# Adaptive Content Management for Collaborative 3D Virtual Spaces

Jarkko M. Vatjus-Anttila, Seamus Hickey, Timo Koskela
Intel and Nokia Joint Innovation Center
Center for Internet Excellence
University of Oulu, Finland
{jarkko.vatjus-anttila, seamus.hickey, timo.koskela}@cie.fi

**Abstract**

Collaborative 3D virtual spaces and their services are often too heavy for a mobile device to handle. The burden of such services is divided between extensive amounts of data, which need to be downloaded prior to using the service, and the complexity of the resulting graphical rendering process. In this paper, a proxy based architecture for collaborative virtual spaces is used to manipulate graphical data demand-time to favor both network bandwidth usage and graphical rendering process. In addition, a proof of concept test is shown, regarding how the simplification process gains savings for different client device profiles, including laptops, tablets and mobile devices.

**Index Terms:** Collaboration, Adaptation, Virtual space, 3D, Architecture.

## I. Introduction

3D virtual space-based services around the world are becoming more popular. The latest statistics in MMOData1 service reveals that nearly all services around this field are growing constantly as measured by their user base. Gaming has been the driving technology in Massively Multiplayer On-line (MMO) solutions. For instance, the statistics for World of Warcraft are showing nearly ten million subscribers. Services, which support collaboration and include the extensibility aspect to the virtual space, like Second Life, are also very popular. For example, Second Life hosts nearly one million monthly active users. Services based on the extensible virtual spaces -concept, allow end users to create and share content in the 3D space. As indicated by Mouton et al. [1], in those services, it is not only about gaming, it is also about sharing and consuming online content, collaborating and being social while enjoying it all in visually appealing 3D graphical format. Similar phenomena have been noticed by Suznjevic et al. [2] in the field of gaming, where social aspects (chatting, spending time on-line with others) have been found to be a widely popular activity. For end users, the collaboration means that they can contribute in real time to the content being manipulated in the 3D space, and at the same time, they see everyone else's contribution. Combining this phenomenon with end-users' willingness to use the services via any kind of a remote device, also mobile, sets difficult requirements for the 3D data processing [1].

As the user base of the virtual spaces grows, so does the number and types of devices accessing the services. Due to highly heterogeneous features of mobile devices, it is difficult to create content which is optimal for all of the devices. There are differences, for example, in the amount of available memory, processing power and GPU features. In theory, it is possible to create optimal content for every mobile device, but due to variations mentioned above, the amount of required content (e.g. texture) combinations climb high. As the device types continue to vary further, it becomes unpractical to store all of the variations in a cache. Instead, the optimal solution would be to recognize the mobile device run-time, and be able

to deliver optimized content for each device type on demand-time. Hence, if the virtual space service providers want to serve all customers well, they need to increasingly pay attention to how the services are being built, in order to make sure the service if fluent for all kinds of device types. In collaborative virtual spaces, where users create the content, the content itself is no longer fully in the hands of the service provider.

The main contribution in this paper is a proposal for an automatic content adaptation solution, which simplifies texture files demand-time. The proposal for such a system is demonstrated using three device classes: laptop, tablet device and mobile phone, and it operates on image data types. The aim of the demonstration is emphasize how 3D graphics-based content can be optimized regarding demand-time for different device types. As a result, achieved asset compression ratios are presented together with brief image quality analysis.

Even though this research focuses on image files only, there are other data types which need attention as well, such as geometry and their surface material definitions. These data types will be part of the follow-up work. This kind of an adaptation service would allow service providers to address a wider range of devices automatically; hence, better serve the growing user base. All this will result in more efficient bandwidth usage and more efficient rendering of 3D graphics on a mobile device.

First, we take a look at previous research, then the problem is described in detail, a solution architecture is proposed, and a demonstration is executed to show the benefits of 3D content adaptation.

## II. Previous research

Optimization of the web based content has been implemented in the literature for several different types of content. Most of the existing solutions are focusing on regular web content, such as web pages and images in them. The basic form of caching has been discussed by Hung et al. [3]. Image based optimization and automated level-of-detail creation is proposed by Rauschenbach et al. [4]. Those methods focus on individual files (e.g. images) in the web pages, but they do not try to optimize the overall web document. Moreover, there does not seem to be research results available about the overall optimization of a 3D based web service, where also the geometry of the models is involved and re-produced in the client device.

