GLOBAL MODELS AND THE W3DS SPECIFICATION - CHALLENGES AND SOLUTIONS

G. Misund*, M. Granlund, H. Kolås

Østfold University College, Halden, Norway - (gunnar.misund, morten.granlund, herman.kolas)@hiof.no

KEY WORDS: W3DS, OGC, VRML, GeoVRML, X3D, 3D Navigation, Levels of Detail, Digital Earth, Heterogeneous Geodata Integration

ABSTRACT:

The recently proposed Web 3D Service (W3DS) specification defines how to access and retrieve geospatial 3D scenes integrating terrain data, textures and local content such as buildings and city furniture. In this paper we discuss some of the issues arising when using W3DS to access huge models, potentially of the entire globe, where high resolution content is embedded. We claim that the only sound approach to deploying such models is to generate the content on demand, based on the requested geographic area and other parameters. Examples are presented, including constructing 3D buildings on-the-fly from the original 2D/2.5D sources, along with empirical results, indicating that this is a feasible strategy. We also show that efficient content caching may reduce response times significantly. Based on lessons learned, we propose a minor extension of the W3DS specification in order to assure seamless navigation in global models assembled from distributed, autonomous data sources. The most crucial components of the proposed strategies are implemented and tested on a real life model.

1 INTRODUCTION

The globe has for long been considered the most descriptive and telling representation of Mother Earth, and the advent of high performance computers and networks has paved the way for virtual versions of the globe. In 1998 US Vice President Al Gore launched the idea of the Digital Earth (DE), a *"multiresolution, three-dimensional representation of the planet, into which we can embed vast quantities of geo-referenced data"* (Gore, 1998).

Digital Earth was described as a common ground for both consumers and providers of a wide variety of geospatial data: *"The Digital Earth would be composed of both the 'user interface' - a browsable, 3D version of the planet available at various levels of resolution, a rapidly growing universe of networked geospatial information, and the mechanisms for integrating and displaying information from multiple sources"*. Gore also rekognized the distributed and non-bureaucratic aspect of the Digital Earth: *"Obviously, no one organization in government, industry or academia could undertake such a project. [...] Like the Web, the Digital Earth would organically evolve over time, as technology improves and the information available expands"*. The snapshots in Figure 1 illustrates the experience of browsing the global model used for testing some of the concepts presented later in the paper.

The Digital Earth white paper spawned a set of promising projects, where both academia, vendors, and political bodies participated, see for instance (Foresman, 2003, DeVarco, 2004, Misund et al., 2005) for overviews of past and present DE activities. However, only few of the projects are, as we write, still active.

The recently proposed Web 3D Service specification (W3DS) (Quadt and Kolbe, 2005) presents new opportunities for web based deployment of 3D models generated from geospatial data. In short, the user specifies a geographic area, viewing parameters and layers, and then retrieves the content formatted according to the VRML97 standard (Carey and Bell, 1997). GeoVRML (Reddy and Iverson, 2002) and X3D (X3D, 2005) are proposed as optional, however more suitable, formats[1].

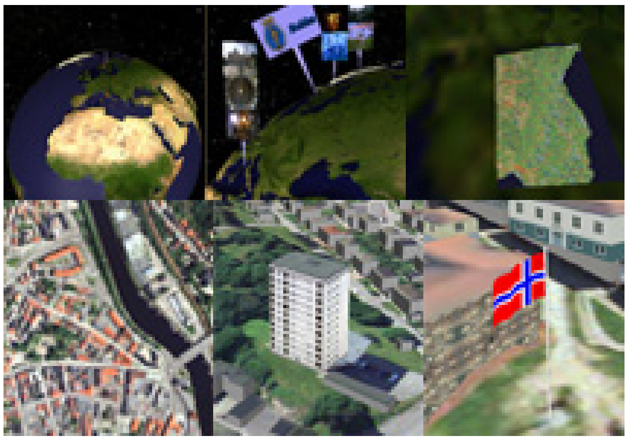
Figure 1: Browsing a Digital Earth

In this paper we discuss certain aspects of the W3DS specification in a Digital Earth context. Even though the specification at first glance seems to target retrieval of smaller models covering limited areas, we think that the users, as soon as they start to explore their models, would like to zoom out or pan around to investigate neighborhood areas. They might even want to do a fly-over from their residence to a vacation destination, several thousand kilometers away. The main purpose of this paper is to investigate the suitability of W3DS as the main mechanism for compiling and browsing global models carrying scattered geospatial data of varying categories and resolution.

