Examensarbete i geografisk informationsteknik nr 28

Visualisering av stadsmodeller på webben

- Jämförande studie mellan CityGML och CityJSON

Alfred Hildingson & Patrik Sylve

Civilingenjörsutbildningen i Lantmäteri
Lunds Tekniska Högskola

Institutionen för Naturgeografi och Ekosystemvetenskap
Lunds Universitet

LUNDS UNIVERSITET
Lunds Tekniska Högskola

Visualisering av stadsmodeller på webben

-Jämförande studie mellan CityGML och CityJSON

EXTM05 Masteruppsats, 30 hp
Civilingenjörsutbildningen i Lantmäteri

Alfred Hildingson & Patrik Sylve

Handledare:
Lars Harrie
Institutionen för Naturgeografi och Ekosystemvetenskap

April 16, 2020

Opponent: Gustav Forsberg
Examinator: Micael Runnström

Institutionen för Naturgeografi och Ekosystemvetenskaper
Lunds Universitet
Sölvegatan 12
223 62 Lund

Telefon: 046-222 30 30
Fax: 046-222 03 21
Hemsida: http://www.nateko.lu.se

Examensarbete i geografisk informationsteknik nr 28
Tryckt av E-tryck, E-huset, 2020

Abstract

As the technological developments and the access to 3D geographical data is increasing the usage of city models in planning is becoming more important. City models have traditionally mainly been used for visualization but are now an important tool for analysis. The increased application of city models set higher requirements on standardized formats to effectively exchange data.

City models are commonly stored in the information model CityGML. However, CityGML has been described as problematic to work with, especially on the web. The criticism is mainly due to the XML-coding being hard to work with, its verbose and hierarchic structure and the large file size.

The purpose of this report is to examine if CityJSON is an appropriate alternative to the XML-implementation of CityGML 2.0 for visualization of geometries and attribute data on the web. In this study, two strategies for visualization are compared. The evaluation is thus being done indirectly by comparing the two strategies.

In the first strategy an existing tool for visualizing CityJSON is being further developed. City models are in this strategy being visualized directly in the web browser in CityJSON format. In the second strategy an application for CityGML is developed using the platform Cesium. In this strategy CityGML is converted to 3D-tiles. The two strategies are then evaluated from a set of requirements.

The study shows that the CityJSON-format has multiple advantages compared to using CityGML and the Cesium platform on the web. CityJSON data in this study is up to ten times more compact compared to the corresponding file in CityGML-format. This results in faster transfer and management on the web. The study also shows that Cesium has problem with handling attribute data and is more suitable for pure visualizations. This is due to information being lost in the converting to 3D-tiles and that the data is saved remotely and being streamed to the client.

In conclusion, this study shows that CityJSON is a promising format and have potential to ease the development of applications for city models on the web.

Sammanfattning

I takt med den tekniska utvecklingen och större tillgång till 3D geografiska data har tillämpningen av stadsmodeller i samhällsplaneringsprocessen ökat. Stadsmodeller som traditionellt sett använts för visualisering är nu även ett viktigt verktyg för analys. Den ökade tillämpningen av stadsmodeller ställer högre krav på enhetliga standarder och dataformat för att effektivt kunna utbyta och arbeta med data.

Stadsmodeller lagras idag vanligen i informationsmodellen och implementeringen CityGML. CityGML beskrivs som problematisk att hantera och arbeta med, framförallt på webben. Kritik som riktats mot formatet är bland annat svårigheter med att arbeta i XML-kodningen, formatets hierarkiska uppbyggnad samt stora filformat.

Som ett alternativ har CityJSON utvecklats. Formatet lanserades 2019 och är en implementering av CityGML 2.0 i JSON-format. Den nya implementeringen CityJSON utvecklades då man såg ett behov av att arbeta med JSON som är det vanligaste formatet för utbyte av data på webben.

Rapporten har som syfte att undersöka om CityJSON är ett lämpligt alternativ till XML - implementationen av CityGML 2.0 för visualisering av geometri- och attributdata. Undersökningen sker genom en jämförelsestudie mellan två visualiseringsstrategier. Utvärdering av formaten sker således indirekt genom jämförelse mellan de två strategierna.

I den första strategin vidareutvecklas en befintlig visualiseringstjänst för CityJSON. Stadsmodeller visualiseras då direkt i webbläsaren i CityJSON-format. I den andra strategin utvecklas en tjänst för CityGML genom plattformen Cesium. I denna strategi konverteras CityGML till 3D-tiles. De två metoderna utvärderas sedan utifrån en framtagen kravställning.

Fallstudien visar att CityJSON har ett flertal fördelar i jämförelse med CityGML på webben. Studien visar att CityJSON är 10 gånger mer kompakt än motsvarande fil i CityGML-format, vilket innebär snabbare överföring och hantering på webben. Studien visade även att plattformen Cesium har problem vid hantering av attributdata och lämpar sig bättre för ren visualisering av geometriska data. Detta beror på informationsförluster vid konvertering till 3D-tiles samt att data lagras extern och strömmas till klienten.

Sammanfattningsvis visade studien att CityJSON är ett lovande format som har potential att underlätta utveckling av applikationer för stadsmodeller på webben.

Förord

Denna rapport är resultatet av det slutliga examensarbetet inom civilingenjörsutbildningen Lantmäteri med specialisering inom geografisk informationsteknik på Lunds Tekniska Högskola. Studien är gjord i samarbete med BIM- och GIS-avdelningen på Tyréns i Malmö.

Vi vill tacka vår akademiske handledare Lars Harrie på Lunds Universitet för vägledning, stort engagemang och alla värdefulla diskussioner.

Vi vill även rikta ett stort tack till vår handledare Johan Larsson Wallin, och Pär Hagberg på Tyréns för allt stöd och hjälp med att tackla de problem som uppstod under projektets gång. Vidare vill vi tacka resterande medarbetare på BIM- och GIS-avdelningen på Tyréns för gott mottagande och sällskap.

Lund 2020-04-14
Alfred Hildingson & Patrik Sylve

Innehåll

Figurer

Tabeller

Ordlista

3DCityDB - *3D City Database* - Implementation av CityGML
BIM - *Building Information Model* - Byggnadsinformationsmodellering
CAD - *Computer-aided design* - Datorstödd konstruktion
CLI - *Command-line interface* - Program som tar emot kommandon genom rader kod
DWG - *Drawing* - Filformat för CAD
GML - *Geography Markup Language* - XML-baserat språk för att beskriva geografiska egenskaper
IFC - *Industry Foundation Classes* - Filformat för BIM
IKT - *Information- och kommunikationsteknik* - IT som bygger på kommunikation mellan människor
JSON - *JavaScript Object Notation* - Textbaserat format för utbyte av data
LOD - *Level of Detail* - Beskriver detaljnivå på stadsmodell
OGC - *Open Geospatial Consortium* - Internationellt konsortium för geospatiala tjänster
SGP - *Svensk geoprocess* - Samverkanprojekt för att förenkla användningen av geodata i samhället tjänster
VCS - *Version Control System* - Kategori av verktyg för versionshantering
WFS - *Web Feature Service* - Gränssnitt för att begära geografisk data över internet
XML - *Extensible Markup Language* - Textbaserat märkspråk för överföring av data

Del I

Inledning

1. Introduktion

1.1 Bakgrund

Teknisk utveckling och större tillgång till geografiska data i 3D har lett till ökad tillämpning av stadsmodeller för analys och visualisering. Inom samhällsbyggnad kan användningen av stadsmodeller i många hänseenden effektivisera samhällsbyggnadsprocessen. Visualisering av geodata i 3D kan öka förståelse för planerade projekt och förbättra medborgardialogen. Många analyser kan även utföras på ett kontor istället för ute på fält, vilket innebär stora ekonomiska nyttor [Johansson och Lithén, 2016].

En förutsättning för att stadsmodeller ska kunna användas i samhällsbyggnadsprocessen på ett effektivt sätt är att det finns gemensamma standarder för lagring och överföring av data. Idag är infrastrukturer för geografiska data bristfälliga både internationellt som nationellt. Inom EU pågår arbete för att bygga upp ramverk för utbyte av geografiska data mellan länder enligt bestämmelser från EU-direktivet INSPIRE. I Sverige har Lantmäteriet i samråd med Geodatarådet och andra berörda organisationer tagit fram Nationella geodatastrategin 2016-2020, med syftet att uppnå en välfungerande nationell infrastruktur för geografiska data [Geodatarådet, 2016].

Som en del av arbetet med den Nationella geodatastrategin håller Lantmäteriet i dagsläget (2020) på att ta fram nya nationella specifikationer för att underlätta utbytet av geografiska data för aktörer inom samhällsbyggnad [Lantmäteriet, 2019a]. Specifikationerna för temat Byggnad kommer att kunna mappas över till befintliga informationsmodellen och standarden CityGML. Standardiseringsarbetet bygger vidare på det tidigare samverkansprojektet Svensk geoprocess (SGP).

CityGML är en internationell standard som upprätthålls av organisationen Open Geospatial Consortium (OGC). CityGML är en informationsmodell som används för att beskriva 3D-objekt i stadsmiljöer och förhållandet mellan dessa. Det finns två officiella implementationer av modellen, CityGML och 3D City Database (3DCityDB). I CityGML lagras data i XML-kodning, där objekt i staden är ordnade i en hierarkisk struktur. 3DCityDB, är ett databasschema som implementerar CityGML ovanpå en spatial relationsdatabas.

Användare av CityGML har riktat viss kritik mot XML-implementeringen av standarden. Exempel på kritik är att kodningen är omständlig att arbeta med samt att det är svårhanterligt på webben [Ledoux et al., 2019]. CityGML var inte designad för visualisering utan skapades för lagring och överföring av data. För visualisering på webben krävs det att data konverteras till visualiseringsformat, exempelvis glTF, COLLADA eller KML, vilket innebär informationsförluster.

Som ett alternativ har en forskningsgrupp inom geoinformation i 3D vid Delft tekniska universitet (TU Delft) utvecklat CityJSON. CityJSON är kodad i JSON-format och bygger på informationsmodellen CityGML 2.0. Formatet stödjer stora delar av informationsmodellen, dock med vissa förenklingar. Syftet med CityJSON är att det ska vara ett mer lätthanterligt alternativ till XML-kodningen av CityGML-modellen [CityJSON, uå]. Syftet med CityJSON är att fungera som ett mer lätthanterligt format för att underlätta tillämpningar av informationsmodellen CityGML i praktiken [Ledoux et al., 2019]. Den första versionen av CityJSON lanserades i början av 2019. Sedan dess har ett antal öppna programvaror för konvertering från CityGML, enkel webbvisualisering samt program för validering släppts [CityJSON, uå].

1.2 Problemformulering

I takt med ökad tillämpning av stadsmodeller för analys och visualisering ställs högre krav på de dataformat som används. Formaten måste vara enhetliga för att ett utbyte av data mellan olika aktörer ska ske effektivt. Samtidigt är det fördelaktigt med flexibla format som kan användas i olika syften för att undvika informationsförluster som sker vid konvertering.

För att lagra stadsmodeller används vanligen formatet och informationsmodellen CityGML. Data lagrade i XML-kodade CityGML-filer är ofta stora vilket kan vara problematiskt vid hantering på webben. Vidare innehåller modellen både semantiska och geometriska data om staden. I samband med visualisering krävs det att dessa hanteras separat. Detta påverkar tillgängligheten och möjligheterna att via webben visualisera och analysera data i formatet. Som ett alternativ har CityJSON, en implementering av CityGML i JSON-format tagits fram. Vi vill undersöka hur det nya formatet hanterar de problem som existerar med CityGML vid visualisering av geometri- och attributdata på webben.

En jämförelsestudie mellan två metoder genomförs för att studera visualisering av stadsmodeller i CityGML- respektive CityJSON-format. Metoden för CityGML genomförs med plattformen Cesium och CityJSON med en egenutvecklad webbtjänst. Strategierna analyseras utifrån en kravställning som utgår från projektet IRIS - Smart Cities[1].

[1] `https://irissmartcities.eu/`

1.3 Syfte

Det övergripande syftet med denna studie är att undersöka och jämföra de två formaten CityJSON och CityGML. Undersökningen sker med hänsyn till tillgänglighet och visualisering på webben. Mer specifikt har studien som syfte att:

- Utvärdera vilka verktyg och metoder som finns tillgängliga för att visualisera data i formaten CityJSON och CityGML.
- Utvärdera metoder för visualisering av CityGML genom plattformen Cesium samt egenutvecklad tjänst för CityJSON på webben utifrån en framtagen kravställning.
- Undersöka om CityJSON är ett lämpligt alternativ till XML-implementeringen av CityGML 2.0 för användning på webben

1.4 Avgränsning

Begränsningar med visualisering av CityGML på webben presenteras i avsnitt 3.5.2. I samma avsnitt beskrivs några metoder som hanterar dessa problem. Fallstudien utgår från *en* av dessa strategier för att visualisera CityGML som innebär konvertering till formatet 3D-tiles för visualisering genom plattformen Cesium. Jämförelsen mellan formaten sker således indirekt genom en jämförelse mellan två olika visualiseringsmetoder.

Visualiseringsapplikationerna är framtagna utifrån krav och behov inom projektet IRIS - Smart Cities som presenteras i avsnitt 2.3.1. Studien är geografiskt avgränsad till en del av centrala Göteborg. Data som används kommer från IRIS-projektet samt öppna dataset från Lantmäteriet. Samtliga verktyg som används i studien har antingen öppen källkod eller gratis licens för studenter.

Eftersom studien genomförs under en begränsad tidsram innebär det att implementeringar av vissa funktioner inte får vara alltför komplicerade eller tidskrävande för att uppfyllas enligt kravställningen. Krävs en mer tidskrävande lösning bedöms kravet som icke uppfyllt och diskuteras endast i teori.

1.5 Studieupplägg

Projektets teoretiska del innefattar en litteraturstudie. I denna del presenteras bland annat relevanta format och standarder. Informationen hämtas från artiklar, specifikationer och akademisk litteratur.

I fallstudien tas en kravställning fram. Utifrån kravställningen utvecklas sedan två tjänster, en

tjänst i CesiumJS för stadsmodeller i CityGML-format samt en JavaScript-applikation för stadsmodeller i CityJSON. Tjänsterna utvärderas och diskuteras därefter utifrån de framtagna kraven.

1.6 Disposition

Denna rapport utgörs av fyra delar. I den första delen *Inledning*, introduceras studiens bakgrund, syfte, problemformulering, avgränsning, disposition samt studieupplägg. I följande del *Teoretisk bakgrund*, presenteras litteraturstudien. Avsnittet tar upp tillämpningar av stadsmodeller i samhällsbyggnadsprocessen och relevanta standarder för geografiska.

I rapportens tredje del *Fallstudie*, beskrivs det praktiska arbetet och resultaten presenteras. Rapporten avslutas därefter med *Diskussion och slutsatser*.

Del II

Teoretisk bakgrund

2. Tillämpning av stadsmodeller

2.1 Användningsområde och tillämpning

En stadsmodell är en tredimensionell representation av en stad och innehåller beskrivningar av urbana objekt och miljöer, med stort fokus på stadens byggnader [Biljecki et al., 2015]. Då en tredimensionell modell är närmre verkligheten än traditionella kartor, kan användning av stadsmodeller för visualisering vara ett effektivt sätt att kommunicera geografisk information [Billen et al., 2014; Johansson och Lithén, 2016]. Visualisering har historiskt sett varit det primära användningsområdet för stadsmodeller, men med utvecklad teknologi har tillämpningar för olika typer av analyser ökat. Nedan presenteras några exempel på tillämpningar som är beroende eller drar stor nytta av 3D-data.

Registrering av 3D-fastigheter
Digitala modeller kan användas för att beskriva och visualisera 3D-fastigheter. I många länder beskrivs 3D-fastigheter genom textbeskrivningar och ritningar i 2D. I Sverige lagras 3D-fastigheter i fastighetsregistret med information om dess läge på kartan och i förhållande till marken. När 3D-fastigheter skapas är det många aktörer som är inblandade (markägare, arkitekter, lantmätare mm) och det är vanligt att Lantmäteriet i samband med skapandet av en 3D-fastighet får in 3D-data från byggnadsentreprenörer. Många av dessa data går dock förlorade när fastigheten slutligen beskrivs i 2D i fastighetsregistret [El-Mekawy et al., 2014]. Idag saknas det dock standarder och metoder för att integrera digitala modeller i fastighetsregistret.

Sun et al. [2019] presenterar ett ramverk för att integrera information från fastighetsregistret. Dessa modeller kan användas för att tydligare definiera och visualisera fastighetsgränser samt andra rättsliga förhållanden. Vidare kan det även vara önskvärt att länka 3D-fastigheter med stadsmodeller för att genomföra olika typer av analyser i samband med exempelvis stadsplanering. Ett standardiserat ramverk kan även effektivisera utbyte av data mellan olika aktörer som är involverade i en fastighetsbildning i 3D [Sun et al., 2019a].

Bygglovshandläggning
När byggnader projekteras är det vanligt att en digital byggnadsmodell skapas, men vid ansökning om bygglov är det oftast ritningar i pappers- eller PDF-format som skickas in. Genom användning

av digitala byggnadsmodeller som underlag vid bygglovshandläggningar kan många delar av processen automatiseras, som exempelvis kontrollera att byggnaden uppfyller kraven för tillåten byggnadshöjd och volym [Olsson et al., 2018].

Analys i 3D
När nya vägar eller järnvägar byggs kan bulleranalyser användas för att välja lämpliga placeringar av ljudbarriärer för att minska buller. Data i 3D är fördelaktigt vid den här typen av analyser eftersom ljud kan variera vid olika höjdlägen beroende på hur ljudet färdas i luften. Vidare kan en bättre analys genomföras om information om ytors material och hur dessa absorberar ljud används [Biljecki et al., 2015].

Exempel på andra typer av analyser som kan genomföras med hjälp av stadsmodeller är översvämningsanalyser. Vattenflöden kan simuleras och eventuella skador på byggnader beräknas. Vid olyckor kan stadsmodeller användas för att avgöra exempelvis optimala placeringen av brandbilar vid en brandutryckning [Biljecki et al., 2015].

Stadsplanering
För stadsplanering har utnyttjande av stadsmodeller flera tillämpningar. 3D-analyser kan vara ett bra underlag vid beslut om exempelvis vägdragningar. Vidare kan visuella analyser användas för att simulera hur projektförslag passar in i relation till befintlig bebyggelse [Liu et al., 2018].

