DRUMM: Dynamic Viewing of Large-scale 3D City Models on the Web

Timo Koskela, Matti Pouke, Arto Heikkinen, Toni Alatalo, Paula Alavesa, Timo Ojala
Center for Ubiquitous Computing
University of Oulu
Oulu, Finland
firstname.lastname@oulu.fi

***Abstract*—3D city models have become an important user interface for various applications, ranging from entertainment to civil engineering. Today, 3D city models can also be accessed on the web without installing any additional software, which has significantly widened their potential audience. However, visually accurate 3D city models are typically large in terms of file size, and hence, require ample network bandwidth for minimizing download delays and providing a smooth user experience. In this paper, we introduce a method called DRUMM, which enables dynamic resource management for viewing 3D city models on the web. DRUMM supports both the use of varying criteria for prioritizing the download order of 3D graphics, and the division of 3D graphics into chunks facilitating parallel downloads that can be suspended and later continued. The performance of DRUMM was evaluated in terms of (1) starting delay; (2) used network bandwidth; and (3) the number of buildings with textures in the view using the developed prototype implementation. Based on the results, DRUMM improves the usability of 3D city applications, particularly when the network bandwidth is scarce.**

Keywords— virtual reality; web3D; 3D graphics; interest management; performance evaluation

I. Introduction

3D graphics have rapidly made their way on the web due to the introduction of WebGL, a JavaScript API for rendering 3D graphics in a web browser without installing any additional software. Thanks to WebGL, several 3D city models are now easily accessible on the web for a wide audience (e.g. [1][2][3]). During the last decade, the 3D city models have also gained a lot of attention in the research community. The popularity of the 3D city models is at least partially explained by their versatile application areas. In a recent survey, almost 30 use cases and more than 100 applications were identified for 3D city models [4].

3D building models utilized in interactive 3D applications can be roughly categorized into three levels of detail ranging from the highest to the lowest: (1) Building Information Models (BIM); (2) game-engine models; and (3) procedurally generated models. BIM models are very detailed and, besides architectural properties, can include semantic information that is utilized in planning and construction, such as heating, ventilation and air conditioning (HVAC) or electric installations. BIM models can utilize very large polygon counts resulting in large file sizes. Game-engine models usually have no planning or construction related information while still aiming to deliver aesthetically pleasing results. While polygon counts for game engine models are lower than BIM models, they still result in large file sizes, which is partly due to texture image data that is coupled with the models. Procedurally generated buildings are usually generated on runtime from a data source and such require minimal bandwidth [5]. However, while procedural models can be useful for analytical and planning purposes, they are not suitable for 3D applications that require high level of aesthetics.

For acquiring the required 3D models (i.e. 3D assets) used in a 3D application, two different approaches can be taken: (1) downloading all 3D assets prior to the use which is typical with 3D games; or (2) downloading 3D assets dynamically on a need basis during runtime. First, depending on the network speed, downloading all 3D assets prior to the use may take up to several minutes or even hours [1]. Second, a large number of 3D assets are downloaded that are not potentially needed at all as the 3D assets are only available as long as the 3D application is in use or as long as they populate the cache of the web browser. Third, in relation to 2D Web, it should also be noted that a page load time longer than 1 second is already considered poor performance [6]. In this respect, downloading 3D assets dynamically on a need basis is typically more efficient and more user-friendly approach particularly when the network bandwidth is scarce. However, in this case, it is crucial to intelligently manage the download order of 3D assets that the delays in the rendering of 3D graphics can be minimized and a smooth user experience can be provided [1][7].

Particularly with mobile devices, network bandwidth is not the only limiting factor, but also the amount of graphics memory may become a performance bottleneck. If the whole 3D city model cannot fit into the graphics memory, the 3D application must be capable of unloading some momentarily unimportant 3D assets from the memory before it runs out. If the memory capacity is exceeded, the 3D application crashes [1].

In this paper, we present a dynamic resource management method called DRUMM designed for viewing 3D city models on the web. DRUMM supports dividing the 3D assets into chunks, which facilitates (1) downloading 3D assets from multiple sources in parallel; and (2) suspending and later continuing the suspended downloads. It should be noted that

DRUMM is data type agnostic, which enables use of any arbitrary data formats for 3D assets. For prioritizing the 3D assets, DRUMM uses Euclidian distance, view frustum, download progress and landmark value. However, as the 3D asset download is independent of the 3D asset prioritization, any equivalent prioritization criteria can be used. A prototype of DRUMM was implemented as part of a 3D city application. The performance of DRUMM was evaluated by implementing an automated walk-through in the 3D city application. During the walk-through, we recorded (1) the starting delay, (2) the used network bandwidth as well as (3) the number of buildings with textures in the view as a function of time. The walk-through was conducted with varying settings for DRUMM and the available network bandwidth.

