VirtualOulu: Collaborative, Immersive and Extensible 3D City Model on the Web

Toni Alatalo* Timo Koskela Matti Pouke Paula Alavesa Timo Ojala
Center for Ubiquitous Computing
University of Oulu, Finland

Figure 1: *One half of the picture is from the virtual model and the other half a photograph — which way?*

Abstract

In this paper, we describe the creation of a photorealistic digital 3D representation of a real world city and its subsequent publication as an open and collaborative 3D virtual world on the web using an open source software platform. We present the guidelines and conventions used in the collaborative development process of the model. We report the design and implementation of the web user interface of the model that exploits on dynamic on demand (un)loading of assets during navigation to reduce memory consumption on user device. We demonstrate the extensibility of the model with example applications. We report an empirical performance evaluation of the web user interface in terms of download latencies and memory consumption, and frame rate achieved with three different types of user devices during navigation. We identify a rendering bottleneck in the current implementation and present a candidate solution for fixing it.

Keywords: virtual city model, 3D web interface, Tundra, WebTundra, open source

Concepts: •**Applied computing** → **Cartography; Computer games;** •**Computing methodologies** → **Image manipulation;** *Computational photography;*

1 Introduction

In this paper we introduce VirtualOulu, the 3D virtual city model of Oulu, Finland (Figure 2). VirtualOulu is published on the web as a persistent, collaborative and immersive 3D virtual environment using an open source game engine. We offer VirtualOulu as an open and extensible general purpose platform for developing new applications for such web-based collaborative and immersive virtual city models. In our ongoing research we regard VirtualOulu as a "mirror world" [Gelernter 1991] that together with the corresponding real world constitutes a "hybrid reality".

3D data from cities have been gathered and utilized in interactive computerized visualizations since the 1990's [Arponen 2002]. Virtual city models have gained lots of attention since mid 2000's when their semi-automatic construction at city-scale became feasible (e.g. [Döllner et al. 2006]). The need and application scenarios for such city scale models are well documented (e.g. [Bourdakis 2001; Horne et al. 2014]). Carrozzino *et al.* [2009] listed urban planning, communication, emergency management, architecture, documentation, social networking and education as examples. Döllner *et al.* [2006] stated that the benefits of 3D city models can be found in the fields of telecommunication, disaster management, homeland security, facility management, real estate portals, logistics, as well as entertainment and education. A recent extensive literature review on the applications of 3D city models identified at least 29 different use cases and over 100 different applications [Biljecki et al. 2015].

Carrozzino *et al.* [2009] described various technical requirements that differ slightly according to which purpose a city model is used. Urban planning requires building models equipped with semantic data and spatial-volumetric dimensions that correspond accurately to their real-world counterparts. Architectural models have typically more intense visual requirements in the form of visual details and realistic lighting. Social applications alternatively might have more loose requirements in spatial accuracy, but high demands for aesthetic visual quality for engaging a strong sense of presence [Slater and Wilbur 1997; Carrozzino et al. 2009]. It has also been observed that high visual quality makes it easier for navigating in virtual cities [Wallet et al. 2011]. Besides the capability to render visually high quality building models with realistic lighting, modern game engines contain many other properties that are useful for immersive applications, such as skeletal animations for humanoid avatars, particle systems for visual effects and synchronization between multiple simultaneous users. This has made game engines an attractive option for certain types of city visualizations.

CityGML [Kolbe et al. 2005; Kolbe 2009] is a popular contemporary format for representing 3D city models. Since CityGML supports object semantics and data queries, it is a suitable for many

*e-mail:toni.alatalo@ee.oulu.fi

Web3D '16,, July 22-24, 2016, Anaheim , CA, USA
ISBN: 978-1-4503-4428-9/16/07
DOI: http://dx.doi.org/10.1145/2945292.2945305

(a) *Bird's eye view with very light models without textures*

(b) *Street level with avatars present*

Figure 2: *VirtualOulu*

urban planning applications. In terms of the visual quality of the building models, CityGML defines a 5-level system for describing the level of detail (LOD0-4) of a building model. More recently, Biljecki *et al.* [2014] proposed a system of 10 discrete LOD levels for the visual quality of city models. Many large-scale city models such as 3D Berlin [Döllner et al. 2006] and Rotterdam 3D [1] are available in the web in CityGML format. However, immersive applications rendering virtual locations at pedestrian level using game engines do not usually use CityGML as such. This is due to the fact that native game engines currently lack support for CityGML. Consequently, the CityGML data has to be separately converted for applications utilizing game engines. In addition, the verbose XML format used by CityGML makes the file size of 3D graphics with high LODs rather large for web publishing which means that some kind of conversion process is usually required for web viewers as well.

Interactive 3D applications, such as those attributed to contemporary game engines, fall in somewhat mid-level when it comes to the complexity of the 3D models. On one hand, the building models found in 3D map applications such as Google Earth, are typically somewhat rough low-polygon models with inconsistent lighting. On the other hand, the CAD and BIM models utilized in architecture, engineering and construction are typically too detailed and complex for interactive applications. When generating models for interactive applications, the complexity of the polygon meshes should be kept to minimum. [Bourdakis 2001; Palmquist and Shaw 2008]

So far, the process of generating virtual cities consisting of buildings with reasonable visual quality has not been completely automated. While methods such as laser scanning and photogrammetry can be utilized for geometry and texture construction, large amounts of manual modeling work with applications such as Blender, 3Ds Max or Maya is often required. However, bridging the gap between visually coarse but spatially accurate visualizations and immersive, visually aesthetic applications is seen as a relevant problem (e.g. [Virtanen et al. 2015; Heo et al. 2013]).

The large-scale of virtual city models has inspired their collaborative modeling and crowdsourcing. For example, Palmquist and Shaw [2008] reported the collaborative modeling of a city model by students that was steered with high-level modeling guidelines. Crowdsourcing of 3D models can be considered as a particular type of collaborative modeling that typically has less control over the quality and the consistency of the resulting models [Maher et al. 2011]. Uden and Zipf [2013] as well as Goetz and Zipf [2013] discuss large scale collaborative modeling in crowdsourcing 3D building databases for OpenStreetMap. In turn, The Trimble SketchUp (formerly Google Sketchup) was developed for acquiring crowdsourced 3D building data for Google Earth. Finally, examples of non-collaborative generation of urban 3D models can be found in [Sahin et al. 2012] and [Sheng et al. 2013].