Webbaserade stadsmodeller kan också vara ett bra sätt för stadsplanerare att kommunicera med medborgare. Traditionellt sett har papperskartor och fysiska presentationer använts för att förmedla planerade stadsförändringar. Fördelar med att använda webbaserade lösningar är att deltagare inte är bundna till att vara fysiskt närvarande vid en presentation och det finns möjligheter att vara anonym, vilket kan öka benägenheten hos en del medborgare att dela sina åsikter [Lafrance et al., 2019].

2.2 Datainsamling och modellering

Det finns olika typer av stadsmodeller och valet av modell beror på i vilket syfte den ska användas. Tre vanliga typer av 3D-modeller är: objektorienterad modell, punktmolns- och bildmodell. En objektorienterad modell innehåller objekt som exempelvis byggnader i 3D. 3D-objekten kan tilldelas beskrivningar av deras funktioner och attribut. Eftersom det är möjligt att länka data till objekten är denna modell användbar för analyser. Punktmolns-modeller utgörs av data insamlat med bland annat laserskanning. Data från skanningen består av punkter med rumsliga koordinater som ritas ut i en 3D-miljö. En fotorealistisk modell kan skapas genom att färglägga punkterna utifrån bilddata. Bildmodeller i 3D skapas genom att lägga bilder ovanpå en digital ytmodell som är modellerad utifrån punktdata. Bildmodeller används främst för visualisering då man kan uppnå en fotorealistisk modell av staden [Persson och Lithén, 2016].

2.3 Smarta städer

Idag bor 75% av Europas befolkning i stadsområden, en siffra som ständigt ökar. För att städerna ska hänga med i denna förändring krävs det bra lösningar för mobilitet och produktion av energi. Smarta städer är ett branschöverskridande ramverk för att med hjälp av den pågående digitaliseringen ta fram lösningar till de problem som uppstår vid den ökande urbaniseringen. I en smart stad arbetar olika branscher inom bland annat energi, mobilitet och transport tillsammans för att ta fram gemensamma lösningar som mynnas ut till förbättringar för städernas medborgare. Bland annat information- och kommunikationsteknik (IKT) används för att ta fram nya och förbättrade lösningar inom exempelvis kollektivtrafiken och vattendistribuering. Det innebär också lösningar för att förbättra kommunikationen mellan invånare och myndigheter. Runt om i världen sker det ett flertal olika smarta städer projekt, varav ett projekt som Sverige är involverad i är *IRIS - Smart Cities* [IRIS - Smart Cities, 2019a; Gemalto, 2020].

2.3.1 IRIS - Smart Cities

2015 antog Europeiska unionen en policy för att arbeta för att EU-medlemmar ska ha tillgång till säker, ekonomisk och klimatvänlig energi. Målet med policyn är att Europa ska ligga i framkant inom förnybar energi. Idag identifieras området som bristfällig vilket bland annat beror på att infrastrukturer inte har hängt med i den tekniska utvecklingen [IRIS - Smart Cities, 2019a; Europakonventionen, 2015].

Dagens centraliserade produktion av energi är i takt med den förnybara energins frammarsch på väg mot en mer decentraliserad produktion där energi kan skapas lokalt med hjälp av exempelvis vindturbiner. En sådan övergång sätter hög press på lokala elnät och förutsätter en högre flexibilitet än vad som finns idag. För att öka intresset för investeringar inom förnybar energi krävs det att aktörer inom olika områden arbetar tillsammans för att ta fram gemensamma lösningar. Investeringar som leder till innovationer som mynnar ut till förbättringar för städernas medborgare [IRIS - Smart Cities, 2019a].

2017 påbörjades det EU-finansierade projektet *IRIS Smart Cities* med syftet att ta fram tekniska lösningar inom energi, mobilitet samt IKT för att hantera den skiftande miljön. Projektet är en del av Europas största forskning och innovationsprogram Horizon 2020. I det fem år långa IRIS-projektet medverkar sju städer i samverkan med 43 aktörer. IRIS – Smart Cities är ett Lighthouse-projekt vilket innebär att tre städer, Göteborg, Utrecht och Nice har pekats ut att ligga i framkant i utveckling inom projektets områden. Fyra städer, Vaasa, Alexandroupolis, Santa Cruz de Tenerife samt Focsani har identifierats som följestäder och ska under projektets gång ta lärdom från och återskapa lösningar från städerna i framkant. I projektet har 16 problemområden identifierats och brutits ner till fem grupper, se figur 2.1 [IRIS - Smart Cities, 2019a].

Inom projektet arbetar nio olika aktörer med att ta fram en testpilot av en digital informationsmodell för Göteborg. Syftet med den digitala modellen är att med hjälp av geografiska data underlätta utbyte av information, öka medborgardialogen och innovation. Med tillgängliga data över infrastruktur och planerade projekt är målet att öka tjänster och smarta lösningar för mobilitet i staden.

Figur 2.1: IRIS-projektets fem fokusgrupper [IRIS - Smart Cities, 2019b]

3. Standarder och Programvaror

Kommuner och andra myndigheter samlar in och dokumenterar stora mängder data som har stor potential att stödja beslutsfattanden inom stadsplanering och andra administrativa aktiviteter inom städer. Insamlade data dokumenteras i olika format och kvalité. Ett stort problem som finns idag är att integrera information från myndigheters dokumenterade data i stadsmodeller på ett effektivt sätt [Billen et al., 2014]. Användning av enhetliga standarder och specifikationer är därför av stor vikt för att kunna utnyttja och uppdatera den data som finns och kontinuerligt produceras. Idag saknar många länder infrastrukturer för geodata och generellt är det problematisk att utbyta data mellan länder då det saknas gemensamma standarder [Lantmäteriet, 2019b].

Det pågår många projekt för att implementera infrastrukturer för utbyte och lagring av geografiska data, både nationell och internationellt. Inom EU finns sedan 2007 direktivet INSPIRE som är ett ramverk för infrastrukturer för geografiska data. I Sverige är Lantmäteriet den myndighet som har ansvaret för frågor som rör samordning av geografiska data. De driver arbetet för ett nationellt ramverk, som kan ses som en nationell förlängning av INSPIRE. Idag arbetar Lantmäteriet med att ta fram nationella specifikationer för geografiska data (se avsnitt 3.1.3).

3.1 Standardisering inom geodataområdet

3.1.1 Open Geospatial Consortium - OGC

OGC är en organisation som arbetar aktivt med att ta fram internationella öppna standarder för spatiala tjänster. Organisationen startade 1994 och består idag utav över 530 företag, forskningsorganisationer och universitet. Vid arbetet med spatiala data har OGC en central roll och dess arbete har lett till utveckling av ett flertal internationella standarder som exempelvis CityGML och Web Map Service (WMS) [OGC, 2019].

Vid framtagning av spatiala tjänster och information arbetar OGC enligt en samling dataprinciper *FAIR - Findable, Accessible, Interoperable och Reusable*. Principerna togs fram av en grupp akademiker och industriutövare 2016. Syftet med principerna är att vetenskapliga data som produceras,

eller metoder som används för att skapa data, ska vara tillgängliga och kunna återanvändas inom forskning. Stor vikt läggs vid att data ska gå att hitta automatiskt med maskiner med hjälp av lämpliga metadata [Wilkinson et al., 2016].

3.1.2 INSPIRE

INSPIRE är ett EU-direktiv från 2007 som anger bestämmelser för upprättandet av en infrastruktur för geografiska data inom Europeiska Unionen för att skapa en positiv påverkan på miljön. Syftet med direktivet är att effektivisera utbytet av geografiska data mellan myndigheter. Direktivet adresserar totalt 34 olika teman som på olika sätt berör miljön. Arbetet förväntas bli färdigställt 2021.

3.1.3 Nationella specifikationer

Lantmäteriet fick 2018 i uppdrag av Regeringen att utreda nationella lösningar för tillgängliggörande av geografiska data som en del i arbetet för en effektiv samhällsbyggnadsprocess [Regeringen, 2018]. I slutrapporten för uppdraget föreslås ett ramverk som tillhandahåller enhetliga geografiska data oberoende av producent. För att ett effektivt utbyte av data mellan olika aktörer ska kunna ske krävs det att data är standardiserad. Från januari 2019 har Lantmäteriet i samverkan med andra organisationer arbetat med att ta fram nationella specifikationer för geografiska data inom samhällsbyggnadsprocessen [Lantmäteriet, 2019b].

Arbetet med att ta fram nationella specifikationer bygger vidare på SGP, som var ett samverkansprojekt mellan Lantmäteriet, kommuner och Sveriges Kommuner och Landsting [SKL, 2019]. I projektet tog man fram specifikationer för nio olika geodatateman, bland annat Byggnad, Höjd och Djup samt Detaljplanering. Syftet med projektet var att underlätta hanteringen av ärenden inom samhällsbyggnadsprocessen genom enhetliga arbetsmetoder för myndigheter och aktörer. SGP Byggnad 3.0 innehåller specifikationer för byggnader i 2D och 3D. Specifikationerna för byggnader i 3D bygger vidare på INSPIRE:s specifikationer för byggnader samt informationsmodellen CityGML [Svensk geoprocess, 2016].

3.2 Kodningsformat

3.2.1 Extensible Markup Language - XML

Extensible Markup Language (XML) är ett flexibelt märkspråk i textformat som är läsbar för både datorer och människor. Språket härstammar från Standard Generalized Markup Language men är något förenklad för att lämpa sig bättre för användare. XML har en central roll vid överföring av data på internet och är ett av de format som används mest. Överföringen sker mellan människor, människor - maskin och maskiner. XMLs syntax ligger till grund för ett flertal andra format, bland annat SVG och GML [W3C, 2016; W3C, 2015].

Formatet är hierarkiskt uppbyggt utifrån ett taggsystem. Ett element skapas genom en starttagg och avslutas sedan med en sluttagg. Allt innanför start- och sluttaggen tillhör då elementet. Ett element kan vara tomt, ha ett värde eller innehålla andra element. Ett element kan även tilldelas attribut som definieras i elementets starttagg. XML har strikta syntaxregler vilket reducerar mängden fel och innebär att det kan pålitligt köras av program [W3C, 2016; W3C, 2015].

3.2.2 Geography Markup Language - GML

Geography Markup Language (GML) är ett märkspråk i XML-kodning som upprätthålls av OGC. GML används för att överföra och lagra geografiska data. Objekt och fenomen beskrivs som *Features* (se figur 3.1) med ett definierat geografiskt läge. En *Feature* kan innehålla olika typer av egenskaper och attribut eller andra *Feature*-objekt.

Konkreta objekt kan även beskrivas geometriskt enligt GML:s geometriska modell. Geometrier i GML är definierade till stora delar utifrån ISO-standarden 19107. I modellen finns det fyra geometriska primitiver, *Point*, *Curve*, *Surface* och *Solid* [OGC, 2007]. Dessa geometrier kan kombineras till *aggregates*, *complexes* eller *composites* som ställer vissa krav på geometrin som exempelvis de topologiska förhållandena mellan primitiverna. Ytor beskrivs i modellen med trianglar och polygoner medan 3D-geometrier beskrivs av dess avgränsande ytor (*Boundary representation*, eller *B-rep*) [Ohori et al., 2018].

```
<Feature fid="142" featureType="school" >
    <Description>Balmoral Middle School</Description>
    <Property Name="NumFloors" type="Integer" value="3"/>
    <Property Name="NumStudents" type="Integer" value="987"/>
    <Polygon  name="extent" srsName="epsg:27354">
        <Coordinates  name="extent" srsName="epsg:27354">
            <Coordinates>
                491888.999999459,5458045.99963358
                ...
                491953.999999466,5458017.99963357
            </Coordinates>
        </LineString>
    </Polygon>
</Feature>
```

Figur 3.1: Beskrivning av en skolbyggnad i GML. Modifierat exempel från [Lake, uå]

3.2.3 JavaScript Object Notation - JSON

JavaScript Object Notation (JSON) är ett textbaserat filformat som är läsbar för både människor och maskin. Formatet är härledd från JavaScript men stödjer idag i princip alla programmeringsspråk. Även fast formatet inte är bundet till ett enskilt programmeringsspråk används konventioner som efterliknar C-programmeringsfamiljen som exempelvis Java, JavaScript och C++.

JSON har under de senaste åren fått ett rejält uppsving i popularitet bland utvecklare och är idag det främsta formatet för överföring av data på webben. År 2017 levererade nio av de tio mest populära webb-APIer data i JSON-format. Det beror på att det finns fullt stöd för formatet i JavaScript, som är det vanligaste skriptspråket för webbsidor [Target, 2017].

JSON har följande datatyper: Objekt, Sträng, Nummer, Lista, Boolean och Null. Ett objekt är en samling med nyckel - värdepar. Nyckeln är en sträng och är unik inom objektet. Nyckeln är kopplad till ett eller flera värden. Objekt avgränsas med måsvingar där varje nyckel-värdepar separeras med kommatecken. En nyckel tilldelas värde genom semikolon [W3School, uå].

3.2.4 GeoJSON

GeoJSON är ett JSON-baserat format för utbyte av geografiska data på webben framarbetad av Internet Engineering Task Force och stöds av de flesta stora kartverktygen på webben. I GeoJSON finns det tre typer av objekt: Geometry, Feature och FeatureList. En Geometry är ett objekt som endast har en geometri, vilken kan vara av typen Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon eller GeometryCollection. Ett Geometry-objekt innehåller även en

lista med koordinater som beskriver objektet. Ett objekt av typen Feature definieras också av geometrier, men kan även ha andra attribut (*properties*). Det finns inga begränsningar i vilka attribut som en Feature kan ha, men de måste definieras som ett JSON-objekt (3.2.3). En FeatureCollection innehåller flera objekt av typen Feature [Butler et al., 2016].

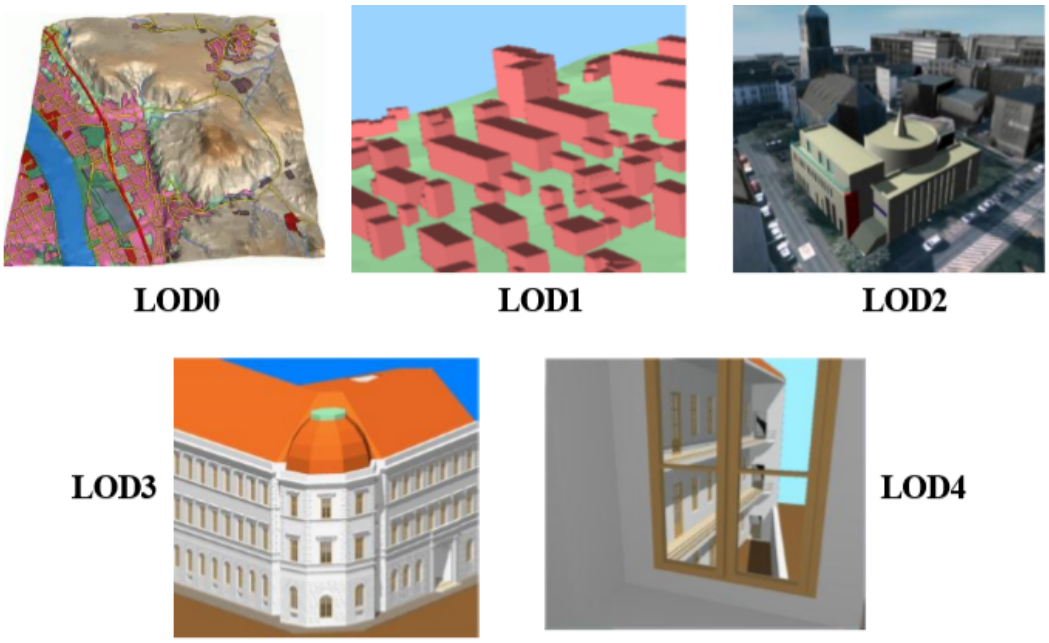
GeoJSON hanterar inte topologi, för detta har en utökning av formatet skapats, TopoJSON. I formatet representeras delade linjesegment mellan geometrier med *arcs* [Bostock, 2018].

3.3 Format för stadsmodeller

3.3.1 CityGML

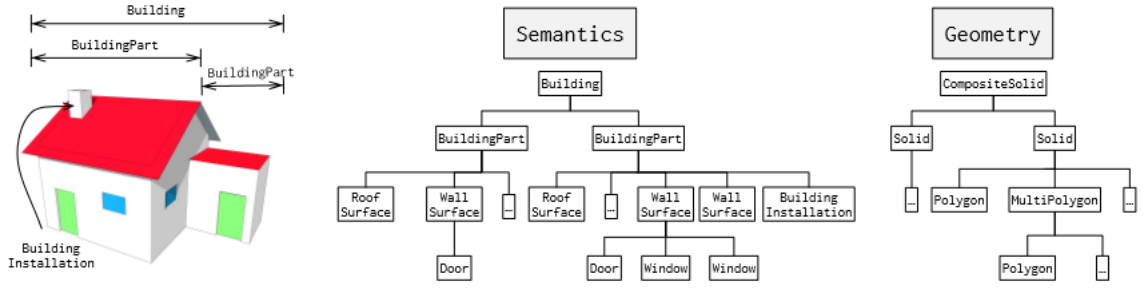
CityGML är en öppen informationsmodell och format för utbyte samt lagring av stadsmodeller i 3D som upprätthålls av OGC. Modellen hanterar sätt att beskriva vanligt förekommande stadsobjekt samt tillhörande attribut och geometrier. I modellen kan även andra egenskaper som tillhör en stad beskrivas, som exempelvis vattenmassor, vegetation och terräng [OGC, 2012]. CityGML är en av de vanligaste standarderna för lagring av stadsmodeller och används bland annat av kommuner och myndigheter för underhåll och planering av städer [Ohori et al., 2018]. Den nuvarande versionen för CityGML är 2.0 men en ny version, 3.0, är under utveckling.

En stadsmodell i informationsmodellen CityGML 2.0 kan beskrivas i fem olika detaljnivåer, LOD (figur 3.2). Den lägsta nivån, LOD0, är en översiktlig representation i 2,5D och innehåller en modellerad terräng i 3D med en karta, vanligen satellitbild, avbildad ovanpå. I LOD1 är byggnader rektangulära. I LOD2 beskrivs delar av byggnader så som tak i högre geometrisk detalj. I LOD3 är byggnaders arkitektoniska egenskaper beskrivna, med detaljerade tak- och fasadytor. I den sista detaljnivån, LOD4, beskrivs även byggnaders interiörer med rum, fönster, dörrar och trappor. Detaljnivåer gäller även för semantiska data, men det finns inga krav på att någon sådan information måste tillhöra objekten [Ohori et al., 2018].