The rest of the paper is organized as follows. In Section II, the related work is presented. In Section III, VirtualOulu is introduced. VirtualOulu is an example of game-engine-based 3D city models published on the web. In Section IV, the principles of DRUMM are presented in detail. In Section V, the experimental setup is described, and in Section VI, the results are summarized. Finally, Section VII concludes the paper and provides some ideas for future work.

II. Related Work

The 3D asset delivery is closely linked with the concept of interest management (IM) whose goal in 3D virtual worlds is to limit the propagation of state updates only to the relevant ones from the standpoint of each individual user [8]. This can be conducted based on proximity [8][9], regions [10], occlusion [11] or a combination of these [12][13]. In addition to propagating state updates, IM methods can also be used in prioritizing the download order of 3D assets.

In proximity-based IM, the importance of state updates is determined based on their distance from the user, which is also utilized in DRUMM. In region-based IM, the environment is divided into regions or tiles that can be of various shapes [10]. In DRUMM, region-based IM as such is not used, but regions can be used for limiting the range of 3D asset downloads. When using regions, the rules that determine which 3D assets are visible from every other region can also be precomputed. However, region-based solution that rely on precomputations are not feasible for 3D city models whose contents can change dynamically [13]. In occlusion-based IM, state updates are only propagated from entities that are visible to the user. In DRUMM, occlusion-based IM is not used as occlusion is not very suitable for prioritizing 3D asset downloads due to its more dynamic nature. For instance, when a user turns around a city block, a large number of previously occluded (and thus not downloaded) 3D assets may become visible simultaneously. However, it should be noted that using proximity or region-based IM for prioritizing 3D asset downloads does not prevent using occlusion culling in rendering.

In [14], Barchetti et al. have examined the challenge of downloading a large number of 3D objects in real-time when the network bandwidth is limited. They propose four priority policies for 3D assets: field of view, distance, file size and priority class, which are also applied in DRUMM. In their work, they also present a modular structure for managing the 3D asset prioritization and downloading. However, no extensive experiments are conducted for evaluating the performance of their proposed solution.

In [15], Rahimi et al. present a context-aware prioritization scheme for 3D asset downloading. In their solution, they take advantage of the game-context and transfer only the most relevant 3D assets in each frame of gameplay. Compared to our work, their focus is strongly on gaming experience and real-time interactivity of the game. However, the presented prioritization scheme could also be applied in DRUMM if the 3D city models are used for real-time gaming purposes. In their continuation work [7], the prioritization scheme is enhanced in a way that it also takes into account the energy consumption and bandwidth constraints of mobile devices.

In [16], Blast is proposed that is a general container format for binary data transmission for the web. The proposed binary format could also be utilized with DRUMM as DRUMM is data type agnostic. In similar to DRUMM, Blast supports transmission of 3D assets in chunks. However, Blast does not prioritize the download order 3D assets in any way.

In [17], spatial data structures are used on the client side to increase efficiency of determining which 3D assets are visible to the user. Spatial data structures can also be taken advantage in DRUMM for prioritizing 3D assets, however, it should be noted that the proposed approach currently supports only static 3D scenes.

For visualizing large scale CityGML models in a web browser, a framework is proposed in [18]. For prioritizing download of 3D assets, the framework implements a scheduler that operates on three different queues: low, high and top priority. The top priority queue is used only for unloading 3D assets from the memory when it is becoming full. The 3D assets to be downloaded are divided into top and low priority queues based on a user specified strategy. In DRUMM, the same ideology is followed, but the priorities are calculated separately for geometry and textures. In the framework, the 3D scene is divided into tiles of which the nearest ones are considered the highest priority. In comparison, DRUMM uses more versatile set of prioritization criteria and enables use of chunks to improve the efficiency of the download process.

