textury

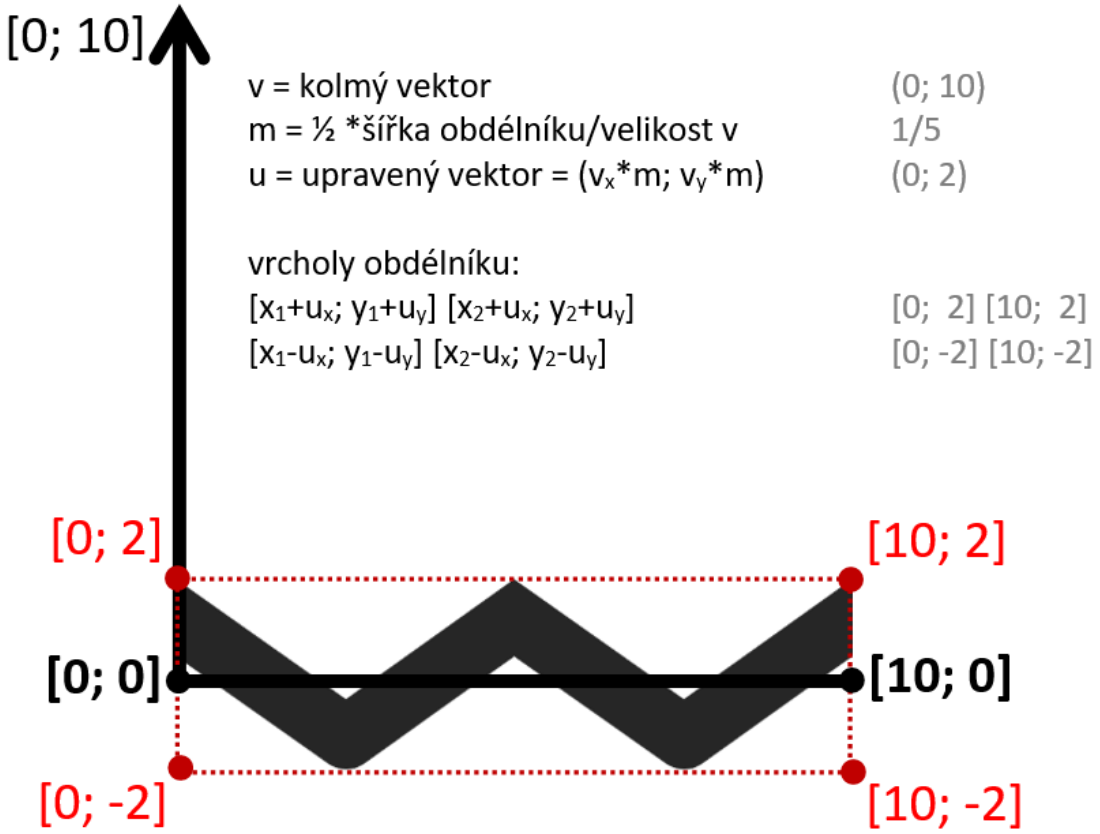
6.4 Animace s napínáním textury

Animace s napínáním textury vychází z myšlenky, že by každé dva body byly spojeny pružinou, která by se při posunu bodů napínala. Napnutí pružiny by pomohlo vytvořit představu o míře deformace v daných bodech.



Obrázek 19: Animace napínání textury

Mezi každými dvěma body trojúhelníkové sítě byl vykreslen obdélník, na který byla namapována textura. Pro zjištění vrcholů obdélníku bylo nejprve nutné vypočítat vektor kolmý na danou hranu a následně vydělit polovinu požadované velikosti hrany velikostí vektoru. Tímto výpočtem bylo získáno měřítko, kterým byl následně vynásoben získaný kolmý vektor. Díky tomuto výpočtu budou mít všechny obdélníky stejnou šířku bez ohledu na vzdálenost mezi původními body. Ukázka kódu se nachází v příloze č. 6. Výsledný počet namapovaných textur na obdélník je závislý na vzdálenosti mezi původními dvěma body.