Figur 3.2: De fem olika detaljnivåerna i CityGML 2.0 [OGC, 2012]

CityGML är hierarkiskt uppbyggd med staden, *CityObject*, som den högsta nivån. En stad innehåller objekt för exempelvis byggnader som i sin tur består av byggnadsdelar. De olika stadsobjekten är grupperade i tretton semantiska huvudmoduler: *Appearance, Bridge, Building, CityFurniture, CityObjectGroup, Generics, LandUse, Relief, Transportation, Tunnel, Vegetation* och *WaterBody*. En delmängd av GML3 (se avsnitt 3.2.2), som är baserad på standarden ISO 19107, används för att beskriva objekten geometriskt. Figur 3.3 visar hur en byggnad som består av två byggnadsdelar hierarkiskt byggs upp genom en semantisk och geometrisk del.



Figur 3.3: En byggnads semantiska- och geometriska uppbyggnad i CityGML 2.0 [Ledoux et al., 2019]

CityGML har två officiella implementationer. Vanligast är att modeller i CityGML lagras i XML-format. Denna implementation används främst vid överföring och publicering av data [Ohori et al., 2018]. Den andra implementationen är 3DCityDB, vilket är en geografisk databas som gör det möjligt att lagra 3D stadsmodeller ovanpå en spatial relationsdatabas. Databasen har utvecklats med öppen källkod och är plattformsneutral. I 3DCityDB kan man importera, analysera och exportera 3D-modeller enligt standarden CityGML 1.0 och 2.0. Det går även att exportera till olika visualiseringsformat, bland annat KML, COLLADA och glTF.

Vid praktiska tillämpningar behövs informationsmodellen oftast utökas för att möta specifika behov. Detta kan göras genom att använda klassen *GenericCityObject* samt *genericAttributes* för att definiera objekt som inte redan finns i informationsmodellen. Ett mer formellt sätt att utöka modellen är genom Application Domain Extensions (ADE) som tillåter definiering av nya klasser samt utökning av de redan existerande klasserna i datamodellen. En ADE specifieras genom ett XML-schema eller ett UML-diagram [Biljecki et al., 2018].

3.3.2 CityJSON

CityJSON är en JSON-kodad implementering av informationsmodellen CityGML 2.0. Kodningen har utvecklats vid TU Delft och introducerades under 2019. CityJSON är idag ingen officiell OGC-standard. CityJSON utvecklades för att skapa en alternativ implementering till CityGML som beskrivs som svårt och komplex att arbeta med. En implementering i JSON förenklar arbetet för utvecklare och främja användningen av den officiella OGC-standarden och informationsmodellen CityGML 2.0 i praktiken [Ledoux et al., 2019; CityJSON, uå].

Utvecklarna såg ett behov av övergå från XML till JSON av främst tre anledningar: den första anledningen är att JSON är det dominerande formatet för överföring av data på webben. JSON används i betydligt fler fall än XML. Populariteten med JSON har gjort att antalet bibliotek och programvaror som stödjer formatet är betydligt högre. Den sista anledningen är att JSON är baserad på endast två datastrukturer, Lista och Objekt som stöds av i princip alla programmeringsspråk. I figur 3.4 visas ett exempel på en CityJSON-fil [Ledoux et al., 2019; CityJSON, uå].

I CityJSON har man gjort förändringar från CityGML. Bland annat gått ifrån CityGMLs djupa hierarki och ersatt med en platt struktur samt att koordinater lagras i en separat lista. Dessa anledningar i kombination med att man går ifrån XMLs taggsystem resulterar i att en CityJSON-fil är i genomsnitt 6 gånger mer kompakt än en CityGML-fil [Ledoux et al., 2019; CityJSON, uå].

```
{
    "type": "CityJSON",
    "version": "1.0",
    "metadata": {
        "geographicalExtent": [
            ...
        ]
    },
    "CityObjects": {
        "Building_1": {
            "geometry": [{
                "boundaries": [...],
                "lod": 2,
                "semantics": {
                    "surfaces": [
                        {"type": "GroundSurface"},
                        ...
                    ],
                    "values": [...]
                },
                "texture": {"Rhino texturing": {
                "values": [
                    [[...]],
                    ...
                    ]}},
                "type": "MultiSurface"
            }],
            "type": "Building"
        }
    },
    "vertices": [
        [...],
        ...
    ]
}
```

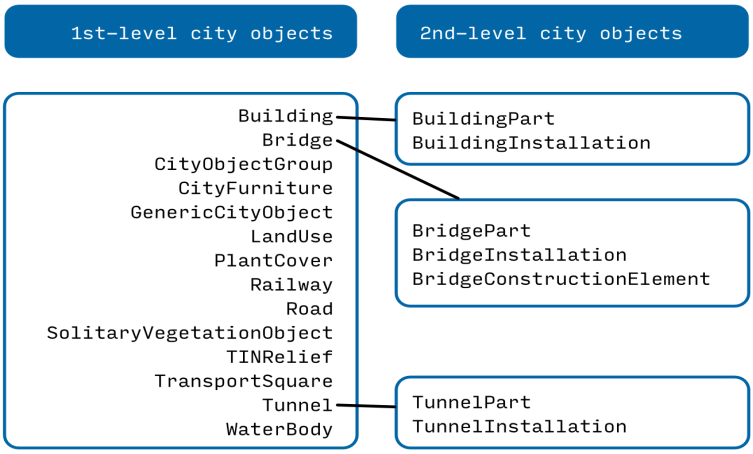
Figur 3.4: Exempel på en CityJSON-fil som beskriver en byggnad

CityJSON är ett JSON-objekt som måste innehålla minst fyra nycklar:

- *type* - måste ha värdet ”CityJSON”
- *version* - en sträng med värde ”X.Y” som beskriver den version av CityJSON som används.
- *CityObjects* - en samling av JSON-objekt som innehåller ett unikt ID som nyckel och ett stadsobjekt som värde.
- *vertices* - en lista som innehåller koordinater för alla punkter som beskriver stadsobjektens geometrier.

Stadsobjekt

Stadsobjekten består av tolv klasser som motsvarar de moduler som finns i CityGML (*Appearance* tillhör inte stadsobjekt i CityJSON, men går att definiera i ett CityJSON-objekt). I en CityJSON-fil radas samtliga objekt upp som värden till nyckeln *CityObjects*. Ett stadsobjekt måste ha en *type* som ska vara en av de klasser som redovisas i figur 3.5. Objekt måste även ha en geometri, och kan ha attribut som är tillåtna enligt specifikationerna i CityGML. I CityJSON har vissa delar av CityGML inte implementerats på grund av att dessa inte används i praktiken, som exempelvis transportklassen *Track*.



Figur 3.5: I CityJSON finns det två hierarkiska nivåer som beskriver förhållanden mellan stadsobjekt [Ledoux, 2019]

Ett objekt tillhör antingen första- eller andra nivån av stadsobjekt baserat på dess förhållanden till andra klasser. Ett förstanivå-objekt kan finnas enskilt medan ett andranivå-objekt behöver existera ”under” ett objekt av första nivån och ha relationen ”parent-child” (figur 3.5) [Ledoux, 2019]. I figur 3.6 ges ett exempel på ett bro-objekt av typen *Bridge* som består av två brodelar av typen *BridgeConstructionElement*. Bron och dess delar länkas samman genom nycklarna *parents* och *children*.

```
...
"CityObjects": {
    "id1-parent": {
        "type": "Bridge",
        "children" : ["id2-child", "id3-child"],
        "geometry" : [{..}]
    }
},
{
    "id2-child": {
        "type" : "BridgeConstructionElement",
        "parents" : "id1-parent",
        "geometry" : [{
            "type": "MultiSurface",
            "lod" : 2,
            "boundaries":[
                [[26,27,28,29]]...[[82,94,92,95]]
            ]
        }]
    }
    "id3-child": {
        "type" : "BridgeConstructionElement",
        "parents" : "id1-parent",
        "geometry" : [{..}]
        }
    }
...
```

Figur 3.6: Delar av stadsobjekt i CityJSON-format

Geometrier

Stadsobjekt har geometrier som definieras i *Geometry*-objekt. Geometrier kan ha en av typerna: *MultiPoint*, *MultiLineString*, *MultiSurface*, *CompositeSurface*, *Solid*, *MultiSolid*, *CompositeSolid* eller *GeometryInstance*. Enskilda geometrier som exempelvis *Point* i CityGML, beskrivs istället som *MultiPoint* i CityJSON. En geometri måste ha en definiera LOD samt *boundaries* som definierar geometrin. *boundaries* innehåller listor med referenser till punkter i koordinatlistan *indices*.

Transformationsobjekt

För att reducera storleken på en fil kan ett transformationsobjekt användas. Ett transformationsobjekt innehåller två matriser, *Scale* och *Translate*, där respektive matris innehåller tre värden. Skalfaktorn möjliggör att koordinaterna ska sparas som heltal och skalfaktorn som flyttal. Matrisen

translate reducerar mängden värdesiffror som sparas i varje koordinat. I figur 3.7 illustreras ett exempel på hur ett transformobjekt kan se ut.

För att beräkna koordinatens riktiga värde används följande formel:

Koordinat[0] = KomprimeradKoordinat[0] * [*transform*][*scale*][0] + [*transform*][*translate*][0]

```
"transform": {
    "scale": [0.01, 0.01, 0.01],
    "translate": [2168443.12, 4571224.12, 13.05]
}
```

Figur 3.7: Transformationsobjekt i CityJSON

Metadata

CityJSON-fil kan kompletteras med information om data, så kallade metadata. I metadata kan bland annat referenssystem, omslutande rektangel och geografisk plats lagras. I figur 3.8 illustreras ett exempel på hur ett referenssystem är definierad i metadata, där refereringssystem anges som EPSG-kod. Till skillnad från CityGML måste alla data i en CityJSON-fil vara definierad i samma koordinatsystem.

```
"metadata": {
  "referenceSystem": "urn:ogc:def:crs:EPSG::3007"
}
```

Figur 3.8: Referenssystem definieras under nyckeln ”metadata” i CityJSON

Utseendeobjekt

Utseendeobjekt (eng *Appearance Object*) är ett JSON-objekt som:

- Har nyckeln *materials* och har ett värde som är en lista med material-objekt. I listan kan värde tilldelas *ambientIntensity, diffuseColor, emissiveColor, specularColor, shininess, transparency* och *isSmooth*.
- Har nyckeln *textures* och har ett värde som är en lista med textur-objekt. I listan måste ett värde för *type* vilket antingen är av värdet PNG eller JPG. Objektet måste även innehålla *image* som har värdet av en string som är länken till filen.
- Har båda nycklarna *materials* och *textures* likt beskrivet ovan.
- Har nyckeln *vertex-texture* och har ett värde som är en lista med koordinater för varje *UV vertex* av stadsmodellen

- Har nyckeln *default-theme-texture* och har en sträng som värde som motsvarar namnet på standardtemat för utseendet. Standardtemat används för att bestämma vilken textur som ska visas när en geometri har fler.
- Har nyckeln *default-theme-material* och har en sträng som värde som motsvarar namnet på standardtemat för materialet. Standardtemat används för att bestämma vilket material som ska visas när en geometri har fler [Ledoux, 2019].

```
"appearance": {
  "materials": [],
  "textures":[],
  "vertices-texture": [],
  "default-theme-texture": "",
  "default-theme-material": ""
}
```

Figur 3.9: Utseendeobjekt i CityJSON

En yta i CityJSON kan tilldelas ett eller flera material. Materialet tilldelas ytan genom att ha en nyckel *material* som har ett värde som innehåller nyckel-värdepar. Nycklarna är temat på materialet och värdet är ett JSON-objekt som innehåller antingen *values* som har värdet en lista med heltal eller *value* som har värdet ett heltal. Heltalen refererar sedan till platsen i *material* som är placerad i utseendeobjektet [Ledoux, 2019].

Utöver material kan även en eller flera texturer tilldelas en yta. Objektet har då nyckeln *texture* som har ett värde som innehåller nyckel-värdepar där nyckeln är temat på texturen. Värdet är ett JSON-objekt med värdet *values* som är en lista med listor som innehåller helttal. I varje lista så refererar det första talet till *texture* i utseendeobjektet, resterande siffror refererar till texturens position på ytan [Ledoux, 2019].

Utökningar

Det finns möjlighet att utöka filen utöver specifikationen. För att validera filen mot utökningarna används utökningar (*eng Extension)*, motsvarande ADE i CityGML. Extensions är en fristående fil i JSON-format som innehåller regler för filens syntax. I filen som innehåller utökningar finns ett JSON-objekt som måste innehålla följande sju nycklar:

type - Måste ha värdet *"CityJSON_Extension"*
name - Är av typen string och identifierar extension
uri - Är av typen string och med URI till var JSON-filen är lokaliserad
version - Är av typen string och identifierar extensionens version
extraRootProperties - Har värdet JSON-objekt och beskriver tilläggningen av en egenskap i roten av dokumentet. Objektet kan också vara tomt.

extraAttributes - Har värdet JSON-objekt och beskriver tilläggningen av attribut. Objektet kan också vara tomt.
extraCityObjects - Har värdet JSON-objekt och beskriver tilläggningen av ett nytt stadsobjekt. Objektet kan också vara tomt.

I figur 3.10 visas ett exempel på hur en fil med utökning kan se ut.

```
{
  "type": "CityJSON_Extension",
  "name": "Suncell",
  "uri": "https://url.se/suncell.json",
  "version": "1.0",
  "description": "Suncell"
  "extraRootProperties": {},
  "extraAttributes": {},
  "extraCityObjects": {}
}
```

Figur 3.10: Utökning i CityJSON

En CityJSON-fil länkas samman med en utökning genom att lägga till *namn, url* och *version* under *Extensions*, se figur 3.11. En CityJSON-fil kan vara länkad till en eller flera extensions [CityJSON, 2019].

```
"extensions": {
    "suncell": {
      "url" : "https://url.se/suncell.json",
      "version": "1.0"
    }
}
```

Figur 3.11: Exempel på hur en CityJSON-fil länkas samman med en utökning

3.4 Format för CAD och BIM

Inom byggsektorn används byggnadsinformationsmodellering (BIM) för dokumentering och lagring av information under byggnadsprocessen. Informationen kan lagras i en BIM-modell som är en beskrivning av en byggnad i 3D. Byggnader som är modellerade i BIM kan vara önskvärda att använda som underlag till stadsmodeller då data inte behöver samlas in på nytt. Vid integrering från BIM-modeller till stadsmodeller uppstår problematik [Sun et al., 2019b]. Eftersom områdena har olika syften, stämmer informationsmodellerna för att lagra data inte alltid överens och konvertering mellan dessa kan medföra informationsförluster.

3.4.1 Industry Foundation Classes - IFC

IFC är en öppen internationell standard (ISO 16739-1:2018) framtagen av den internationella ideella organisationen BuildingSMART för att beskriva BIM. I ett stort byggnadsprojekt arbetar ett flertal aktörer med varandra. Då varje aktörer arbetar med sina egna programvaror med respektive format kan det innebära problem vid överföring av data. IFC löser detta problem genom att vara ett neutralt format vilket innebär att användaren inte blir bunden till en enskild leverantörs produkt. IFC kan både beskriva fysiska ting som exempelvis del av en byggnad samt abstrakt information som byggnadskostnad. Standarder kan implementeras i ett flertal olika format, bland annat XML och STEP, där STEP är vanligast [buildingSMART, 2019; buildingSMART, 2019].

3.4.2 Drawing - DWG

Autodesk lanserade 1982 AutoCad som med tiden har växt till det mest använda CAD-programvaran. AutoCAD lanserades med filformatet drawing (.dwg), vilket idag är standardformatet för överföring av data mellan CAD-program. DWG är ett binärt filformat med användarskapad information för 2- 3D-dimensionell design data. Eftersom det är binärt är det inte läsbart för människor istället används specifika program för translation av filerna. Formatet påträffas främst i designmiljöer och är ett av de mest använda formatet inom konstruktion [Open Design Alliance, 2019; Scan2CAD, 2016]. Filerna kan bland annat bestå av konstruktion, geometriska data samt kartor och foton [Autodesk, uå].

3.5 Format och strategier för visualisering av geografiska data på webben

Visualisering av stadsmodeller lagrade i CityGML-format kräver vanligen att data först konverteras till ett visualiseringsformat för visualisering på webben. Det beror dels på att filer i CityGML innehåller stora mängder data, vilket gör det ohållbart att genom en server-klient applikation överföra data för visualisering. Vidare är det komplicerat att tolka och hämta information från XML-filer i webbapplikationer [Gaillard et al., 2015]. För visualisering används istället standarder inom datorgrafik som X3D, COLLADA, eller KML [Ohori et al., 2018]. Formaten har dock vissa begränsningar när det kommer till att beskriva mer komplexa geografiska förhållanden och semantisk information [Billen et al., 2014].

3.5.1 Visualiseringsformat

KML (Keyhole markup language) är ett öppet format och utformades för visualisering av geografiska objekt i virtuella glober som exempelvis Google Earth [OGC, 2015]. I KML finns det ett element för att beskriva semantik och det är även möjligt att definera egna element. 3D-formatet X3D (Exstensible 3D-graphics) stödjer också visualisering av geografiska data. Även i X3D finns möjligheter att beskriva semantisk information, men liksom KML finns inga standarder för hur denna information ska mappas över från CityGML. CityGML är primärt designat för representation av städer och överföring av information som hör till staden, till skillnad från X3D och KML som är skapade för effektiv visualisering. Dessa ses därför som ett komplement till informationsmodellen CityGML [Gröger and Plümer, 2012].

Slutligen finns 3D Tiles, en öppen specifikation designat för effektiv strömning och visualisering av stora spatiala dataset över nätet. I formatet lagras data hierarkiskt i ett *tileset* som består av ett metadataobjekt i JSON samt ett eller flera *tile*-objekt lagrade i en trädstruktur som refererar till data i ett av fyra olika *tile*-format: *Batched 3D-models* (ytor, byggnader eller andra typer av unika 3D-modeller), *Instanced 3D models* (3D-modeller som återanvänds på flera ställen, som exempelvis träd, lyktstolpar eller parkbänkar), *Point Clouds* för punktmolnsdata samt *Composites* som är två eller flera *tiles* sammanslagna [Cesium, uå].