III. VirtualOulu

DRUMM was originally designed for VirtualOulu (see Figure 1), which is a 3D city model of Oulu, Finland [1]. VirtualOulu is published on the web as an open and extensible general-purpose platform for developing new 3D applications. VirtualOulu uses WebGL and three.js for rendering and realXtend WebTundra for enabling synchronization in multi-user 3D applications.

In VirtualOulu, the geometry has been modelled on a granularity of a single block. This is due to the fact that creating a block as a seamless entity is very challenging when constructed from multiple parts. Buildings in a block are typically tightly connected to each other and may even share some parts such as stairs. The geometry in each block is presented as a JSON file of which typical size ranges from 1 to 4 MBs. For textures, each JSON file includes several URLs for texture atlases that were created for each building. Texture atlases were deliberately created per building in order to ease the modelling process, but also to avoid creating huge texture atlases of which downloading would delay the rendering of textures for every building in a block. The texture atlases for each building are presented as PNG files of which typical size ranges from 1 to 2 MBs. For a typical block, the total file size of textures is around 10 MBs. Although originally designed for VirtualOulu, DRUMM can be applied for all kinds of 3D city models on the web. In addition, use of DRUMM is not tied to any specific data formats used for 3D assets.

Fig. 1. A screen shot of VirtualOulu.

IV. Dynamic Resource Management Method (DRUMM)

A. *Block and Building Prioritization*

In DRUMM, the prioritization is conducted within two zones (*R1* and *R2*) as illustrated Figure 2. *R1* is determined dynamically based on the available bandwidth, whereas *R2* is determined based on the available graphics memory of the client device. The geometry data download always precedes texture data download that something meaningful can be quickly shown to the user. However, when the available bandwidth is scarce, *R1* should be rather short that the downloading of texture data can be quickly started after finishing with the geometry data. With WebGL, an application crashes if it tries to overuse the available graphics memory [1]. Therefore, *R2* is not only used for limiting the downloading of 3D assets, but also for limiting the storage of 3D assets. When a 3D asset falls outside *R2*, it is unloaded from the memory.

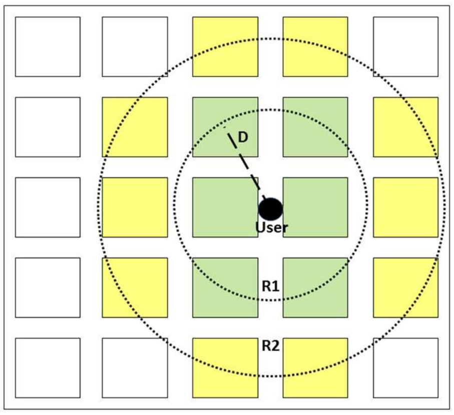

Fig. 2. Prioritization zones R1 and R2.

Within the prioritization zones, the priority of a block is determined by the score that is calculated as a weighted sum of different importance factors as shown in (1).

$$S_{block}^{TOT} = \sum_{i=1}^{n} \omega^i S_{block}^i,$$
$$\text{where } \sum_{i=1}^{n} \omega^i = 1 \text{ and } \forall\, i\colon S_c^i \in [0,1] \tag{1}$$

In (1), S_{block}^{TOT} denotes the overall importance of a block; S_{block}^i is an importance factor *i* calculated for the block. ω^i is a weight for the importance factor S_{block}^i.

In DRUMM, the importance factors used in the prioritization of a block include, but are not limited to (a) Euclidian distance; (b) view frustum; and (c) download progress. In (2), the importance factors for the blocks are presented with their weights.

$$S_{block}^{TOT} = \omega^{distance} S_{block}^{distance} + \omega^{view} S_{block}^{view} + \omega^{download} S_{block}^{download} \tag{2}$$

When the user is moving, priority of each block within *R1* is calculated at every *t* seconds. All blocks are then sorted out based on their overall importance. After this, the geometry data is downloaded starting from the most important block.

After completing the geometry data download, the prioritization is continued with buildings. In DRUMM, the importance factors used in the prioritization of a building include, but are not limited to (a) Euclidian distance; (b) view frustum; (c) download progress; (d) landmark value. In (3), the importance factors for the buildings are presented with their weights.

$$S_{building}^{TOT} = \omega^{distance} S_{building}^{distance} + \omega^{view} S_{building}^{view} + \omega^{download} S_{building}^{download} + \omega^{landmark} S_{building}^{landmark} \tag{3}$$

Next, buildings belonging to the blocks within *R1* are sorted out based on their importance. After this, the texture data is downloaded starting from the most important building. In cases, when all geometry and texture data has been downloaded within *R1*, the same downloading process is continued within *R2*.