The rest of the paper is organized as follows. Section 2 reports the creation of VirtualOulu as a collaborative modeling process involving university researchers, companies and city planners, including detailed modeling guidelines and the division of modeling labor at city block level. Section 3 describes the publication of VirtualOulu on the web as an immersive and collaborative virtual space using open source software, including the optimization of memory and rendering resources, the web user interface of the city model and example applications developed using the model. Section 4 presents an empirical performance evaluation of VirtualOulu with respect to download latencies, memory consumption, and rendering speed achieved on three different types of user devices. Section 5 concludes the paper with a discussion.

2 Creation of VirtualOulu

2.1 Motivation and design objectives

Our overarching motivation to create VirtualOulu is to create a "mirror world" [Gelernter 1991], a detailed large-scale digital representation of the real world downtown Oulu as a persistent, collaborative and immersive 3D virtual space. Google Earth and Microsoft Visual Earth are well-known examples of such mirror worlds that have proven very useful for a wide range of application domains [Roush 2007]. Such digital metaworlds are forecast to become an integral part of our reality in the near future (e.g. [Ricci et al. 2015]). We refer to the combination of a real world space and its detailed mirror world realized as a persistent and collaborative virtual 3D space constitute as the "hybrid reality". We do not assume that reality and virtuality would be coupled in a particular strictly defined way or according to a particular taxonomy. In other words, virtuality is not necessarily blended with reality into a 'single' augmented or mixed reality experience, although we do consider such couplings, as well. Instead, the two parallel but distinct realities can be complete unto themselves with their distinct interfaces, but whose interplay via some mechanism (e.g. a game or a sensor network) serving as a communication channel between the two realities yields a hybrid reality, where the two realities are enriched by their ability to mutually influence one another. Lifton and Paradiso [2009] referred to such bi-directional coupling of reality and virtuality as "dual reality". Prior visions of coupling a real city and its virtual model include the "visual city" [Hudson-Smith 2007] and the "humane city" [Streitz 2011] but neither of them ever matured into a persistent realization that would have been accessi-

[1] www.rotterdam.nl/rotterdam_3d

ble to the general public. In our long-term vision VirtualOulu will eventually develop into a persistent and collaborative virtual city and a living lab for the co-creation of virtual and hybrid reality applications by users, city planners, researchers and businesses.

This meta objective is pursued with the following design objectives for VirtualOulu:

Open and collaborative creation process: A city is constantly living as buildings are demolished and new ones built. Various parties are responsible for the different areas, for example different architect companies make the plans for different new buildings. Furthermore alternative usage needs require different features: some applications need traffic data, whereas in others it is irrelevant but for example correct shadows from sunlight are needed instead. Therefore we strive to create an open and collaborative process where the base city model can be updated by anyone, and it serves as an open platform for anyone to add functionality.

Real-world realism at street level: To serve as a mirror city the virtual model has to match the real city and be accessible at street level similar to contemporary first-person games, not only from a bird's eye view typical to most virtual 3D city models (Figure 1). That is why we have matched it with the official terrain height map provided by the national land survey institute, and scanned the streets by land laser scanning to have the correct building dimensions with good accuracy. This should facilitate for example AR applications where the modelled geometry information can support the feature recognition and matching process. Correct dimensions support also simulation, for example to check if a certain snow ploughing vehicle can fit to make a turn in a corner where some change is planned, or acoustic simulations to determine noise levels. Visually the model must be realistic to be recognizable and fitting to blend with the real world in AR applications. Graphics should be good quality for a pleasing experience in social and commercial applications.

Web-based user interface: we have chosen web as the primary user interface for VirtualOulu so that general public can use it anywhere in the Internet with web-enabled devices without having to install any plugins or native viewer applications. However, we do have plans for developing interfaces for more specialized devices such as AR viewers.

2.2 Collaborative modeling process

The acquisition of the digital source materials needed in modeling involved (1) photographing the facades of all buildings, (2) digitalizing the blueprints of buildings' facades and floor plans, and (3) terrestrial laser scanning of 3D surfaces at open areas. The source materials were used in a typical manual 3D modeling process. In the first phase, a 3x3 blocks area totaling nine city blocks was modeled by a single company for a reference. In the second phase, the model was expanded to the current size of 30 blocks by three companies working in parallel. One reason for using multiple companies in the modelling process was to simply divide the workload. Similar to [Palmquist and Shaw 2008], we wanted to develop and study the best practises how to create large models as a collaborative effort. The idea is that in the future any party can add new parts to the city, or update some building or area that has changed. This is similar to how Google Earth started as a collaborative effort where anyone could upload models which, after review for correctness, then show on that virtual globe. The difference here is that the collaborators were creating a larger continuous area together, with a LOD suitable for immersive real-time 3D applications. The main difference in comparison to [Palmquist and Shaw 2008], is that in our work, the modeling effort was carried out by commercial modeling companies instead of students. Also all the models are licensed under Creative Commons allowing any kind of use for free to best facilitate application development. The modeling process has been steered by a working group comprising of us researchers and the representatives of the City of Oulu's Urban Planning division and companies.

2.3 Modeling guidelines and conventions

To enable different parties to contribute parts to the overall city model, we created a set of modeling guidelines in collaboration with the modelers from the participating companies (Table 1).

We have made few minor changes to the initial guidelines during the project: 4 (modified) - We noticed that upturned normals were a more common flaw in the models than gaps in the mesh structure. 6 (modified) - Originally street partitions for a block were done separately by the party who modeled the buildings of the block. As a result, street partitions did not match exactly and there were small mismatches in the elevation of visual and physical terrain. 7 & 9 (relative prioritization added) - the original guidelines appeared contradictory to the modellers thus we decided to prioritize them them so that 7 (atlassing textures) has priority over 9 (tiling textures).