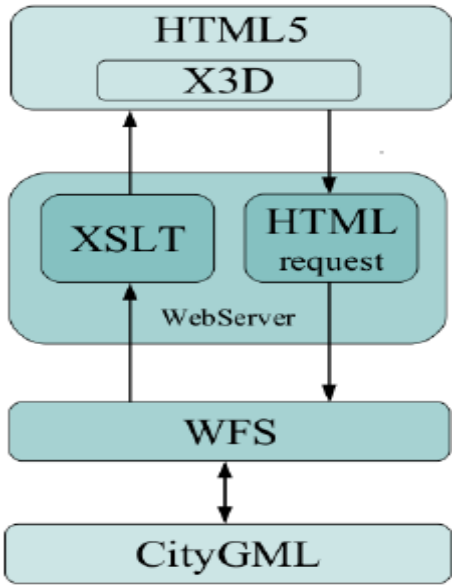
3.5.2 Strategier för visualisering på webben

För visualisering av stadsmodeller på webben finns det både öppna och kommersiella verktyg. Några exempel på JavaScript-bibliotek med öppen källkod är CesiumJS[1], Mapbox GL JS[2] och Leaftlet [3]. Dessa riktar in sig på visualisering av geografiska data och bygger på WebGL, ett API för 2D- och 3D-visualisering i webbläsare.

I CityGML läggs stort fokus på semantiska data och vilka relationer dessa har (exempelvis en byggnad består av väggar, tak, fönster osv.). Vid konvertering från CityGML till visualiseringsformat går ofta semantiska data förlorad [Ohori et al., 2018]. Dessa data är framförallt viktiga för vissa typer av analyser, men i rent visualiseringssyfte räcker det oftast med geometriska data. Nedan presenteras några strategier som kan användas för visualisering av data i CityGML-format.

Rodrigues et al. [2013] tar fram en lösning där data lagras i CityGML-format på en server. I ett första skede hämtas geometrier från CityGML-filen genom standarden Web Feature Service (WFS). En XML-tolkare, XSLT, används därefter för att representera geometrierna och vissa semantiska data i visualiseringsformatet X3D. Data i X3D skickas därefter till en JavaScript-applikation på klientsidan och visualiseras med hjälp av WebGL (se figur 3.12).

[1] `https://cesium.com/cesiumjs/`
[2] `https://docs.mapbox.com/mapbox-gl-js/api/`
[3] `https://leafletjs.com/`



Figur 3.12: Förslag på strategi för visualisering av stadsmodell i CityGML på webben [Rodrigues et al., 2013]

Gaillard et al. [2015] hanterade problematiken med stora CityGML-filer genom att i ett första skede spatialt dela upp data i rutnät (*tiles*). Geometrier, koordinater och semantiska data konverteras och lagras i JSON-format. En webbtjänst utvecklades i JavaScript-biblioteket ThreeJS och data i JSON hämtas kontinuerligt utifrån behov.

2015 introducerade Cesium det öppna formatet 3D Tiles. Liksom Gilliard et al. [2015] baseras denna lösning på en strömning av *tiles* vid behov, vilket möjliggör interaktiv visualisering av stora dataset.

3.6 Programvaror och ramverk

3.6.1 ThreeJS

ThreeJS är ett JavaScript-bibliotek för att skapa 3D-innehåll i webbläsare. I ThreeJS finns det bland annat stöd för att hantera geometrier, ljussätta scener, tilldela objekt material och texturer. Ramverket bygger på WebGL, vilket är ett rasteriserings-API för grafik på webben. I WebGL finns endast tre primitiva geometrier: punkter, linjer och trianglar. För att rita mer komplexa geometrier måste man utgå utifrån de tre grundläggande primitiverna. WebGL kan därför vara krävande att arbeta med och mycket kod krävs för att rita objekt. ThreeJS är därför fördelaktigt om man vill skapa innehåll i 3D utan att skriva särskilt mycket kod.

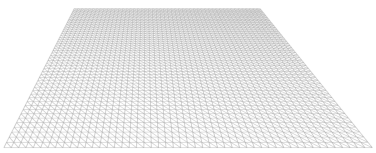
För att rita ett objekt i ThreeJS krävs en scen och en kamera. I scenen placeras alla objekt som ska ritas. Kamera-objektet definierar från vilken position och i vilken riktning 3D-miljön ska visas i. En *renderer* skapar därefter en bild av innehållet i scenen utifrån kameran, och ritar den i ett HTML canvas-element.

Ett plan kan enkelt skapas genom att definiera ett geometriobjekt av typen PlaneGeometry, se figur 3.13.

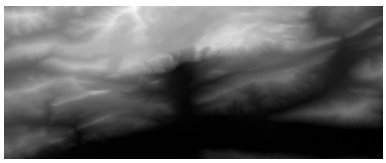
```
1    // skapa ett plan av storlek 10x10 med 50x50 segment
2    var planeGeometry new THREE.PlaneGeometry(10, 10, 50, 50);
3    var planeMaterial = new THREE.MeshPhongMaterial({wireframe: true});
4    var plane = new THREE.Mesh(planeGeometry, planeMaterial);
5    scene.add(plane);
6    renderer.render(scene, camera);
```

Figur 3.13: JavaScript-kod för att skapa och rita ett plant objekt i ThreeJS

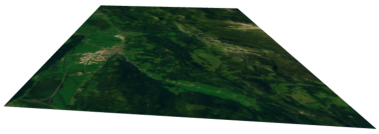
I ramverket består objekt av en geometri som beskriver objektets form samt ett material som beskriver dess textur och färg. På rad två i figur 3.13 skapas en plan geometri. Attributen i konstruktorn bestämmer planets utbredning och antal segment i x- och y-led. Därefter skapas ett material som beskriver hur geometrin ska ritas. Som attribut till materialet kan man även ha *map* som man tilldelar en bild, eller *texture*. Texturen draperas då ovanpå den geometri som är definierad. Ett annat attribut är *displacementMap*, som fungerar likt *map*- attributet, men istället för att ändra geometrins färg ändras höjderna för punkterna som utgör geometrins segment. Objektet, eller *Mesh*, skapas därefter utifrån materialet och geometrin. I figur 3.14(d) har en terrängbild samt höjdmodell i form av en bildfil 3.14(b) lagts till som attribut i objektets material.



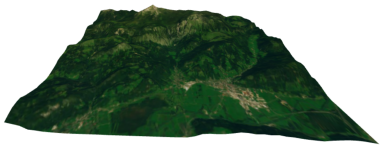
(a) En plan geometri



(b) Höjdmodell i rasterformat



(c) Plan med texture



(d) Plan med texture och höjddata (displacement map)

Figur 3.14: Visualiseringar i Three.js. (b) är hämtad från tjänsten Tanagram Heightmapper [4]

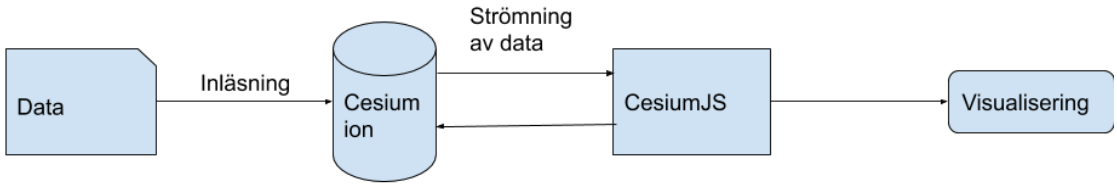
[4] https://tangrams.github.io/heightmapper/

3.6.2 Feature manipulation engine - FME

FME är en programvara från Safe Software för att bearbeta geografisk information. FME gör det möjligt att sätta ihop led av operationer vilket förenklar hanteringen av stora mängder data. Exempel på operationer man kan göra i FME är byta koordinatsystem, byta filformat samt strukturera om definitioner. FME stödjer över 450 olika format [Esri, uå; Safe, uå].

3.6.3 Cesium

Cesium är en plattform för visualisering av geografiska data i 3D på webben. Tjänsten lanserades 2011 med öppen källkod och var ursprungligen utvecklad av ett team inom Analytical Graphics Inc, men blev ett eget företag 2019. Cesium är uppdelat i ett JavaScript API, CesiumJS, samt Cesium ion, en plattform för uppladdning och lagring av data. Informationsflödet i en Cesiumapplikation uppbyggd med Cesium ion och CesiumJS kan ses i figur 3.15.



Figur 3.15: Informationsflödet i en Cesiumapplikation

CesiumJS

CesiumJS är ett öppet källkodat JavaScript-bibliotek som kan användas för att integrera data och skapa interaktiva kartor på webben. Med biblioteket går det att visualisera 3D-modeller med hög precision och kvalité. CesiumJS har licensen Apache 2.0.

Cesium ion

Cesium ion är en webbaserad plattform för att lagra geografiska data som lanserades 2018. Cesium ion är molnbaserad vilket medför att alla data som lagras effektivt kan strömmas till webbapplikationer som exempelvis CesiumJS. Tjänsten stödjer lagring av över 19 olika format bland annat CityGML, COLLADA och GeoTIFF. Data som laddas upp på plattformen konverteras till en av följande tillgångar *3D-tiles, Imagery, Terrain, glTF* eller *Native*.

I Cesium ion går det att lagra upp till 5GB data kostnadsfritt. Vill man ha ytterligare lagringsmängd krävs det en prenumeration där det tillkommer en månadskostnad. Vid upprättande av ett Cesium ion-konto ingår fyra kostnadsfria dataset. Tre av dataseten, *Bing Map Aerial, Bing Map Aerial with Labels, Bing Map Roads*, med typen *Imagery* är världstäckande bakgrundskartor. Det fjärde datasetet, *Cesium World Terrain*, är av typen *Terrian* och är världstäckande höjddata. Höjddata har enligt Cesium en upplösning på cirka 30m.

3.7 Versionshantering av stadsmodeller

Stadsmodeller är i behov av ständiga uppdateringar för att korrekt representera en stad i förändring. Det kan också vara aktuellt att uppdatera en modell om nya och bättre data finns tillgängliga. Vidare tar stadsplanerare och samhällsbyggare fram nya förslag på bebyggelser som representerar olika framtida scenarion för städer. I sådana fall är det önskvärt med system som hanterar de olika versionerna av en stadsmodell samt historiska versioner av staden för att kunna se förändringar över tid.

Vitalis et al. [2019] observerade att det är vanligare att kommuner och myndigheter väljer att skapa nya modeller från grunden istället för att arbeta vidare på befintliga stadsmodeller. Orsakande faktorer, enligt författarna, är bland annat svårigheterna med att arbeta i CityGML som format samt att det finns begränsat stöd för versionshantering av stadsmodeller.

I CityGML 2.0 finns det inget stöd för versionshantering, däremot är det möjligt att beskriva byggnaders tillstånd över tid genom attributen *creationDate* och *terminationDate*. I den kommande versionen CityGML 3.0 kommer en versionsmodul introduceras i vilken olika versioner kan beskrivas och länkas med olika stadsobjekt som tillhör specifika versioner [Kutzner et al., 2020]. IFC-standarden har dock i dagsläget inget stöd för olika versioner av BIM-modeller, men ett antal olika förslag på utökningar av standarden med stöd för versionshantering har presenterats [Chaturvedi et al., 2017].

Några exempel på nuvarande system för versionshantering av spatiala data är GeoGIG[5], som stöder data från Shapefiler, PostGIS och SpatiaLite. Därutöver finns utökningar av plattformen QGIS [6] som FastVersion och QGIS versioning plugin. Det är i praktiken möjligt att använda dessa system för att hantera olika versioner av stadsmodeller. Dock är det svårt att integrera detta med stadsmodeller som beskrivs i CityGML-modellen [Vitalis et al., 2019].

[5] `http://geogig.org/`
[6] `https://qgis.org/en/site/`

3.7.1 Versionshanteringssystem - VCS

Ett versionshanteringssystem (VCS) hanterar olika versioner av ett projekt. Ett VCS innehåller en datakatalog (*repository*) av alla versioner som skapats inom ett projekt och information om vem som skapat vad. Projektets aktuella och stabila version ligger vanligen i en *master*-gren. Från denna kan man bygga vidare på olika delar av projektet genom förgreningar (*branches*). Förgreningar är självständiga och påverkar inte master-grenen, vilket möjliggör experimentella ändringar utan att påverka projektets aktuella version. När man är nöjd med en gren kan man inkorporera den med master-grenen, är man inte nöjd kan man enkelt ta bort grenen från projektet.

Ett populärt VCS är *git*. git är ett decentraliserat system, och kräver därför inte, till skillnad från ett centralt system, att användaren är uppkopplad till internet. Användare kan därför arbeta i samma projekt parallellt i lokala kataloger av projektet. När en användare har gjort ändringar i ett projekt görs en *commit*. Commiten innehåller information om vem som gjort ändringen, vad som ändrats och en referens till den version av projektet ändringen bygger på [Spinellis, 2012; Vitalis et al., 2019].

Del III

Fallstudie

4. Bakgrund till fallstudien

I den här fallstudien jämförs två strategier för att visualisera stadsmodeller i CityJSON- samt CityGML-format på webben. Syftet med fallstudien är att undersöka för- respektive nackdelar med att arbeta med de två olika formaten. I den första strategin visualiseras stadsmodeller direkt i webbläsaren i CityJSON-format. I den andra strategin, konverteras stadsmodeller i CityGML till formatet 3D-tiles för att därefter strömmas från Cesium ion till en applikation byggd med CesiumJS.

För att utvärdera de två strategierna har ett antal användarfall identifierats och en kravställning arbetats fram. Dessa krav utgår delvis från IRIS-projektet där en lösning ska tas fram för att visualisera inkomna data som producerats inom Göteborg stad.

I den första delen av fallstudien, kapitel 5, presenteras användarfall och krav för visualiseringstjänsten som tagits fram. Data och studieområde som använts i studien beskrivs i kapitel 6 och programvaror i kapitel 7. I kapitel 8 presenteras fallstudiens metodik, här beskrivs utvecklingen av de två tjänsterna. Tjänsterna jämförs och utvärderas mot kravställningen. Resultaten presenteras i kapitel 9. I kapitel 10 diskuteras resultaten. Slutsatser presenteras avslutningsvis i kapitel 11.

5. Kravställning

För att kunna jämföra och utvärdera de två olika strategierna har ett antal krav för tjänsterna arbetats fram. Kraven har utarbetats från ett användarfall som utgår från en användare på en kommun som använder stadsmodeller för visualisering. Kraven har sedan delats in i två underkategorier: *producent* samt *användare*.

5.1 Användarfall - Visualisering av stadsmodeller

En anställd på en kommun vill visualisera hur ett pågående byggnadsprojekt av en bro smälter in i den befintliga bebyggelsen. Den anställda har en fil som innehåller bron samt en fil för närliggande bebyggelse. Båda filerna finns lagrade lokalt på den anställdas dator. Filerna laddas in i en webbtjänst och visualiseras i en navigerbar virtuell 3D-miljö.

Användaren vill ta reda på vilket byggnadsskede som brons olika delar är i. Genom en lista över objektets attribut klickar användaren på attributet ”byggnadsskede” och får upp en lista över unika värden som är kopplade till attributet. När användaren klickar på värdet ”färdig” syns en ruta med olika valbara färger. Användaren klickar på röd färg och samtliga objekt med attributet ”byggnadsskede” och värdet ”färdig” visas i rött.

I samband med visualisering upptäcker användaren att en del av brons färdiga grundkonstruktion inte har uppdaterats. Användaren klickar konstruktionsdelen och får upp en informationsruta med objektets attribut. Användaren redigerar attributvärdet för ”byggnadsskede” genom att klicka på attributet i informationsrutan och skriver in det nya värdet ”färdig”.

Användaren vill spara ner den redigerade filen som en ny version av den inladdade filen. Användaren klickar på en sparknapp och fyller i ett formulär med beskrivning av vem och vad som har ändrats. Den uppdaterade filen laddas sedan ner.

5.2 Kravställning

Kraven är indelade i två kategorier. Första kategorin, *producentkrav (P)*, är krav framtagna ur utvecklaren av tjänstens perspektiv. Den andra kategorin, *användarkrav (A)*, utgår ifrån användarens perspektiv. Några av kraven kan tillhöra båda kategorierna, men har i denna studie valts att endast ingå i en.

5.2.1 Producentkrav

P.1 **Georeferering**
Data som laddas in ska ha korrekt geografisk position.

P.2 **Bakgrundskarta**
Inlästa data ska kunna kompletteras med en georefererad bakgrundskarta.

P.3 **Markhöjd**
Det ska vara möjligt att visualisera markhöjd utifrån en höjdmodell.

P.4 **Objekt under marknivå**
Det ska vara möjligt att hantera och visualisera objekt som är lokaliserade under marknivå.

P.5 **Webbaserad**
Tjänsten ska kunna användas direkt i en webbläsare.

P.6 **Hantera ett eller flera objekt**
Det ska var möjligt att läsa in flera filer separat i tjänsten.

5.2.2 Användarkrav

A.1 **Stöd för lokala filer**
Användaren ska kunna visualisera lokala filer direkt i webbtjänsten.

A.2 **Snabb och effektiv**
Inläsning och visualisering av data ska ske snabbt och effektivt.

A.3 **Läsa attributdata**
Attribut för enskilda objekt ska kunna avläsas.

A.4 **Färgsättning utifrån attribut**
Det ska vara möjligt att färgsätta objekt utifrån attributdata.

A.5 **Redigera attribut**
Det ska var möjligt att redigera attribut som tillhör inladdade data.

