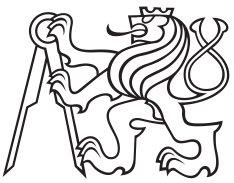
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Webová aplikace pro vizualizaci dynamického KML

Tom Nováček
Otevřená informatika

Květen 2015
Vedoucí práce: RNDr. Michal Čertický, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Tom Nováček**

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: **Webová aplikace pro vizualizaci dynamického KML**

Pokyny pro vypracování:

Navrhněte a implementujte webovou aplikaci schopnou vizualizovat konkrétní podmnožinu Keyhole Markup Language (KML), která odpovídá dynamickým změnám v čase. Aplikace by v porovnání s existujícími alternativami měla být schopná zobrazit znatelně větší množství pohybujících se prvků.

1. Seznamte se s KML jako prostředkem k vyjádření časově závislých geografických dat.
2. Analyzujte omezení existujících vizualizačních nástrojů a dostupných webových frameworků, které by umožnily vytvořit aplikaci pro velký počet dynamických prvků.
3. Navrhněte datový model, který bude minimalizovat požadavky na systémové zdroje.
4. Navrhněte uživatelské rozhraní pro aplikaci.
5. Navrhněte architekturu s ohledem na analýzu a datový model z kroků 2 a 3.
6. Implementujte aplikaci a otestujte její chování a výkon na různých vstupních datech.

Seznam odborné literatury:

- Nolan, Temple Lang. XML and Web Technologies for Data Sciences with R. 2014.
- Ying-jun, D., Yu, C. C., & Jie, L. A study of GIS development based on KML and Google Earth. NCM'09. Fifth International Joint Conference on (pp. 1581-1585). IEEE. 2009.
- John Rohlf, Brian McClendon. Markup language for an interactive geographic information system. 2008.

Vedoucí: RNDr. Michal Čertický, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 3. 2015

Poděkování / Prohlášení

Rád bych touto cestou poděkoval panu RNDr. Michalovi Čertickému, Ph.D. za cenné rady a motivující vedení bakalářské práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22. 5. 2015

.....

Abstrakt / Abstract

Tato bakalářská práce se věnuje problému vizualizace velkého počtu geografických jevů a objektů. V poslední době se zvyšuje poptávka po nástrojích umožnujících provádět simulace s reálnými daty, která však bývají rozsáhlá a pro jejich zobrazení je často nutné použít paralelismus grafických karet. Dříve bylo prakticky nemožné vytvořit tyto nástroje a pracovat s nimi jinak než za použití specializovaného softwaru, dnes je možné díky pokroku v oblasti webových technologií provádět některé výpočty na grafické kartě přimo z prostředí webového prohlížeče. Aplikace vyvinutá v rámci této práce využívá těchto technologií ke zpracování a animování velkého množství dynamických objektů.

Klíčová slova: KML, vizualizace, GIS, geodata, WebGL, Cesium

This thesis deals with the problem of rendering a large number of geographic phenomena and objects. There has been an increase in demand for visualization tools allowing to perform simulations with real data. These data can be very extensive so their rendering requires the use of parallel computing on the graphics card. Thanks to advances in Web technologies, it is now possible to perform some calculations on the graphics card using just a Web browser. The purpose of the devised application is to use these technologies to process and animate a large number of dynamic objects.

Keywords: KML, visualization, GIS, spatial data, WebGL, Cesium

Title translation: Web-based visualizer for dynamic KML

Obsah /

Tabulky /

Kapitola 1
Úvod

S rozvojem internetu a potřebou zpracovávat geografická data online začaly vznikat informační systémy, které souhrnně označujeme jako geografické informační systémy (GIS). S těmito systémy přímo souvisí geografická data, tzv. *geodata*, která můžeme definovat jako soubor dat obsahující prostorovou složku vyjadřující jejich vztah ke konkrétnímu místu nebo oblasti na zemském povrchu. Pod pojmem geodata si tedy můžeme představit většinu objektů a jevů reálného světa [1]. Zdrojem vstupních dat mohou být například letecké snímky nebo geodetická měření. GIS se zabývá problémy, při kterých využíváme znalosti polohy a vzájemných prostorových souvislostí mezi objekty.

Prostorová složka však není jedinou součástí těchto dat, pro maximální přínos geografické analýzy obsahují dále složku časovou a atributovou. První z nich může odkazovat na konkrétní okamžik či časový interval (v tom případě mluvíme o *dynamických jevech*), druhá pak obsahuje popisné informace prostorové složky. Při vývoji GIS se musíme vypořádat s podstatným problémem, a to uchováváním a distribucí samotných geodat. V průběhu let proto vzniklo mnoho formátů, které se snažily vytvořit jednotný způsob zápisu geografických informací do souboru, čímž zjednodušily tvorbu a výměnu těchto dat mezi vývojáři.

Praktické užití GIS je velmi různorodé, například ve státní správě (katastr nemovitostí), pro správu inženýrských sítí, v urbanismu (tvorba územního plánu) nebo při plánování dopravy (sledování pohybu vozidel, osob). Pro všechny zmíněné případy užití je velmi důležité umět geodata také detailně a přesně zobrazovat. Zejména při modelování a simulaci komplexních dopravních systémů se však vzhledem k objemu těchto dat (například pohyb všech pasažérů v městské aglomeraci), která jsou navíc dynamická a jejich časová složka je typicky velmi rozsáhlá, jedná o náročný úkol. Výstupem takového zobrazení může být mapa, trojrozměrný model území nebo animace konkrétního jevu. Trendem posledních let je prezentace dat v prostředí internetu, proto jsou pro aplikace využívající mapové podklady spravovány mapové servery (tzv. Web Map Services) [2].

1.1 Cíl práce

Cílem práce je vytvořit open-source aplikaci schopnou zobrazit velké množství dynamických prvků z geodat. V současné době je k dispozici mnoho nástrojů, které vizualizaci umožňují, většina z nich se ale soustředí spíše na podporu široké škály prvků a zpracování velkých souborů jim činí zásadní problém. Tato práce by tak na základě analýzy těchto nástrojů měla přijít s výpočetně méně náročným způsobem animování při zachování plynulosti pohybu.

Pokrok ve vývoji webových prohlížečů, především pak díky technologiím WebGL, HTML5 a JavaScript, umožnil provádět některé grafické výpočty bez nutnosti instalace dalších programů [3]. Tím se vývojářům otevřela možnost řešit výpočetně náročné úlohy, jako je například práce s 3D grafikou, přímo v rámci webové aplikace. Značně se tak zvýšila dostupnost pro široký okruh uživatelů, navíc lze tento výkon využít ke

zpracování velkého počtu prvků. Z těchto důvodů bude nově vzniklá aplikace vytvořena jako webová.

Jádro aplikace by mělo být vytvořeno tak, aby v budoucnu v případě potřeby a zájmu umožňovalo snadné rozšíření. S ohledem na stabilitu by měly nejnáročnější výpočty probíhat na straně klienta, čímž se přenese zátěž ze serveru.

Kapitola 2
Keyhole Markup Language

Jedním z dnes velmi rozšířených formátů GIS souborů je Keyhole Markup Language (KML), který byl vyvinut firmou Keyhole, Inc. pro využití ve vizualizačním nástroji Keyhole Earth Viewer. Jednalo se o trojrozměrný virtuální glóbus který umožňoval zobrazit geodata z KML na mapový podklad. Firmu následně v roce 2004 převzala společnost Google, která o rok později představila novou verzi glóbu, tentokrát pod označením *Google Earth* [4].

Jak již z názvu vyplývá, KML patří do skupiny značkovacích jazyků, konkrétně je rozšířením známého jazyka XML. Vzhledem k velkému množství nástrojů a programovacích jazyků, které zpracování XML podporují (což souvisí s jeho velkou oblibou při tvorbě webových stránek společně v kombinaci s HTML), je manipulace s KML poměrně jednoduchá. Rozmachu navíc velkou mírou přispívá popularita Google Earth mezi vizualizačními aplikacemi. I přesto, že KML byl původně určen pouze pro jednu aplikaci, byl v roce 2008 přijat ve verzi 2.2 jako standard mezinárodní organizací Open Geospatial Consorcium (OGC).

KML však není jediným značkovacím jazykem pro geodata, dalším je například Geography Markup Language (GML), který je také zařazen do standardů OGC. Oba se liší v účelu a vzájemně se spíše doplňují - KML je zaměřen více na vizualizaci, zahrnuje třeba i reference na obrázky. Určuje také způsob, jakým budou data uživateli zobrazena (např. z jaké výšky a v jakém měřítku). Oproti tomu GML popisuje a rozlišuje široké spektrum geografických objektů, které v KML nenajdeme. Soustředí se spíše na obsah a strukturu geodat [5].

2.1 Struktura XML

Značkovací jazyk XML slouží jako výměnný formát dat. Základem jsou párové značky (tzv. tagy), které vyznačují význam jednotlivých částí textu. Standardně se těmto označeným částem říká *elementy*. Elementy do sebe mohou být navzájem vnořené a tím dle potřeby zachycovat strukturu informací uložených v dokumentu. Sada značek, na rozdíl od HTML, však není pro XML pevně daná, což uživateli umožňuje tvorbu vlastních značek, jejichž sémantiku pak určuje aplikační logika. Jazyk tedy obecně určuje pouze syntaxi pro tvorbu elementů. Celý dokument musí být obsažen v jednom elementu, kterému se říká kořenový [6].

```
<?xml version="1.0" encoding="UTF-8"?>
   <note>
       <to>Tove</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don't forget me this weekend!</body>
   </note>
```

Předchozí ukázka [7] zachycuje jednoduchou poznámku převedenou do dokumentu XML. Na jeho začátku určíme způsob kódování znaků pro potřeby zobrazení. Poté

následuje kořenový element „note“, do něhož jsou zanořeny další elementy. Tento způsob vytváří stromovou strukturu, ve které je možné se poměrně rychle pohybovat. Jakým způsobem bude interpretován význam např. elementu „body“ je pouze věci obsluhující aplikace.

2.2 Struktura KML

Jazyk KML převádí geodata do struktury XML dokumentu. Navíc zavádí nová pravidla pro tvorbu značek. Pokud název elementu začíná velkým písmenem, jedná se o komplexní element, což znamená, že může obsahovat další elementy (není tedy možné zanořit do sebe dva elementy s malým počátečním písmenem). Jedinou výjimku tvoří element kml, který je zároveň vždy kořenem celého souboru.

Pokud se v souboru KML vyskytne relativní adresa odkazující na důležitou součást vizualizace (např. obrázek), bez které by nešlo obsah zobrazit, je nutné jej komprimovat společně se všemi potřebnými soubory a adresáři do ZIP archivu s koncovkou *KMZ*. Google Earth podporuje načítání těchto archivů, není tedy nutné jej před použitím znovu dekomprimovat. Důležité však je zachovat podmínku jednoho KML souboru v archivu, nejčastěji pojmenovaném jako „doc.kml“ [8].