B. *3D Asset Download*

The basic idea in DRUMM is to conduct the downloading of 3D assets in chunks of which size *C* can vary according to the available bandwidth, see Figure 3. Although a 3D asset falls lower in the priority order, downloading of the ongoing chunk is always first finished. These design choices facilitate (1) the use of multiple sources for 3D asset download; (2) the suspension and continuation of 3D asset download; and (3) minimizing the delays and the load caused by the increased number of requests for 3D assets. For (1), the performance of 3D asset download can be improved by using WebRTC (i.e. P2P) in parallel with the client-server download [19], however, this topic is not in the scope of this paper. For (2), when the user moves, the rendering delays can be minimized as downloading can be quickly switched to the new high priority 3D assets without losing any effort on partially downloaded 3D assets in case the user decides to double back. In the latter case, the 3D asset downloading can be continued and completed by downloading only the remaining chunks. This is particularly beneficial when the 3D assets are large and the available bandwidth is scarce. For (3), when ample bandwidth is available, *C* can be increased in order to decrease the number of requests for 3D assets.

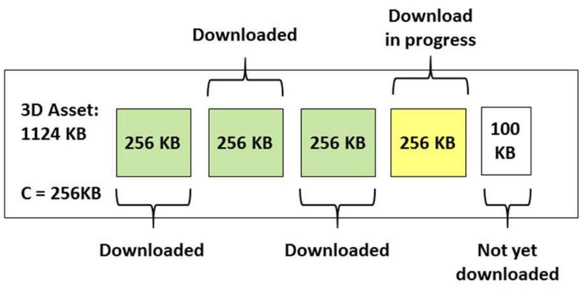

Fig. 3. 3D asset download using chunks.

In DRUMM, only a single 3D asset is downloaded at once. The only exception to this is when the downloading of one or more rearmost chunks is still in progress, but there are no new chunks to request that belong to the same 3D asset. The focus is strictly on downloading a single 3D asset at a time that (1) the maximum number of relevant 3D assets could be rendered as soon as possible; and (2) the number of suspended downloads could be minimized. Depending on the characteristics of the available bandwidth, DRUMM can establish up to *S* parallel downloads of chunks. If the download bandwidth of the client is not the limiting factor, it is beneficial to establish multiple parallel downloads from different sources. However, it should also be noted that a large *S* may result in additional delays in the rendering when the user moves. This is because of the fact that downloading of chunks in progress is always first finished and downloading a large number of chunks in parallel increases the average download time per chunk. Therefore, switching to downloading of a new 3D asset would take some more time with a large *S*. The detailed algorithms guiding the dynamic selection of *S* are not in the scope of this paper.

In DRUMM, the downloading of 3D assets using chunks is implemented with HTTP HEAD and HTTP GET methods. HTTP HEAD returns the header fields of the resource in question (i.e. 3D asset). In the response, Content-Length field contains the total size of the 3D asset in MBs. This information is then used with HTTP GET to request for a chunk having the size of *C* (or less if the chunk is the last one in the 3D asset). The size of the chunk is indicated using the RANGE parameter in the HTTP GET request.

V. Experimental Setup

A. Environment

Our testing environment consists of (1) a virtual server running a Apache Web Server v.2.4.23; and (2) a laptop computer (MacBook Pro i7, 16GB, Win10) with Chrome v.56.0 running a 3D city application implemented with three.js and the prototype of DRUMM. For adjusting the available network bandwidth, Chrome's network throttling feature was used. We did not use a smartphone in the experiments, as Chrome Android did not allow for network throttling. The web server was used for hosting all the 3D assets required by the 3D city application

B. 3D City Application and DRUMM

For evaluating the performance of DRUMM, a 3D application was implemented. For evaluation purposes, we created a consistent test scene of 25 blocks utilizing a similar block across the entire scene. While actual city models might contain blocks of alternating sizes and larger deviations in building heights, we chose to use a homogeneous city model for analysis purposes. This allowed us to evaluate the runtime performance of DRUMM with different parameters, without different geometry and texture sizes affecting the evaluation. The geometry and texture file sizes were adapted from VirtualOulu presented in Section III.