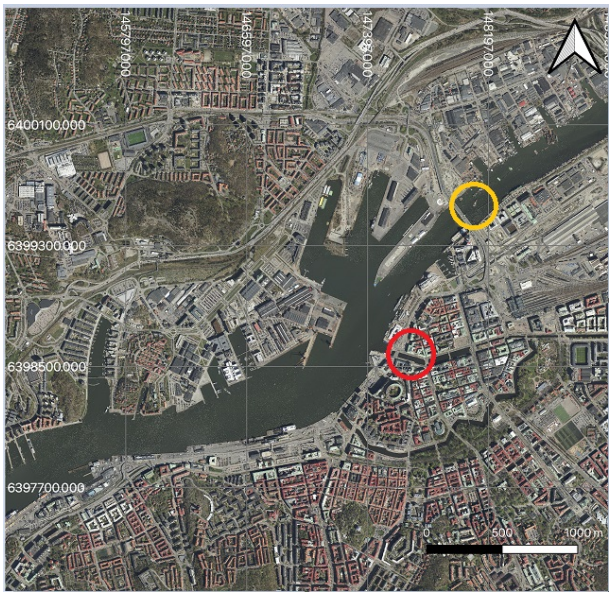
A.6 **Versionshantering**

Det ska finnas stöd för versionshantering av redigerade filer enligt versionshanteringssystemet git.

6. Studieområde och data

6.1 Studieområde

Fallstudien begränsas geografiskt till ett 4x4 km stort område i centrala Göteborg. Området definieras utifrån tre inkomna datamängder från Göteborgs stad inom IRIS projektet. I figur 6.1 illustreras det utpekade studieområdet. Den gula ovalen markerar ut Hisingsbron och den röd ovalen markerar ut Skeppsbron. Området har relativt platt terräng med en del kullar. Högsta höjdskillnaden i studieområdet är cirka 85m.



Figur 6.1: Ortofoto över studieområdet. Hisingsbron är markerat i gult och Skeppsbron rött.[1]

[1]Referenssystem: SWEREF 99 12 00

6.2 Beskrivning av data

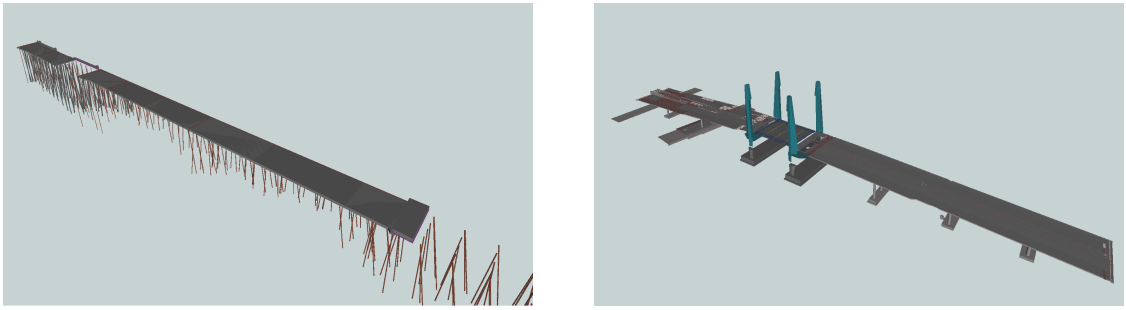
I fallstudien visualiseras tre modeller från IRIS-projektet: Skeppsbron, Hisingsbron samt befintliga byggnader i Göteborg. De tre modellerna är lagrade i Göteborgs lokala projektion SWEREF 99 12 00. Utöver de tre modellerna har höjddata från Lantmäteriet och ett högupplöst ortofoto över Göteborg använts.

Skeppsbron pålplan

Filen innehåller en 3D-modell av en pålplanskonstruktion under Skeppsbron, en kaj vid Göta älv i Göteborg. Filen är ursprungligen lagrad i IFC-format och har efter konvertering till CityGML storleken 66,3MB. CityGML-filen består av ett objekt av klassen Bridge som representerar hela bron. Bron består i sin tur av 978 objekt av underklassen till Bridge, BridgeConstructionElement, som beskrivs av enskilda geometrier. Modellen har ett höjdläge mellan -10 och +12 meter och visualiseras i figur 6.2a.

Hisingsbron

Filen innehåller en 3D-modell av Hisingsbron i Göteborg. Filen är ursprungligen lagrad i IFC- och DWG-format och är efter konvertering till CityGML 90,9MB stor. Filen består av ett Bridge-objekt och 5033 BridgeConstructionElement. Filen visualiseras i figur 6.2b.

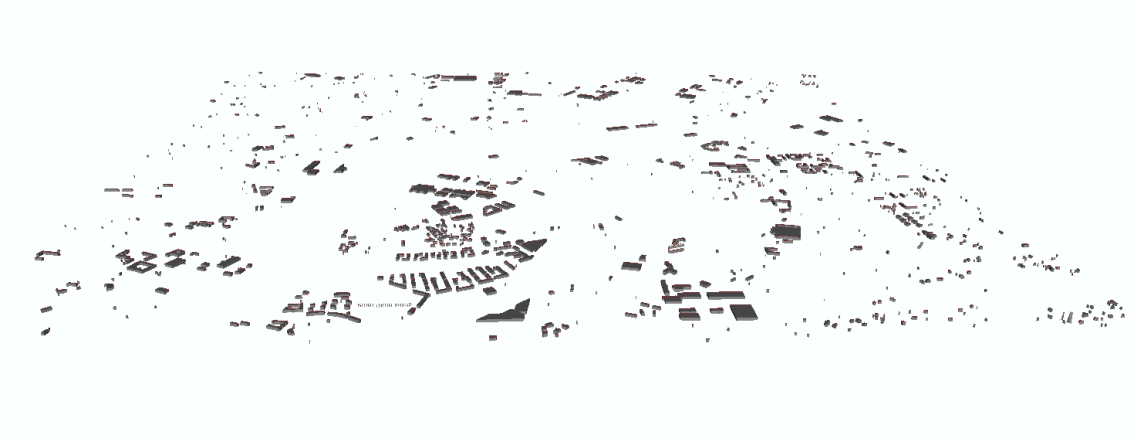


(a) Skeppsbron (b) Hisingsbron

Figur 6.2: Skeppsbron och Hisingsbron visualiserade i FME Data Inspector

Byggnader i Göteborg

Filen består av byggnader med LOD2 som överlappar området i figur 6.1. Modellen är i CityGML-format och är 5,0MB stor. Filen består av 1047 antal objekt av klassen Building. Filen visualiseras i figur 6.3.



Figur 6.3: Byggnader inom studieområdet visualiserade i FME Data Inspector

Konvertering

De tre dataseten beskrivna ovan har sitt ursprung från IFC- och DWG-format. För att konvertera om filerna till CityGML användes FME. I en serie FME-operationer hämtas geometrierna ut från IFC och DWG-filerna, trianguleras för att sedan mappas manuellt över till en förvald *Feature Role* i CityGML. I FME-skriptet hämtas även klassifiseringssystemet CoClass med API som adderas till CityGML-filen.

Övriga data

Ortofotot som används i fallstudie är i JPEG-format. Fotot har filstorleken 86MB, antal pixlar 20,000 x 20,000 och geografisk upplösning 0,2 meter.

I fallstudien användes även GSD-höjddata, grid 50+, från Lantmäteriet som är en rikstäckande höjdmodell (DEM) baserad på laserinskannad data med upplösning 50 meter. Höjdmodellen är indelad i 100x100 km indexrutor och hämtades i GeoTIFF-format från Lantmäteriets portal för öppna geodata. Inhämtade höjddata är angivna i Sveriges nationella höjdsystem RH2000.

7. Programvaror

7.1 Konvertering mellan CityGML och CityJSON

Applikationen Citygml-tools används för att konvertera filerna från CityGML till CityJSON. CityGML-tools är ett Command Line Interface-program (CLI) där användaren har möjlighet att konvertera mellan CityGML och CityJSON. Utöver det kan man ändra spatiala referenssystem, filtrera på LOD nivåer mm. Konverteringsprogrammet är licensierad under Apache 2.0, vilket innebär att användaren får använda, modifiera och vidareförmedla programvaran utan förbindelse om betalning.

7.2 Visualiseringsverktyg

För att visualisera data i CityJSON-format finns en webbaserad visualiseringstjänst. Applikationen är till stor del utvecklad i kursen Research Assignment vid Delfts Tekniska Universitet och har öppen källkod. Webbtjänsten är utvecklad i JavaScript-biblioteket Three.js.

Data visualiseras genom att användaren laddar in filen direkt i webbtjänsten. Användaren kan sedan rotera, zooma in och ut på visualiserade data. Användaren kan även tända/släcka samt få ut attribut på enskilda objekt.

Figur 7.1: Bild på visualiseringsverktyget för CityJSON

7.3 Validering

För att validera filer i CityJSON-format används Cjio. Cjio är ett python baserat CLI där användaren har möjlighet att sätta ihop led av operationer. I Cjio valideras CityJSON-filen mot inbyggda CityJSON-scheman i programmet. Utöver att validera filer i CityJSON-format kan man bland annat exportera filen i annat format samt sammanfoga två CityJSON-filer. Cjio har MIT licens, vilket innebär att man får ändra och dela programmet för kommersiellt eller icke kommersiellt bruk.

7.4 GDAL

GDAL är ett bibliotek med MIT/X licens för hantering av geografiska data framtagen av Open Source Geospatial Foundation. Genom kommandotolken kan man nå funktioner för att konvertera och processa vektor- och rasterdata i över 200 olika format. GDAL används i många av de stora programvarorna för geografiska data, som bland annat PostGIS, ArcGIS, Google Earth och QGIS. [GDAL, uå].

7.5 CityJSON versionshantering

Vitalis et al. [2019] presenterade ett förslag till ett versionssystem för att hantera uppdateringar i CityJSON. Förslaget bygger på versionshanteringssystemet *git* (se avsnitt 3.7).

Enligt förslaget lagras alla versioner av en stad i en vanlig CityJSON-fil som även innehåller nyckeln *versions*. I *versions* listas alla versioner av staden. En version är länkad till en lista innehållande de stadsobjekt som tillhör just den versionen. Under nyckeln *CityObjects* lagras alla unika stadsobjekt för samtliga versioner.

I figur 7.2 presenteras ett exempel på en versionsfil som innehåller en version, *version_1*. I den versionen finns två stadsobjekt, en byggnad och ett garage. Under nyckeln *versioning* beskrivs vem och vad som har lagts till i versionen. I *object* listas alla stadsobjekt, som är definierade i *CityObjects*, som är med i respektive version.

```
{
 "type": "CityJSON",
   ...
 "CityObjects": {
   "Byggnad" : {... },
   "Garage" : {... },
 },
 "versioning": {
   "versions": {
     "version_1": {
       "author": "John Doe",
       "message": "Första versionen av byggnaden",
       "objects": [
         "Byggnad",
         "Garage"
         ]
     }
   },
   "branches": {
     "master": "version_1"
   }
 },
 "vertices": [],
 ...
}
```

Figur 7.2: Exempel på en versionshanterad CityJSON-fil

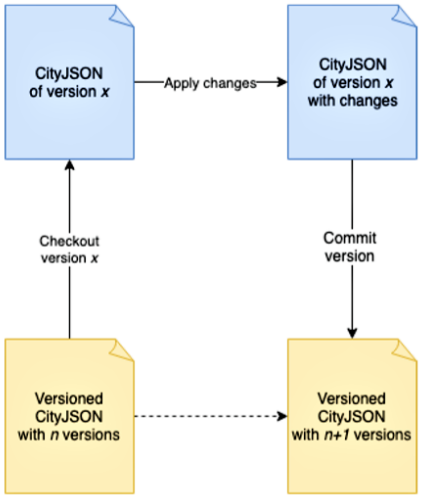
I figur 7.3 har byggnaden, som beskrivs i figur 7.2, renoverats. Den nya versionen har lagts till som ett eget stadsobjekt, *Byggnad-renoverad*. I versionsfilen beskrivs detta i "*version_2*. Den ursprungliga byggnaden har då ersatts av den nya, renoverade modellen. I *branches*" pekar nu den nuvarande versionen, (*master*), på den tillagda "*version_2*. Den tillagda versionen har även en referens till den version av modellen som den är härledd ifrån. Genom att följa dessa referenser kan man spåra alla ändringar ända till den ursprungliga modellen.

```
...
"CityObjects": {
"Byggnad" : {... },
"Garage" : {... },
"Byggnad-renoverad": {... }
},
...
"versioning": {
"versions": {
  "version_1": {
    "author": "",
    "message": "Första versionen av byggnaden",
    "objects": [
      "Byggnad",
      "Garage"
      ]
  },
  "version_2": {
    "author": "",
    "message": "Byggnaden renoverades",
    "parents": "version_1",
    "objects": [
      "Byggnad-renoverad",
      "Garage"
      ]
  }
}
}
...
 "branches": {
  "master": "version_2"
}
```

Figur 7.3: Byggnaden från figur 7.2 har ändrats och en ny version har lagts till

En prototyp av förslaget implementerades som ett CLI-program i Python. Programmet är publicerat och går att hämta på GitHub[1].

Programmet har fem kommandon: *log*, som returnerar en CityJSON-fils versionshistorik, *checkout*, som exporterar en given version som CityJSON, *diff*, visar skillnaderna mellan två versioner av en stadsmodell, *rehash*, som *hashar* identifierare samt *commit*, som lägger till en ny version av stadsmodellen i versionsfilen. Figur 7.4 visar hur versionshanteringen av en CityJSON-fil går till.



Figur 7.4: Förslag av system för versionshantering av CityJSON enligt Vitalis et al. (2019)

[1] https://github.com/tudelft3d/cityjson-versioning-prototype

8. Metod

I avsnitt 8.1 och 8.2 beskrivs hur tjänsterna har arbetats fram utifrån de två strategierna. Vid utveckling av applikationerna har data som beskrivs i avsnitt 6.2 använts.

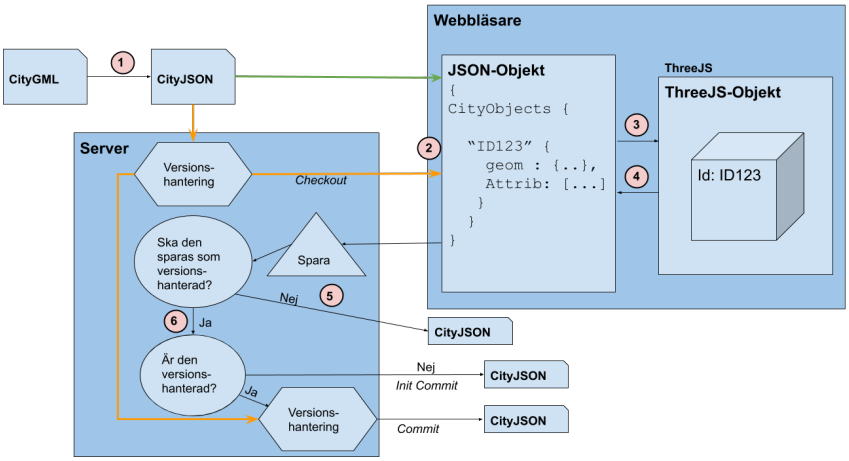
8.1 Strategi 1 - CityJSON

I första strategin visualiseras stadsmodeller i CityJSON-format direkt i en webbläsaren med hjälp av 3D-biblioteket ThreeJS. Informationsflödet sker enligt figur 8.1. I det första steget konverteras stadsmodellen från CityGML till CityJSON-format med applikationen CityGML-tools (1).

Genom en dra- och släppfunktion i tjänsten laddas CityJSON-filer in. Först kontrolleras om filen är versionshanterad, se orangea spåret, eller inte, se gröna spåret. Är filen versionshanterad hämtas endast den senaste versionen ut. Filen laddas sedan in i visaren och lagras i webbläsarens interna minne på klientsidan (2). I JavaScript hämtas koordinaterna för varje stadsobjekt från CityJSON-filen. Utifrån koordinaterna skapas för varje stadsobjekt geometrier i ThreeJS. Utifrån dessa skapas ett ThreeJS-objekt som tilldelas en unik identifierare som länkar samman den med motsvarande stadsobjekt i CityJSON (3).

ThreeJS-objektets unika identifierare används därefter för att hämta ut stadsobjektet från det lagrade JSON-objektet (4). Attributdata hämtas från JSON-objektet och visas i webbläsaren. Genom gränssnittet kan attributdata redigeras i webbläsaren vilket ändrar det lokalt lagrade JSON-objektet.

Vid ändring av attributdata kan den redigerade filen sparas ner på två olika sätt. Det första alternativet är att direkt spara ner den redigerade filen i CityJSON-format (5). Det andra alternativet är att spara den som en versionshanterad CityJSON-fil (6). Var filen versionshanterad vid inladdning så uppdateras den inladdade filen utifrån den redigerade versionen. Var den inte versionshanterad vid inläsning skapas en ny versionshanterad CityJSON-fil. Filen kan sedan hämtas av användaren.



Figur 8.1: Flödesschema - visualisering av stadsmodeller i CityJSON

8.1.1 Utveckling av visualiseringstjänsten

Motivering

För visualisering av stadsmodeller i CityJSON-format vidareutvecklas den befintliga visaren *CityJSON viewer* som beskrivs i avsnitt 7.2. Visaren valdes att vidareutvecklas eftersom det är den enda webbaserade visualiseringstjänsten med öppen källkod som hanterar CityJSON. Den har en effektiv lösning för visualisering i JavaScript-biblioteket ThreeJS som lämpar sig bra för JSON-format. Från CityJSON-filen kan geometrier hämtas ut direkt i JavaScript. Objekt i ThreeJS kan sedan skapas utifrån hämtade geometridata. Webbvisaren innehåller endast grundläggande funktioner vilket underlättar vid påbyggnad av koden då man får god överblick över applikationens struktur. Vidare är ThreeJS ett väldigt formbart bibliotek vilket möjliggör utformning av visaren enligt den framtagna kravställningen.

Beskrivning av applikationen

När applikationen laddas initialiseras en ThreeJS miljö och ett element som ThreeJS ritar till skapas i ett HTML-dokument. En händelsehanterare väntar på att en fil ska laddas in. Vid inläsning kontrolleras först att filen är i JSON-format. Därefter lagras filen i webbläsarens minne som en DOM-sträng. Strängen behandlas med en JSON-tolkare för att skapa ett JSON-objekt som lagras i ett JSON-lexikon. Alla inladdade filer lagras i detta lexikon med filnamn som nyckel och filens innehåll som värd

När filen är inladdad hämtas alla *vertices* från filen och lagras i en lista. För varje stadsobjekt hämtas dess *vertices* från listan utifrån varje stadobjekts *boundaries*, vilket generar en repre-

sentation av objektets geometri. Trianguleringsverktyget *earcut* [1] används för att arrangera alla *vertices* för att därefter kunna skapa en ThreeJS geometri av stadsobjektet. Därefter skapas en Three-Mesh som tilldelas ett material färglagt utifrån en förbestämd lista där CityJSON-objektens typ är länkad med en viss färg. Avslutningsvis läggs stadsobjekten till i en *scen* för att sedan ritas i HTML -taggen.