Navigation from a satellite view down to street level obviously calls for efficient methods for handling levels of detail (LOD). Thus, we start our discussion in section 2 with a brief introduction to LOD management in VRML. In section 3 we promote the concept of transient models, where the 3D content is generated on demand from distributed data sources, based on the requested

*Corresponding author

[1] In the remainder of the paper, we use VRML as a generic term for both VRML 97, GeoVRML and X3D and its components.

geographic area and geospatial content. However, this approach imposes some challenges, and these constitute the main focus of this paper. Many of these problems arise when integrating distributed, autonomous W3DS sources to compile global LOD hierarchies. We address some of these issues in section 4, and propose a minor, but significant, extension of the current W3DS specification as means to solve some of the LOD issues.

Building content on demand may slow down response times, and in order to reduce these types of delay, we introduce a server side caching strategy in section 5. We also present some empirical results from applying the method on real life data. In section 6 we describe a method for on-the-fly generation of buildings, along with a study of the overhead incurred by this technique. The paper is closed with some final remarks in section 7.

The presented work is part of Project OneMap (OneMap, n.d., Misund, 2002). Some of the case models are available online (OneMap, 2005).

2 LEVELS OF DETAIL IN VRML

LOD mechanisms are crucial for efficient management of huge 3D gespatial models. As a backdrop for the discussions in this paper, we give a brief outline of Quad Tree LOD hierarchies, which is the VRML strategy for efficient management of huge 3D gespatial models. We also point to some issues arising when accessing LOD hierarchies with W3DS.

2.1 Quad Tree LOD Hierarchies

In the original VRML97 documentation (Carey and Bell, 1997), little or no information is provided on effective use of LOD in large scale models or terrain visualizations in general. Example code is limited to relatively small objects that can be presented by single VRML nodes, for instance an airplane with 5000, 1000, 250 and 50 polygons respectively. However, available literature, regardless of underlying technology, seems to agree on the fact that a hierarchical data structure, in particular the Quad Tree, is the most appropriate structure for terrain applications (Falby et al., 1993, Hitchner and McGreevy, 1993, Leclerc and Lau, 1995, Reddy et al., 1999a). Both GeoVRML, the geospatial extension to VRML, and the geospatial module in X3D, provides Quad Tree LOD constructs by default.

Unfortunately, the memory management of many VRML browsers prevents efficient utilization of Quad Tree data structures in visualization of large models. The VRML97 specification states *"Exactly when [referenced resources]* [2] *are read and displayed is not defined"*. This means that it is left to those who implement a VRML browser to decide when to download external resources, typically a representation of an object with higher level of detail. Several of the available VRML viewers are optimized for visualization of relatively small objects, such as cars or engines. They prepare and load every representation of an object on startup, hence, making browsing of global multiresolution terrain models infeasible. Both the GeoVRML and X3D specifications clearly state that a rational loading of referenced resources is imperative. In addition, the specifications also emphasize the importance of *unloading* the resources from memory whenever they are not rendered. To the authors' knowledge, none of the freely available VRML clients have implemented a purging strategy that prevents exhaustion of internal memory after prolonged navigation in large LOD scenes.

[2] In VRML, external resources are referenced using the `Inline` node.

VRML is modular in the sense that one may incorporate other resources into any model simply by specifying their URIs. These resources do not need to be static files, and can even be services delivering dynamic content. This opens for a distributed approach to the Digital Earth concept, where topography with embedded objects is regarded as a set of chained web services, preferably W3DS services, rather than a set of static files distributed on the Internet or in local repositories.

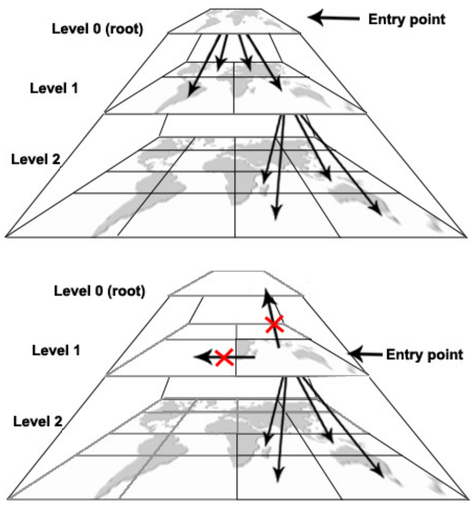

Figure 2: Traversals of a quad tree hierarchy with three levels of detail

LOD hierarchies have typically been implemented as *one-way*, top-down Quad Trees in VRML models. This is a very efficient and easy-to-implement solution, but it inflicts certain limitations. Currently, there are no way of replacing a high-resolution model automatically and seamlessly by a larger and less detailed model when zooming out, if the less detailed tile has not been previously loaded. As a result, the initially loaded tile is locked as the root tile, and isolates it and its descendant tiles. This is illustrated in Figure 2. While the client is able to retrieve any representation of the model as long as the root tile is being loaded initially (the upper hierarchy), the client is, after loading a level 1 tile initially (the lower hierarchy), locked to this particular tile and its descendants.