Pro určení pozice využívá KML tři souřadnice - zeměpisnou šířku, délku a výšku. První dvě souřadnice jsou definovány podle World Geodetic System 1984 (WGS84). Zápis výšky je uveden v metrech, nejedná se však o skutečnou nadmořskou výšku, ale vzdálenost od elipsoidu, což může oproti skutečné výšce způsobit odchylku až 100 m [9]. Pokud není v pozici výška specifikována, automaticky se stanoví na hodnotu 0.

2.3 Elementy dynamického KML

Počet objektů a jevů reálného světa, které KML prostřednictvím svých elementů pokrývá, je velmi rozsáhlý. Jejich celkový přehled je dostupný v online dokumentaci[1]), a není účelem této práce všechny podrobně popsat.

Pro potřeby vizualizace dynamických geodat je důležitá podmnožina prvků jazyka KML skládající se z následujících elementů.

2.3.1 Placemarks

Placemark patří mezi nejčastěji používané elementy v KML a je také zásadní pro potřeby této práce. Určuje místo na zemském povrchu, které pak Google Earth standardně označí ikonou žlutého špendlíku. V nejjednodušším provedení může obsahovat pouze Point element, u něhož specifikujeme souřadnice dle WGS84.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
 <Placemark>
<name>Simple placemark</name>
<description>Attached to the ground. Intelligently places itself
     at the height of the underlying terrain.</description>
<Point>
  <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
</Point>
 </Placemark>
</kml>
```

[1]) https://developers.google.com/kml/documentation

Struktura předchozího jednoduchého souboru se dělí následovně:

- První řádek vždy obsahuje hlavičku XML dokumentu.
- Druhý řádek deklaruje jmenný prostor KML pro verzi 2.2.
- Objekt Placemark obsahující následující elementy:

a) name - jméno, které bude automaticky zobrazeno
b) description - popis objektu, bude zobrazen při kliknutí
c) Point - obsahuje souřadnice zeměpisné délky, šířky a volitelně výšky

Obrázek 2.1. Ukázka Placemark elementu v aplikaci Google Earth.

Důležitosti Placemark elementu navíc přidává fakt, že se jedná o jediné místo v KML, do kterého můžeme vložit geometrické prvky [10]. Kromě již zmíněného „Point“ mezi ně patří ještě například Polygon, LineString a gx:Track. Posledně jmenovaný zajišťuje dynamiku objektu, pro jejíž vizualizaci byla primárně vyvíjena aplikace pro tuto práci.

2.3.2 Style

Styl definuje skupinu pravidel, která ovlivňují vzhled a způsob zobrazení jednotlivých elementů (pro geometrické prvky navíc ve 3D režimu). Je žádoucí, aby styly mohly být sdílené a nemusely tak být uváděny u každého prvku zvlášť. Proto bývají zahrnuty v elementu Document, v jehož rámci pak mohou být snadno referencovány podle unikátního id přes styleUrl.

```
...
<Document>
<Style id="exampleStyleDocument">
  <LabelStyle>
        <color>ff0000cc</color>
  </LabelStyle>
 </Style>
 <Placemark>
    ...
 <styleUrl>#exampleStyleDocument</styleUrl>
    ...
 </Placemark>
</Document>
...
```

V této ukázce je vytvořen jeden Style element se specifikací barvy (LabelStyle) pro element name. Element Placemark, který je vnořený do stejného Document elementu jako Style pak obsahuje pouze referenci tohoto pravidla.

2.3.3 Track

Tento element popisuje pohyb objektu po zemském povrchu v daném časovém intervalu. Jeho trajektorie je zaznamenána pomocí množiny bodů, která odpovídá souřadnicím v konkrétním čase. Elementy <*gx:coord*> (poloha) a <*when*> (čas) tvoří vždy dvojici a je důležité zachovat jejich pořadí. Změna polohy mezi dvěma souřadnicemi je interpolována, proto se výsledný pohyb jeví jako plynulý na celém intervalu.

2.3.4 LineString

Jedná se o množinu úseček, z nichž každá je určena vždy dvěma souřadnicemi. Tyto úsečky na sebe logicky navazují podle pořadí, v jakém jsou zapsány v <coordinates> elementu, tedy dvě po sobě jdoucí souřadnice vždy označují jednu stranu celkového řetězce. Jeho vizualizací je pak lomená čára. Následující příklad zachycuje řetězec se třemi vrcholy.