Konvertering

Data konverteras från CityGML till CityJSON med Citygml-tools som beskrivs i avsnitt 7.1. Kommandot som används är *to-cityjson filnamn.gml* och resultatet är motsvarande fil i CityJSON-format.

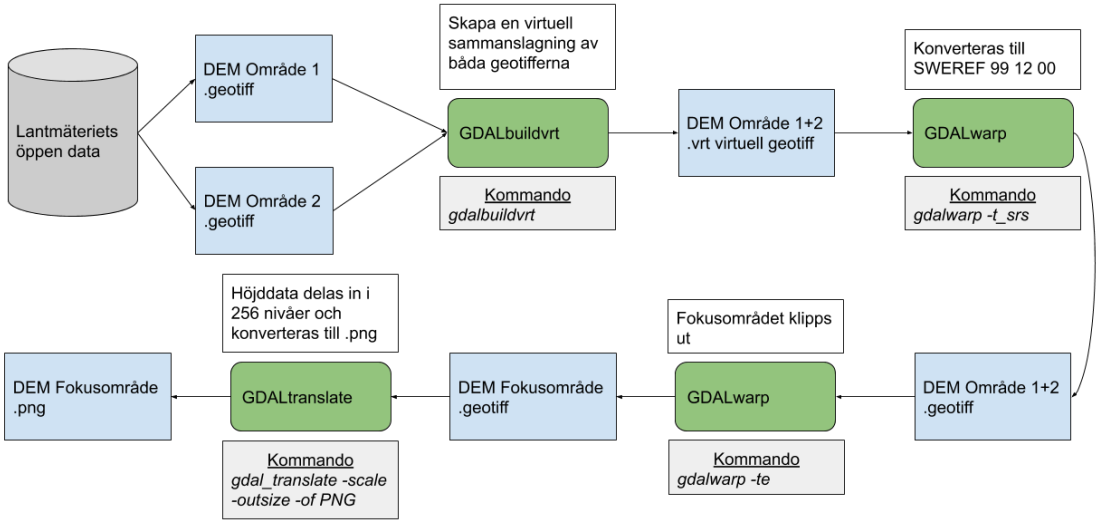
Georeferering

Det ursprungliga visualiseringsverktyget för CityJSON läste samtliga koordinater i filen och normaliserade dessa, vilket resulterade i att objekten placerades i origo och som högst hade längden ett i verktygets referenssystem. När flera objekt laddades in separat i visaren överlappades dessa och fick oproportionella skalor i förhållande till varandra. För att möjliggöra inläsning av flera filer separat används istället objektens verkliga, dvs, icke-normaliserade koordinater. Objekten har således samma koordinater i visarens referenssystem som i dess projektion. Samma princip gäller även för objektens höjddata.

Höjddata och bakgrundskarta

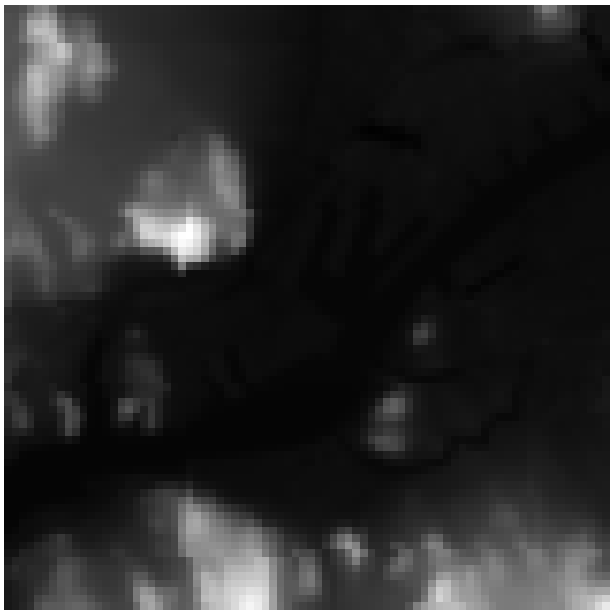
Lantmäteriets höjddata levereras i GeoTIFF-format indelade i olika indexrutor. Eftersom fokusområdet för den här fallstudien faller inom två indexrutor används verktyget GDAL för att hämta ut rätt område, se figur 8.2. Med *GDALbuildvrt* sammanfogas de två filerna till ett virtuellt dataset med täckning över hela fokusområdet. Med *GDALwarp* konverteras det virtuella datasetet från SWEREF 99 TM till SWEREF 99 12 00 för att överensstämma med de stadsmodeller från IRIS-projektet som används.

[1] `https://github.com/mapbox/earcut`



Figur 8.2: Framtagning av höjddata till bakgrundskartan

Fokusområdet klipps därefter ut med *gdalwarp* och konverteras till en PNG-fil i gråskala där färgintensiteten motsvarar höjddata, se figur 8.3. Vid denna konvertering används *GDALtranslate* där även höjddata skalas till ett heltalsvärde mellan 0 och 255, där 255 innebär hög färgintensitet. Eftersom höjdskillnaden mellan lägsta och högsta punkterna i vårt område är 84,578m blir den största värdefelet vid konvertering 0,3m.



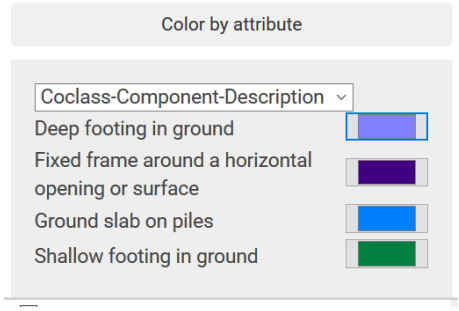
Figur 8.3: Höjdmodell i PNG-format över studieområdet

I webbvisaren skapas ett plant rutnät som motsvarar höjdmodellens geografiska utsträckning. Rutnätet skapas i ThreeJS med ett PlaneGeometry-objekt. I visaren skapas även ett materialobjekt av typen *MeshPhongMaterial*. Höjddata tilldelas materialobjektet genom att ladda in PNG-filen som en *displacementMap*.

Som bakgrundskarta används ortofotot över fokusområdet. Eftersom ortofotot har en storlek på 86MB reduceras bilden för att effektivisera inläsning i applikationen. Efter reduceringen har ortofotot storleken 3,5MB, storleken 2400 x 2400 och geografisk upplösning 1,6m. Ortofotot sparas som en *texture* och tilldelas materialobjektet. Avslutningsvis skapas en *mesh* med materialobjektet och rutplanet.

Färgsättning utifrån attributdata

ThreeJS-objekt kan färgsättas i scenen utifrån objektens attribut. Vid inläsning traverseras alla objekt och alla unika attribut lagras i ett *set*, som är en samling med unika värden. En lista skapas från *setet* och genom ett gränssnitt kan alla tillgängliga attribut väljas. När ett attribut är valt från listan traverseras alla objekt ytterligare en gång och alla unika attributvärden tillhörande det valda attributet hämtas ut. Attributvärdena visas sedan i en lista tillsammans med en färgväljare, se figur 8.4. Vid förändring av attributets färgsättning i färgväljaren, uppdateras stilsättningen av alla ThreeJS-objekt som har samma unika identifierare som ett JSON-objekt med det valda attributet samt värdet. Stilsättningen sker således endast i visaren och uppdaterar inget värde i JSON-objektet i den inlästa filen.



Figur 8.4: Funktion för färgsättning utifrån attributdata i CityJSON-applikationen

Redigering av attribut

Attribut kan hämtas ut från ThreeJS-objekt med musklick genom funktionen *ray cast*. I funktionen beräknas först musens position utifrån kamerans projektion och orientering. Från musens position skapas sedan en linje med riktning in mot scenen och en kontroll görs för vilka av objektens ytor som korsar linjen. Den första ytan som korsar linjen tillhör det ThreeJS-objekt som har klickats på. Utifrån den unika identifieraren i ThreeJS-objektet hämtas attribut med attributvärde ut från JSON-objektet med motsvarade identifierare i den inlästa filen. Resultatet redovisas i en lista i gränssnittet. Attributvärdet kan redigeras genom att klicka på det existerande värdet.

Vid inmatning av ett nytt värde uppdateras objektens attributvärde i listan och lokalt lagrade JSON-objektet.

Versionshantering

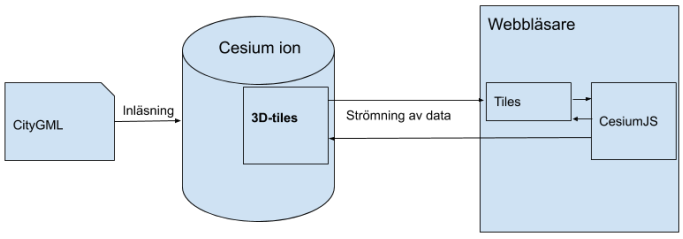
I applikationen kan både vanliga och versionshanterade CityJSON-filer laddas in, se avsnitt 7.5. Inladdade filer kan exporteras som vanlig eller versionshanterad CityJSON-fil.

Vid inladdning av en versionsfil, skickas filen till en lokal webb-server genom en POST-förfrågan. I ett PHP-skript skrivs data till en fil och en CityJSON av den senaste versionen hämtas ut genom kommandot *checkout*. Klienten hämtar därefter filen genom en GET-förfrågan och den senaste versionen av stadsmodellen laddas in i tjänsten. Det globala JSON-objekt som innehåller inladdade data uppdateras när objekt redigeras genom tjänsten. Ska filen sparas som en ny version skickas den uppdaterade versionen till servern. Genom kommandot *commit* sparas ändringarna som en ny version i den versionsfil som tidigare laddades in. Filen kan därefter hämtas genom dess URL. Om inladdade data är i CityJSON, utan versionshantering, skapas istället en ny versionsfil på servern genom kommandot *init commit*.

8.2 Strategi 2 - CityGML-Cesium

I strategi två utvecklas applikationen med hjälp av plattformen Cesium. I figur 8.5 presenteras dataflödet för applikationen. I processens första steg laddas CityGML-data upp på Cesium ion där det konverteras till 3D-Tiles.

En stadsmodell strömmas till klienten som en Batched 3D-model. En *tile* av modellen hämtas med en GET-förfrågan vid behov. *Tilen* som hämtas består av en metadata-del (*header*) samt dess huvuddel (*body*). Attributdata lagras binärt i en Batch Table. En Batch Table består av en del i JSON som specificerar vilka attribut som lagras, samt antingen värdet till dessa attribut eller en referens till tabellens binära del. Värden är kopplade till geometrier med hjälp av identifierare (*batchID*).



Figur 8.5: Dataflödet för visualisering av CityGML med en applikation byggd i Cesium

8.2.1 Utveckling av visualiseringsverktyg

Motivering

Cesium valdes som verktyg i fallstudien då det används i Tyréns processförslag för visualisering i IRIS-projektet. Cesium är ett etablerat verktyg för visualisering av 3D-modeller, vilket medför att det finns bra stöd och instruktioner på webben. Cesium stödjer konvertering av CityGML genom Cesium ion för visualisering.

Beskrivning av applikationen

Visualiseringen sker i en applikation utvecklad i JavaScript med biblioteket CesiumJS. Applikationen länkas samman med Cesium ion genom en unik tillgångstoken. Genom den upprättade länken kan uppladdade data hämtas till applikationen med en URL som innehåller objektens unika identifierare. Data som hämtas sparas i ett *viewer-objekt*, huvudkomponenten i CesiumJS som innehåller en virtuell jordglob, som ritas upp i ett -element.

Konvertering

Objekt som ska visualiseras lagras först i Cesium ion. Objekten är ursprungligen i CityGML-format och konverteras vid lagring i Cesium ion till 3D-tiles av typen Batched 3D Model. När data läggs in i Cesium ion kan man välja om objekten ska projiceras parallellt med marknivån eller med sitt lagrade höjdvärde. Ska objekten projiceras parallellt med marken väljs vilken terrängdata av typen Terrain som ska användas som marknivå.

Georeferering

Objekt som laddas in i Cesium ion och visualiseras i CesiumJS har korrekt geografisk positionering. Detta gäller även när flera objekt laddas in separat och har olika ursprungliga koordinatsystem. Detta tyder på att alla objekt konverteras automatiskt till ett enhetligt koordinatsystemet i något steg av processen.

Höjddata och bakgrundskarta

För att jämföra spåren används höjddata över studieområdet som togs fram i avsnitt 8.1.1. Data laddas in i Cesium ion och konverteras till typen Terrain, vilket är en typ av *tileset* innehållande DEM i olika detaljnivåer. I CesiumJS skapas ett objekt av typen CesiumTerrainProvider som har åtkomst till inladdade terrängdata via en länk till Cesium ion.

Som bakgrundskarta används ortofotot över studieområdet. Fotot laddas in i Cesium ion där det konverteras från JPEG-format till datatypen Imagery. Ortofotot laddas in genom ett SingleTileImageryProvider-objekt (ett *tileset* med endast en *tile*) och positioneras över fokusområdet. Ortofotot placeras ovanpå Cesiums ursprungliga bakgrundskarta.

Färgsättning utifrån attributdata

Som beskrivet i avsnitt 8.2, lagras inlästa data lokalt i binärt och JSON-format. *Header* som är i JSON-format pekar på var attributvärdet är lagrat i *batch-tabellen* i binärt-format. Utifrån tabellen kan objektens färg redigeras med Cesiums inbyggda språk för stilsättning. Stilsättningen sker genom JSON-objekt, se 8.6.

```
{
    "show" : " ${Area} > 0",
    "color" : {
        "conditions" : [
            [" ${Height} < 60", "color('#13293D')"],
            [" ${Height} < 90", "color('#1B98E0')"],
            [" ${Height} < 120", "color('#1424E0')"],
            ["true", "color('#E8F1F2', 0.5)"]
        ]
    }
}
```

Figur 8.6: Exempel på stilsättning i CesiumJS baserad på höjdattribut

Redigering av attribut

Det är inte möjligt att redigera stadsmodeller genom plattformen Cesium. Detta diskuteras vidare i avsnitt 10.1.

Versionshantering

Det är inte möjligt att implementera ett system för versionshantering i plattformen Cesium. Detta diskuteras vidare i avsnitt 10.1.

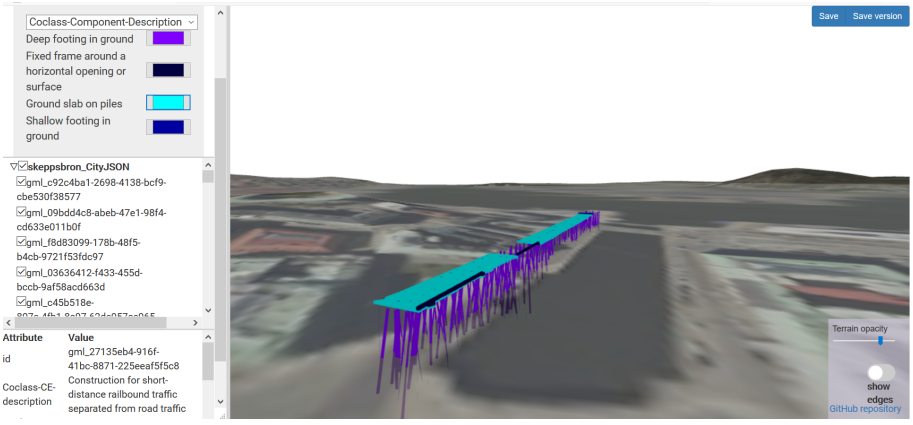
9. Resultat

9.1 Applikationerna

Nedan presenteras de två applikationerna som har utvecklats i fallstudien. Källkod för applikationen finns att hämta på `https://github.com/PatrikSylve/visualisering-av-stadsmodeller`.

9.1.1 CityJSON

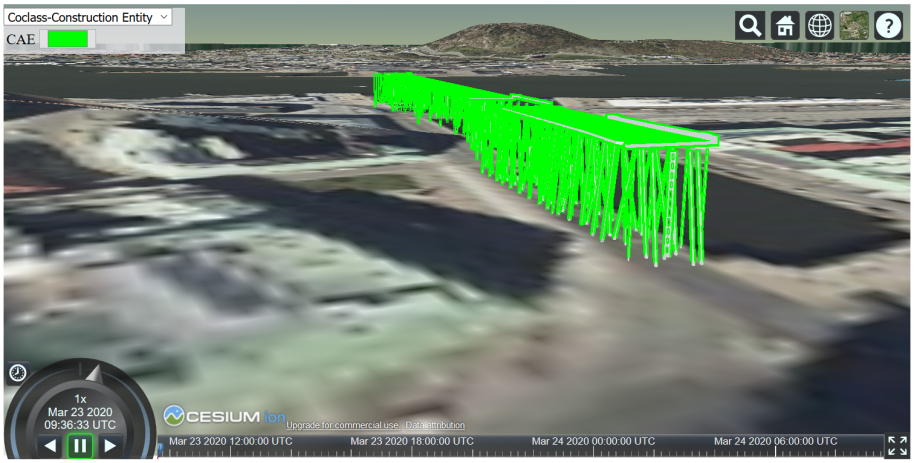
I figur 9.1 visas en skärmdump från den utvecklade tjänsten för CityJSON. I bilden har skeppsbron laddats in och färgsatts utifrån attribut. I den vänstra panelen kan användaren välja attribut att färgsätta utifrån. Där finns även en lista över olika objekt i stadsmodellen tillsammans med kryssrutor som tänder och släcker objektet. Längst ner i panelen visas ett markerat objekts attribut och attributvärde. Attributvärden kan redigeras genom att klicka på värdet och fylla redigera texten. Uppe i det högra hörnet kan filen sparas ner som antingen vanlig eller en versionshanterad fil i CityJSON-format. Längst ner till höger finns kontroller för att styra markens transparens samt en knapp som fyller i objektens konturer.



Figur 9.1: Utvecklad applikation för CityJSON. Skeppsbron är färgsatt utifrån attributdata

9.1.2 CityGML-Cesium

Tjänsten utvecklad med hjälp av Cesium visas i figur 9.2. Högst upp i vänstra hörnet kan användaren välja attribut att färgsätta utifrån. I bilden är skeppsbron färgsatt utifrån attribut. Tabell över attributdata visas genom att klicka på ett objekt, se figur 9.3.