We also needed to specify how the data is organized and composed to form the overall city model. This was done to meet two criteria: (1) the modeling work should follow the common practises used by the 3D artists; and (2) it must be ensured that the overall city model can be composed in a robust, straightforward and scalable manner.

We decided to work with the granularity of a scene file per city block. The whole city as a single scene would have been too burdensome for the 3D modeling applications to handle. The initial model consisting of nine blocks was already too large to handle as a single project file for Blender, as reported by the company who created the model. A scene per building was seen as impractical in the modeling work as it would have been a significant distraction to continuously switch the currently open project file while working on various passes on multiple nearby buildings (which are often similar). Furthermore, it would be very challenging to create a block as a seamless entity when constructed from multiple parts. Buildings are often next to each other, sometimes even sharing parts, and how the staircases and pavements connect to the inner yards or streets is best – from the perspective of the modeling work – to create in the same scene.

Based on the modelling specification, each building model was composed as a single standalone 3D object and created in a proper way to enable efficient rendering (culling). This supports the use of the building models both individually, and in other kind of overall scene compositions which are based on geo-coordinates of individual buildings and do not have nor need the concept of a block at all. For example, in systems like Google Earth or MAPGETS (mapgets.com) buildings are placed individually on a map that is created separately. We find the granularity of blocks useful for (manual) creation of seamless models for larger areas.

To facilitate interorganizational collaboration, we used a version control system for storing the 3D model files. At the beginning of the project we elected to use Subversion (SVN) as it was known to work quite fast and reliably with large files, in contrast to Git and other more recent distributed versioning systems. Recently, we have changed to Git Large File Storage (LFS) which has become available later. The model repository is organized according to so-called areas which refer to particular parts of the city center. Each area is a directory with a subdirectory for each block in that area. The scene file and accompanying texture files of a block are contained in the block's directory.

Table 1: *Modeling guidelines*

Modeling guideline	Motivation
1. Each block with its neighboring terrain should be included in a single Blender-file (or in some other file format if Blender is not used in the modeling).	Merging of the data is likely to be needed at the production phase and the original files should be available.
2. The 3D city model should not contain any objects that are not visible to the user	To avoid unnecessary burden on the CPU and the GPU.
3. Any two objects (such as buildings) should not overlap with each other.	To avoid overlapping triangles that result in visual artifacts in the GPU rendering.
4. All objects should be closed and there should not be any holes in the objects. →(modified during the project) There should not be any holes or normals backwards in the objects.	To avoid visual flaws.
5. Proper object grouping, each building should be grouped as an object that contains all of its parts (windows, doors, etc.).	The purpose is to know any given object will fit into some predefined 3D space partition, and its groups (submeshes) are fractions of the given geometry sorted by differing surface materials.
6.Terrain should be divided according to city blocks. →(modified during the project) Streets should be done by one organization and they should match the terrain. The building models should be matched to the streets.	Not to waste resources used for rendering the complete terrain all the time. →(modified during the project) To avoid mismatch of separate street partitions. To avoid mismatch in the elevation of visual and physical terrain.
7. Grouping materials for each object, materials and textures should be grouped (atlas). Materials and textures should be reused as much as possible. This guideline has priority over guideline 9.	To make rendering faster on the GPU.
8. The size of textures or texture maps should be power of 2 (2, 4, 8, 16, 32, 64, 128, ...).	To achieve optimal performance with desktop hardware and a working solution with mobile GLES2. [Koskela and Vatjus-Anttila 2015]
9. Texture UV mapping should be done using "tiling" of textures, i.e. repeating the texture as much as possible. (prioritization added during the project) This guideline defers to guideline 7.	To reduce texture memory consumption significantly.
10. Adjacent triangles in a mesh should be mapped side by side also on the texture map.	To avoid filtering problems. Furthermore, this results in reduced number of vertices.
11. Texel density should match the surface area of a single triangle.	Use of a too small texture results in poor visual quality and use of a too large texture wastes extra pixels.
12. A large number of vertices should not be used for modeling buildings that typically have a lot of plain surface.	To make rendering faster on the GPU.
13. For creating tiny details, use of textures should be preferred.	To avoid visual flaws when the models are observed from distance.
14. Use repository with version control for all the models.	To take the reusability of the models in consideration. To make it easier to update details like changes in signage or new construction.

2.4 Composing the city model from block models

The city model is composed from block models as follows. The city's urban planning division has specified for each block a geospatial reference point in a particular coordinate system, typically near one corner of the block. We decided to compose the city model so that each block scene has its origin point in the corresponding spot where the pivot point for that block is in the overall model (Figure 3). This way for the city model we can simply instantiate every scene at the corresponding point. We learned from 3D modelers that it is a common practice to have a local origin in each scene, instead of a single global pivot point somewhere in the middle of the city. In a large-scale model such a single pivot point could be very distant, which would make placing new blocks or buildings very difficult.

To set the orientation of the buildings correctly, we decided to apply geo-orientation directly in the model. Alternatively, we could model them so that the walls of rectangular buildings would match the axis in the scene coordinates. Then the orientation information would be applied separately when instantiating the buildings or scenes in the overall city model. This seemed like a viable option because downtown Oulu has a quite strict grid layout. However, the grid layout has a peculiar caveat in that street corners are not exactly 90 degrees, but either 89 or 91 degrees approximately. The building used as a reference back in 1783 had a structural deviation of about one degree (of which no one knew at the time) and this erroneous reference has propagated through the whole city center. Thus, it was simpler to apply the geographical orientation (as in rotation) directly in the models. We regard this as a suitable approach for any city model, regardless if the city has a grid plan or not.