```
<Placemark>
 <LineString>
  <coordinates>
   -76.05178073508411,42.09494608432735,0
   -76.05203053658903,42.09552528912739,0
   -76.05139398680267,42.09632132125732,0
   -76.05101125304151,42.09563504340319,0
   -76.05163726591572,42.09542358349865,0
  </coordinates>
 <LineString>
</Placemark>
```

Obrázek 2.2. Ukázka LineString elementu v aplikaci Google Earth.

Protože první a poslední souřadnice nejsou v LineString propojeny, není tento řetězec uzavřený. Pokud bychom takový chtěli vytvořit, musíme použít element LinearRing. Pro upřesnění grafické podoby řetězce se používá element LineStyle. Ten může obsahovat informace například o barvě nebo tloušťce čáry.

Kapitola 3
Analýza dostupných technologií a služeb

Tato kapitola přiblíží důvody, které jsou zmíněny v 1.1 a které vedly k tomu, že aplikace pro vizualizaci dynamického KML bude webová. Práce s 3D grafikou vždy patřila mezi výpočetně nejnáročnější oblasti informatiky a její tvorba se neobešla bez specializovaných programů s podporou knihoven jako OpenGL nebo Direct3D.

S příchodem HTML5 (nové specifikace značkovacího jazyka pro tvorbu webových stránek), se však velmi rozšířily možnosti pro kreslení grafických prvků díky elementu canvas. Ten slouží k dynamickému vykreslení do bitmapového plátna za použití skriptu. Canvas vykresluje pixely v tzv *„immediate mode“*, což znamená, že se pro každý jednotlivý snímek překreslí vždy celá plocha. Rozdíl oproti *„retained mode“*, který využívají třeba Flash nebo SVG[1]), je v tom, že canvas po vykreslení neumožňuje manipulaci s konkrétním tvarem [11]. Místo reference na vytvořený grafický objekt zná pouze instrukce pro vykreslení každého snímku. Název canvas (česky plátno) tedy odkazuje na vlastnosti reálné malby.

3.1 JavaScript a Canvas

Spojení těchto dvou technologií vychází z toho, že instrukce pro kresbu do elementu canvas se předávají skriptem, a vzhledem ke snadnému přístupu JavaScriptu do Document Object Modelu (kterého je canvas v rámci HTML stránky součástí), jde nejčastěji právě o něj. Výhodou tohoto přístupu je možnost interakce JavaScriptové funkcionality (vstupů z klávesnice, myši, matematických funkcí) s grafickou částí stránky, což přináší zajímavou možnost při tvorbě animací nebo her přímo ve webové aplikaci.

Obrázek 3.1. Obrázek vytvořený v canvas HTML5 [12].

[1]) Scalable Vector Graphics škálovatelná vektorová grafika je značkovací jazyk a formát souboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML

3.2 WebGL

Canvas byl vytvořen primárně ve 2D kontextu, který však pro potřeby náročnějších aplikací není dostatečný. Vývojáři proto začali experimentovat s 3D kontextem ve webových stránkách, který by zároveň nevyžadoval instalaci dalších pluginů. Problémem ale byla velká náročnost takových stránek na samotný prohlížeč a procesor počítače (CPU). Vzhledem k růstu výpočetních výkonů grafických karet (přesněji grafických procesorů na nich umístěných) se jako nejlepší řešení ukázalo odstranění zátěže z CPU a její předání nevyužité grafické kartě. Tomuto zrychlení vykreslování grafického obsahu se říká *hardwarová akcelerace*.

I když je primárním účelem hardwarové akcelerace podpora 3D grafiky, lze této technologie díky poskytnutému výkonu grafické karty využít i pro zpracování velkého počtu prvků - což je případ naší aplikace.

V roce 2011 se objevila oficiální verze nové technologie WebGL, jejímž účelem je právě podpora hardwarové akcelerace pro webové prohlížeče [13]. Překvapením pak není, že WebGL je implementováno v JavaScriptu, čímž je zajištěn rychlý přístup a komunikace se všemi prvky webové stránky. Není také nutné tento kód v prohlížeči kompilovat do binární podoby pro konkrétní platformu, navíc se WebGL stará o automatickou alokaci paměti (na rozdíl třeba od jazyka C). K vývoji 3D grafických aplikací je tak možné použít pouze textový editor a prohlížeč.

3.3 Google Earth

Patrně nejznámější virtuální glóbus současnosti. Nabízí nejširší podporu KML souborů mezi vizualizačními nástroji, což je pochopitelné z důvodů již zmíněných v kapitole o KML. Mezi jeho hlavní výhody patří především rychlost zpracování souborů, široké pokrytí mapových podkladů s detailním rozlišením a velká komunita vývojářů. Je k dispozici zdarma v základní verzi a s licenčním klíčem ve verzi Pro, která přináší řadu pokročilých funkcí určených především pro firmy. Vzhledem k propojenosti Google Earth a Google Maps se podpora pro KML dostala i do druhé zmíněné aplikace, pochopitelně však v omezené míře (především v mobilní verzi). Se soubory se dá pracovat jako s vrstvami, které se do aplikace mohou jednotlivě nahrát a podle potřeby odkrývat/skrývat. Typický případ užití pak zahrnuje několik KML a KMZ souborů.

Nevýhodou je nedostatečný výkon při zpracování velkého (řádově desítky až stovky MB) souboru s dynamickými prvky, při kterém je znatelně snížena plynulost animace celého pohybu.

Vzhledem k tomu, že vývojáři prohlížečů Chrome a Firefox oznámili ukončení podpory pro NPAPI framework, který umožnoval začlenit Google Earth *API*[1]) do aplikací za použití JavaSriptu, byl nucen Google ukončit podporu pro toto rozhraní k prosinci 2015.

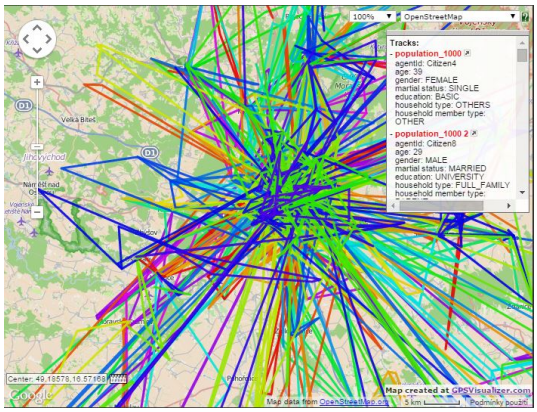
3.4 GPS Visualizer

Online[2]) dostupná služba je určená především pro vizualizaci výstupů z GPS navigací, navíc ale také nabízí podporu Google Earth elementů gx:Track a LineString v souborech KML a KMZ. Mezi výhody patří možnost tvorby vlastních map a souborů (pro KML

[1]) Application Programming Interface označuje v informatice rozhraní pro programování aplikací
[2]) `http://www.gpsvisualizer.com/`

pomocí Google Earth API) a množství pokročilých funkcí, jako výpočet vzdálenosti, převod mezi souřadnicovými systémy nebo označení oblasti na mapě s daným rádiusem. Užitečná je i možnost výběru z několika mapových podkladů.

Nevýhodou je ale naprostá absence podpory dynamické vizualizace, služba nerozlišuje mezi časovými značkami jednotlivých elementů a zobrazí je všechny naráz. Navíc ignoruje grafický styl určený souborem, a elementům přiřazuje několik základních barev, což zapříčiní velmi nepřehledný výsledek, obzvlášť u většího počtu Placemark elementů. Pro velké soubory se tedy nehodí a je určen spíše pro turistické plánování trasy.



Obrázek 3.2. Ukázka aplikace GPS Visualizer.

3.5 CesiumJS

Za končící Google Earth API bylo nutné najít odpovídající alternativu. Tohoto úkolu se zhostil JavaScriptový framework Cesium[1]), který využívá hardwarové akcelerace grafiky pomocí WebGL. Navíc je vyvinut jako multiplatformní a nezávislý na použitém prohlížeči. Přichází s vlastním jazykem Cesium Markup Language (CZML), který je primárně určen k popisu dynamické scény. Od verze 1.7 také nabízí základní podporu pro KML.

Obrázek 3.3. Ukázka virtuálního glóbu Cesium.

Výhodou tohoto frameworku je výborná dokumentace s množstvím návodů a ukázek, které umožňují snadné vytvoření vlastní aplikace. Vyznačuje se velkou podporou pro tvorbu geometrických prvků a díky WebGL je také poměrně výkonný. Pro tyto důvody byl vybrán jako *základ při tvorbě našeho visualizeru.*

[1]) `http://cesiumjs.org/`

Podobně jako Google Earth, Cesium je limitován při použití velkých souborů, na rozdíl od něj by však nedokázal takové KML vůbec zpracovat díky vzrůstajícím paměťovým nárokům. Tento problém je způsoben především dvěma důvody. Zaprvé tím, že Cesium parsuje XML soubory pomocí tzv. DOM parseru, který se pro velké soubory nehodí (podstata je vysvětlena v kapitole 6.2.1), zadruhé proto, že všechny dynamické prvky animuje se stejnou frekvencí, což logicky vede ke ztrátě plynulosti při jejich větším počtu.

Kapitola 4
Analýza požadavků a rizik

Nově vzniklá aplikace by měla mít podobnou funkcionalitu jako Google Earth, stejně tak uživatelské rozhraní by mělo být co nejvíce intuitivní bez nutnosti předchozí znalosti dalšího nástroje. Počáteční analýza a specifikace nezbytných částí aplikace slouží především jako výchozí bod pro pozdější návrhovou a implementační část.

4.1 Funkční požadavky

Funkční požadavky definují základní funkce, které by měla aplikace obsahovat. Každý požadavek by měl popisovat pouze jednu samostatnou a měřitelnou funkcionalitu.

- Aplikace umožní načítat soubory KML i KMZ.
- Aplikace bude schopna zobrazit velké množství gx:Track elementů.
- Aplikace bude také schopna zobrazit LineString elementy.
- Součástí aplikace bude interaktivní posuvník po časové ose.
- Obsah z elementu Description bude zobrazen při kliknuti v samostatném okně.
- Pohyb elementu gx:Track bude interpolován.
- Aplikace bude podporovat zobrazování KML ve vrstvách.
- Aplikace bude informovat uživatele o průběhu zpracování KML souboru.
- Animace bude v detailu plynulá i při velkém počtu Placemark elementů.
- Na časové ose bude animován pohyb vpřed i vzad.
- Aplikace zvládne na běžném osobním počítači zpracovat soubory velké řádově stovky MB.

4.2 Obecné požadavky

Tato sekce specifikuje omezující podmínky, které jsou kladeny na systém, nikoliv na jeho funkcionalitu. Většinou se jedná o použité technologie a omezení týkající se kvality a výkonu.

- Aplikace bude implementována v jazyce JavaScript za využití frameworku Cesium.
- Aplikace ponese název *KML visualizer*, který by měl usnadnit uživateli její nalezení přes vyhledávače.
- Aplikace bude funkční v prohlížečích Chrome a Firefox.
- Aplikace nebude zatěžovat server a výpočty bude provádět na straně klienta.
- Zdrojový kód aplikace bude zpřístupněn ve službě GitHub.

4.3 Analýza problémů a rizik

Pro vývoj KML visualizeru, tedy aplikace určené ke zpracování velkých souborů a velkého počtu dynamických objektů, je zásadní především zachování uživatelsky přijatelné

obnovování frekvence. Pro malé soubory není problém zachování této frekvence na nejvyšší možné úrovni pro všechny objekty, pro naše potřeby by ale na toto už výpočetní výkon nestačil. Je tedy nutné nějakým způsobem rozlišit množinu prioritních objektů, pro které bude výkon využit, a to tak, aby uživatel nepřišel o žádná důležitá data a zároveň se mu animace zdála stále plynulá.

Dalším problémem je samotné načítání KML souboru a tvorba na něj navazujícího datového modelu. Pokud bychom chtěli v průběhu zpracování uchovat v paměti celý soubor, nároky prohlížeče by vzrostly natolik, že by byl výpočet ukončen a vizualizace byla nekompletní. Je tedy nutné zajistit uchování dat v paměti pouze po dobu nezbytně nutnou k okamžitému zpracování. Vzhledem k velkému počtu dynamických objektů, které bude nutné zobrazit na mapový podklad, by uživatel měl mít z důvodu větší přehlednosti možnost skrýt neaktivní objekty, případně skrýt celou vrstvu, pokud bude souborů více.

Velkou nevýhodou pro potřeby této aplikace je fakt, že JavaScript neumožňuje vícevláknové zpracování. Funkce starající se o animování objektů z tohoto důvodu nesmí zahltit výpočet a znemožnit tak současně ovládání aplikace.

Omezující bude také podmínka na přenesení většiny aplikační logiky směrem ke klientovi, díky čemuž nebude možné použít například některé technologie run-time prostředí Node.js[1]), který je určen pro serverovou část. Výhodou tohoto prostředí je zavedení asynchronního způsobu zpracování vstupu/výstupu, což by umožnilo neblokujícím způsobem načítat KML soubory a zároveň je zpracovávat.

V neposlední řadě je třeba zmínit, že aplikace založené na frameworku Cesium pro svůj chod potřebují podporu WebGL. Tento problém se netýká většiny moderních webových prohlížečů, nicméně i u nich se může stát, že je tato technologie deaktivována.

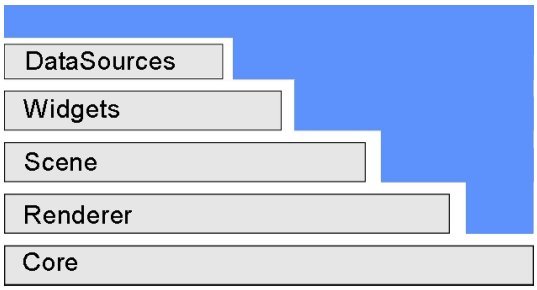
[1]) `https://nodejs.org/`

Kapitola 5
Návrh

5.1 Popis architektury

KML visualizer je založen na frameworku Cesium, a proto by měl při rozšiřování funkcionality pro náš konkrétní případ užití zohledňovat stávající architekturu tak, aby novinky související s příchodem dalších verzí Cesium bylo možné začlenit i do visualizeru.

Struktura jednotlivých JavaScriptových souborů se dělí do několika balíčků, které tvoří vrstvy aplikace. Obecně platí, že každá vrstva přidává funkcionalitu, zvyšuje úroveň abstrakce a závisí na vrstvách pod ní.



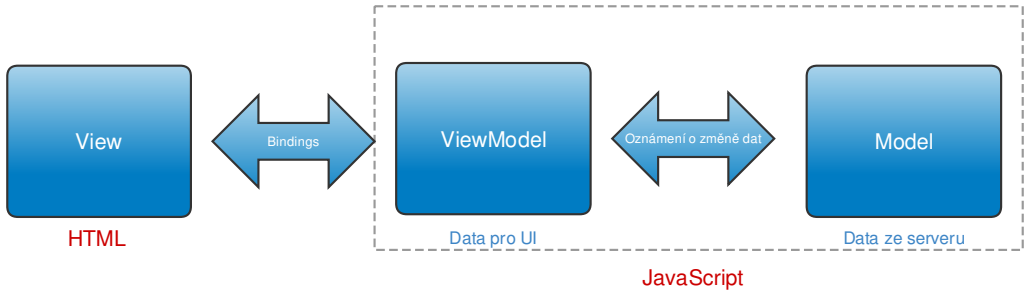
Obrázek 5.1. Hierarchie balíčků ve frameworku Cesium.

Mimo těchto základních balíčků je součástí architektury ještě složka ThirdParty. V té nalezneme několik externích knihoven, jako například KnockoutJS[1]). Ta přináší do aplikace architektonický vzor pro tvorbu uživatelského rozhraní (UI - user interface) Model-View-ViewModel (MVVM). Principem je udržení přehlednosti (i pro velmi rozsáhlá UI) rozdělením do třech částí:

- Model - popisuje objekty z doménového modelu, se kterými aplikace pracuje. Tyto data jsou nezávislá na UI.