Figur 9.2: Applikation för CityGML i Cesium. Skeppsbron är färgsatt utifrån attributdata

06a9f027-7b3f-4675-8f5d-c61a899c8a4e	
Coclass-CE-description	Construction for short-distance railbound traffic separated from road traffic
Coclass-Construction Entity	CAE
Height	22.836366246842946
Latitude	57.705944361831186
Longitude	11.960821688961497
end_date	2020-01-01
gml:name	skeppsbron
start_date	2018-01-01
state	bygghandling
TerrainHeight	40.350735776844814

Figur 9.3: Attributtabell för skeppsbron i Cesium-tjänsten

9.2 Kravställning

De två strategierna har utvärderats mot kravställningen som tagits fram i avsnitt 5. I avsnitt 9.2.1 och 9.2.2 presenterats resultaten utifrån ett producent- respektive användarperspektiv.

9.2.1 Producent

Kravet för georefering uppfylls i båda applikationerna. I CityJSON projiceras objekten i ett koordinatsystem med X, Y och Z-axlar. I Cesium skapas en virtuell glob med referenssystemet WGS84. Inlästa objekt projiceras utifrån den virtuella globen.

Cesium tillhandahåller ortofoton från öppna dataset som visualiseras automatiskt vid skapandet av en glob. Egna data kan läggas till i Cesium ion i bland annat GeoTiff- eller PNG-format. Raster som inte är georefererade, exempelvis PNG-filer, kan mappas ovanpå ett georefererat plan. I CityJSON-tjänsten tillhandahålls inga förinställda bakgrundsdata. Användaren måste lägga till egna data i något bildformat (PNG, JPEG etc.).

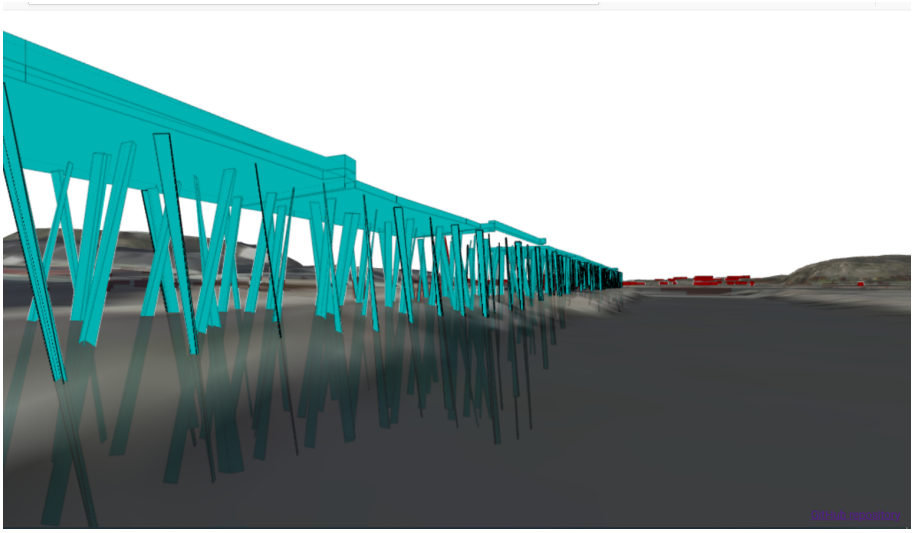
I båda spåren går det att ladda in egna höjddata. I den första strategin, CityJSON, måste inlästa data vara en DEM i PNG-format. I det andra spåret laddas höjddata i GeoTIFF-format in i Cesium ion.

Visualisering av objekt under mark stöds inte i Cesium. Vid inläsning av data i Cesium ion kan objekt placeras ovanpå terräng eller med sin lagrade höjddata. Det är inte möjligt att se igenom eller flytta kameran under marken. I CityJSON placeras objekt utifrån deras definierade höjdlägen. Objekt kan ses under mark genom att flytta kameran eller göra marken transparent, se figur 9.4.

Båda strategierna är webbaserade och hanterar inläsning av flera objekt från olika filer. I tabell 9.1 sammanställs resultaten utifrån producentkraven.

Tabell 9.1: Sammanställning av resultat utifrån producentkraven

Kravmatris - Producent		
Krav	CityJSON	CityGML
P1 - Georeferering	X	X
P2 - Bakgrundskarta	X	X
P3 - Markhöjd	X	X
P4 - Objekt under marknivå	X	-
P5 - Webbaserad	X	X
P6 - Läsa in flera objekt	X	X



Figur 9.4: Krav P4 objekt under mark. Skeppsbron visualiserad i CityJSON-applikationen

9.2.2 Användare

I visualiseringstjänsten framtagen för CityJSON kan lokala filer läsas in med en dra och släpp-funktion. I Cesium måste data först laddas in i Cesium ion. Applikationen har således inget stöd för inläsning av lokala filer. Inläsningen sker snabbt och effektivt för de två strategierna.

Vid inläsning av bromodellerna i Cesium-applikationen är det endast huvudobjektens (Bridge) attributdata som kan läsas och färgsättas utifrån. Eftersom krav A3 och A4 inte uppfylls för underobjekten till Bridge, anser vi att kraven endast är delvis uppfyllda.

Redigering av attribut kan endast utföras i applikationen för CityJSON. I Cesiumtjänsten finns det endast stöd för att redigera metadata. Det sista kravet *Versionshantering* uppfylls endast av CityJSON-strategin.

I tabell 9.2 har resultaten sammanställts utifrån användarkraven.

Tabell 9.2: Sammanställning av resultat utifrån användarkraven

Kravmatris - Användare		
Krav	CityJSON	CityGML
A1 - Stöd för lokala filer	X	-
A2 - Snabb och effektiv	X	X
A3 - Läsa attributdata	X	(X)
A4 - Färgsättning utifrån attribut	X	(X)
A5 - Redigering av attribut	X	-
A6 - Versionshantering	X	-

9.3 Filstorlek

I tabell 9.3 sammanställs stadsmodellernas filstorlek före och efter konvertering från CityGML och CityJSON.

Tabell 9.3: Sammanställning av filstorlek i CityJSON- och CityGML-format

Filstorlek		
Fil	CityJSON (MB)	CityGML (MB)
Skeppsbron pålplan	6,6	66,3
Hisingsbron	7,3	90,9
Göteborg byggnader	1,0	5,0

Del IV

Diskussion och slutsatser

10. Diskussion

10.1 Utvärdering utifrån kravställning

Utifrån kravställningens resultat framkommer att visualiseringsstrategin för CityJSON uppfyller alla producent- samt användarkrav. I det andra spåret, där CityGML visualiseras genom plattformen Cesium uppfylls fem av sex producentkrav och tre av sex användarkrav varav två endast uppfylls delvis. I följande avsnitt diskuteras respektive krav.

P1 - Georefering

Båda strategierna uppfyller kravet för georefering. I det egenutvecklade verktyget för CityJSON måste inlästa data vara i samma koordinatsystem som bakgrundskarta för att georefereras korrekt. Detta gäller även när två filer läses in separat till visaren. Plattformen Cesium har löst detta problem genom att automatiskt konvertera alla data till samma koordinatsystem. Detta medför att användaren inte behöver ta hänsyn till vilket koordinatsystem data är lagrade i.

P2 - Bakgrundskarta

I de två strategierna läses ortofoto över studieområdet in i webbläsaren vid start. Ortofotot läggs ovanpå ett georefererat plan som en *texture*. Ursprungliga ortofotot hade en filstorlek på 86MB men komprimerades till 3,5MB på grund av prestandaproblem. Metoden, som används i de båda strategierna, är därför inte optimal för att visualisera data med hög upplösning. I Cesium ion är det möjligt att konvertera rasterdata till TMS- eller WTMS-lager för att därefter ladda in data genom *imagery providers* i CesiumJS. Data måste då vara georefererad i exempelvis GeoTIFF-format.

P3 - Markhöjd

I båda strategierna uppfylls kravet för att visualisera markhöjd utifrån en höjdmodell. I CityJSON måste höjddata vara DEM i PNG-format. Cesium ion stödjer också PNG-format men kräver att filen är georeferad. Plattformen stödjer även GeoTIFF vilket är formatet som Lantmäteriets höjddata levererades i. Resultatet blev således detsamma i båda strategierna.

P4 - Objekt under marknivå

Producentkravet som inte uppfylls genom plattformen Cesium är *Objekt under marknivå*. Cesium har ingen inbyggd funktion för att visualisera objekt under marknivå, det är heller inte möjligt att flytta kameran under jordytan. Men i och med att CesiumJS har öppen källkod skulle det vara möjligt att utveckla en sådan funktion. Eftersom en sådan lösning kräver en ändring i källkoden bedömdes kravet som icke uppfyllt. I CityJSON-applikationen implementerades en funktion som ökar transparensen på marknivån. Objekt under mark kan således visualiseras.

P5 - Webbaserad

Båda tjänsterna är webbaserade. En fördel med CityJSON-tjänsten är att data inte behöver överföras mellan klienten och en server för att kunna visualiseras. Nackdelen med att lagra alla data i det lokala minnet är dock att mängden data som kan visualiseras är begränsad. I Cesium är man begränsad till att använda deras plattform vilket exempelvis kan medföra en kostnad om man vill utöka lagringsmängden i Cesium ion.

P6 - Hantera ett eller flera objekt

Båda strategierna hanterar inläsning av flera objekt från separata filer.

A1 - Stöd för lokala filer

Cesiumplattformen har inget stöd för att läsa in lokala filer direkt i webbläsaren då data istället strömmas från Cesium ion. Detta problem skulle eventuellt kunna lösas genom att använda Cesium ions REST API. Inlästa data måste då först lagras i en *Amazon S3 bucket*. Från lagringen kan data sedan läsas in till Cesium ion, för att sedan returnera en unik identifierare på lagrade data. Utifrån den unika identifieraren kan objekten läsas in i CesiumJS. Problematiken med denna lösning är att det tar lång tid att konvertera data från CityGML till 3D-Tiles, vilket inte skulle vara hållbart i praktiken. Då implementeringen av lösningen är omfattande och kräver flera led som resulterar i en ohållbar lösning i praktiken bedömdes kravet som icke uppfyllt.

A2 - Snabb och effektiv

Båda strategierna visualiserar data snabbt och effektivt. Från att applikationen startar till att inladdade data visualiseras är i båda fallen under tio sekunder. Däremot är det större skillnader när data konverteras från CityGML till CityJSON respektive 3D-tiles. För skeppsbron tog konverteringen från CityGML till CityJSON cirka tio sekunder, motsvarande konvertering till 3D-tiles tog runt 80 sekunder.

A3 - Läsa attributdata

Som nämnt i avsnitt 9.2.2 är det endast huvudobjektets attributdata som går att avläsa. Detta beror med stor sannolikhet på hur data konverteras till 3D-tiles. I fallstudien testades även

att i skeppsbron-modellen konvertera varje *BridgeConstructionElement* i CityGML-filen till typen *Bridge*, dvs att varje komponent blev ett eget objekt. Det gjorde att alla attribut tillhörande respektive objekt gick att utläsa korrekt, dock innebär det att man frångår informationsmodellen CityGML. Det medförde även att de objekt som vid inläsning till Cesium ion bestämdes att projiceras mot markytan får en ojämn konstruktion på grund av att alla de olika delarna följer marknivån, se figur 10.1. Att kunna avläsa attributvärden för underobjekt är en viktig egenskap för att exempelvis kunna studera hur olika komponenter är klassificerade. Därav bedömdes kravet som endast delvis uppfyllt med plattformen Cesium.

Figur 10.1: Felaktig projicering av skeppsbron i Cesium när BridgeConstructionElement konverteras till Bridge

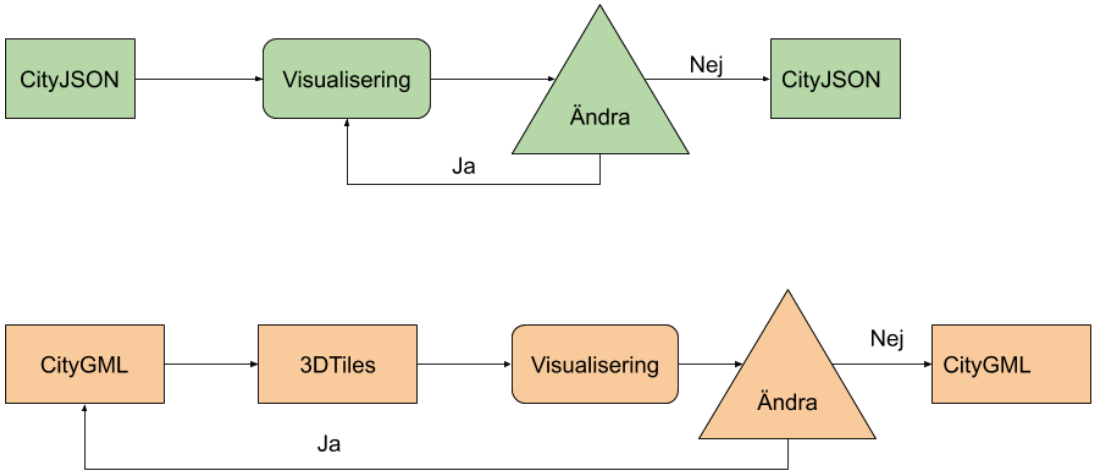
A4 - Färgsättning utifrån attributdata

Enligt samma princip som i avsnitt 10.1 kan underobjekt inte färgsättas enskilt utifrån dess attributvärde i Cesium. Däremot är det möjligt att färgsätta övriga objekt utifrån attribut. Därför anses kravet delvis uppfyllt.

A5 - Redigering av attribut

I plattformen Cesium är att data lagrad separat från klienten. Det tar bort möjligheterna att implementera en funktion för redigering av attribut på klientsidan, likt visualiseringsstrategin för CityJSON. Redigering av attributdata skulle således behöva ske i Cesium ion där data lagras. Men detta är något som plattformen inte stödjer. Det är i dagsläget endast möjligt att via Cesium ions REST API redigera metadata. För att ändra en stadsmodell med plattformen Cesium krävs det

att man går tillbaka och redigerar den ursprungliga CityGML-filen. Den redigerade CityGML-filen måste sedan konverteras och laddas in i Cesium ion för att sedan visualiseras på nytt. Eftersom CityJSON lagrar alla data lokalt i JSON-format går det att redigera data direkt i webbläsaren för att sedan exportera filen som en ny CityJSON. Det krävs alltså inte att man måste gå tillbaka något steg i processen, vilket illustreras i figur 10.2.



Figur 10.2: Process för visualisering och redigering i de två applikationerna för CityJSON och CityGML.

A6 - Versionshantering

Vid inladdning av en versionsfil visualiseras endast den senaste versionen som är lagrad i filen. Tjänsten kan vidareutvecklas genom att låta användaren skifta mellan olika versioner lagrade i en versionsfil. Man hade då kunnat visualisera en stads utveckling genom tid, eller jämföra olika alternativa utvecklingar. Systemet är dock endast experimentellt och inget stöd finns i specifikationerna för CityJSON.

I Cesium-tjänsten implementerades inget versionshanteringssystem. Det beror dels på att det i dagsläget finns begränsat stöd för versionshantering av CityGML-filer, och inget stöd finns heller i informationsmodellen. En förutsättning är också att det går att ändra och spara data i en CityGML-fil, vilket inte är möjligt genom Cesium-tjänsten.

10.2 Sammanställning och generell diskussion

Att bygga en applikation för visualisering med plattformen Cesium var smidigt. Plattformen har tydliga instruktioner hur data laddas upp till Cesium ion och hämtas i en CesiumJS-applikation. Med Cesium ion tillkommer även världstäckande bakgrundskarta samt höjddata.

Även om det är fördelaktigt att arbeta med en etablerad plattform som Cesium då projekt går snabbt att sätta upp tillkommer en del problematik. Det främsta problemet som uppkom under fallstudien var kopplat till hur data strömmas med 3D-tiles från Cesium ion. Vid konvertering förlorades viss information, som exempelvis avsaknaden av attribut i undergrupper.

Att arbeta med ThreeJS och formatet CityJSON medförde en betydligt större flexibilitet vilket man tydligt kan se då strategin uppfyllde alla ställda krav. Att arbeta med data som är lagrade lokalt direkt i lagringsformatet gjorde det möjlig att bland annat redigera data direkt utan att gå tillbaka något steg i processen. Strategin var däremot mer tidskrävande att implementera då de flesta funktioner behövdes byggas från grunden.

För att visualisera en stadsmodell i Cesium krävs det att användaren har tillgång till Cesium ion för att lagra data. Därefter krävs det att användaren har tillgång till källkoden för att lägga in objekten för visualisering. Detta understryker att användaren av en sådan tjänst måste ha kunskap inom programmering vilket i många fall inte är en självklarhet. Med den motsvarande strategin för CityJSON är det en tydligare indelning mellan utvecklare och användare vilket öppnar upp möjligheterna att nå ut till en bredare användargrupp.

10.2.1 Format och filstorlek

Under fallstudien uppstod problem med hur en korrekt definierad CityGML-fil ska vara uppbyggd. Biljecki et al. [2016] validerade 37 dataset från olika källor och visade att samtliga filer var inkorrekta. Det är svårt att bygga CityGML-tolkare för att hantera och hämta information. Rouault [2014] visade att det är möjligt att definiera en rektangel i GML på 25 olika sätt, alla giltiga enligt GML-standarden. En tolkare som helt stöder formatet måste hantera alla dessa fall. I CityGML förekommer främst objekt i tre dimensioner, vilket innebär ännu flera möjliga variationer för att definiera geometrier [Ledoux et al., 2019].

CityJSON var i förhållandevis till CityGML betydligt lättare att få en korrekt definierad fil. Vilket bland annat beror på att geometrier endast kan beskrivas på ett sätt. Vidare är det enkelt att tolka filer i JSON-format, vilket gör det lättare att utveckla applikation som stödjer formatet.

I CityJSON-format har Skeppsbron pålplan en filstorlek som är en tiondel av motsvarande data i CityGML. En förklaring till att filen är mer kompakt än förväntade 6 gånger, kan vara att bron har hög detaljnivå med många upprepande koordinater. Indexsystemet som används i CityJSON är effektivt då koordinater endast lagras en gång och refereras till genom index.