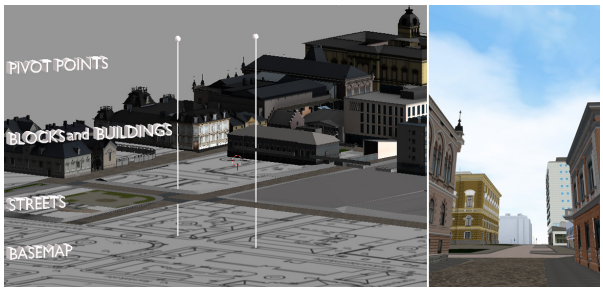


Figure 3: *The city model is composed by positioning the streets and blocks using a base map.*

3 Publication of VirtualOulu on the Web

VirtualOulu is published on the web as a collaborative 3D web application that strives to provide immersive street level experience of downtown Oulu.

3.1 Open source platform: realXtend Tundra and WebTundra

We chose to use an existing WebGL engine for publishing VirtualOulu on the web and to serve as a base for applications. Since a comprehensive review of all candidate engines is outside the scope of this article, we just observe that nowadays there are several good alternatives including Three.js, Babylon.js, Cesium, Turbulenz, xml3d.js and X3DOM. Also Unity and Unreal Engine feature Javascript exports for using WebGL. It should be noted that VirtualOulu model itself is independent of any engine used for publishing it in the web. The model can serve as both a benchmark for renderers, and as freely available content for anyone to use with their engine of choice.

To publish VirtualOulu as a 3D web application we use the WebTundra client and the Tundra multi-user application server from the realXtend open source platform [Alatalo 2011]. This choice is partly motivated by our pivotal role in developing the realXtend platform in recent years. Today, WebTundra and Tundra are a part of EU's FIWARE initiative as open source libraries and open specifications for 3DUI and Synchronization Generic Enablers in the Advanced Web-based User Interface chapter [2]. That is, Tundra's entity-component model is the specification and reference implementation for real-time multi-user applications in FIWARE.

WebTundra is a browser based Javascript client that can use a Tundra server for multi-user applications (Figure 4). WebTundra uses the popular Three.js library for rendering with WebGL and implements the Tundra protocol over WebSockets for real-time networking. An extensible entity-component model is used for the data and automated network synchronization [Alatalo 2011]. We use Meshmoon (meshmoon.com), a commercial WebTundra cloud hosting service, to host VirtualOulu in the web. Meshmoon currently uses Amazon's Simple Storage Service (S3) for distributing 3D files to web clients.

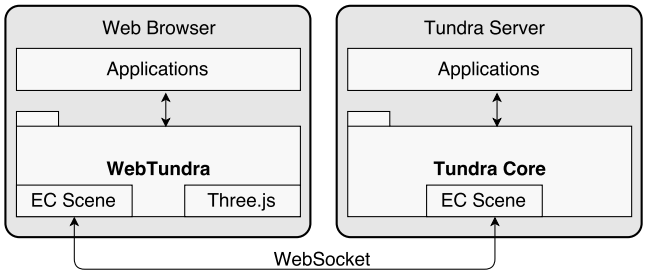


Figure 4: *WebTundra and Tundra are a platform for real-time multiuser applications on the web*

WebTundra applications are multi-user out of the box. It basically adds a layer of networked entities atop Three.js renderer. WebTundra can be used for creating, for instance, a multiplayer game without the need to implement new client and server platform nor a new protocol for the game functionality. For an application, the developer can define her own data components with attributes for the application data. Application Javascript code can access and manipulate the attributes both on client and server side, and the changes are replicated automatically to all clients in the session. In the core and default distribution, WebTundra includes basic components common in 3D applications such as support for meshes and rigid body physics simulations (either on client or server side, using the Bullet open source physics library, as the ammo.js Emscripten build on the client side).

For graphics, WebTundra uses Three.js which is the most popular open source WebGL library. Three.js is simple to use and complete in features typically needed in game-like applications. Besides basic meshes, it supports sprites, particles, morph and skeletal animations and various post-processing effects. The material system features both ready-made shaders for typical e.g. bump, light and environment mapping, but also using own custom shaders. Three.js is a very active project with a healthy developer community. Besides various projects for web advertisements, it is used in services such as Microsoft's Photosynth and in a commercial on-line 3D modeling application Clara.IO (from where many recent code improvements have come).

Notably for 3D web, neither WebTundra nor the Three.js renderer included there use browser DOM by default. Instead, they feature a scene graph and an associated Javascript API. One reason is performance: manipulating thousands of objects in browser Javascript is faster with pure Javascript objects, compared to manipulating the DOM (although detaching parts of the DOM tree from the document, or using the shadow DOM, can solve this). Other reason is that the Javascript API is friendlier for application developers when the 3D objects have associated functionality directly in them, for example, the lookAt() method in a camera. Particularly in case of WebTundra, the idea is to have the same scene model with a similar API in both the web client and on the server side for real-time multi-user applications.

There is, however, an optional DOM integration plugin for WebTundra which synchronizes the scene to the DOM either in the verbose machine readable TXML or the human friendly XML3D format. It's also possible to load the scene from a XML3D description embedded in the HTML page, similarily to the XML3D.js reference implementation. The Tundra entity-component model and XML3D have been found to be almost identical and their partial mapping is described in the FIWARE Wiki [3]. Furthermore the maturing of Web Components and their utilization for 3D scenes is now bringing new interesting opportunities for DOM integration in WebTundra.

In the context of 3D city models WebTundra is also used by the commercial MAPGETS 3D map application platform offered by FCG City Portal Ltd. Related research on integrating 3D scenes automatically generated from GIS data to the Tundra application platform was published by Virtanen *et al.* [2015]. MAPGETS extends WebTundra with autogenerated 3D buildings from map and other data, supporting both Open Street Maps (similarily to ViziCities) and official city databases (for example Trimble Locus) over WFS. It also aggregates several open data sources for application developers to use. In the future, they plan to include the immersive VirtualOulu models as well and add the functionality for street level views. To support this, we plan to provide the detailed and textured models from a database via an API that supports spatial queries and works via geographical coordinates.