Obrázek 20: Namapování textury na obdélník vytvořený mezi dvěma body

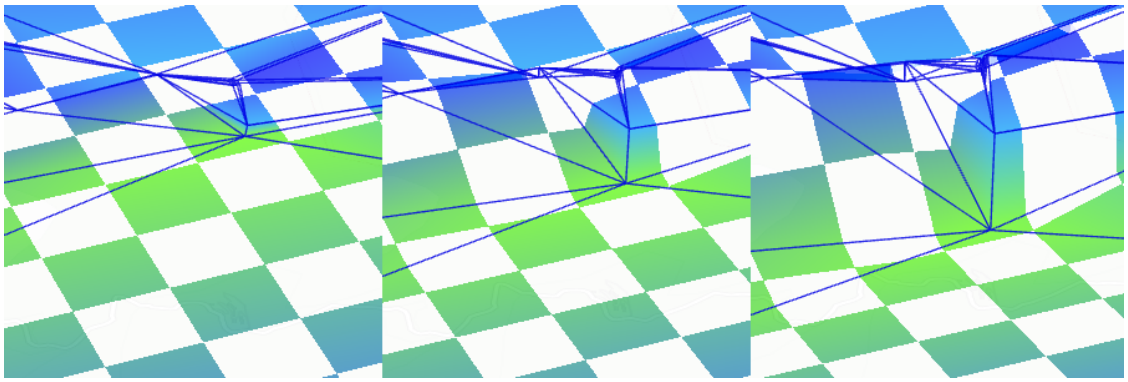
Ve fragment shaderu byla vytvořena podmínka, díky které je v případě bílé barvy textury nastavena průhlednost na nulovou hodnotu. Během průběhu animace zůstávají souřadnice do textury stejné, čímž dochází k napínání textury podle změny výšky bodů.

7 Testování a zhodnocení

Trojúhelníková síť byla vytvořena pomocí Bowyer-Watsonova algoritmu podle souřadnic jednotlivých bodů. Poskytnutá data obsahovala několik různých měření, která se vztahovala ke stejným souřadnicím. Po odstranění duplicitních hodnot zbylo pro triangulaci 1631 unikátních bodů. Pro ověření, zda při triangulaci v rovině byl vytvořen správný počet trojúhelníků, lze použít rovnici $2n-2-k$ (viz kapitola 3.1). Za n musí být dosazen počet triangulovaných bodů a za k počet bodů, které se nacházejí na hranici trojúhelníkové sítě. Výpočet bodů ležících na hranici trojúhelníkové sítě může být komplikovaný a pro jednoduché ověření stačí použít počet trojúhelníků vytvořených před odstraněním super trojúhelníku. Hranici bude tvořit daný super trojúhelník tedy $k=3$ a dané 3 body musí být přičteny k množině triangulovaných bodů $n=1634$. Po dosazení do rovnice je získáno 3263 trojúhelníků ($1634*2-2-3=3263$). Vypočítána hodnota se shoduje s hodnotou získanou implementovaným algoritmem před odstraněním super trojúhelníku. Na základě tohoto výpočtu lze říct, že implementovaný algoritmus vytvořil pro danou množinu bodů správný počet trojúhelníků.

Při pozorování vizualizace trojúhelníkové sítě se projevil problém Z-fighting, který byl úspěšně vyřešen a jeho problematika byla popsána v kapitole 6.1. Během dalšího pozorování nebyla nalezena žádná nespojitá část, žádné překrývaní prvků nebo jiné nežádoucí efekty.