- View - Viditelná část aplikace (v našem případě tedy HTML stránka).
- ViewModel - rozhraní mezi View a Modelem. Ovládací prvky z view jsou pomocí tzv. bindingu propojeny s ViewModelem a čerpají z něj svůj obsah.

Důležité je zmínit, že ViewModel je jakýsi prostředník, který udržuje aktuální stav aplikace. Provázanost mezi View a ViewModelem se projevuje tím, že při změně stavu aplikace se změní i UI, a naopak, při interakci uživatele s UI (například kliknutí myší), pošle View informaci ViewModelu. Na rozdíl od jiného známého vzoru Model - View - Controller, kde View získává data přímo z Modelu, probíhá u MVVM veškerá komunikace skrze ViewModel.

[1]) `http://knockoutjs.com/`



Obrázek 5.2. Architektura Model-View-ViewModel.

5.1.1 Core

Nejnižší vrstva, která obsahuje široce využívané matematické funkce a poskytuje tak základ celé aplikaci. Příklady zahrnují:

- Maticové a vektorové výpočty
- Transformace, jako je například převod zeměpisných souřadnic do soustavy kartézských souřadnic (Cartesian).
- Manipulaci s datem (podle Juliánského kalendáře - JulianDate).
- Triangulaci a další metody pro zjišťování souřadnic a vzdáleností v rámci glóbu.

5.1.2 Renderer

Tato vrstva je důležitá především z hlediska použití technologie WebGL. Poskytuje abstrakci pro obsluhu shader programů, což jsou programy, které plně běží na GPU a starají se například o transformace vrcholů nebo zpracování textur.

Aplikace by, až na malé množství výjimek, neměla přistupovat k těmto souborům přímo, ale využívat jednotlivých konstrukcí z vyšších vrstev, především pak z vrstvy Scene.

5.1.3 Scene

Vrstva Scene poskytuje poměrně vysokou úroveň abstrakce pro manipulaci s glóbem. Zahrnuje v sobě všechny 3D grafické objekty a stavy, které můžeme přidávat do virtuální scény aplikace. Tato virtuální scéna obvykle není vytvořena přímo, ale jako součást jiných prvků.

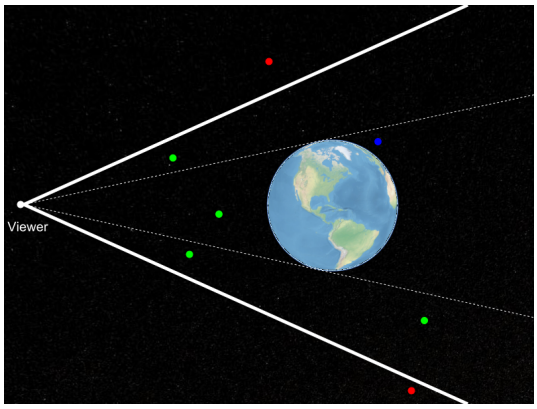
Mezi hlavní funkcionalitu patří:

- API pro práci s 3D glóbem a 2D mapou.
- Nahrávání mapových podkladů ve vysokém rozlišení - Cesium standardně používá Bing Maps, možností je ale nepřeberné množství (např. OpenStreetMap, Web Map Service atd.)
- Tvorba křivek, polygonů, elipsoidů a jejich uchovávání v kolekcích.
- Obsluha kamery a její nastavení v závislosti na vstupu.

Je důležité zmínit, že objekty z této scény jsou přímo svázány s elementem canvas 3.1, který vymezuje jejich prostor v rámci aplikace. Zajímavá je také možnost přidávání jednotlivých mapových podkladů ve vrstvách (tzv. Imagery Layers), kterým můžeme nastavovat průhlednost a podle potřeby měnit jejich pořadí. Scéna navíc dokáže prostřednictvím svých funkcí ověřit, zda se na dané pozici (například při kliknutí myši) nachází jakýkoliv objekt, a případně na něj poskytnout referenci pro další manipulaci.

Z hlediska aplikací, které vyžadují nahrávání ikon nebo obrázků z externích zdrojů (což je i případ KML) a jejich zobrazení ve scéně, je zásadní objekt Billboard.

Kamera, která je také součástí scény, je definována svou pozicí a určuje viewing frustum, tedy jakýsi komolý kužel, který určuje aktuálně zobrazovanou plochu z celé virtuální scény.



Obrázek 5.3. View frustum, zelené objekty jsou viditelné, modré jsou skryty za glóbem a červené leží mimo frustum [14].

5.1.4 Widgets

Tato vrstva obaluje většinu widgetů v aplikaci, což jsou základní ovládací prvky pro interakci programu s uživatelem. Nejdůležitější je patrně widget Viewer, který v sobě sdružuje další elementy, jako je například časová osa (Timeline). Viewer tak tvoří určitý základní balíček, díky kterému uživatel vidí kompletní uživatelské rozhraní. Zároveň zpřístupňuje vývojáři potřebné kolekce objektů, protože jeho součástí je i virtuální scéna 5.1.3.

5.1.5 DataSources

Jedná se o rozhraní pro přístup aplikace k datům z externích zdrojů, což mohou být například soubory GeoJSON a TopoJSON (psány v JavaScriptové objektové notaci) nebo soubory zpracované dle Cesium jazyka CZML. Pro potřeby KML visualizeru byl implementován nový typ externího zdroje typu DataSource, který zpracovává jak soubory .kml, tak i .kmz.

5.2 Datový model

Navrhnout datový model pro dynamické objekty ve virtuální scéně frameworku Cesium by bylo velmi snadné v případě, že by nám postačilo zobrazit menší množství současně aktivních prvků. Pro tyto účely je dostupné high-level API nazývané jako Entity API. Cílem je propojit vizualizaci se všemi související atributy (jako je jméno, popisek) do jednotné datové struktury, které říkáme Entita. Vývojář se tak může soustředit pouze na způsob prezentace dat a je odstíněn od vizualizačních mechanismů. Vytvořit entitu a manipulovat s ní je rychlá a poměrně snadná záležitost, příkazy jsou pak vnitřní logikou delegovány na prvky low-level API, které v sobě entita sdružuje. Tyto označujeme jako *Primitive API*.

Rozdíl oproti Entity API je především v míře abstrakce poskytované pro požadované úkoly - jejich provedení je v Primitive blíže vývojáři, což na druhou stranu vyžaduje

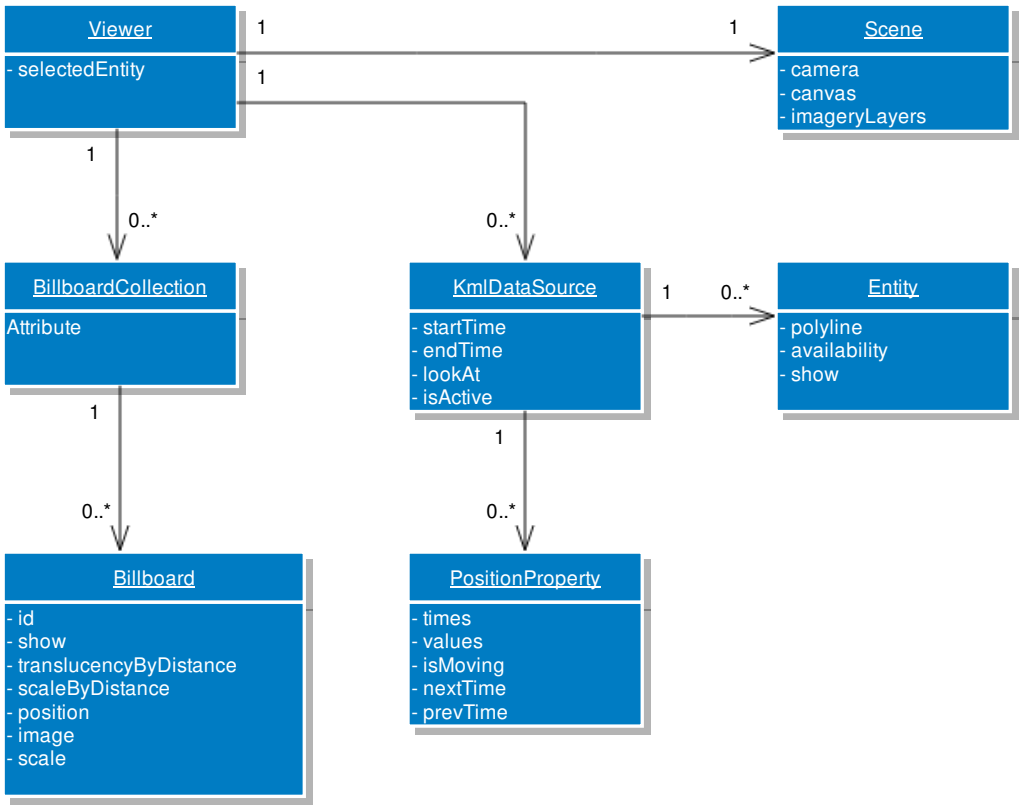
větší důraz na efektivní analytickou a návrhovou část. Jednotlivé druhy primitiv, tedy například Billboard, Model nebo Polygon, totiž mají rozdílný způsob konstrukce a jejich vizualizace se vyznačují specifickými rysy [15]. Každý typ navíc klade odlišné nároky na výpočetní prostředky a s každým je potřeba pracovat jiným způsobem pro dosažení optimálního výsledku (v případě KML Visualizeru tedy co největšího počtu zobrazených objektů).

Entity API je doporučováno pro vývoj aplikací, které pracují s širokou škálou grafických objektů, a zároveň nebalancují na hraně poskytovaného výkonu a paměti. Tento přístup má výhodu z hlediska budoucí rozšiřitelnosti, některé ovládací prvky (například camera) poskytují entitám lepší podporu. Použití Primitive API je stavění na naprostých základech frameworku a přinese výkonnější způsob řešení, ale pouze v případě, kdy jsme si jisti účelem jeho použití a kdy nám nevadí komplikovanější manipulace a transformace.

Datový model pro KML visualizer vychází z druhého přístupu, a to z toho důvodu, že hlavním cílem aplikace je dosažení co nejvyššího výkonu pro konkrétní podmnožinu jazyka KML, která se dá pro potřeby vizualizace převést na poměrně malý počet primitiv.

5.3 Popis datového modelu

V této sekci jsou popsány všechny využité třídy, které byly použity při implementaci KML visualizeru, a společně tak vytváří datový model.



Obrázek 5.4. Konceptuální datový model.

Obrázek zachycuje návrh datového modelu zvoleného pro tvorbu KML visualizeru. Při načtení KML souboru se vytvoří nová instance třídy KmlDataSource, která se stará o zpracování jednotlivých elementů.

Description a Style jsou převedeny do objektového zápisu a uloženy v poli jako veřejně přístupné atributy třídy KmlDataSource. Pro každý Placemark, který obsahuje gx:Track (tedy záznam o trajektorii) se vytvoří instance třídy PositionProperty. Ta v sobě uchovává dvojice hodnot souřadnice - čas, a její vnitřní logika bude zajišťovat interpolaci pohybu.

KmlDataSource si lze tedy představit jako unikátní soubor KML vyjádřený formou JavaScriptové třídy. Pro jednotlivé dynamické Placemark elementy se vytvoří zástupný objekt Billboard, jehož pozice bude čerpat zdrojová data z odpovídajícího objektu PositionProperty.

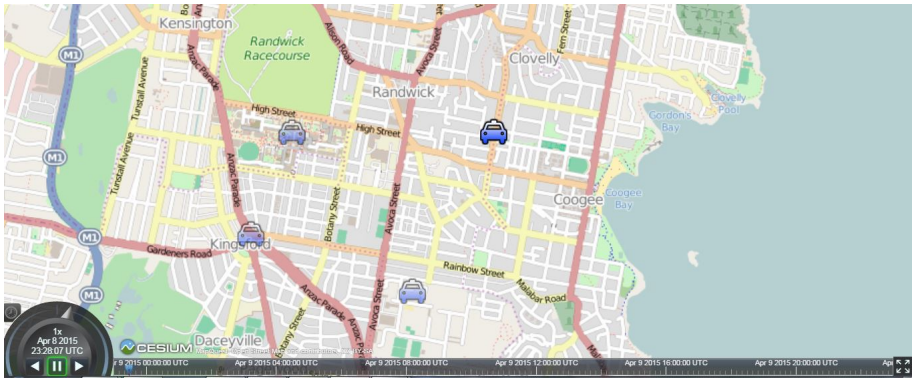
Viewer obsahuje přímé reference jak na všechny vytvořené KmlDataSource, tak i na jednotlivé BillboardCollection (prostřednictvím scény), což z widgetu Viewer dělá přirozené jádro aplikace.

5.3.1 PrimitiveCollection

Kolekce pro shromažďování primitiv, nejčastěji využívaná jako součást virtuální scény ve widgetu Viewer. Vzhledem k tomu, že kolekce jsou samy o sobě primitivem, lze je vkládat do sebe a vytvářet tak požadovanou hierarchii dat. Kvůli výkonu je vhodné vytvářet kolekce tak, aby v sobě každá sdružovala co možná nejvíc podobná primitiva.

5.3.2 Billboard

Tato třída vkládá do scény externí obrázek a přidává ho jako atribut do uceleného objektu, se kterým je možné dále pracovat. Tomuto objektu můžeme snadno nastavit pozici včetně výšky nad mapovým podkladem, dále například průhlednost nebo měřítko.



Obrázek 5.5. Ikona automobilu jako součást objektu Billboard.

5.3.3 BillboardCollection

Kolekce určená výhradně pro objekty třídy Billboard, zároveň může být přidána do PrimitiveCollection, tedy i do množiny objektů obsažených ve scéne. Stará se o aktualizaci objektů, což může být vyvoláno změnou polohy nebo atributu show. Objekty v jedné kolekci automaticky sdílejí texturu obrázků se stejným identifikátor, což značně optimalizuje jejich vizualizaci (je výhodné podle toho objekty Billboard třídit). Pozitivní vliv na výkon také přinese seskupení Billboard objektů podle jejich obnovovací frekvence (kolekce totiž při zavolání funkce update() obnoví všechny primitiva).

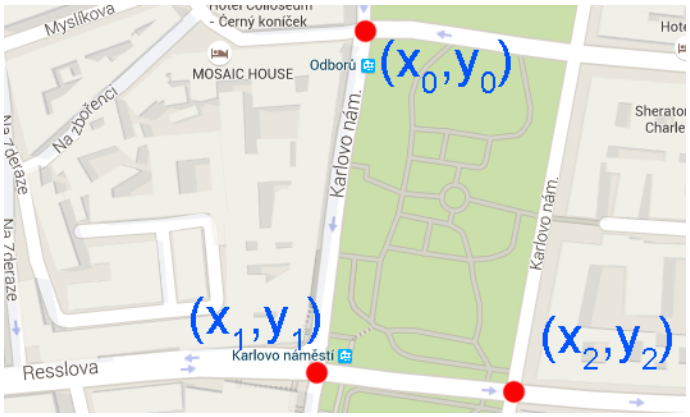
Kapitola 6
Implementace

6.1 Interpolace pohybu

Zaznamenat celou trajektorii pohybu například cestujícího po městě tak, abychom měli k dispozici dostatečně velké množství dat pro plynulou animaci by bylo natolik náročné, že by paměť prohlížeče byla schopná pojmout nejspíš jen zlomek požadovaného počtu dynamických objektů. Označíme-li každý bod takového pohybu (x_i, y_i), pak bychom potřeboval zaznamenat ideálně $i = 25$ hodnot za vteřinu (aby lidské oko vnímalo animaci jako spojitou).

Pokud bychom si představili město jako ohodnocený graf, jehož vrcholy by byly například křižovatky a hrany by představovaly jednotlivé ulice mezi nimi, pohyb cestujícího by nejspíš vedl většinu času přes po částech hladkých křivkách, velké změny směru by nastaly pouze na křižovatkách. Jde spíše o ideální model, reálné ulice většinou nejsou dokonalé rovné, nicméně i tak není pro členitější terén nutné zaznamenávat jednotlivě každou souřadnici.

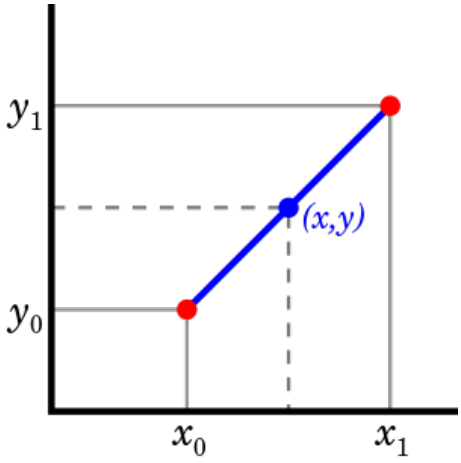
Z tohoto důvodu se do elementu gx:Track většinou zapíšou jen klíčové hodnoty (například souřadnice křižovatek), což vede ke značné úspoře paměti.



Obrázek 6.1. Body postačující k vyznačení trasy.

Tímto postupem získáme několik hraničních bodů, pro celou animaci však budeme potřebovat znát i hodnoty mezi nimi (stačí nám ale pouze hodnota pro aktuálně vykreslovaný snímek). Způsobu výpočtu, nebo spíše odhadu těchto hodnot se říká interpolace. Pokud pracujeme s nějakou reálnou funkcí, jejíž explicitní vyjádření by bylo příliš složité (což je případ naši „dopravní“ funkce), ale máme ji zadanou tabulkou hodnot, interpolace vrátí hodnotu pro argument, který v tabulce není [16].

Interpolačních algoritmů existuje celá řada, nejjednodušším je lineární interpolace. Pokud jsou dány dva známé body souřadnicemi (x_0, y_0) a (x_1, y_1), lineární interpolace je přímka mezi těmito dvěma body.



Obrázek 6.2. Lineární interpolace [17].

Následuje jednoduchý vzorec pro výpočet interpolované hodnoty. Nezávislá proměnná je x, v našem případě čas, pro který chceme vědět odpovídající polohu.

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

Výsledná hodnota y obsahuje souřadnice v požadovaném čase.

6.2 Parsování XML

Parsování je proces, při kterém analyzujeme řetězec znaků a porovnáváme jeho strukturu vůči předem dané formální gramatice. Proto se také tento proces někdy nazývá syntaktická analýza. Programu, který tuto úlohu vykonává, se říká parser. Při parsování vlastně kontrolujeme, zda daná posloupnost symbolů odpovídá definovaným pravidlům, podle kterých lze složit výraz, a pořadí, ve kterém se musí v takovém výrazu objevit. Například tedy v XML dokumentu bude parser předpokládat, že za symbolem „<“ bude následovat název elementu, a že stejný element musí být ukončen párovou značkou „</..>“. Umožní nám to „rozbít“ soubor na jednotlivé elementy daného jazyka a získat tak v něm zakódované informace, které můžeme převést do námi požadovaného zápisu (např. JavaScriptových objektů).

6.2.1 DOM parser

V našem Visualizeru potřebujeme zpracovávat soubory KML, obecně tedy jazyk XML. Vzhledem k tomu, o jak rozšířený formát pro výměnu dat se jedná, není překvapením velké množství dostupných parserů pro tento jazyk. Většina moderních prohlížečů má dokonce jeden parser přímo zabudovaný v sobě, konkrétně jde o tzv. DOM parser. Ten využívá stromové struktury elementů XML, a převádí je do objektů v rámci Document Object Modelu dané stránky. Výhodou je rychlost, se kterou v takto převedeném dokumentu můžeme procházet jednotlivé uzly, neboť každý obsahuje seznam svých potomků (vnořených uzlů) a rodičů. Programátor má navíc jednoduše k dispozici všechen textový obsah jednotlivých elementů, jejich atributy, komentáře atd, protože ty jsou také součástí vytvořených objektů.