There is, however, research results about image based complexity reduction methods, which have revealed several solutions to the problem. In these methods, the basic principle is that the majority of the processing is done on the server end and only a stream of images or video of the output is passed to the client device. For example, Paravati et al. [5] have presented a method for a video stream adaptation, whereas Hildebrandt et al. [6], [7] have proposed another approach utilizing extended cube map based rendering. As in these methods the virtual space is re-produced in the client device as a video or image stream, the client does not need to know about the geometric complexity of the original data. While this is good from the complexity point of view, it has drawbacks in the client's ability to interact with individual 3D objects in the space, and it also requires a constant bandwidth to be usable, which is not always available in mobile wireless networks.

The second type of approach is asset based methods, where the aim is to optimize the 3D assets of the virtual space individually to allow re-production of the scene graph locally in the device in an optimized form [4], [8]. Asset based methods are better suitable for networks where the access is not always guaranteed, since the device can simulate the 3D space also locally. Especially in wireless mobile networks, bandwidth, latencies and access to network

are limited [8]. However, this method requires that the client device is able to handle all the assets the server sends to it.

Rahimi et al. [9] have studied intelligent streaming of 3D objects to a mobile device and its effect on the overall performance. They prove how the mobile gaming experience in a virtual space can be made better by selectively not

delivering all the 3D content for the mobile device, but instead making decisions on the network side what to deliver at each given moment. While this is not content adaptation as such, because the original asset content is not modified, it is still virtual space adaptation as a whole for the mobile device. It also requires intelligence about what can be left out from the simulation, and universally that may be a very challenging task to accomplish.

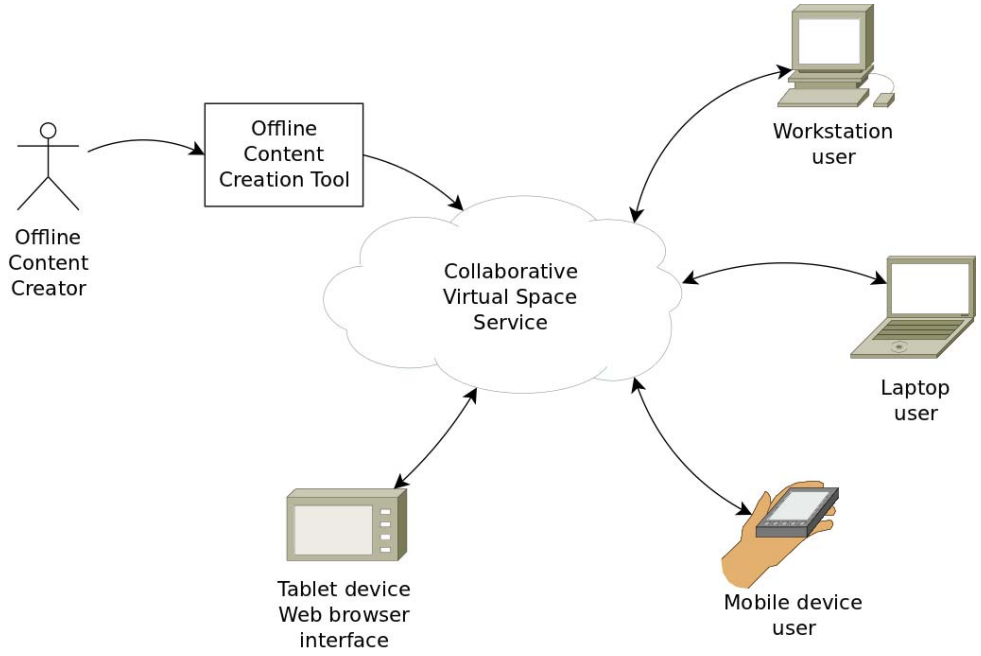


Fig. 1. High level architecture of a collaborative virtual space

## III. Collaborative virtual spaces

Fig. 1 presents a high level illustration of a collaborative virtual space. The virtual space is a network-based service, to which remote clients are connecting to. The client device types can vary. They can be workstations, laptops, tablet devices or other mobile devices, like mobile phones. The term collaboration refers to interaction and content creation and sharing. Each one of the mentioned clients is different, and hence can implement a different feature set. At the same time, each of the mentioned clients will consume and contribute to the virtual space content. This means participation and interaction with other clients in the virtual space (e.g. chatting), or creating new content, which is meant to be published in the space (e.g. a new building for the space). This kind of online editing of the virtual space is implemented already in the Second Life client application. As Fig. 1 shows, the content can also be manipulated via off-line tools. Those results can then later (when network is available), be uploaded into the virtual space.