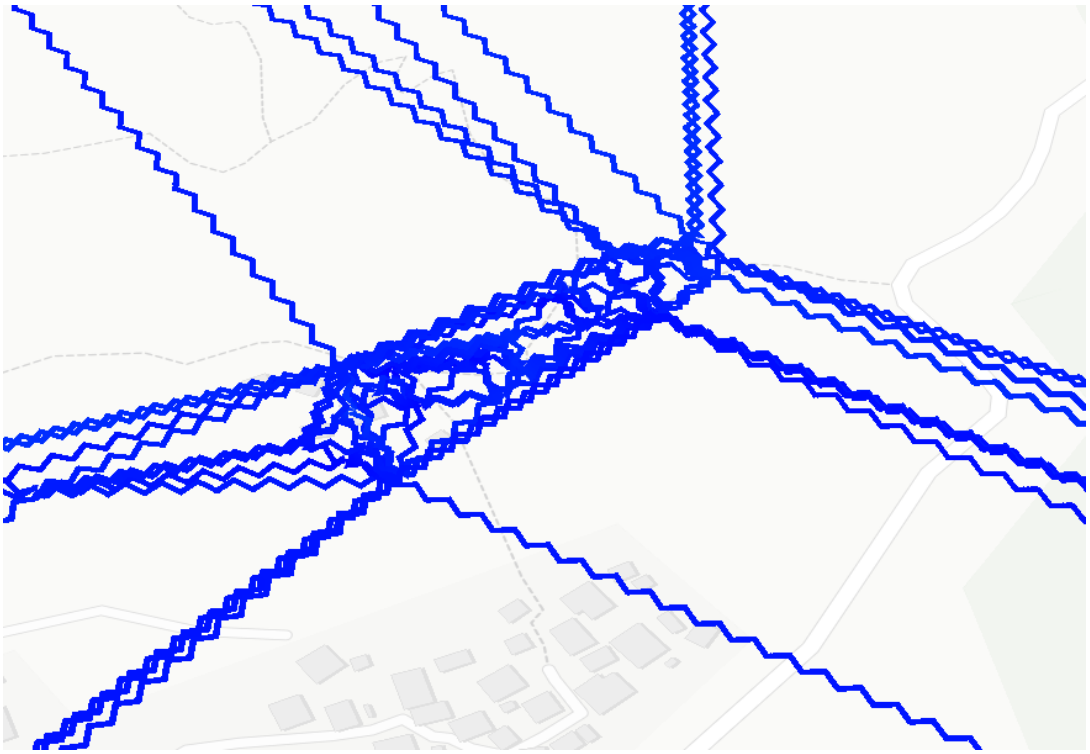
Vzhledem k tomu, že naměřená deformace byla velmi malá, tak bylo potřeba pro zviditelnění posunu bodů trojúhelníkové sítě několikanásobně zvětšit měřítko deformace, což je nutné brát v potaz při případné analýze dané oblasti. V ostatních směrech se druhá vizualizace chovala podle předpokladů.



Obrázek 21: Stejná oblast ve stejném čase animace v různých měřítkách, 1:1 vs. 1:2000 vs. 1:4000

Animace s deformací pravidelné textury splnila očekávání a u oblastí s velkým posunem bodů je možné pozorovat značnou deformaci textury. Navíc interaktivní nastavení pozice terénu vůči pozorovateli umožňuje uživateli zvolit správný úhel pohledu pro pozorování deformace.

Animace s napínáním textury nenaplnila původní představu. Přestože čtverce byly úspěšně vytvořeny a textura byla bez problémů namapována, tak animace zobrazuje očekávaný efekt napínání pouze u ojedinělých hran. U oblastí, kde se nachází více bodů blízko sebe, tak se čtverce dokonce navzájem překrývají. Výsledná vizualizace neukazuje dobře celkovou deformaci daného terénu. Jedním z potencionálních způsobů, jak by mohl být výsledný efekt vylepšen, by bylo změnit výpočet souřadnic do textury, tak aby více zohledňoval velikost deformace a vzdálenost mezi jednotlivými body.



Obrázek 22: Nesprávné zobrazení při animaci s napínáním textury

Možná vylepšení vytvořených programů by se mohla zaměřit na zlepšení výkonu a snížení paměti. Vzhledem k tomu, že vrstva trojúhelníků a vrstva linií pracuje se stejnými daty, tak by pro snížení vytíženosti paměti mohly být tyto vrstvy spojeny. V takovém případě by bylo nutné najít jiné řešení pro problém Z-fighting popsaný v kapitole 6.1. Pro zlepšení výkonu by mohl být výpočet barvy a pozice bodů přenesen na grafickou kartu.

8 Závěr

Bakalářská práce zpracovala téma vizualizace geografických dat ve webovém prostředí. Cílem práce bylo prozkoumat, popsat a využít dostupné technologie pro vytvoření interaktivních vizualizací geografických dat běžících v reálném čase.