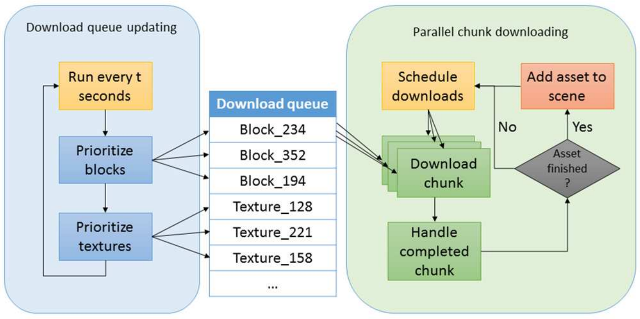

Fig. 4. Software architecture of the DRUMM prototype.

The implementation of DRUMM consists of two main loops: download queue updating (Figure 4: left) and parallel chunk downloading (Figure 4: right). These loops run independently, communicating via the download queue. The download queue is updated every *t* seconds based on the asset prioritization criteria described in Section IV A. The chunk download scheduler maintains *S* simultaneous chunk downloads as long as there are 3D assets to download in the download queue. When the scheduler starts a new chunk download, it always selects the first not yet started chunk from the first 3D asset in the download queue. When the last remaining chunk of a certain 3D asset is downloaded, the 3D asset is added to the scene.

The prototype implementation of DRUMM currently supports the use of arbitrary chunk size, download suspension and continuation as well as using Euclidean distance, view frustum and download progress as means for prioritizing the download order of 3D assets for both blocks and buildings. For the prototype, we also implemented a simple algorithm for predicting the users' movement. When the user is on the move, Euclidian distance is calculated from a spot that is *p* units in front of the user's true position.

C. Test Cases

For evaluating the performance of DRUMM, we conducted an automated walk-through (see Figure 5 and a video recording from a bird's eye view with DRUMM and 20Mb/s network speed https://youtu.be/vkTPnQmMu9o) in the 3D city application with three different test cases (TCs). In TC1, we used DRUMM with all available features. In TC2, we used DRUMM without chunks and only with the plain Euclidian distance based prioritization as a benchmark [7]. In TC3, no DRUMM was used (i.e. all 3D assets are downloaded in the beginning). With each configuration, three emulated networked speeds of 10, 20 and

50 Mb/s were used. For each of the nine test cases, the starting delay and the overall use of the network bandwidth were recorded. In case of (1) and (2), also a video was captured that was used for drawing a timeline showing the number of textured buildings in the view at every *t* seconds.

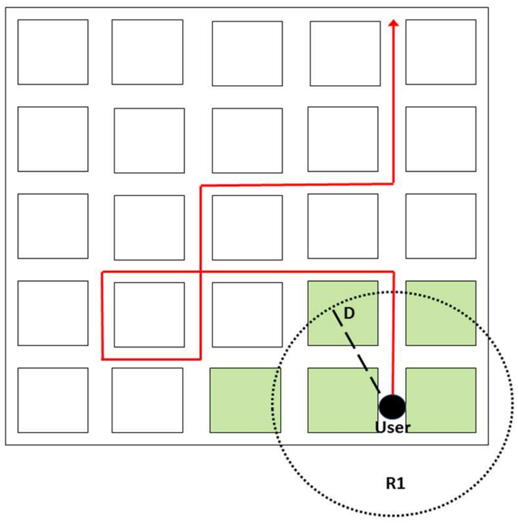

Fig. 5. An automated walk-through in the 3D city application.

The parameters used for DRUMM are listed in Table 1. In the experiments, *R2* was set to be the same as *R1* as the used 3D scene was small enough to fit all 3D assets into the memory. We experimented with various values of *S*, and noticed that increasing *S* beyond three did not improve the performance in our test case when downloading 3D assets only from a single source (i.e. the web server). When downloading from multiple sources in parallel, the value of *S* can have a significant impact on the performance. For the test cases TC1 and TC2, the values of weight factors ω are also shown in Table 1. For TC1, the values were determined experimentally.