2.2 W3DS Access To LOD Hierarchies

Let us assume that we have a W3DS provider capable of providing a global LOD hierarchy. A user would pass a `GetScene` request to retrieve the wanted content, and one of the mandatory parameters is the bounding box of the requested scene. View parameters may optionally be specified, for instance to simulate the view from a certain viewpoint.

If the user specifies a small (relative to the geographic extent of the LOD hierarchy) bounding box, the W3DS might handle this in the following way: first, locate the smallest tile covering the requested area, then crop the tile to fit the bounding box, and finally adjust the LOD references to be consistent with the position and extent of the served root tile. The user will receive a chunk of the main model, and (if not already at the highest resolution level) it would be possible to zoom in at more detailed levels. However, when zooming out, the model just becomes a small island in a void. Likewise, panning to neighborhood areas will not be possible.

However, if the user wants to get access to the complete model, but still start at street level, the solution would be to specify a large bounding box, for instance encompassing the entire globe, but keep the view parameters for local examination. The result of this is that the root tile would be returned, however along with the view parameters from the request translated to the corresponding position of the avatar[3]. The viewer would then recursively load the appropriate LOD references, until the correct level of detail is reached. In this case the user would be free to move around in the LOD hierarchy. The two solutions are illustrated in Figure 3.

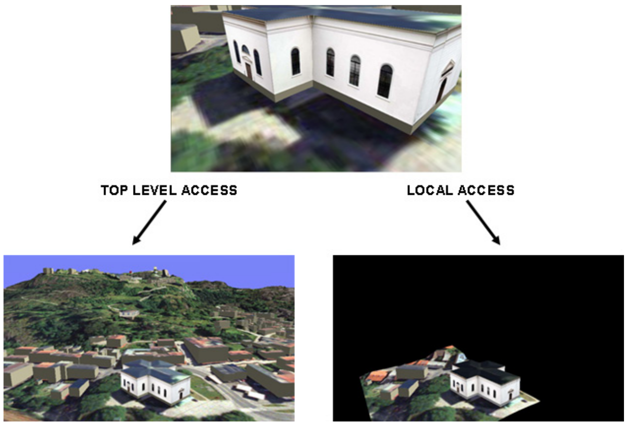

Figure 3: Zooming out from a local viewpoint

The drawback with the last mentioned strategy is that considerable processing resources may be required to drill down in the levels of detail, and without sophisticated memory management, this approach might turn out infeasible applied to large models and deep LOD hierarchies. We will return to issues concerning VRML LOD management in Section 4.

3 TRANSIENT GLOBAL MODELS

The conceptual view of a transient global 3D model is a VRML representation of the entire Earth provided in a range of levels of detail (LOD), starting with a global view and ending up at street level. In practice, with an approximately doubling of the accuracy for each level, this results in 20 - 25 levels. Such a model represents an incomprehensive amount of data[4]. It is obviously not possible to preprocess and explicitly generate all the needed files, hence the only feasible approach is to generate the tiles on demand, a fact also alluded to in (Reddy et al., 1999b).

A tile in a transient geospatial LOD hierarchy is constructed from basically three types of data: 1) Terrain data, 2) Image textures and 3) 3D objects. The needed content could preferably be retrieved using the family of the Open Geospatial Consortium (OGC) (OGC, n.d.) Web Services family, respectively the Web Coverage Service (WCS) (Evans, 2003), the Web Map Service (WMS) (de La Beaujardiere, 2004) and the proposed W3DS specification. In addition, the Web Feature Service (WFS) specification (Vretanos, 2002) could serve as a basis for generating 3D objects on-the-fly from traditional map data sources (see section 6). Compiling 3D scenes from heterogenous geospatial sources is treated in more detail in for instance (Schilling and Zipf, 2003, Altmaier and Kolbe, 2003), and the concept is illustrated in Figure 4.

Although there is rising number of providers of various OGC web services, in particular WMS's, there is a general problem of locating and integrating these heterogeneous data sources. This challenge is the main rationale for introducing the concept of the Federating Geodata Service, treated in more detail in the following

[3]The avatar is a construct representing the user's view of the 3D scene.
[4]25 levels of tiles with an average size of 250 KB would yield approximately 100 Exabyte, or 10^{14} MB.

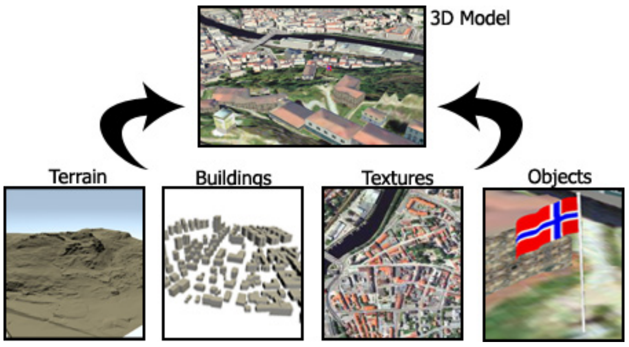

Figure 4: Compiling a 3D scene from distributed sources

section. In this paper we are primarily concerned with Federating Web3D Services (FW3DS), but the principle is applicable to all of the OGC geodata services.