10.2.2 Visualiseringsstrategier

I avsnitt 3.5.2 presenteras tre strategier som på olika sätt hanterade problematiken med att visualisera stadsmodeller i CityGML på webben. Gaillard et al. [2015] använde en metod som strömmade data likt Cesium med 3D-tiles, en metod som visade sig vara effektiv för att hantera stora mängder data. JSON valdes som överföringsformat i den nämnda studien på grund av komprimeringsmöjligheter och enklare hantering i JavaScript. Denna fördel med JSON-formatet märktes även i denna studie. Det var enkelt att bygga vidare på applikationen genom implementering av funktionerna färgsättning av attribut och versionhantering.

Den strategi som presenteras av Rodrigues et al. [2013] utgår från data i CityGML-format och bygger på att semantiska data hanteras separat från geometriska data. Semantiska data har stor vikt vid olika typer av analyser, som exempelvis beräkningar av solcellspotential. Cesium verkar inte fullt ut hantera det problemet med 3D-tiles. Istället krävs en liknande lösning med en WFS för att hämta ytterligare data för vissa typer av analyser. I CityJSON-tjänsten är detta problem löst då alla data lagras i webbläsaren.

11. Slutsatser

Ett syfte med rapporten är att utreda olika metoder för visualisering av stadsmodeller på webben. Studien visade att en vanlig metod för visualisering av CityGML är att konvertera data till ett visualiseringsformat. Vid en konvertering uppstår informationsförluster. För att hantera denna förlust kan förlorade data hämtas med hjälp av webbtjänster. En annan lösning är att bitvis strömma data till en applikation vid behov med hjälp av ett strömningsformat. I studien valdes den etablerade plattformen Cesium ut för att studeras närmare då den kombinerar två metoder, konvertering till visualiseringsformat samt strömning. CityJSON-formatet möjliggör genom dess kompakta struktur en annan metod för visualisering. Stadsmodeller i formatet kan laddas in och lagras lokalt hos klienten utan behov av konvertering till annat format.

Studiens andra syfte är att utvärdera metoder för visualisering av CityGML genom plattformen Cesium samt egenutvecklad tjänst för CityJSON.Metoden för visualisering av CityJSON implementerades helt på klientsidan och förlitar sig lite på extern kod. Det innebär större kontroll över applikationen men att många funktioner måste implementeras från grunden. Genom att arbeta med ett etablerat verktyg som Cesium finns mer kod tillgänglig vilket minskar tiden för utveckling av en applikation. Däremot medför det en mindre flexibilitet vilket kan ses i utvärderingen av studiens kravställning.

Vid konvertering från CityGML till 3D-tiles förlorades viktig information som krävs för att kunna visualisera utifrån attributdata. Däremot möjliggör Cesiums strömningslösning visualisering av större mängd data. Eftersom data lagras på klientsidan i CityJSON-applikationen, finns istället begränsningar i mängden data som kan visualiseras. Däremot krävs ingen konvertering av formatet och det sker således ingen informationsförlust.

Filstorleken mellan motsvarande filer i vår studie är upp till tio gånger mer kompakt i CityJSON i jämförelse med CityGML-format. Detta innebär effektivare lagring och överföring av data. Lokal lagring av CityJSON-data i applikationen öppnar upp möjligheterna till att redigera och sedan spara data. Motsvarande är inte möjligt i Cesium på grund av att data lagras externt i ett strömningsformat. Redigeringen måste således ske i ursprungliga CityGML-filen som sedan måste konverteras till 3D-tiles på nytt.

Rapportens tredje syfte är att undersöka om CityJSON är ett lämpligt alternativ till XML-implementeringen av CityGML 2.0 för användning på webben. Studien visar att JSON-formatet har stora fördelar i jämförelse med XML. I takt med ökad användning av GIS-applikationer på webben är CityJSON ett intressant komplement till CityGML.

Litteraturförteckning

[Autodesk, uå] Autodesk (uå). *Vad är DWG?* Tillgänglig: `https://www.autodesk.se/products/dwg`. [2019-12-05].

[Biljecki et al., 2018] Biljecki, F., Kavisha, K., and Nagel, C. (2018). CityGML Application Domain Extension (ADE): overview of developments. *Open Geospatial Data, Software and Standards*, 3(1):1–17.

[Biljecki et al., 2015] Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3D city models: State of the art review. *Internationaljournal of Geo-Information*, 4(4):2842–2889.

[Billen et al., 2014] Billen, R., Cutting-Decelle, A. F., Marina, O., Almeida, J. P., Caglioni, M., Falcuet, G., Leduc, T., Metral, C., Moreau, G., Perret, J., Jose, R. S., Yatskiv, Y., and Zlatanova, S. (2014). 3d city models and urban information: Current issues and perspectives.

[Bostock, 2018] Bostock, M. (2018). *TopoJSON*. Tillgänglig: https://github.com/topojson/topojson. [2019-12-10].

[buildingSMART, 2019] buildingSMART (2019). *Industry Foundation Classes*. Tillgänglig: `https://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/`. [2019-12-05].

[buildingSMART, 2019] buildingSMART (2019). *Industry Foundation Classes (IFC) - An Introduction*. Tillgänglig: https://technical.buildingsmart.org/standards/ifc/. [2019-12-05].

[Butler et al., 2016] Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., and Schaub, T. (2016). *The GeoJSON format*. Tillgänglig: `https://tools.ietf.org/html/rfc7946#section-3.1`. ISSN: 2070-1721, [2019-12-10].

[Cesium, uå] Cesium (uå). *3D Tile Format Specification*. Tillgänglig: https://github.com/AnalyticalGraphicsInc/3d-tiles/tree/master/specification. [2020-01-30].

[Chaturvedi et al., 2017] Chaturvedi, K., Smyth, C. S., Gesquière, G., Kutzner, T., and Kolbe, T. H. (2017). *Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML*. Lecture Notes in Geoinformation and Cartography. Springer. ss.191-206.

[CityJSON, 2019] CityJSON (2019). *Extensions.* Tillgänglig: `https://www.cityjson.org/extensions/`. [2019-12-05].

[CityJSON, uå] CityJSON (uå). *A JSON-based encoding for 3D city models.* Tillgänglig: `https://www.cityjson.org/`. [2019-12-05].

[CityJSON, uå] CityJSON (uå). *What is CityJSON?* Tillgänglig: `https://www.cityjson.org/about/`. [2019-12-05].

[El-Mekawy et al., 2014] El-Mekawy, M., Paasch, J. M., and Paulsson, J. (2014). Integration of 3d cadastre, 3d property formation and bim in sweden. In *Proceedings of the 4th International FIG 3D Cadastre Workshop, November 2014, Dubai, UAE : ss.17-34.*

[Esri, uå] Esri (uå). *FME® - Konvertera, transformera och förädla geografisk information.* Tillgänglig: `http://www.esri.se/Produkter/ovriga-produkter/fme`. [2019-12-05].

[Europakonventionen, 2015] Europakonventionen (2015). *State of the Energy Union - questions and answers.* Tillgänglig: `https://ec.europa.eu/commission/presscorner/detail/en/MEMO_15_6106`. [2019-12-05].

[Gaillard et al., 2015] Gaillard, J., Vienne, A., Baume, R., Pedrinis, F., Peytavie, A., and Gesquière, G. (2015). Urban data visualisation in a web browser. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15. Association for Computing Machinery. ss.81–88.

[GDAL, uå] GDAL (uå). *Software using GDAL.* Tillgänglig: `https://gdal.org/software_using_gdal.html#software-using-gdal`. [2020-01-05].

[Gemalto, 2020] Gemalto (2020). Tillgänglig: `https://www.gemalto.com/iot/inspired/smart-cities`. [2019-03-11].

[Geodatarådet, 2016] Geodatarådet (2016). *Utvecklad samverkan för öppna och användbara geodata via tjänster.*

[Gröger and Plümer, 2012] Gröger, G. and Plümer, L. (2012). Citygml – interoperable semantic 3d city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71:12 – 33.

[IRIS - Smart Cities, 2019a] IRIS - Smart Cities (2019a). Tillgänglig: `https://www.irissmartcities.eu/content/context`. [2019-03-10].

[IRIS - Smart Cities, 2019b] IRIS - Smart Cities (2019b). *THE 5 IRIS TRANSITION TRACKS .* Tillgänglig: `https://www.irissmartcities.eu/content/5-iris-transition-tracks`. [2020-02-05].

[Johansson and Lithén, 2016] Johansson, M. R. and Lithén, T. (2016). *Ramverk för Nationella geodata i 3D.*

[Kutzner et al., 2020] Kutzner, T., Chatuverdi, K., and Kolbe, T. H. (2020). *CityGML 3.0: New Functions Open Up New Applications.*

[Lafrance et al., 2019] Lafrance, F., Daniel, S., and Dragićević, S. (2019). Multidimensional web gis approach for citizen participation on urban evolution. *ISPRS International Journal of Geo-Information*, 8(6):253.

[Lake, uå] Lake, R. (uå). *Introduction to GML - Geography Markup Language*. Tillgänglig: https://www.w3.org/Mobile/posdep/GMLIntroduction.html.

[Lantmäteriet, 2019a] Lantmäteriet (2019a). *Nationella Specifikationer*. Tillgänglig: `https://www.lantmateriet.se/sv/Om-Lantmateriet/Samverkan-med-andra/lantmateriet---utvecklingsmyndighet-for-samhallsbyggnadsprocessen/nationella-specifikationer/`. [2019-12-05].

[Lantmäteriet, 2019b] Lantmäteriet (2019b). *Nationellt Tillgängliggörande av Geodata i Samhällsbyggnadsprocessen*.

[Ledoux, 2019] Ledoux, H. (2019). *CityJSON Specifications 1.0.1*. Tillgänglig: https://www.cityjson.org/specs/1.0.1/. [2019-11-05].

[Ledoux et al., 2019] Ledoux, H., Ohori, K. A., Kumar, K., Dukai, B., Labetski, A., and Vitalis, S. (2019). CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*. 4(4).

[Liu et al., 2018] Liu, Y., Mateo-Babiano, I. M.-B., and Darchen, S. (2018). [2020-01-05].

[OGC, 2007] OGC (2007). *Geography Markup Language (GML) Encoding Standard 3.2.1*.

[OGC, 2012] OGC (2012). *OGC City Geography Markup Language (CityGML) Encoding Standard* .

[OGC, 2015] OGC (2015). *OGC KML 2.3*. Tillgänglig: http://docs.opengeospatial.org/is/12-007r2/12-007r2.html. [2020-12-19].

[OGC, 2019] OGC (2019). Tillgänglig: `https://www.opengeospatial.org/about`. [2020-01-11].

[Ohori et al., 2018] Ohori, K. A., Biljecki, F., Kumar, K., Ledoux, H., and Stoter, J. (2018). Modelling cities and landscapes in 3d with citygml. *Building Information Modeling: Technology Foundations and Industry Practice*. Springer, Cham.

[Olsson et al., 2018] Olsson, P.-O., Axelsson, J., Hooper, M., and Lars Harrie, L. (2018). Automation of building permission by integration of bim and geospatial data. *ISPRS International Journal of Geo-Information*, 7(8).

[Open Design Alliance, 2019] Open Design Alliance (2019). *Open Design Specification for .dwg files Version 5.4.1*. `https://www.opendesign.com/files/guestdownloads/OpenDesign_Specification_for_.dwg_files.pdf`.

[Persson and Lithén, 2016] Persson, C.-G. and Lithén, T. (2016). Teknisk rapport 2016:4 - i gränslandet bim-gis- geodesi.

[Regeringen, 2018] Regeringen (2018). *Regeringen utökar satsningen på en smartare samhällsbyggnadsprocess*. Tillgänglig: https://www.regeringen.se/pressmeddelanden/2018/04/regeringen-utokar-satsningen-pa-en-smartare-samhallsbyggnadsprocess/. [2020-02-05].

[Rodrigues et al., 2013] Rodrigues, J. I. J., Figueiredo, M. J. G., and Costa, C. P. (Jul 2013). Web3dgis for city models with citygml and x3d. pages 384–388. IEEE.

[Safe, uå] Safe (uå). *What is data conversion*. Tillgänglig: https://www.safe.com/what-is/data-conversion/. [2019-12-05].

[Scan2CAD, 2016] Scan2CAD (2016). *The DWG File Specification*. Tillgänglig: https://www.scan2cad.com/dwg/file-spec/. [2019-12-05].

[SKL, 2019] SKL (2019). *Svensk geoprocess*. Tillgänglig: https://skl.se/samhallsplaneringinfrastruktur/planerabyggabo/geodatalantmateri/svenskgeoprocess.4215.html. [2019-12-05].

[Spinellis, 2012] Spinellis, D. (2012). Git. *IEEE Software*, 29(3):100–101.

[Sun et al., 2019a] Sun, J., Mi, S., Olsson, P.-o., Paulsson, J., and Harrie, L. (2019a). Utilizing bim and gis for representation and visualization of 3d cadastre. *ISPRS International Journal of Geo-Information*, 8(11).

[Sun et al., 2019b] Sun, J., Olsson, P., Eriksson, H., and Harrie, L. (2019b). Evaluating the geometric aspects of integrating bim data into city models. *Journal of Spatial Science*, 0(0):1–21.

[Svensk geoprocess, 2016] Svensk geoprocess (2016). *Geodataspecifikation Byggnad*. https://www.lantmateriet.se/contentassets/e4b0b7cb457045c7bd3ce794e625b154/sgp_geodataspecifikation_byggnad_v3.0test1.pdf. [2019-12-05].

[Target, 2017] Target, S. (2017). The rise and rise of json. Tillgänglig: https://twobithistory.org/2017/09/21/the-rise-and-rise-of-json.html. [2020-03-26].

[Vitalis et al., 2019] Vitalis, S., Labetski, A., Ohori, K. A., Ledoux, H., and Stoter, J. (2019). A data structure to incorporate versioning in 3d city models. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W8:123–130.

[W3C, 2015] W3C (2015). *XML Essentials*. Tillgänglig: https://www.w3.org/standards/xml/core. [2019-12-15].

[W3C, 2016] W3C (2016). *XML*. Tillgänglig: https://www.w3.org/XML/. [2019-12-02].

[W3School, uå] W3School (uå). *JSON Data Types*. Tillgänglig: https://www.w3schools.com/js/js_json_datatypes.asp. [2019-12-9].

[Wilkinson et al., 2016] Wilkinson, M., Dumontier, M., and Aalbersberg, I. e. a. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Sci data 3*.

Institutionen av naturgeografi och ekosystemvetenskap, Lunds Universitet

Student-examensarbete (seminarieuppsatser) i geografisk informationsteknik.

Uppsatserna finns tillgängliga på institutionens geobibliotek, Sölvegatan 12, 223 62 LUND. Serien startade 2010. Hela listan och själva uppsatserna är även tillgängliga på LUP student papers och via Geobiblioteket (www.geobib.lu.se).

Serie examensarbete i geografisk informationsteknik

1. Patrik Carlsson och Ulrik Nilsson (2010) Tredimensionella GIS vid fastighetsförvaltning

2. Karin Ekman och Anna Felleson (2010) Att välja grundläggande karttjänst - Utveckling av jämförelsemodell och testverktyg för utvärdering

3. Jakob Mattsson (2011) Synkronisering av vägdatabaser med KML och GeoRSS - En fallstudie i Trafikverkets verksamhet

4. Patrik Andersson and Anders Jürisoo (2011) Effective use of open source GIS in rural planning in South Africa

5. Nariman Emamian och Martin Fredriksson (2012) Visualiserig av bygglovsärenden med hjälp av Open Source-verktyg - En undersökning kring hur man kan effektivisera ärendehantering med hjälp av en webbapplikation

6. Gustav Ekstedt and Torkel Endoff (2012) Design and Development of a Mobile GIS Application for Municipal FieldWork

7. Karl Söderberg (2012) Smartphones and 3D Augmented Reality for disaster management - A study of smartphones ability to visualise 3D objects in augmented reality to aid emergency workers in disaster management

8. Viktoria Strömberg (2012) Volymberäkning i samhällsbyggnadsprojekt

9. Daniel Persson (2013) Lagring och webbaserad visualisering av 3D stadsmodeller - En pilotstudie i Kristianstad kommun

10. Lisette Danebjer och Magdalena Nyberg (2013) Utbyte av geodata - studie av leveransstrukturer enligt Sveriges kommuner och landstings objekttypskatalog

11. Alexander Quist (2013) Undersökning och utveckling av ett mobilt GISsystem för kommunal verksamhet

12. Nariman Emamian (2014) Visning av geotekniska provborrningar i en webbmiljö

13. Martin Fredriksson (2014) Integrering av BIM och GIS med spatiala databaser – En prestandaanalys

14. Niklas Krave (2014) Utveckling av en visualiseringsapplikation för solinstrålningsdata

15. Magdalena Nyberg (2015) Designing a generic user interface for distribution of open geodata: based on FME server technology

16. Anna Larsson (2015) Samredovisning av BIM- och GIS-data

17. Anton Lundkvist (2015) Development of a WEB GI System for Disaster Management

18. Ellen Walleij (2015) Mapping in Agricultural Development – Introducing GIS at a smallholders farmers' cooperative in Malawi

19. Frida Christiansson (2016) Lagring av 3D - geodata - en fallstudie i Malmö Stad

20. Lisette Danebjer (2016) Methodology for creating and modifying distributed topologically structured geographical datasets

21. Jeanette Dunn Ekelund (2016) En jämförelse av algoritmer och resultat för flödesberäkning i QGIS/GRASS och ArcGIS

22. Ebba Gröndahl och Frida Thorman (2016) Verksamheters optimala läge i staden och hur de är lokaliserade idag

23. Gunnar Rolander (2017) Data transformation using linked data ontologies

24. Måns Andersson och Moa Eklöf (2017) Stilsättning av geografiska data

25. Josefine Axelsson (2018) Automatisering av bygglovsansökningsprocessen med stöd av BIM och GIS

26. Leonard B. O. Berge (2018) Uppdatering och visualisering av stadsmodell med stöd av konverterade BIM-modeller

27. Rickard Ingesson & Gabriella Olsson (2019) Publicering av geografiska data på webben : En utvärdering av programsystem med fokus på öppen källkod

28. Alfred Hildingson & Patrik Sylve (2020) Visualisering av stadsmodeller på webben : Jämförande studie mellan CityGML och CityJSON