V teoretické části práce byly popsány geografická data, nepravidelné trojúhelníkové sítě a softwarové knihovny využívané k vizualizaci. V kapitole o geografických datech byl představen geografický informační systém, software a hardware používaný pro vizualizaci geografických dat. Část o geografických datech popisovala především souřadnicové systémy, datové modely a způsoby modelování terénu. Další kapitola byla věnována nepravidelným trojúhelníkovým sítím a způsobům jejich tvorby, což bylo potřebné pro vypracování praktické části. Kapitola o softwarových technologií pro vizualizaci ve webovém prostředí popsala grafické rozhraní WebGL a framework deck.gl.

Praktická část se zabývala tvorbou vlastních vizualizací. Použitá geografická data byla poskytnuta soukromou firmu, která zároveň uvedla, jaké technologie mají být použity při implementaci. V rámci práce byly vytvořeny vizualizace geografických dat založené na animaci trojúhelníkové sítě. Výsledné vizualizace používají pro znázornění terénních deformací změnu výšky, barvy, souřadnic do textury a vytvoření obdélníku s namapovanou texturou.

Implementované metody byly testovány na dostupných datech. Triangulace jednotlivých bodů proběhla správně a vizualizace výsledné trojúhelníkové sítě byla v pořádku. Výsledná aplikace umožňuje zobrazení v reálném čase s možností interakce a animace trojúhelníkové sítě podle jednotlivých měření. Metoda mapování pravidelné textury pro lepší znázornění deformací vytvořila předpokládaný efekt. Poslední metoda vizualizace využívající napínání textury mezi jednotlivými body trojúhelníkové sítě nenaplnila očekávaní. Řešení vyskytnutých problémů bylo popsáno v části implementace. V části zhodnocení byly uvedeny možné způsoby vylepšení poslední vizualizace a další změny v implementaci, které by mohly zlepšit výkon programu.

Možné rozšíření vizualizací by se mohlo zaměřit na využitích dalších parametrů měření. Jak bylo zmíněno v kapitole 5.2, při implementaci byly použity pouze atributy vztahující se k pozici bodu a velikosti deformace, přičemž byly zanedbány atributy jako sklonitost svahu nebo úhel dopadu radarového paprsku.

9 Seznam použité literatury

[1] *deck.gl* [online]. Dostupné z: https://deck.gl/

[2] BOLSTAD, Paul. *GIS Fundamentals: A First Text on Geographic Information Systems*. 5. vyd. Minnesota: Eider, 2016. ISBN 978-1-5066-9587-7.

[3] DENT, Borden, Jeffrey TORGUSON a Thomas HODLER. *Cartography: Thematic Map Design*. 6. vyd. New York: McGraw-Hill Education, 2008. ISBN 978-0-07-294382-5.

[4] The Ultimate List of GIS Formats and Geospatial File Extensions. *Gisgeography* [online]. 2022. Dostupné z: https://gisgeography.com/gis-formats/

[5] *GEOJSON* [online]. Dostupné z: https://geojson.org/

[6] EL-SHEIMY, Naser, Caterina VALEO a Ayman HABIB. *Digital Terrain Modeling: Acquisition, Manipulation and Applications*. Illustrated edition. Florida: Artech House Publishers, 2005. ISBN 978-1-58053-921-0.

[7] LI, Zhilin, Qing ZHU a Christopher GOLD. *Digital Terrain Modeling: Principles and Methodology*. 1. vyd. Norwood: CRC PRESS, 2005. ISBN 0-415-32462-9.

[8] BERG, Mark. *Computational geometry: algorithms and applications*. 3. vyd. Berlin: Springer, 2008. ISBN 978-3-540-77973-5.

[9] SCHÄFER, Michael. *Computational Engineering – Introduction to Numerical Methods*. 2006th edition. Darmstadt: Springer, 2006. ISBN 978-3-540-30685-6.

[10] CHEN, Li. *Digital and Discrete Geometry: Theory and Algorithms*. 2014th edition. Washington: Springer, 2014. ISBN 978-3-319-12098-0.

[11] MILES, J. P. a M. FARRASHKHALVAT. *Basic Structured Grid Generation: with an introduction to unstructured grid generation*. 1. vyd. Oxford: Butterworth-Heinemann, 2003. ISBN 0-7506-5058-3.