3.1 Federating Geodata Services

A Federating Geodata Service (FGS) is basically an OGC compliant service, with some additional internal functionality, however hidden from the requesting clients. The main purpose with an FGS is to facilitate service level federation of a set of distributed and autonomous data sources. To enable this, it must implement a Federation Module, whose main component is a dynamic registry of relevant service providers. This is indeed a challenging task. However, specifications and tools have been proposed, implemented and tested in order to support discovery and description of geospatial web services based on the WSDL/UDDI paradigm (Lieberman et al., 2003).

In addition, for performance purposes, an FGS should implement a Content Cache which provides mechanisms for pre-fetching, caching and purging of content. An example of a caching implementation is given in section 5. The general structure of a Federating Geodata Service is illustrated in Figure 5.

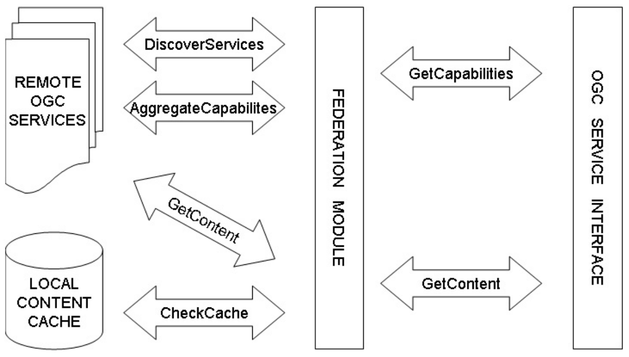

Figure 5: Federating Geodata Service

By using the well defined OGC services, which again relies on well known and widely adopted formats and specifications, syntactic integration of content from various providers is a relatively trivial task. However, semantic heterogeneity remains a serious challenge. A common way to address this problem is to use ontologies[5], see for instance (Klien et al., 2004) for details on search and retrieval of geographic information using this approach.

Based on the available ontology, or perhaps a more simple feature catalog, the capabilities documents[6] retrieved from the contribut-

[5]An ontology may be defined as an explicit formal specification of a shared conceptualization.
[6]The `GetCapabilites` request is common to all OGC web services and returns capabilites document providing crucial information such as geographic extent, layer information and spatial reference systems.

ing sources should be analyzed, transformed and aggregated into a single capabilities document, which again the requesting clients can use to formulate their requests. With this approach, the FGS will hide the underlying semantic heterogeneity and enable efficient search and retrieval.

The federating W3DS is the most complex of the federating geodata services, as it uses other OGC services to retrieve and adjust content for consistent integration. It would also have to deal with LOD issues, which we discuss in section 4.

4 INTEGRATING DISTRIBUTED W3DS SOURCES

There are essentially two ways to realize large geospatial models with W3DS. The first strategy is to build a monolithic and centralized solution, in practice based on a single provider with knowledge and control over all the required data. For each generated tile, the reference to the four LOD tiles with higher resolution may be "hard-coded", either as file locations or dynamic services generating the content on-the-fly. The advantage with this model is that the provider has complete control of the process, and will be able to utilize a variety of optimization techniques to ensure good response times. However, the drawbacks are numerous, the most prominent being the fact that the sheer size of the data volumes involved would render the approach infeasible when speaking of huge Digital Earth like models.

The other approach is to consistently make use of federating W3DS requests as LOD references. For each generated tile, the FW3DS would use the dynamic registry to select appropriate providers for the four LOD URIs. The drawback is that the initial service would have no control of the cascading process of building the hierarchy. The main advantage is that the approach is inherently scalable, and would be fault tolerant in the sense that the W3DS would be able to detect off-line services and pass the the request on to an active service. Needless to say, this concept requires a fairly high number of distributed (F)W3DS providers. Considering the rapidly growing supply of WMS servers around the world, this might not be as unrealistic as it may seem at first glance.

Obviously, real life applications would probably use a mix of the two approaches, typically passing FW3D requests to build the coarser levels, and then trust the local providers to take care of the high resolution part of the LOD hierarchy.

The fully distributed strategy will, however, require some additional functionality in the current W3DS specification. First of all, we need to know if a provider can serve quad tree based LOD content. The natural way of doing this, is to extend the response from the `GetCapabilites` request to incorporate information on the LOD aspects of the served content, typically the number of levels (from zero and up), and preferrably some additional metadata, such as measures of the accuracy of each level. It is important to note that the size of the served LOD tiles must be adjusted to ensure consistency with the size of the initially requested tile.