[2] http://catalogue.fiware.org/enablers/3dui-webtundra

[3] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.MiWi.3D-UI#Mapping_Synchronization_GE_data_to_XML3D_objects

3.2 Web user interface

VirtualOulu and any applications built on top of it are intended to be used in the Web. Therefore, the user interface of the 3D city model and its applications should meet generic criteria for web usability, including support for a range of popular devices, reasonable loading times, and simple and clear navigation. Web-based 3D city models are problematic on all these fronts. WebGL requires suitable hardware and browser support and its performance varies greatly between difference devices. The size of downloaded files and memory requirements of large 3D scenes with detailed building models and textures are huge compared to conventional web pages with just text and images. Navigating in 3D spaces is relatively complex and there are little conventions for generic web outside the genre specific specialized controls that only gamers are familiar with.

We address the challenge of large amounts of data to be downloaded in two ways: (1) The web UI sequence is designed so that the first scene to be loaded is a light bird's eye view that loads quickly and performs well on most devices (Figure 2(a)), and (2) during navigation in the immersive street level view (Figure 2(b)) only 'relevant' nearby objects of the scene are loaded dynamically on-demand. This is opposite to conventional 3D computer games that typically commence with a large download up front (even gigabytes) before the user is allowed any interaction with the system. This is the default behaviour for example when publishing a 3D application made with Unity on the web. We find this approach as web-unfriendly and employ dynamic on-demand (un)loading of assets instead. The large amount of data of a detailed 3D model is also critical for memory use in the user device. With WebGL, if an application tries to use too much graphics memory it simply crashes (the browser tab, in case of Chrome). Our dynamic (un)loading of assets on-demand works as follows. Objects are prioritized according to camera view frustum and distance, so that only nearby objects are loaded to fit within memory constraints. This also reduces the CPU load of the render er when it processes the scene. A k-d tree is used to get nearby objects efficiently. When user navigates to a new area in the scene, corresponding new objects are loaded on-demand and the ones left behind are removed from the scene and from the memory. Neither Tundra nor WebTundra feature this kind of dynamic scene management so it was implemented for VirtualOulu specifically in the application code. In the future, it would be desirable to refactor this functionality into a generic scene manager.

Navigation in the 3D scene is implemented as follows. In the initial bird's eye view the controls mimic common web map applications, where mouse drag or touch swipe pans the view. Right-mouse-button drag rotates the view and mousewheel controls the height of the camera to "zoom" the view in/out. Certain key locations in the city are marked with their names 'floating' in the air that serve as shortcuts to be clicked/touched to enter the street level view in that location. The interface also has GUI buttons for switching between the bird's eye view and the street level view. The street level view uses a first person camera that is moved left/right/back/forward with arrow keys. This navigation paradigm is identical to Second Life and suitable for non-gamers, as it can be used with one hand only. However, also the common WASD+mouse controls are implemented for powerful two-hand control typical for first and third person games. There W and A move forward and backward, but A and D strafe sideways instead of rotating. Moving the mouse with the right button pressed rotates the camera both vertically and horizontally. This leaves moving the mouse without any keys pressed to control the cursor normally, and left mouse press works to act on either active objects in the 3D view or 2D GUI elements. Role playing games typically have this same mouse control for camera rotation as they require a lot of different interactions with the GUI, whereas first person shooters map mouse movement directly to rotation without any button presses as they do not have 2D GUI elements that the player should also click in the middle of the action.

Current web user interface provides basic tools for real-time collaboration between users in form of simple lightweight avatars, text chat, and voice and video chat using WebRTC. The lightweight avatars correspond to a line sticking up from the ground and a curved picture plate to optionally show the user's Facebook profile picture. We decided to start without complex realistic human avatars because they are both challenging to design for the general public (to not distract people's focus) and heavy to render with WebGL on commodity user devices.

3.3 Application development

VirtualOulu, similar to any client side web library, supports application development in Javascript. WebTundra packages several libraries, including Three.js for the 3D rendering and polymer for 2D GUI widgets using web components. Additionally it provides a high-level scene API for the automatically synchronized multiuser entity-component system, and an asset module that manages file downloads before feeding the data to Three.js. Ready made visual components such as meshes (from files) and particle systems can be utilized directly by using the WebTundra scene API. Generating custom geometry in code or manipulating the graphics rendering in other ways is done directly using the Three.js API which is available normally in the Javascript namespace. WebTundra's scene API provides access to the underlying Three.js scene to facilitate this.

A number of applications have been implemented to verify the feasibility of VirtualOulu and its underlying publication platform as an application development platform. The web interface comes with a set of default applications, including a guided city tour of selected cultural objects represented with textual stories, images and videos, a spatial voting application, and a future view of a new building to be built at downtown Oulu. They are implemented as WebTundra applications and for example the voting application uses the underlying entity-component system to store the votes for a block in an application specific component.

Other applications have been developed atop VirtualOulu as research prototypes and student projects, including Service Fusion, 3D user interface integrating real-world online services into VirtualOulu [Hickey et al. 2012]; Props, a 3D-game-like system for improvisational storytelling [Alavesa et al. 2014]; Street Art Gangs, a location based hybrid reality game played with mobile phones at downtown Oulu and in VirtualOulu [Alavesa and Ojala 2015]; and Historical Oulu where portals planted around the city allow peeking into historical photographs superimposed at their real world locations in the 3D scene.

4 Empirical performance evaluation

We have empirically evaluated the quantitative performance of web browsing in VirtualOulu in two ways, first in terms of typical download times and memory loads imposed on user device at three different stages of navigation and then in terms of frame rates achieved by three different user devices during navigation.

4.1 Download latencies and memory use

We recorded download time and data, GPU memory consumption and number of HTTP requests in three test cases corresponding to three successive phases in a typical browsing session: (C1) initial bird's eye view; (C2) enter stationary street level view at particular location so that detailed textured models are loaded; and (C3)

navigation along a particular route in the street level view so that assets are dynamically (un)loaded on demand during navigation. The three test cases are specified as follows.

(C1) Browse to http://beta.adminotech.com/virtuaalioulu/. The browser opens a welcome page for the user to login into the 3D scene. At this point record measurements as a baseline for determining the values for loading the actual 3D scene. Click the login button and wait for the initial bird's eye view to be fully loaded before recording measurements. It should be noted that at this point all default applications of the user interface have been loaded.