```
var parser = new DOMParser();
var kml = parser.parseFromString(text, 'application/xml');
```

V ukázce je nejprve vytvořena instance parseru, poté parser začne zpracovávat zdrojový soubor KML (text.kml) a uloží referenci na vytvořenou strukturu do proměnné kml. Z té pak můžeme snadno procházet celý dokument.

S tímto přístupem však přichází největší nevýhoda tohoto parseru; paměťové nároky. Pro vytvoření DOM reprezentace XML dokumentu je nejdříve nutné celý soubor načíst a nechat rozparsovat, do té doby není možné s daty jakkoliv pracovat. Aby bylo možné k jednotlivým uzlům libovolně přistupovat, musí zůstat celá reprezentace dat v paměti, což vytváří velké omezení pro použití na velkých souborech.

6.2.2 SAX parser

Předchozí část popisovala tvorbu stromové reprezentace dokumentu, existuje však alternativní způsob parsování souborů XML, který se nazývá Simple API for XML, zkráceně označovaném jako SAX. Jde o tzv. událostmi řízené rozhraní, kdy parser postupně čte zdrojový text, a pokud narazí na definovanou položku, což může být například začátek nebo konec elementu, vyvolá událost, kterou musí obsloužit funkce v naší aplikaci [18]. Parser předá funkci všechny důležité parametry, jako je název elementu, seznam atributů apod. Událostmi řízené zpracování XML dokumentu má dvě velké výhody – je rychlé (soubor je zpracováván lineárně) a má malé paměťové nároky, respektive je pouze na vůli programátora, která konkrétní data uloží. Nevýhodou je logicky nutnost provést s daty vše potřebné při jednom sekvenčním průchodu, možnost vrátit se zpět k nějakému elementu již není.

Z důvodů zmíněných výše jasně vyplývá, že DOM parser je vhodný pro aplikace, při kterých se opakovaně a náhodně vracíme k elementům relativně malého souboru KML, zatímco SAX parser vyžaduje větší pozornost ze strany vývojáře, na oplátku ale umožní načíst i velké soubory, *což vedlo k jeho použití v KML visualizeru.*

Nyní se podíváme, jak konkrétně probíhá zpracování KML souboru v naší aplikaci. Uživatel má v zásadě dvě možnosti, jak takový soubor z disku načíst. První je prostřednictvím grafického uživatelského rozhraní (GUI), konkrétně stisknutím tlačítka „Load KML“. To vyvolá standardní HTML vstupní formulář pro soubor. Druhá možnost je tzv. „drag and drop“, což je postup, při kterém uživatel „přetáhne“ soubor ze složky přímo do okna prohlížeče. Oba způsoby vedou ke stejnému cíli - předání souboru třídě LoadFileMixin, která slouží jako vstupní rozhraní aplikace. Tato třída vytvoří novou instanci SAX parseru a předá mu náš zdrojový soubor k procesu parsování, zároveň se stará o to, aby byl uživatel informován o průběhu zpracování.

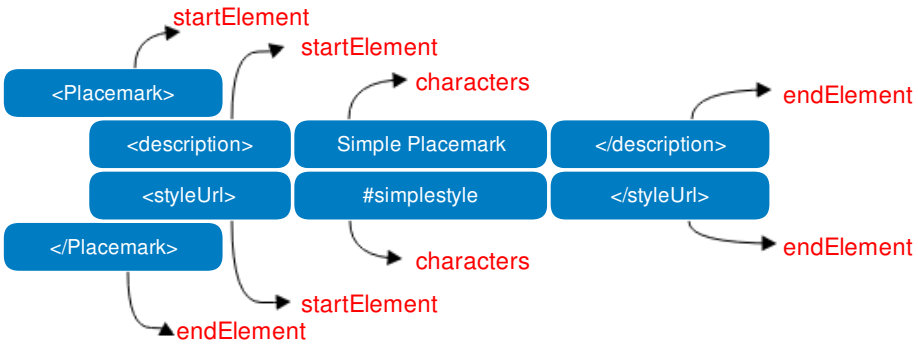
Jako SAX parser byl použit externí, open-source JavaScriptový jsXmlSaxParser[1]). Ten úzce spolupracuje s třídou KmlDataSource, která je určena k transformaci geodat do podoby JavaScriptových objektů, z 1nichž jsou následně tvořeny primitiva Billboard atd.

Následující kod zachycuje definici funkce, jejíž volání je řízeno událostí zaznamenanou parserem, v tomto konkrétním případě začátkem elementu XML.