On the other hand, it should be possible to specify if the response should incorporate LOD references (if applicable) or not. This could easily be accomplished by adding the appropriate (optional) parameters to the `GetScene` request.

One could argue that it is not strictly necessary to implement these addition to the specification, since it would be possible for the FW3DS to replace the served LOD references with other pointers. However, extending the specification is probably a cleaner approach, making it it easier to implement more efficient applications.

5 CONTENT CACHING

Transient and distributed models are subject to delays due to several factors. As terrain meshes, textures, and local content like buildings, forest, roads, or power lines, may all be retrieved from physically separated servers, delay caused by data transfer is inevitable. In addition, on-demand generation of content, as described in section 6, will also slow down response times, as will calculations and transformations in order to adjust and assemble heterogeneous geospatial data. In this section we describe a server side caching mechanism to compensate for lag.

5.1 Likelihood of Utilization (LoU)

We have called our chosen strategy *Likelihood of Utilization* (LoU). The main idea is to perform qualified guesswork on how the user is going to navigate in the model, thus being able to maintain a dynamic cache of tiles that is estimated to be requested in the near future.

LoU is, in short, the *value* of the tiles in the cache, where the term value refers to how likely it is that the tiles will be requested in relatively near future. The goal is try to keep LoU as high as possible for the cache content at all times. Several factors play a role when estimating the LoU of a tile, e.g. the distance between the topography and the client's current viewpoint, or the number of times a tile has been requested. Since the distance between topography and the client's viewpoint is continuously in alteration, we need some means to periodically update the values of the tiles, and keep them organized in a priority queue. This way, we can maintain a sound cache by purging the tiles with lowest LoU every time a new tile is inserted into the cache. In addition, we introduce a background process that tries to generate 3D content in advance, before the client requests it. If the daemon generates content a relatively long time in advance, it will have to take into consideration that the newly generated 3D content may be considered to have a low LoU (because there may be a long distance between the client's viewpoint and the content). One possible solution to this is to let the time since a tile was created be one of the factors when estimating a tile's LoU.

5.2 Effects of Server-Side LoU Caching

A prototype implementation, along with a set of test cases, have been carried out in order to get some empiricism on the suggested LoU Caching. The prototype focuses on the generation of a global terrain, and is a transient, quad tree LOD model with 16 levels, returning random terrain for 1.4E09 potential tiles. The size of the tiles was varied from approximately 300 to 350 kilobytes. The model is implemented as a web service, accessed through a server which accepts highly simplified W3DS requests, and returns VRML content. The process of generating the random terrain is deliberately designed to take somewhat more than two seconds per tile. This is done by adding an artificial delay of two seconds to the tile building process.

A test case performed with 18 different types of LoU Cache setup is shown in Figure 6. A static, predefined fly-through, taking 300 seconds, is run through for each of the setups. This takes the client through the model, visiting arbitrary tiles of all the sixteen levels. By repeating exactly the same navigation for all the setups, we end up with results suitable for direct comparison. The three first bars show the average response time (the time between the server received the HTTP request from the client, until it was able to respond with the VRML file) for servers with little, or no, LoU Caching. The difference between these three setups is that the first one utilizes no cache at all, the second setup has a cache capacity of 2,000 tiles, whereas the third has a capacity of 15,000. However, no daemon is predicting which tiles will be requested in advance in these setups. The average response time was, rounded off to one decimal, 2.1 seconds for all. For all these setups, 0% of the requested tiles were fetched directly from the cache, which implies that the client memory never had to be flushed. These three setups will function as a basis, to which the more intelligent LoU Cache setups can be compared.

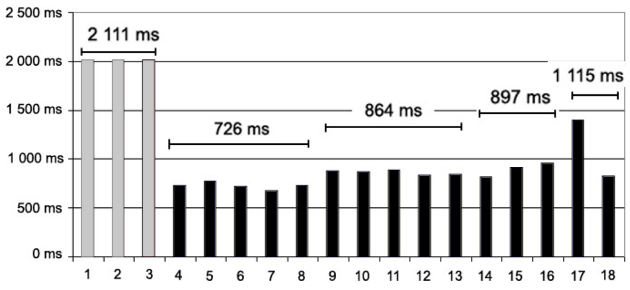

Figure 6: Average response times with and without LoU Caching

Different approaches have been tested when it comes to predicting which tiles should be built in advance. Although pre-caching is a common strategy in client-server applications, it proves difficult to find literature about similar implementations, where the optimization mechanisms are located on the server-side. Complex approaches, such as using extrapolation of recent navigation, did not tend to result in better performance than the more simple approach of pre-building one or more neighbors of the tiles actually being requested. Often, the extra workload introduced by more complex heuristics, results in equally high, if not higher, response times than that of more simple ones. Which tiles are defined to be neighboring tiles depends solely on how many tiles the server are allowed to prepare in advance, which in the end is a matter of resource allocation. Figure 7 shows a rather extensive definition of neighboring tiles. One way to dynamically control the amount of neighboring tiles to be prepared is to select only a selection of the neighboring tiles, depending on the current workload on the server.