[12] JAYAWANT, Pallavi, Martha KOSA a Christine SHANNON. *Reconstructing Curves from Sample Data: Implementing Algorithms using Voronoi Diagrams and Delaunay Triangulations* [online]. B.m.: DIMACS. 2008. Dostupné z: http://archive.dimacs.rutgers.edu/Publications/Modules/Module08-5/dimacs08-5.pdf

[13] THE KHRONOS GROUP INC. *WebGL Overview* [online]. Dostupné z: https://www.khronos.org/webgl/

[14] PARISI, Tony. *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages*. 1. vyd. Sebastopol: O'Reilly Media, 2014. ISBN 978-1-4493-6296-6.

[15] MATSUDA, Kouichi a Rodger LEA. *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL*. Pap/Psc edition. Michigan: Addison-Wesley, 2013.

[16] ARORA, Sumeet. *WebGL Game Development*. Illustrated edition. Birmingham: Packt Publishing, 2014. ISBN 978-1-84969-979-2.

[17] PARISI, Tony. *WebGL: Up and Running: Building 3D Graphics for the Web*. 1. vyd. Sebastopol: O'Reilly Media, 2012. ISBN 978-1-4493-2357-8.

[18] *Luma.gl* [online]. Dostupné z: https://luma.gl/

[19] *Loaders.gl* [online]. Dostupné z: https://loaders.gl/

[20] *React-map-gl* [online]. Dostupné z: https://visgl.github.io/react-map-gl/

[21] EKR, Jan. *Vizualizace geografických dat v prostředí webového prohlížeče s využitím programovatelných grafických karet* [online]. Hradec Králové, 2022. UHK. Dostupné z: https://theses.cz/id/beuurq/STAG96398.pdf

10 Přílohy

1) Kompletní zdrojový kód
2) Implementace Bowyer–Watsonova algoritmu
3) Změna Buffer
4) Přidání nového atributu pomocí AttributeManageru
5) Výpočet souřadnic do textury pro každý bod trojúhelníkové sítě
6) Výpočet souřadnic vrcholů obdélníku

Kompletní zdrojový kód

Kompletní zdrojový kód je dostupný v podobě git repozitáře na adrese:
https://gitlab.com/kabrtt/geographical-data-visualization

Implementace Bowyer–Watsonova algoritmu

```
//Bowyer-Watson algorithm
/**
 * points = array of points to be triangulated [[x1,y1],[x2,y2]...]
 *
 * triangles = array of mesh triangles [[[x1,y1], [x2,y2], [x3,y3]]...]
 */

function triangulate(points){
  //add super triangle which contains all the points
  let superTriangle = findSuperTriangle(points)
  let triangles = [superTriangle];

  //add points one by one
  points.forEach(point => {

    //find all the bad triangles
    let badTriangles = [];

    triangles.forEach(triangle => {
     if(isPointInsideCircumcircle(...point,
     ...triangle[0],...triangle[1], ...triangle[2])){
        badTriangles.push(triangle);
      }
    });

    //remove bad triangles from triangulation
    badTriangles.forEach(badTriangle => {
      triangles.forEach((triangle,i) => {
        if(badTriangle.every((value,j) => value === triangle[j])){
          delete triangles[i];
        }
      });
      triangles = triangles.filter(x => x !== undefined);
    });

    //find the boundary of the polygon
    let polygon = [];

    badTriangles.forEach(badTriangle => {
      let edges = [
        [badTriangle[0], badTriangle[1]],
        [badTriangle[1], badTriangle[2]],
        [badTriangle[0],badTriangle[2]]
      ]
```

```
      edges.forEach(edge => {
        if(!isEdgeShared(edge,badTriangles)){
          polygon.push(edge);
        }
      });
    });

    //re-triangulate the polygonal hole
    polygon.forEach(edge => triangles.push([...edge, point]));
  });

  //delete triangles which have superTriangle vertices
  triangles.forEach((triangle,index) => {
    if(containsVertexFromOriginal(triangle,superTriangle)){
      delete triangles[index];
    }
  });

  triangles = triangles.filter(x => x !== undefined);

  return triangles;
}
```