(C2) Click 'Kauppatori' shortcut to enter the street level view at the market place. Record measurements once the scene is fully loaded.

(C3) The navigated route was as follows: Kauppatori -> turn towards 'Arkkarikortteli' -> move towards Hotel Radisson Blue and Hallituskatu -> turn to Hallituskatu -> continue to culture house Valve -> stop at Valve. Record measurements once the scene at Valve is fully loaded.

The measurements were done by browsing VirtualOulu with the Chrome browser on a gaming laptop (see next section). We recorded GPU memory consumption, the number of requests and the amount of downloaded data. Table 2 shows the measurements for the baseline (0) and the three test cases. The download times for the three different Internet connection types are estimated from the measured amounts of downloaded data using average real-world download speeds in Finland as obtained from the Testmynet website (http://testmy.net/country/fi): 0.763 Mbps for 3G, 9.183 Mbps for 4G (LTE) and 27.8 Mbps (xDSL). This way we obtain more representative estimates of download times for the general public than what our own measurements in a particular measurement context would provide.

4.2 Frame rate

A key characteristic for the usability of VirtualOulu is frame rate during navigating around in the 3D scene at street level in test case C3. We evaluated the frame rates achieved by three different user devices during navigation as follows. We used Chrome Developer Tool's performance timeline tool to observe and record the frame rate and the amounts of data downloaded. GPU Shark was used to measure the use of GPU memory consumption because Chrome failed to report it on one of the test devices. The three tested user devices were:

Gaming laptop: ASUS ROG G501J Notebook, Intel i7-4720HQ CPU @ 2.60GHz 4 Cores, 16GB, NVIDIA GeForce GTX 960M (4 GB), Windows 10 64bit
Basic laptop: HP EliteBook 840 G2, Intel i5-5300U, 8GB, Intel HD Graphics 5500, Windows 7 64bit
Surface Pro: Intel i7-4650U @ 2.30 GHz, 8GB, Intel HD Graphics 5000, Windows 10 Pro 64bit

Notably, the frame rate is good (60 FPS) on all three devices when the scene is in a stable state so that objects are not being (un)loaded. So the 3D scene and the renderer as such are sufficient for a good user experience. However, frame rate suffers badly during navigation and dynamic (un)loading of assets on demand. In test case C3 the FPS distribution was determined from Chrome's timeline JSON recordings and is presented using 5 categories: 60 (frame duration <17ms, that is 58.8 FPS or higher), 30-58.8, 10-30, 5-10 and <5 FPS. We regard the <5 FPS category as unusable and corresponding periods are typically experienced as glitches or stalls by the user. Figure 5 shows the frame rate categories for the three devices that used indoor WiFi for Internet access. It should be noted that these percentages refer to the proportion of frames in a given category, not to the proportion of real time browsing at a given frame rate. As expected, the gaming laptop performs best but stalls at times. The 'weaker' devices perform acceptably for most frames but have many more and longer stalls.

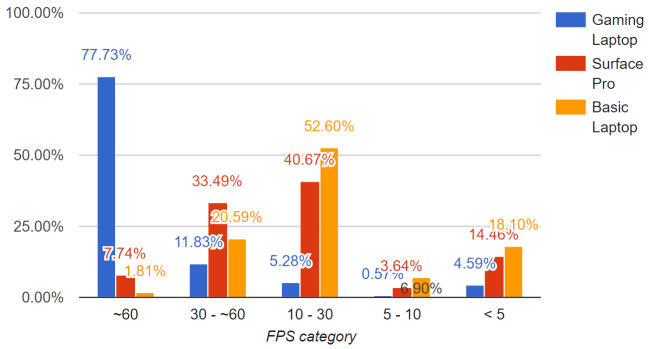


Figure 5: *FPS distribution during navigation (C3)*

The lesson from these measurements is that we should improve the dynamic (un)loading of assets on demand. Studying the Javascript performance profile reveals clearly that the problematic frames are slow due to WebTundra parsing binary mesh assets in the main rendering thread on the CPU. When a new file arrives from the web (or cache), its handling (onTransferCompletedBinary total) takes 331 ms on the gaming laptop, for example. OgreThreeJsUtils.convertOgreMesh (91 ms) and the underlying geometry parsing functions are the heavy ones.

The good news is that there are promising candidate solutions for addressing this problem. Firstly, it is possible to move the parsing to a Web worker to not block the renderer. We have made some initial tests with promising results with the Three.js OpenCTM loader which uses workers, where adding a new large object to the scene caused only a barely noticeable hick-up. However, most Three.js loaders do not currently use workers and we are not using the CTM format for other reasons. Thus, refactoring the loader of choice to use workers is one possible solution.

We measured the impact of using workers on frame rate with a dedicated technology test of loading one city block that Playsign Ltd. had created to test the feasibility of using WebGL and Three.js for large detailed city models. That work was done as open source in early WebTundra development supported by FIWARE, published on GitHub as Playsign/OuluThreeJS. In one of the tests they used one of the VirtualOulu blocks for the test and exported it to the OpenCTM compressed geometry format, we call it CTM-OuluThreeJS here. The scene is managed by a GridManager which keeps the city blocks (or grids in any terrain) near the view location loaded and visible, loading and unloading as necessary during movement. The scene manager in the current VirtualOulu differs there by managing the scene at object (typically building), not block, level. Also CTM-OuluThreeJS uses the OpenCTM format and does not use the WebTundra entity system but Three.js directly, whereas VirtualOulu parses Ogre binary meshes using WebTundra's asset system to load the model to Three.js. However, we expect that the performance impact of using workers to offload mesh parsing away from the main rendering loop would be similar.