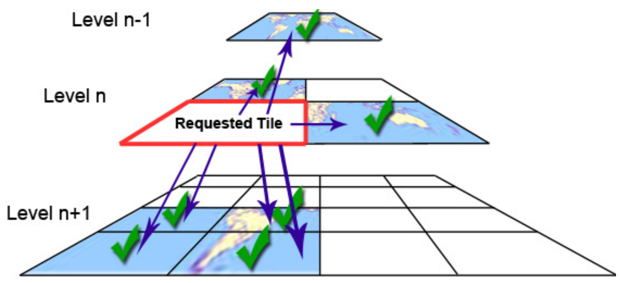

Figure 7: Simple neighborhood pre-fetching

All of the remaining setups have implemented both a mechanism that builds/prepares tiles in advance (based on neighboring tiles), and a periodically updated priority queue (implemented as a heap) which keeps track of the dynamically changing LoU of the already cached tiles. This priority queue enhances the content of the cache because it ensures that the least valuable tiles are the first ones to be purged from the cache. Four different criterions, or *measures*, are used in order to calculate the LoU of the tiles in the content cache, viz. `Distance`, `Reuse`, `IdleTime`, and `CreationTime`. The `Distance` measure is the distance between the last registered viewpoint and the tile, `Reuse` is the number of times that the tile has been requested, `IdleTime` is the time passed since the tile was last requested, and `CreationTime` is the time since it was inserted in the cache.

Setup 4 to 8 all have a cache with a capacity of 15,000 tiles, but have tweaked the weight of the four LoU measures differently. We see that while the tweaking of the measures have no significant effect on the response time, the introduction of the pre-building/caching of the neighbor tiles reduces the response time by 34%. This is a reduction by a factor of three. The server was on average, able to retrieve 65% of all requested tiles from the tile cache.

The next four bars in the figure (9 to 13) show the response times when reducing the cache capacity from 15,000 to 500. The surprisingly small increase in the average response time (726 to 864 milliseconds, which is only a 19% increase) shows that the concept of LoU Caching is scalable, and easily portable to multiuser, session-based applications.

Bar 14 to 16 show the relative increase in performance as the interval between each time the LoU priority queue is updated (to reflect the new avatar position, number of times the tiles have been requested, and the current time). The setups represented by these bars are identical to setup 9, except for the following increase in the update interval from 1.5 seconds to 3, 4.5 and 9 seconds respectively. The increase in performance is not remarkable, probably due to the reduced workload on the server.

Setup 17 emphasizes the importance of a sound and semi-intelligent flushing mechanism, as it is equal to setup 9 except that it expunges the tiles in the cache after the FIFO queue (First in - First out) principle. The removal of the priority queue based flushing results in a 63% increase in the response time (from 878 to 1,402 milliseconds).

Finally, the last setup does not use the Distance measure in its LoU estimations; only the three other measures described above. This results in an insignificant change in the performance, proving that the choice of measure is not one of the major performance factors.

The main lesson learned in the experiments, was that in order to gain significant reduction in response times, it suffices to apply a simple neighborhood pre-fetching method along with a simple LoU metric. In addition, the cache size should be kept at a moderate level. For more detailed information about the tests performed on this prototype, see (Granlund, 2004).

6 ON-DEMAND CONSTRUCTION OF 3D OBJECTS

Some types of 2D and 2.5D geospatial data may be used to generate 3D objects automatically. As an example, a 3D building may be constructed by extruding the footprint according to its given height (Harvey et al., 1999). This obviously yields relatively coarse models, but may still be useful in various applications, for instance city planning. There are three main reasons for considering this approach:

- 2D geodata is much more available than corresponding 3D content.
- 2D data is maintained in various scales, generated by well developed and proven map generalization techniques. 3D generalization if far more complicated and computational intensive (Kada, 2002).
- Many types of geodata are subject to frequent updates. By accessing the original sources, the generated 3D model is guaranteed to be as valid as possible.

We have investigated the feasibility of this approach, in particular focusing on the response times overhead. This was done through a prototype server, offering 3D buildings on the VRML format. The source data was footprint polygons and height for all buildings (approximately 8000) in downtown Halden, Norway. Figure 8 shows two snapshots from the test case, where the buildings are integrated with an elevation model draped with high resolution aerial imagery. The model is a transient quad tree structure, as described in section 3. In the lower picture, buildings in the background are only represented by the texture, but when moving closer, more buildings will be constructed on-the-fly by extrusion and incorporated in the current scene (the upper image). See (Kolås, 2004) for more information on the case model and a detailed discussion on the construction of the buildings.