TABLE I. PARAMETERS FOR DRUMM IN TC1 AND TC2

Parameter	Value	
R1 (prioritization zone)	200	
R2 (prioritization zone)	200	
p (Euclidian distance prediction)	50	
t (priority calculation interval)	1s	
C (chunk size)	256KB	
S (maximum # of simultaneous downloads)	3	
Weight factors	TC1	TC2
$\omega^{distance}$(weight factor for Euclidian distance)	0.5	1
ω^{view} (weight factor for view frustum)	0.3	0
$\omega^{download}$ (weight factor for download progress)	0.2	0

VI. EXPERIMENTAL RESULTS

A. *Starting Delay*

With the starting delay, we mean the time that the user has to wait before anything meaningful is shown on the screen. The starting delays were measured for DRUMM (TC1 and TC2) as well for no DRUMM (TC3) with the three different networks speeds. The measured starting delays are shown in Table 2. It should be noted that having chunks and download suspension has none or a marginal effect on the performance of DRUMM in this test case.

TABLE II. STARTING DELAY

	10Mb/s	20Mb/s	50Mb/s
TC1/TC2 (DRUMM)	2s	1.5s	1s
TC3 (no DRUMM)	4min 35s	2min 17s	55s

It can be clearly seen in Table 2 that the starting delay with a 3D city model consisting of 25 visually detailed blocks is intolerable if no dynamic downloading of 3D assets is used. In case of DRUMM, the starting delay constitutes only the download delay for the first JSON file containing the block geometry and the links to building textures.

B. *Amounts of Downloaded Data*

The total amounts of downloaded data are presented in Table 3 for all test cases with the three different network speeds.

TABLE III. AMOUNTS OF DOWNLOADED DATA

	10Mb/s	20Mb/s	50Mb/s
TC1	203MB	269MB	292MB
TC2	192MB	268MB	290MB
TC3 (no DRUMM)	329MB	329MB	329MB

It can be seen in Table 3 that even with DRUMM in use, large amounts 3D assets are eventually downloaded, however, without the user needing to wait for minutes before anything meaningful is drawn on the screen. Table 3 also shows that the 3D asset downloading is slightly more efficient with chunks (see TC1) particularly when the network bandwidth is less ample. This is due to the fact that the use of parallel TCP connections (used with HTTP) improve the overall throughput to a certain extent [20]. The number of parallel TCP connection should, however, be carefully chosen not to cause network congestion, which may on the contrary decrease the overall throughput.

C. *Texture Visibility*

In Figures 6, 7 and 8, the number of textured buildings in the view are presented for TC1 and TC2 with different network speeds as a function of time. In general, it can be seen in the figures that 3D asset prioritization performs better in TC1. This is due to both the more intelligent prioritization criteria and the capability to suspend and continue downloading of 3D assets. The performance improvement, however, is clearly more significant when the available bandwidth is scarce (see Figures 6 and 7). Particularly with the network speed of 10Mb/s, the 3D application is primarily capable of downloading only those 3D assets that are directly visible to the user. Thanks to chunking, downloading of 3D assets having fallen lower in the priority can be suspended and later continued when the user decides to double back. This way, the scarce available bandwidth can be more efficiently utilized.

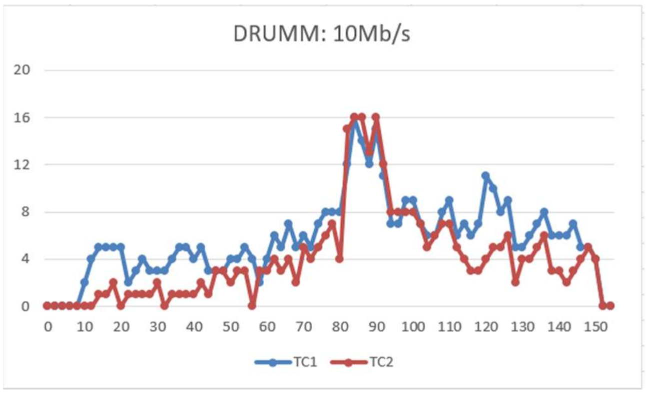

Fig. 6. The number of textured buildings in the view as a function of time (DRUMM: 10Mb/s).

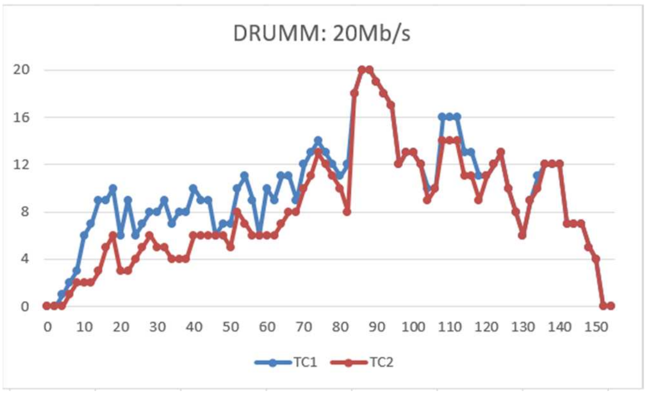

Fig. 7. The number of textured buildings in the view as a function of time (DRUMM: 20Mb/s).