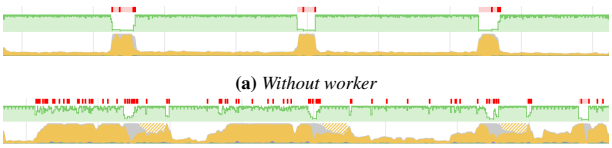
We instrumented this test by adding an automated movement of 17 seconds which covers three times when new blocks are added to the scene. This automated test also annotates the Chrome DevTools timeline record with the start and stop times so we can compare the runs exactly. The test setup is available online[4] with workers on

[4]http://playsign.github.io/CTM-OuluThreeJS/

Table 2: *Measurement data*

Case	GPU (MB)	# Requests	Data (MB)	3G (s)	4G (s)	xDSL (s)
(0) Login page	-	65	3.0	31.5	2.6	0.9
(C1) Initial bird's eye view	138	200	6.5	68.2	5.7	1.9
(C2) Stationary street level view	28	1044	101.5	1064.2	88.4	29.2
(C3) Navigation in street level view	384	765	86	901.7	74.9	24.7

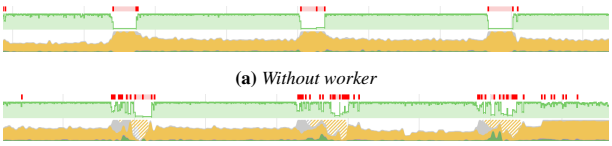
by default, and to run it without workers it supports the parameter worker=false[5]. The test is done by first starting the timeline recording manually, then calling startPerf() in the Javascript console, and stopping the recording when the movement stops. The manual steps with the recording are not a problem because the timeline annotations created by the test script mark exactly the time range to be analyzed.



(a) *Without worker*

(b) *With worker*

Figure 6: *Effect of worker usage on FPS (green) and CPU (yellow) on gaming laptop.*



(a) *Without worker*

(b) *With worker*

Figure 7: *Effect of worker usage on FPS (green) and CPU (yellow) on Surface Pro.*

The graphs taken from Chrome's timeline tool in Figures 6 and 7 illustrate the effect of workers on FPS and CPU usage. The red marks denote long frames. However, despite having more red marks, the runs with the worker provide a smoother user experience especially on the gaming laptop where the FPS remains usable almost all the time even though it drops lower during loading. With the loading code blocking rendering there is a disturbing brief stall. On the powerful gaming laptop the worker loading smoothens experience considerably. On the Surface Pro, there is also significant improvement but more low-FPS periods still remain. This suggests that workers can solve the problem on powerful machines but not fully on weaker commodity hardware. Using the glTF format by Chronos consortium could provide another solution for smooth loading. It is designed so that it does not require parsing the geometry in Javascript because it is already provided as GL arrays in the glTF binary file. We have preliminary tested glTF and were able to load a part of the city model using it but have neither yet thoroughly evaluated its performance nor have versions of the whole VirtualOulu or the dynamic loading code that would use it.

5 Discussion

We presented the creation of a 3D virtual city model and its subsequent publication on the web using the open source realXtend platform and the Three.js engine. The modeling guidelines have proven very useful in steering the collaborative modeling process. We openly acknowledge that our process is still far from completely addressing the ultimate challenge of any virtual city model, namely keeping it up-to-date with the real world. To this effect we are currently exploring automated integration of VirtualOulu with building models automatically generated from the GIS system (Locus Trimble) of the City of Oulu. The guidelines effectively specify the desired 3D models in terms of visual quality and geometry so that they would be suitable for rendering with game engines. However, large textures remain very problematic for web(GL) use due to their large size which induces high download times and memory consumption in user devices. We have addressed the problem with dynamic (un)loading of 3D assets on demand, but the results of the empirical evaluation showed our current solution is not optimal. Background workers for mesh loading are a good improvement based on the additional measurements. In future work we will explore other solutions, including progressive mesh loading, procedural textures and maybe giving up on rich textures altogether, considering recent evidence indicating that the role of textures in 3D city models may be overestimated [Garnett and Freeburn 2014]. It is also interesting to test other engines with background loading of 3D data. This may include web builds of Unreal and Unity C++ engines which are getting better performance thanks to WebAssembly during 2016. Currently the visual appearance of the model is poor due to no light or other shading configuration. We are currently adding use of modern shading but need to be careful to retain adequate performance on common hardware with WebGL. We demonstrated the extensibility of VirtualOulu and its underlying software platform with a number of example applications, both single-user and multi-user (WebTundra provides synchronization between multiple users). Three.js has been perceived as easy to use by application developers. In future application development we will focus on hybrid reality applications that exploit bi-directional real-time communication between reality and virtuality, for example for users physically present in the real city services could be available through an AR viewer such as smart glasses and AR and VR users could exist in the same virtual scene and communicate with each other.

6 Note

Access information and source materials of VirtualOulu are available via http://www.ubioulu.fi/en/VirtualOulu. Videos of a fly through VirtualOulu and its default applications is available at http://goo.gl/MP7ERI and of the test case C3 at https://goo.gl/OcfouH.

Acknowledgements

Adminotech Ltd., Cubicasa Ltd., Evocons Ltd., Oulu3D Ltd. and Ludocraft Ltd. are acknowledged for their contributions to modeling, and the design and implementation of the UI, web hosting and scene management software. This work has been supported by the Open Innovation Platforms spearhead project and the Open City Model as Open Innovation Platform pilot project funded by the ERDF and the City of Oulu under the Six City Strategy program, and the COMBAT project (293389) funded by the Strategic Research Council at the Academy of Finland.

[5] http://playsign.github.io/CTM-OuluThreeJS/?worker=false

References

ALATALO, T. 2011. An entity-component model for extensible virtual worlds. *Internet Computing, IEEE 15*, 5, 30–37.

ALAVESA, P., AND OJALA, T. 2015. Street Art Gangs: Location Based Hybrid Reality Game. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*, ACM, New York, NY, USA, MUM '15, 64–74.

ALAVESA, P., OJALA, T., AND ZANNI, D. 2014. Props: 3d-game-like mediator for improvisational storytelling. *Entertainment Computing 5*, 4, 381–390.

ARPONEN, M. 2002. From 2d Base Map to 3d City Model. In *UMDS '02 Proceedings, Prague (Czech Republic)*, 2–4.