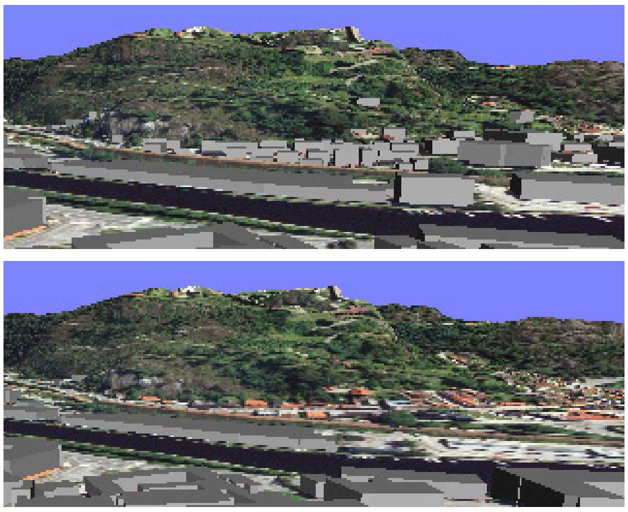

Figure 8: Creating buildings on-demand

In our experiment, 1000 requests were sent to the server. Each request included a bounding box, defining a random size and location within the model. The size of the bounding box was selected from one out of five predefined sizes: level 0, covering 1,920,000 square meters, to level 4, covering 7,000 square meters. Furthermore, each request was sent twice to the server (making a total of 2000 requests), first time with a parameter that told the server to create the building on-the-fly and store the resulting VRML representation to a temporary file. Then, a few seconds later, the second request, with the exactly same bounding box, was sent, this time with a parameter that told the server to fetch the VRML representation from the temporary file. This way, the response times for each request could easily be stored and the on-the-fly requests could be directly compared with the equivalent direct-access requests.

Figure 9 shows the response times difference between fetching static 3D content from the server (direct file access) and using the

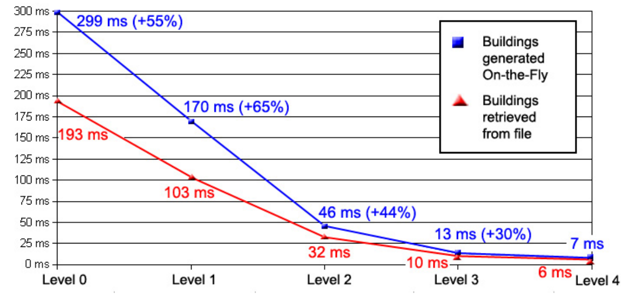

Figure 9: Overhead incurred by on-demand construction of 3D objects

dynamic on-the-fly generation approach. Depending on the size of the bounding box (level 0 to level 4), we see that the overhead is, on average, 55%, 65%, 44%, 30%, and 17% respectively. The average overhead for all levels is, roughly calculated, 50%. In the experiment, the client and the server were connected to the same high performance local area network. In a more realistic setting, where the client is accessing a truly remote server, the relative overhead caused by on-the-fly generation would decrease as the transfer delay would constitute more of the total response delay. Hence, the results clearly indicate that dynamic generation of 3D content is a feasible strategy. In addition, by leveraging LoU caching as described in section 5, one can expect to reduce the response times to a third. This means that a W3DS server building houses on-demand by using LoU Caching, would yield *better* response times than an equivalent server using direct file access and no caching.

7 CONCLUSION

The authors believe that the W3DS specification is a good candidate for a standardized and simple way to access and browse global 3D models with local content. However, there are some remaining issues, of which some have been addressed in this paper.

First of all, we have argued that it not possible to realize huge geospatial 3D models as static, completely pre-processed data sets. The proposed and demonstrated concept is to build transient models, were partial models are assembled on demand based on current user parameters. It is also our opinion that the content, mainly terrain grids, textures and 3D objects such as buildings and artifacts, should be, if possible, generated on-the-fly directly from the source data.

Obviously, this approach requires new types of server side functionality. We have presented some proof-of-concept solutions, more precisely caching methods and on-demand generation of buildings by accessing and processing the original 2D source data. However, a remaining challenge, far from trivial, is to develop methods for content sources management, for instance components acting as dynamic registries able to point to the appropriate data providers. Our proposed framework, the Federating Geodata Services, is a contribution in this direction.

In order to release the full potential of the concept of the Federating Web 3D Service, we have proposed a minor extension of the current W3DS specification. This allows Federating W3DW providers to either use the LOD hierarchy offered by the remote service, or generate an alternative one. However, it would require a substantial effort to test the concept to the full extent, requiring a realistic assortment of varied (F)W3DS servers complying to the recommended extension of the specification.

The requirements imposed by deploying transient models also call for new functionality on the client side. Current VRML viewers are not able to handle huge LOD models properly. The main problem is a general lack of sophisticated memory management. There are several open-source viewers, for instance the XJ3D browser (Hudson, 2003), that could be used as a starting point to solve this problem.