With the network speed of 50Mb/s, the differences between TC1 and TC2 are less significant as the 3D application is capable of downloading all 3D assets in proximity at a high speed. The performance in TC2 falls behind TC1 practically only in the beginning of the walk-through when there is a large number of 3D assets to be prioritized and downloaded. In the later parts of the walk-through the small difference between TC1 and TC2 is due to the prediction feature of the Euclidian distance calculation that enables downloading of 3D assets that are further away in the direction the user is moving to.

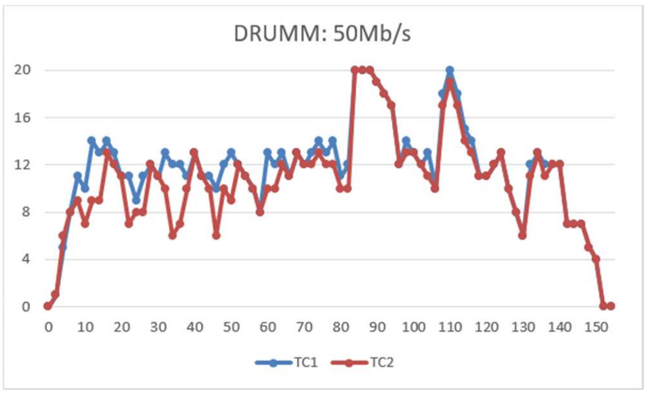

Fig. 8. The number of textured buildings in the view as a function of time (DRUMM: 50Mb/s).

VII. Conclusions

In this paper, we present a dynamic resource management method called DRUMM designed for viewing 3D city models on the web. DRUMM enables dynamic downloading of 3D assets during runtime, which exempts the user from long waiting times when starting the 3D application. DRUMM is data type agnostic, and therefore, can be used with any types of 3D assets. In DRUMM, the order of 3D asset downloading can be steered using various importance factors. DRUMM also supports dividing 3D assets into chunks enabling (1) parallel downloads from multiple different sources; and (2) suspension and continuation of already started downloads. Based on the experiment results, DRUMM improves the usability of 3D city applications particularly when network bandwidth is scarce.

Although the current prototype implementation of DRUMM is fully functional, it is still lacking some important features. In the future, we will implement support for WebRTC to enable P2P-assisted 3D assets downloading, which will harness the true value of dividing 3D assets into chunks. Second, for downloading texture files, use of variable sized chunks will also be examined. In this case, the idea is to assign the size of chunks according to the size of texture files, so that their downloading is efficient, but also suspending their download is possible at an early stage. Third, we will examine how different quality variants for 3D assets (i.e. LODs) could be used in DRUMM to enable far distance real-time rendering for large cities. The most straightforward approach is to define additional radius *R* within both *R1* and *R2* for each LOD. The download order for each LOD would be thus determined according to the proximity to the user. Fourth, we will evaluate the performance of DRUMM with non-uniform city plans (e.g. with the complete VirtualOulu 3D city model [1]). Finally, we will also perform comparative evaluation with other state-of-the art methods, such as Clip Mapping [21].

Acknowledgment

The Strategic Research Council at the Academy of Finland is acknowledged for financial support of the COMBAT project (293389). This work has also been supported by the CADIST3D project (268905) funded by the Academy of Finland and the 6Aika: Open City Model as Open Innovation Platform project (A71143) funded by the ERDF and the City of Oulu under the Six City Strategy program.

References

[1] T. Alatalo, T. Koskela, M. Pouke, P. Alavesa, and T. Ojala, “VirtualOulu: collaborative, immersive and extensible 3D city model on the web,” in 21st International Conference on Web3D Technology (Web3D ’16), 2016, pp. 95–103.

[2] T. Kolbe, “3D City Model of New York City.” [Online]. Available: https://www.gis.bgu.tum.de/en/projects/new-york-city-3d/. [Accessed: 23-Nov-2016].