Změna Buffer

```
const colorBuffer = new Float32Array(data.length * 3);
const positionBuffer = new Float32Array(data.length * 3);

…výpočet hodnot

const colorLocation = gl.getAttribLocation(model.getProgram().handle,
'color');
model.vertexArray.setBuffer(colorLocation, new Buffer(gl, {
      usage: gl.DYNAMIC_DRAW,
      data: colorBuffer
}));

const positionLocation = gl.getAttribLocation(model.getProgram().handle,
'position');
      model.vertexArray.setBuffer(positionLocation, new Buffer(gl, {
            usage: gl.DYNAMIC_DRAW,
            data: positionBuffer,
}));
```

Přidání nového atributu pomocí AttributeManageru

```
const attributeManager = this.getAttributeManager();
attributeManager.add({
    textureCoord: {
       size: 2,
       type: gl.FLOAT,
       update: this.getTextureCoordinates
    },
});
```

Výpočet souřadnic do textury pro každý bod trojúhelníkové sítě

```
 let [minX,minY,maxX,maxY] = [Infinity,Infinity,-Infinity,-Infinity]

 data.forEach(element => {
     minX = Math.min(minX, element.coordinates[0]);
     minY = Math.min(minY, element.coordinates[1]);
     maxX = Math.max(maxX, element.coordinates[0]);
     maxY = Math.max(maxY, element.coordinates[1]);
 });

 let regionSize =
 (maxX-minX) > (maxY - minY) ? (maxX-minX) : (maxY - minY);

const textureCoordBuffer = new Float32Array(data.length * 2);
let textureSize = 50

data.forEach((element, index) => {
   let position = element.coordinates;
   let textureX = (position[0]-minX)*(textureSize)/(regionSize)
   let textureY = (position[1]-minY)*(textureSize)/(regionSize)

   textureCoordBuffer[index * 2 + 0] = textureX;
   textureCoordBuffer[index * 2 + 1] = textureY;
});
```

Výpočet souřadnic vrcholů obdélníku

```
let position1 = data[i].coordinates;
let position2 = data[i+1].coordinates;

let vector = [-1*(position2[1]-position1[1]), position2[0]-position1[0]];

let vectorSize = Math.pow(Math.pow(vector[0], 2)+Math.pow(vector[1], 2),
0.5);

let scale = 0.00005/vectorSize;
let recSize = [scale*vector[0], scale*vector[1]];

let point1 = [position1[0]-recSize[0],position1[1]-recSize[1]]
let point2 = [position1[0]+recSize[0], position1[1]+recSize[1]]
let point3 = [position2[0]-recSize[0], position2[1]-recSize[1]]
let point4 = [position2[0]+recSize[0], position2[1]+recSize[1]]
```

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Tomáš Kábrt**

Studium: I2000369

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Vizualizace geografických dat ve webovém prostředí**

Název bakalářské práce AJ: Visualization of geographical data on the web

Cíl, metody, literatura, předpoklady:

Cíl práce:

Cílem práce je prozkoumat přístupy pro zobrazení 3D scény v prostředí webového prohlížeče. Pro zvolenou technologii navrhnout, implementovat a otestovat řešení pracující v reálném čase.

Postup prací:

1. Prozkoumat principy a metody zobrazení geografických informací

2. Vytvořit přehled softwarových řešení pro práci s 3D geografickými daty v prostředí webového prohlížeče

3. Pro vybranou technologii navrhnout řešení s využitím programovatelných grafických karet

4. Zaměřit se na vizualizaci atributů geografických dat měnících se v čase

5. Navržené řešení implementovat a otestovat

6. Zhodnotit dosažené výsledky

BOLSTAD, Paul. GIS Fundamentals: A First Text on Geographic Information Systems. Eider, 2016.

MATSUDA, Kouichi a Rodger LEA. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. Addison-Wesley, 2013.

GHAYOUR, Farhad a Diego CANTOR. Real-Time 3D Graphics with WebGL 2 - Second Edition. Packt, 2018.

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Ing. Bruno Ježek, Ph.D.

Datum zadání závěrečné práce: 26.1.2021