```
function startElement (namespaceURI, localName, qName, atts) {
   // example of how to get the value of an attribute

   var id = atts.getValue("id");
   KmlDataSource.startElement(localName,id);
}
```

Funkci je předán název začínajícího elementu, hodnota jeho atributů (např. id) a další informace. Dále je zavolána funkce startElement() uvnitř třídy KmlDataSource, která

[1]) `https://github.com/ndebeiss/jsxmlsaxparser`



Obrázek 6.3. Řízení událostí při SAX parsování.

se postará o nastavení aktuálně zpracovávaného elementu, aby bylo možné další příchozí data správně zařadit.

Při zavolání funkce endElement je předáno zpracování celého elementu funkci handleEndElement ve třídě KmlDataSource.

```
function handleEndElement(dataSource,element) {
    if (nodes.length >0 && nodes[nodes.length-1].name == element){
        var elementProcessor = supportedTypes[element];
        if (defined(elementProcessor)) {
            elementProcessor(dataSource);
            return;
         }
    }
}
```

Funkce ověří, jestli je právě ukončený element na seznamu podporovaných (supportedTypes), a jestliže ano, předá jeho zpracování příslušné funkci. KML visualizer v současné době podporuje elementy:

- LookAt - nastaví pohled kamery na určené souřadnice
- Style - specifikace grafických prvků užitých při vizualizaci daného objektu.

  a) IconStyle - např. měřítko, odkaz na použitou ikonu.
  b) LineStyle - např. barva, šířka čáry.

- Placemark - viz 2.3.1.

  a) description - popis objektu
  b) gx:Track - trajektorie pohybu
  c) LineString - souřadnice vrcholů lomené čáry
  d) styleUrl - reference na použitý styl
  e) TimeSpan - časový interval pro zobrazení elementu

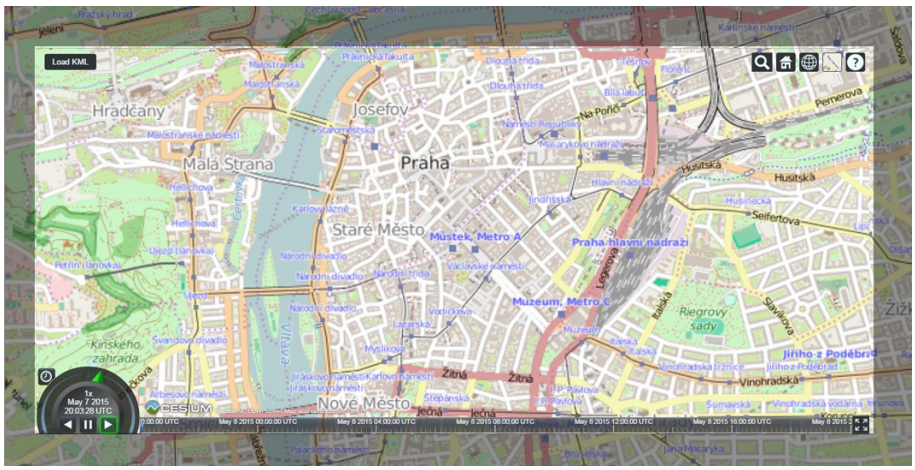
6.3 Animování pohybu

Při vytváření dynamických animací ve frameworku Cesium máme dvě možnosti, jak požadovaný objekt rozpohybovat. Zaprvé je možné vytvořit Enititu 5.2, která díky

svému high-level API sama zajistí volání všech potřebných vizualizačních mechanismů v době, kdy se má objekt podle zadaných hodnot pohybovat. Zásadní nevýhodou tohoto principu je stejná obnovovací frekvence, se kterou se virtuální scéna snaží všechny Entity vykreslovat na plátno. Zajistí nám to sice jejich plynulé animování, nicméně se vzrůstajícím počtem těchto Entit (řádově jednotky tisíc) už není aplikace schopná obsloužit všechny se stejnou rychlostí, a nejenže se tím pádem sníží obnovovací frekvence (což vyvolá pocit trhané animace), ale navíc se výpočtem zahltí vlákno a aplikace tzv. zamrzne.

Druhou možností je vytvořit Billboard 5.3.2. Ten je sice sám o sobě statický, můžeme mu však měnit pozici dle našich potřeb. Každá taková změna je spouštěčem pro BillboardCollection, která zajistí znovuvykreslení celé kolekce (proto je výhodné měnit pozici všem Billboardům v kolekci najednou). Vzniká tím možnost regulovat obnovovací frekvenci, v případě nutnosti ji některým snížit na úkor dalších. Zásadní myšlenka, která vedla k tomu, že KML visualizer je schopný animovat řádově až stovky tisíc Billboardů, spočívá v rozdělení všech objektů na dvě skupiny. První z nich definuje prioritní množinu Billboardů, pro kterou je frekvence vyšší (animace je plynulejší) a druhá obsahuje zbylé Billboardy, kterým je naopak frekvence snížena.

Aby uživatel na první pohled tuto skutečnost nepoznal, je prioritní množina vymezena oblastí, ve které se vyskytují aktuálně viditelné Billboardy (tedy ty, které jsou vykresleny na plátno). Aplikace pak postupně prochází všechny Billboardy, a ty, jejichž pozice odpovídá vymezené oblasti, zařadí do „rychlejší“ skupiny. Pokud je pozice už mimo oblast, přesune se Billboard do „pomalejší“ skupiny. Ve skutečosti je ohraničená oblast o trochu větší, než vymezuje pohled kamery, a to z toho důvodu, aby se co nejvíc eliminovala doba, která uplyne než aplikace přesune Billboard z jedné skupiny do druhé.



Obrázek 6.4. Oblast vymezující prioritní množinu, včetně šedé části, kterou uživatel nevidí.

V aplikaci jsou implementovány dvě obsluhující funkce - draw() pro pomalejší množinu, drawFast() pro rychlejší. Účelem každé je v nastaveném intervalu kontrolovat jednu konkrétní kolekci BillboardCollection (v každém průběhu funkce jinou).

```
for (var i = 0; i < collection.length; i++) {
    var billboard = collection.get(i);

    var dataSource = collection.dataSource;
```

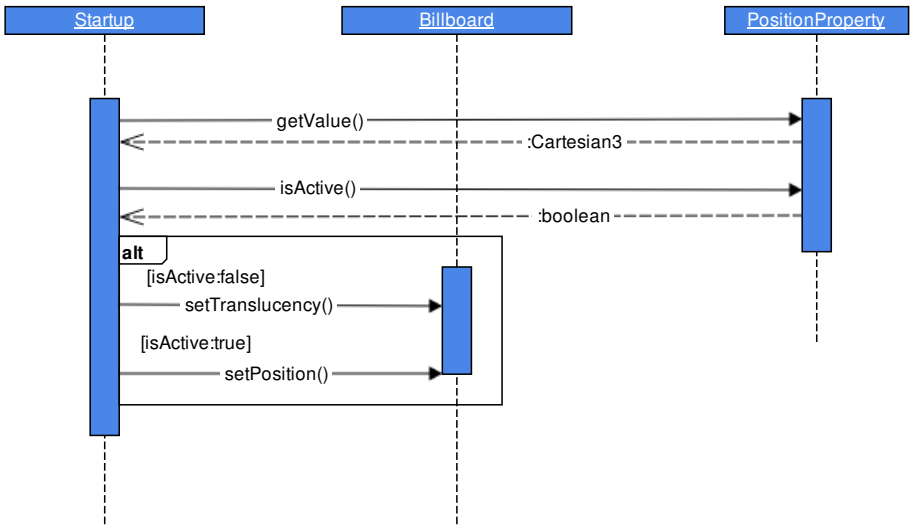
```
    var positionProperty = dataSource.positions[billboard.index];
    var position = positionProperty.getValue(viewer.clock.currentTime);
```

Každému Billboardu přísluší jedna PositionProperty, která nám díky interpolačnímu algoritmu vrátí pozici pro aktuální čas získaný z hodin widgetu Viewer. Jednotlivé BillboardCollection obsahují referenci na KmlDataSource, kde najdeme uložené souřadnice ze souboru KML.

Následně se ověří, zda-li pozice Billboardu odpovídá vymezené oblasti. Souřadnice získáme převodem prohlížeči známých rozměrů elementu canvas (výška a šířka našeho plátna) na povrch Elipsoidu.

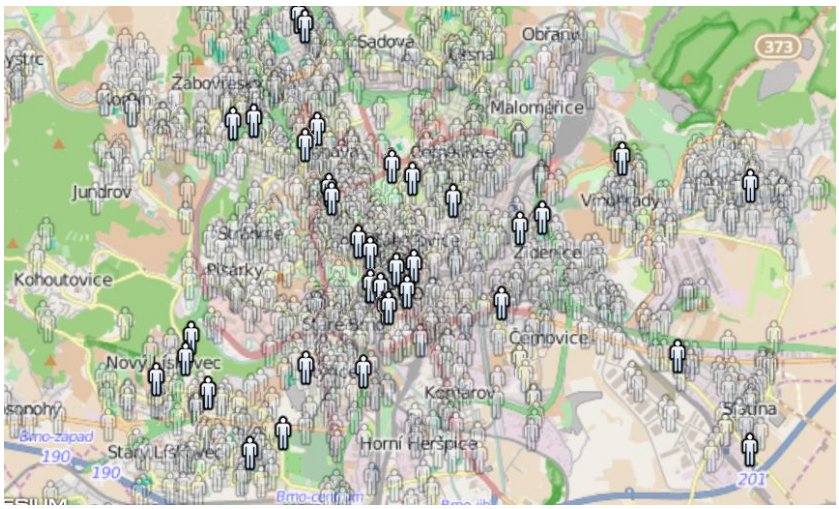
```
var camera = viewer.camera;
var topLeft = camera.pickEllipsoid(Cesium.Cartesian2(0,0));