[3] T. Kolbe, “Semantic 3D City Model of Berlin.” [Online]. Available: http://www.3dcitydb.net/3dcitydb/VisualizationBerlin/. [Accessed: 23-Nov-2016].

[4] F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin, “Applications of 3D City Models: State of the Art Review,” ISPRS Int. J. Geo-Information, vol. 4, no. 4, pp. 2842–2889, 2015.

[5] J.-P. Virtanen, H. Hyyppä, A. Kämäräinen, T. Hollström, M. Vastaranta, and J. Hyyppä, “Intelligent Open Data 3D Maps in a Collaborative Virtual World,” ISPRS Int. J. Geo-Inf, vol. 4, pp. 837–857, 2015.

[6] H. Bakri, C. Allison, A. Miller, and I. Oliver, “Virtual Worlds and the 3D Web -- Time for Convergence?,” in Immersive Learning Research Network: Second International Conference, iLRN 2016 Santa Barbara, CA, USA, June 27 -- July 1, 2016 Proceedings, C. Allison, L. Morgado, J. Pirker, D. Beck, J. Richter, and C. Gütl, Eds. Cham: Springer International Publishing, 2016, pp. 29–42.

[7] M. Hemmati, S. Shirmohammadi, H. Rahimi, and Ali Asghar Nazari Shirehjini, “Optimized Game Object Selection and Streaming for Mobile Devices,” Adv. Inf. Technol. Appl. Comput., vol. 1, pp. 144–149, 2012.

[8] H. Liu, M. Bowman, and F. Chang, “Survey of state melding in virtual worlds,” ACM Comput. Surv., vol. 44, no. 4, pp. 1–25, 2012.

[9] G. Morgan and F. Lu, “Predictive interest management: An approach to managing message dissemination for distributed virtual environments,” in 1st International Workshop on Interactive Rich Media Content Production: Architectures, Technologies, Applications, Tools, 2003.

[10] J. Boulanger, “Interest Management for Massively Multiplayer Games,” McGill University, 2006.

[11] J. Boulanger, J. Kienzle, and C. Verbrugge, “Comparing interest management algorithms for massively multiplayer games,” Work. Netw. Syst. Support Games, p. 6, 2006.

[12] B. Hariri, S. Shirmohammadi, and M. R. Pakravan, “A distributed interest management scheme for massively multi-user virtual environments,” in VECIMS 2008 - IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems Proceedings, 2008, pp. 111–115.

[13] K. Vatjus-Anttila, T. Koskela, S. Hickey, and J. Vatjus-Anttila, “Occlusion Based Message Dissemination Method in Networked Virtual Environments,” in Seventh International Conference on Next Generation Mobile Apps, Services and Technologies, 2013, pp. 44–49.

[14] U. Barchetti, A. Bucciero, S. S. Sabato, and L. Mainetti, “Loading and rendering optimization for networked virtual worlds,” in Proceedings - 2007 International Conference on Cyberworlds, CW’07, 2007, pp. 265–272.

[15] H. Rahimi, A. A. Nazari Shirehjini, and S. Shirmohammadi, “Activity-centric streaming of virtual environments and games to mobile devices,” HAVE 2011 - IEEE Int. Symp. Haptic Audio-v. Environ. Games, Proc., pp. 45–50, 2011.

[16] J. Sutter, K. Sons, and P. Slusallek, “Blast: A Binary Large Structured Transmission Format for the Web,” in 19th International ACM Conference on 3D Web Technologies, 2014, pp. 45–52.

[17] C. Stein, M. Limper, and A. Kuijper, “Spatial data structures for accelerated 3D visibility computation to enable large model visualization on the web,” in 19th International ACM Conference on 3D Web Technologies, 2014, pp. 53–61.

[18] J. Gaillard et al., “Urban Data Visualisation in a web browser,” Web3D ’15 Proc. 20th Int. Conf. 3D Web Technol., 2015.

[19] T. Koskela, A. Heikkinen, E. Harjula, M. Levanto, and M. Ylianttila, “RADE: Resource-aware distributed browser-to-browser 3D graphics delivery in the web,” in 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2015, 2015, pp. 500–508.

[20] T. J. Hacker, B. D. Athey, and B. Noble, “The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network,” in Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2002, 2002, pp. 434–443.

[21] A. Asirvatham and H. Hoppe, “Terrain rendering using GPU-based geometry clipmaps,” in GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, 2005, pp. 27–46.