REFERENCES

Altmaier, A. and Kolbe, T. H., 2003. Applications and Solutions for Interoperable 3D Geo-Visualization. In: D. Fritsch (ed.), Proceedings of the Photogrammetric Week 2003 in Stuttgart, Wichmann Verlag.

Carey, R. and Bell, G., 1997. The Annotated VRML97 Reference Manual. Addison-Wesley Professional.

de La Beaujardiere, J., 2004. Web Map Service (WMS) Implementation Specification. 1.3 edn, Open Geospatial Consortium, Inc.

DeVarco, B., 2004. Earth as a Lens: Global Collaboration, Geo-Communication, and the Birth of EcoSentience. PlaNetwork.

Evans, J. D., 2003. Web Coverage Service (WCS) Implementation Specification. 1.0.0 edn, Open Geospatial Consortium, Inc.

Falby, J. S., Zyda, M. J., Pratt, D. R. and Mackey, R. L., 1993. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. Computer and Graphics 17(1), pp. 65–69.

Foresman, T., 2003. Digital Earth: the Status and the Challenge. In: Proceedings of the Global Mapping Forum.

Gore, A., 1998. The Digital Earth: Understanding our planet in the 21st Century.

Granlund, M., 2004. Perspective Based Level of Detail Management of Topographical Data. Masters Thesis in Computer Science, Østfold University College, Halden, Norway.

Harvey, F., Kuhn, W., Pundt, H., Bishr, Y. and Riedemann, C., 1999. Semantic Interoperability: A Central Issue for Sharing Geographic Information. Annals of Regional Science 33(2), pp. 213–232.

Hitchner, L. E. and McGreevy, M. W., 1993. Methods for User-Based Reduction of Model Complexity for Virtual Planetary Exploration. Proceedings of the SPIE: the International Society for Optical Engineering 1913, pp. 622–36.

Hudson, A. D., 2003. An introduction to the Xj3D toolkit. In: Web3D/Virtual Reality Modeling Language Symposium, p. 190.

Kada, M., 2002. Automatic Generalisation of 3D Building Models. In: Proceedings of the Joint International Symposium on Geospatial Theory, Processing and Applications. University of Stuttgart : Special Research Area SFB 627, Ottawa, Canada.

Klien, E., Einspanier, U., Lutz, M. and Hübner, S., 2004. An Architecture for Ontology-Based Discovery and Retrieval of Geographic Information. In: F. Toppen and P. Prastacos (eds), 7th Conference on Geographic Information Science (AGILE 2004), pp. 179–188.

Kolås, H., 2004. 3D visualisering av kartdata. Master's thesis, Østfold University College. In Norwegian.

Leclerc, Y. G. and Lau, S. Q., 1995. TerraVision: A Terrain Visualization System. Technical Note.

Lieberman, J., Reich, L. and Vretanos, P. (eds), 2003. OWS1.2 UDDI Experiment. Open GIS Consortium, Inc.

Misund, G., 2002. One Map. HØit. ISSN 0805-6692 / 0805-7486.

Misund, G., Granlund, M. and Kolås, H., 2005. Oneglobe - Building and Browsing a Transient Digital Earth from Distributed, Heterogeneous Sources. In: Accepted to ScanGIS 2005.

OGC, n.d. Open Geospatial Consortium. www.opengeospatial.org.

OneMap, 2005. Online OneGlobe Models. www.ia-stud.hiof.no/ hermanko/onemap/onlineGlobalModels.html. Project website.

OneMap, n.d. Project OneMap. www.onemap.org. Project website.

Quadt, U. and Kolbe, T. H., 2005. Web 3D Service (W3DS). Discussion Paper edn, Open Geospatial Consortium, Inc.

Reddy, M. and Iverson, L., 2002. GeoVRML 1.1 Specification. 1.1 edn, SRI International, http://www.geovrml.org/1.1/doc/.

Reddy, M., Leclerc, Y. G., Iverson, L., Bletter, N. and Vidimce, K., 1999a. Modeling the Digital Earth in VRML. The International Society for Optical Engineering 3905, pp. 114.

Reddy, M., Leclerc, Y., Iverson, L. and Bletter, N., 1999b. TerraVision II: Visualizing Massive Terrain Databases in VRML. IEEE Computer Graphics and Applications 19(2), pp. 30–38.

Schilling, A. and Zipf, A., 2003. Generation of VRML city models for focus based tour animations: integration, modeling and presentation of heterogeneous geo-data sources. In: Web3D Conference, pp. 39–48.

Vretanos, P. A., 2002. Web Feature Service (WFS) Implementation Specification. 1.0.0 edn, Open Geospatial Consortium, Inc.

X3D, 2005. X3D International Specification Standards.