BILJECKI, F., LEDOUX, H., STOTER, J., AND ZHAO, J. 2014. Formalisation of the level of detail in 3d city modelling. *Computers, Environment and Urban Systems 48*, 1–15.

BILJECKI, F., STOTER, J., LEDOUX, H., ZLATANOVA, S., AND ÇÖLTEKIN, A. 2015. Applications of 3d City Models: State of the Art Review. *ISPRS International Journal of Geo-Information 4*, 4, 2842–2889.

BOURDAKIS, V. 2001. On developing standards for the creation of VR city models. In *Architectural Information Management [Proceedings 19th eCAADe-Conference, Helsinki (Finland)], Helsinki*, Citeseer, 404–409.

CARROZZINO, M., EVANGELISTA, C., AND BERGAMASCO, M. 2009. The Immersive Time-machine: A virtual exploration of the history of Livorno. In *Proceedings of 3D-Arch Conference*, 198–201.

DÖLLNER, J., KOLBE, T. H., LIECKE, F., SGOUROS, T., AND TEICHMANN, K. 2006. The virtual 3d city model of berlin-managing, integrating, and communicating complex urban information. In *UDMS Proceedings*, vol. 2006.

GARNETT, R., AND FREEBURN, J. T. 2014. Visual acceptance of library-generated citygml lod3 building models. *Cartographica: The International Journal for Geographic Information and Geovisualization 49*, 4, 218–224.

GELERNTER, D. 1991. *Mirror worlds or the day software puts the universe in a shoebox: how will it happen and what it will mean*. Oxford University Press.

GOETZ, M., AND ZIPF, A. 2013. The evolution of geo-crowdsourcing: bringing volunteered geographic information to the third dimension. In *Crowdsourcing Geographic Knowledge*. Springer, 139–159.

HEO, J., JEONG, S., PARK, H.-K., JUNG, J. H., HAN, S., HONG, S., AND SOHN, H.-G. 2013. Productive high-complexity 3d city modeling with point clouds collected from terrestrial lidar. *Computers, Environment and Urban Systems 41*, 26–38.

HICKEY, S., PAKANEN, M., ARHIPPAINEN, L., KUUSELA, E., AND KARHU, A. 2012. Service fusion: an interactive 3d user interface. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, ACM, 53.

HORNE, M., THOMPSON, E. M., AND CHARLTON, J. 2014. Towards a multifunctional virtual city model. *Technologies for Urban and Spatial Planning: Virtual Cities and Territories: Virtual Cities and Territories*, 154–172.

HUDSON-SMITH, A. 2007. Digital urban-the visual city. *UCL Working Papers*, 124.

KOLBE, T. H., GRÖGER, G., AND PLÜMER, L. 2005. CityGML: Interoperable Access to 3d City Models. In *Geo-information for Disaster Management*, P. Oosterom, S. Zlatanova, and E. M. Fendel, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 883–899.

KOLBE, T. H. 2009. Representing and exchanging 3d city models with CityGML. In *3D geo-information sciences*. Springer, 15–31.

KOSKELA, T., AND VATJUS-ANTTILA, J. 2015. Optimization techniques for 3d graphics deployment on mobile devices. *3D Research 6*, 1, 1–27.

LIFTON, J., AND PARADISO, J. A. 2009. Dual reality: Merging the real and virtual. In *Facets of Virtual Environments*. Springer, 12–28.

MAHER, M. L., PAULINI, M., AND MURTY, P. 2011. Scaling up: From individual design to collaborative design to collective design. In *Design Computing and Cognition'10*. Springer, 581–599.

PALMQUIST, E., AND SHAW, J. 2008. Collaborative City Modeling. In *Architecture in Computro*, 26th eCAADe Conference Proceedings, 249–256.

RICCI, A., PIUNTI, M., TUMMOLINI, L., AND CASTELFRANCHI, C. 2015. The mirror world: Preparing for mixed-reality living. *Pervasive Computing, IEEE 14*, 2, 60–63.

ROUSH, W. 2007. Second earth. *Technology Review 110*, 4, 38–48.

SAHIN, C., ALKIS, A., ERGUN, B., KULUR, S., BATUK, F., AND KILIC, A. 2012. Producing 3d city model with the combined photogrammetric and laser scanner data in the example of Taksim Cumhuriyet square. *Optics and Lasers in Engineering 50*, 12, 1844–1853.

SHENG, W., NAKATA, S., TANAKA, S., TANAKA, H. H., AND TSUKAMOTO, A. 2013. Modeling High-Quality and Game-Like Virtual Space of a Court Noble House by Using 3d Game Engine. In *Culture and Computing (Culture Computing), 2013 International Conference on*, IEEE, 212–213.

SLATER, M., AND WILBUR, S. 1997. A framework for immersive virtual environments (FIVE): Speculations on the role of presence in virtual environments. *Presence: Teleoperators and virtual environments 6*, 6, 603–616.

STREITZ, N. A. 2011. Smart cities, ambient intelligence and universal access. In *Proceedings of the 6th international conference on Universal access in human-computer interaction: context diversity-Volume Part III*, Springer-Verlag, 425–432.

UDEN, M., AND ZIPF, A. 2013. Open building models: towards a platform for crowdsourcing virtual 3d cities. In *Progress and New Trends in 3D Geoinformation Sciences*. Springer, 299–314.

VIRTANEN, J.-P., HYYPPÄ, H., KÄMÄRÄINEN, A., HOLLSTRÖM, T., VASTARANTA, M., AND HYYPPÄ, J. 2015. Intelligent Open Data 3d Maps in a Collaborative Virtual World. *ISPRS International Journal of Geo-Information 4*, 2, 837–857.

WALLET, G., SAUZÉON, H., PALA, P. A., LARRUE, F., XIA ZHENG, AND N'KAOUA, B. 2011. Virtual/Real Transfer of Spatial Knowledge: Benefit from Visual Fidelity Provided in a Virtual Environment and Impact of Active Navigation. *CyberPsychology, Behavior & Social Networking 14*, 7/8 (Aug.), 417–423.