var width = viewer.canvas.clientWidth;
var height = viewer.canvas.clientHeight;

var lowerRight = camera.pickEllipsoid(Cesium.Cartesian2(width,height));
```

Pokud je Billboard v tomto výhledu, přesune jej funkce do prioritní množiny a v příštím běhu jej přeskočí (obslouží ho druhá funkce). Aby obě funkce neověřovaly v každém průchodu pozici Billboardů, u kterých víme, že budou dlouho dobu neaktivní (nezmění svou pozici), byl do PositionProperty přidán atribut „isActive“ typu boolean, který je nastaven na hodnotu false, pokud se objekt v danou chvíli nehýbe. Když tato situace nastane, využijeme interpolačního algoritmu k tomu, abychom zjistili další čas, kdy objekt začne být aktivní. To je velmi jednoduché, neboť víme, že pokud se objekt nehýbe, intrpolované hodnoty budou pro dva sousední časy totožné. V ten moment použijeme jako náš aktivační čas krajní hodnotu z intervalu, ve kterém jsme inteprolovali. Musíme sice provést interpolaci pro dva parametry, umožní nám to ale dočasně vyřadit daný objekt z cyklu funkce, a vrátit jej do něj až v potřebný moment.



Obrázek 6.5. Diagram procesu změny polohy.

To vede k další výpočetní úspoře, protože interpolace polohy je poměrně náročná úloha (především z důvodu hledání správného intervalu mezi zadanými hodnotami). Těmto Billboardům dále nastavíme průhlednost (atribut translucencyByDistance), aby byly aktivní objekty zvýrazněny.



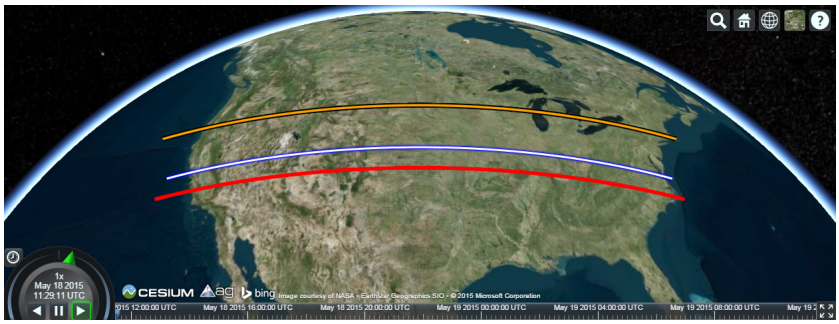
Obrázek 6.6. Neaktivní Billboardy s nastavenou průhledností, aktivní jsou zvýrazněny.

6.4 Vizualizace LineString

Předchozí sekce popisovala princip animování dynamických elementů Placemark, které obsahovaly trajektorii v podobě gx:Track. KML visualizer ale také podporuje zobrazení LineString elementů, což je lomená čára, jejíž pozice se v čase nemění, je pouze omezena časovým intervalem TimeSpan.

Pokud při parsování narazí třída KmlDataSource na element LineString, vytvoří pro něj novou Entitu a přidá ji atribut Polyline. Ten je podobně jako Billboard primitivem, u kterého můžeme nastavit atributy show, position (v tomto případě však všechny souřadnice najednou) a také material, což představuje například barvu nebo šířku čáry.

Následující obrázek zachycuje ukázku použití atributu material pro tři jednoduché Polyline, jejichž pozice jsou určené pouze dvěma souřadnicemi.



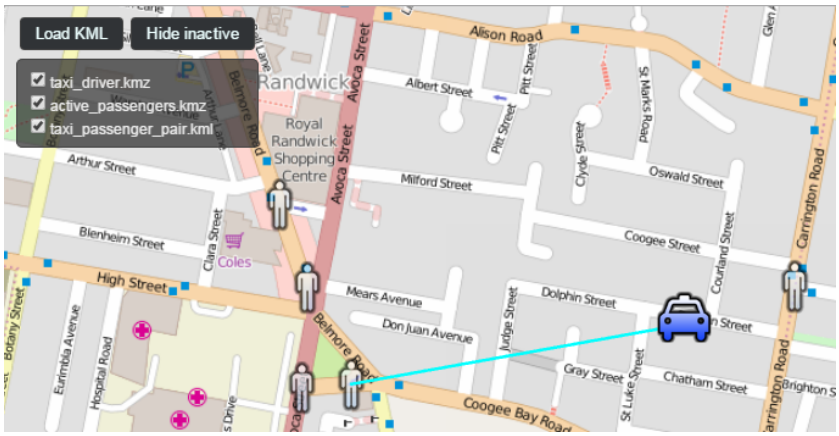
Obrázek 6.7. Ukázka několika objektů Polyline, na které je převeden KML element LineString.

U elementu gx:Track nebylo vhodné použít Entity API, pro statické objekty (což je případ LineString) však jeho nedostatky částečně odpadají. Zaprvé pozice vrcholů je

typicky zadána mnohem menším počtem souřadnic než je tomu u gx:Track. Zadruhé se pozice nemění v čase, není tedy nutné její hodnoty interpolovat. Jediný atribut, který u Polyline hlídáme, je časový interval určující viditelnost daného objektu - o to se ale dobře postará právě Entity API.

6.5 KML vrstvy

Při práci s geodaty se nám může stát, že máme informace rozděleny ve více souborech, které spolu souvisí a měly by tak být vizualizovány společně. Jako příklad můžeme uvést dopravní systém, kde budeme chtít oddělit do jednotlivých vrstev všechny osobní automobily, cyklisty, vozy MHD či taxislužeb. Načítání souborů do vrstev nám umožní v případě potřeby skrýt nebo naopak přidat jednu z těchto částí. KML visualizer, podobně jako Google Earth, přidá každý načtený KML či KMZ soubor do checkbox menu umístěném v levé horní části grafického rozhraní. Uživatel pak zaškrtnutím jednotlivých názvů souborů může zvolit ty, které bude aplikace zobrazovat.



Obrázek 6.8. Několik souborů načtených do vrstev, každá obsahuje jinou kategorii geodat.

Pro každý checkbox je definován poslouchač událostí (EventListener), který se zavolá při kliknutí na název souboru. EventListener pak předá řízení funkci hideElements spolu s referencí na daný KmlDataSource, abychom mohli nastavit atribut show na hodnotu false jen u Billboardů z daného KML souboru. V případě, že součástí soubouru byly i elementy LineString (převedené do datové struktury Entit), stačí nastavit atribut show pro celou kolekci (v případě BillboardCollection musíme jednotlivým objektům nastavit tento atribut individuálně).

6.6 Grafické uživatelské rozhraní

Základními ovládacími prvky KML visualizeru jsou widgety Timeline a Animation. Oba zobrazují aktuální čas naši virtuální scény a umožnují uživateli s ním manipulovat. Změny času v naší aplikaci ve skutečnosti probíhají ve vrstvě Core, konkrétně ve třídě nazvané příznačně Clock. Timeline i Animation tak slouží jako grafické rozhraní této třídy a všechny uživatelovy příkazy předávájí právě jí.

Obrázek 6.9. Timeline s posuvníkem.

6.6.1 Timeline

Časová osa je z hlediska uživatelské orientace v dané simulaci nejdůležitější částí aplikace. Pomocí modrého posuvníku se může pohybovat dopředu i dozadu a libovolně si tak nastavit konkrétní čas, podle kterého se bude aplikace řídit.

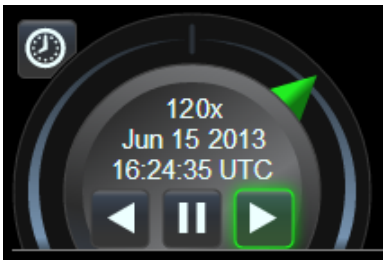
Timeline nabízí funkci zoomTo(), která ohraničí zobrazený časový interval na zadané hodnoty. KML visualizer této funkce využívá při načtení souboru dynamického KML, neboť v průběhu parsování zaznamená počáteční a konečný čas simulace:

```
function setTimeline(dataSource) {
    var timeline = viewer.timeline;
    timeline.zoomTo(dataSource.startTime, dataSource.stopTime);
    viewer.clock.clockRange = Cesium.ClockRange.LOOP_STOP;
}
```

Poslední řádek v předchozí ukázce uzavře časovou osu do smyčky, to znamená, že se ukazatel času při dosažení krajní hodnoty vrátí znovu na začátek animace.

6.6.2 Animation

Widget Animation poskytuje tlačítka play, pause a reverse pro ovládání směru animace. Kromě toho dovoluje nastavit také rychlost animace, a to pomocí tzv. „shuttle ring“, což je ukazatel v podobě zelené šipky. Tento koncept je podobný otáčivému ovladači pro zrychlování a zpomalování např. hudby nebo videa (tzv. „jog wheel“).



Obrázek 6.10. Animation widget, zelený ukazatel slouží k nastavení rychlosti animace.

6.6.3 Další ovládácí prvky

Do uživatelského rozhraní KML visualizeru byly přidány dvě tlačítka, z nichž jedno slouží k načítání souborů a druhé k zobrazení/skrytí neaktivních prvků.Toto tlačítko bylo zařazeno z důvodu zlepšení přehlednosti animace pro velký počet Placemarků, kdy průhledné obrázky částečně zhoršovaly orientaci. Obě tlačítka jsou zarovnána v levém horním rohu, do kterého je umístěn i seznam načtených souborů KML. Všechny tyto prvky společně tvoří základní ovládácí panel aplikace, který je vidět např. v obrázku 6.8.

Jak již bylo zmíněno, aplikace podporuje zpracování elementu Description, který se v Google Earth zobrazí při kliknutí na daný objekt. KML visualizer tuto funkcionalitu nabízí také, při kliknutí na jakýkoliv Billboard či Polyline se tento objekt zvýrazní zeleným ukazatelem a informace z Description se objeví v pravém horním rohu v samostatném rámečku. V tomto rámečku také uživatel může kliknout na ikonku kamery, což nastaví její sledování daného objektu.

Obrázek 6.11. Ukázka zobrazení elementu Description.

Kapitola 7
Zhodnocení aplikace

Testování výkonu aplikace probíhalo v prohlížečích Google Chrome (verze 42.0) a Mozilla Firefox (verze 37.0), pro hardwarovou akceleraci byla použita grafická karta NVIDIA GeForce GT 555M. K zátěžovým testům posloužily následující soubory:

- population10k.kml - soubor o velikosti cca 15 MB obsahuje 10 tisíc dynamických Placemark elementů, jejichž trajektorie byla v průměru zaznamenána 15 body.
- population100k.kml - soubor o velikosti cca 150 MB, obsahuje 100 tisíc dynamických elementů, složitost trajektorie je srovnatelná s předchozím souborem.
- population3k.kml - soubor o velikosti cca 20 MB, obsahující cca 3 tisíce dynamických elementů, trajektorie ale byla zaznamenána v průměru skoro 100 body.

Pro porovnání byly vybrány celkem tři aplikace - Google Earth jakožto primární nástroj pro práci s KML, dále virtuální glóbus Cesium, který od verze 1.7 nabízí základní podporu zpracování KML, a nakonec KML visualizer. První dvě zmíněné aplikace jsou dle zjištění autora této práce jediné, které v současné době umožnují vizualizaci dynamického KML. Zajímavé bude porovnání samotného frameworku Cesium a aplikace KML visualizer, která je na něm založena.

Aplikace	rychlost načtení [s]	plynulá animace	RAM [MB]
Google Earth	5	ano	370
Cesium	20	ne	1500
KML visualizer	10	ano	584

Tabulka 7.1. Soubor population10k.kml.

Z tabulky je zřejmé, že použití Entity API (viz 5.2) ve frameworku Cesium vede při velkém počtu objektů k obrovským pamětovým nárokům, které mohou vést k ukončení procesu celého prohlížeče.

Aplikace	rychlost načtení [s]	plynulá animace	RAM [MB]
Google Earth	90	ne	960
Cesium	nenačte	-	-
KML visualizer	120	ano	1350

Tabulka 7.2. Soubor population100k.kml.

S takto velkým souborem už měl Google Earth problémy, a pohyb jednotlivých objektů byl velmi trhaný. Ovládání programu mělo značnou odezvu. KML visualizer soubor déle načítá, a chvíli mu i trvá než mezi 100 tisíci objekty určí prioritní množinu. Poté ale běží animace plynule a bez problémů.

Aplikace	rychlost načtení [s]	plynulá animace	RAM [MB]
Google Earth	37	ano	340
Cesium	8	ano	670
KML visualizer	11	ano	538

Tabulka 7.3. Soubor population3k.kml.

Z tabulky vyplývá zajímavé zjištění, že Cesium bez problému zvládne animovat řádově jednotky tisíc dynamických objektů, ale s podobně velkým souborem population10k, který má 10 tisíc takových objektů už má nepřekonatelné problémy. Toto potvrzuje správnost volby Primitive API jako základu pro KML visualizer. Znatelné jsou také rozdíly ve využití paměti RAM prohlížečem, kde datový model KML visualizeru umožnil bezproblémové načtení desetkrát většího souboru, než zvládl ten použitý v Cesium.

Ke snížení nároků také přispělo použití SAX parseru, kde rozdíl oproti DOM parseru při načítání souboru population100k.kml činil až 300 MB paměti.

Kapitola 8
Závěr

Cílem práce bylo vytvořit nástroj, který by umožnoval vizualizaci velkého počtu dynamických Placemark elementů. Většina dostupných služeb je zaměřena spíše na podporu široké škály elementů KML, což na jednu stranu vytváří nepřeberné množství možností pro uplatnění jazyka KML, na druhou neumožňuje například simulace velkých dopravních systémů s reálným počtem objektů, který se může vyšplhat až k několika statisícům. Zpracování velkých souborů, ať již XML nebo KML, bude proto v budoucnosti určitě velmi poptáváné, a dá se očekávat rozvoj tomu odpovídajících služeb. Jednou z takových by se mohl stát i KML visualizer, ke kterému v současné době (pokud je autorovi dobře známo) neexistuje podobná opensource alternativa.

Ambicí nebylo vytvořit konkurenci k masově rozšířenému virtuálnímu glóbu Google Earth, který bude díky perfektní podpoře vývojářů nejspíš ještě několik let udávat trend v oblasti GIS a mapových aplikací obecně. Motivaci poskytla myšlenka přijít s aplikací cílenou na konkrétní podmnožinu jazyka KML a v té poskytnout uživatelům snadné a zároveň efektivní řešení. Samotné implementaci předcházelo analyzování dostupných technologií, z nichž se nakonec ukázal být nejlepší volbou framework Cesium, na jehož základech je KML visualizer vytvořen.

Hlavní přínos práce vidíme v navrhnutém a úspěšně implementovaném vlastním způsobu animování dynamických objektů v JavaScriptu. Tento způsob se snaží o minimalizaci pamětových nároků, což se podařilo především díky vhodně zvolenému datovému modelu (Primitive API) a také způsobu parsování KML souboru (SAX). Další důležitou částí představeného způsobu je princip dynamické změny obnovovací frekvence, který zajistí plynulost animace i pro znatelně větší počet objektů, než jsou schopné zobrazit podobné nástroje. Při vývoji KML visualizeru se ukázaly silné stránky JavaScriptových aplikací (použití WebGL, přenesení zátěže ze serveru na klienta), stejně jako ty slabé (jednovláknové zpracování). Co se budoucnosti týče, vytvořené jádro aplikace nabízí možnost rozšíření podpory o další KML elementy. Pokud by se objevil nový optimalizovaný JavaScriptový SAX parser, bylo by vhodné jím nahradit stávající jssaxparser, který vykazoval nepatrně horší rychlost při načítání souborů, což prokázalo i závěrečné porovnání s Google Earth.

Věříme, že práce splnila cíl, který byl vytyčen v úvodu tohoto dokumentu.

Literatura

[1] Introduction to GIS. [online]. [cit. 2015-05-10]. Dostupné z: https://geodata.ethz.ch/resources/tutorials/L1IntroToGeodata/en/html/unit_u1Intro.html

[2] Geoportal Praha. [online]. [cit. 2015-05-10]. Dostupné z: http://www.geoportalpraha.cz/cs/clanek/11/co-je-gis

[3] MATSUDA, Kouchi a Rodger LEA. *WebGL programming guide: interactive 3D graphics programming with WebGL*. xxv, 516 pages. Always learning. ISBN 03-219-0292-0.

[4] HARRINGTON, Surya Michael. *Google earth forensics: using google earth geolocation in digital forensic investigations*. 1st edition. pages cm. ISBN 978-012-8002-162.

[5] The difference between KML and GML. [online]. [cit. 2015-05-11]. Dostupné z: http://www10.giscafe.com/blogs/gissusan/2009/07/30/the-difference-between-kml-and-gml

[6] MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. *XML technologie: principy a aplikace v praxi*. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 978-80-247-2725-7.

[7] XML DTD. [online]. [cit. 2015-05-18]. Dostupné z: http://www.w3schools.com/xml/xml_dtd.asp

[8] ROEBUCK, Kevin. *Service Provisioning Markup Language (SPML): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Publishing, 2012, 748 s. ISBN 978-1-74304-862-7.

[9] ARIEL, Avraham a Nora Ariel BERGER. *Plotting the globe: stories of meridians, parallels, and the international date line*. Westport, Conn.: Praeger Publishers, 2006, xii, 235 p. ISBN 02-759-8895-3. RC Press, c2010, xxix, 277 p. ISBN 14-200-8799-1.

[10] WERNECKE, Josie. *The KML handbook: geographic visualization for the Web*. Upper Saddle River, NJ: Addison-Wesley, c2009, xix, 339 p. ISBN 978-032-1525-598.

[11] FULTON, Steve a Jeff FULTON. *HTML5 canvas*. Second edition. xx, 726 pages. ISBN 14-493-3498-9.

[12] Canvas Cycle: True 8-bit Color Cycling with HTML5. [online]. [cit. 2015-05-08]. Dostupné z: http://www.effectgames.com/demos/canvascycle/

[13] DANCHILLA, Brian. *Beginning WebGL for HTML5*. New York: Distributed to the book trade worldwide by Springer Science Business Media, c2012, xxiii, 329 p. Expert's voice in Web development. ISBN 14-302-3996-4.

[14] *Horizon Culling* [online]. [cit. 2015-05-08]. Dostupné z: https://cesiumjs.org/2013/04/25/Horizon-culling/

[15] Visualizing Spatial Data. [online]. [cit. 2015-05-11]. Dostupné z: http://cesiumjs.org/2015/02/02/Visualizing-Spatial-Data/

[16] DOŠLÁ, Zuzana a Petr LIŠKA. *Matematika pro nematematické obory: s aplikacemi v přírodních a technických vědách*. 1. vyd. Praha: Grada, 2014, 304 s. Expert (Grada). ISBN 978-80-247-5322-5.

[17] Lineární interpolace. In: *Wikipedie* [online]. [cit. 2015-05-18]. Dostupné z: http://cs.wikipedia.org/wiki/Line%C3%A1rn%C3%AD_interpolace

[18] NOLAN, Deborah Ann a Duncan Temple LANG. *XML and Web technologies for data sciences with R*. xxiv, 663 pages. Use R!. ISBN 978-146-1478-997.

Příloha A
Seznam použitých zkratek

API	Application Programming Interface
CZML	Cesium Markup Language
DOM	Document Object Model
GIS	Geografický Informační Systém
GML	Geography Markup Language
HTML	HyperText Markup Language
KML	Keyhole Markup Language
RAM	Random-Access Memory
SAX	Simple API for XML
WGS84	World Geodetic System 1984
XML	Extensible Markup Language

Příloha B
Obsah přiloženého CD

- /text
  - pdf/ - obsahuje elektronickou verzi této práce ve formátu PDF.
  - tex/ - obsahuje plain TEX zdrojové soubory práce, která byla vytvořena pomocí šablony CTUstyle od RNDr. Petra Olšáka.
- /KMLvisualizer - obsahuje zdrojové soubory webové aplikace.
- /KMLexamples - obsahuje vzorové KML soubory, na kterých byla aplikace testována.
- readme.txt