The collaborative virtual space service can be built in several ways. Usually, there is a central server, which maintains all the current client connections and the overall virtual space

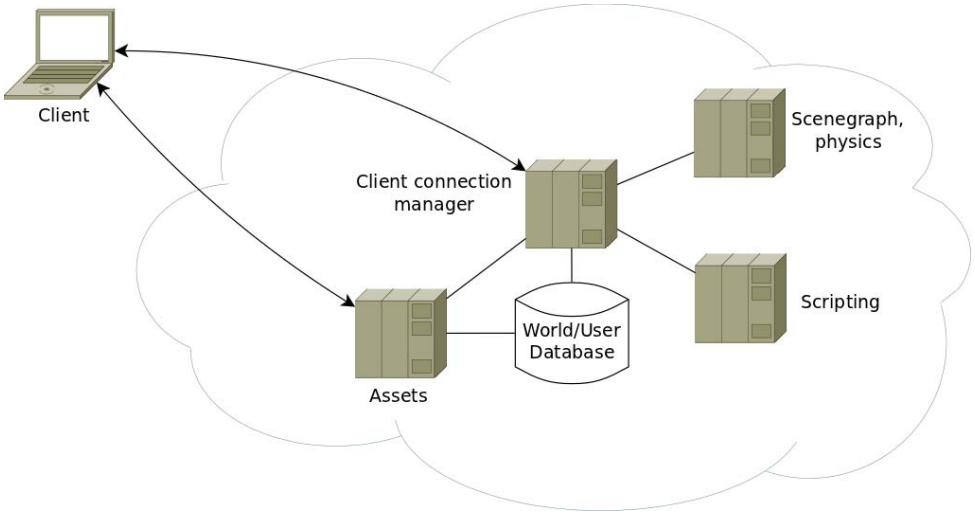


Fig. 2. Generic building blocks for collaborative virtual space service

status, i.e. a list of content (assets) that exist in the space. Then, there is another service, which hosts the assets themselves, which are used to re-create the virtual space during rendering. Examples of such assets include 3D meshes, images, etc. Physics simulation, scripting and scene graph are also components, which exist in the server. They can either be implemented in the core of the server itself (as in realXtend architecture [10]), or they can be separate server entities (e.g. in Second Life1, or OpenSim [11], [12]). Maintenance wise, all of the components can be maintained by one entity (e.g. a service provider), orall of them by several independent entities. For example, the asset storages might be based on cloud storage services (such as Dropbox) and may have nothing to do with the core service itself. These basic building blocks are presented in Fig. 2.

Each virtual space solution (such as realXtend, Open Wonderland, OpenSim) had its own flavor of the server architecture, even though they follow in general the one described above. Likewise, the virtual space architectural distribution inside the dedicated server architecture is implementation specific. The details on how the virtual space is distributed logically in the server grid are out of the scope of this paper.

### *A. The problem of content creation*

Creation of the virtual spaces from scratch is outside of the scope of this paper. There exists several ways of doing it, via off-line modeling tools (such as Blender) or by building a space using existing graphics services (such as Google 3D Warehouse). This paper focuses on the problems of how existing virtual spaces are modified.

All content creators, whether end-users of the service, or original authors of the service, are able to create 3D content to the existing virtual space. When the content is being created, the authors need to address following issues:

- The content complexity in terms of the number of vertices, faces and the complexity of selected materials need to match a wide range of devices at the same time. High polygon 3D models, with high resolution textures, may work in a workstation computer, but the same model does not necessarily fit into mobile device graphics memory at all.
- The content needs to match to the device capabilities. If the content creator is using, for example, a compressed texture format (such as DXT5 in DDS container), it works well

on a workstation, but there is no hardware support for it on the majority of the mobile devices.
- They need to address the fact that in collaborative environment someone else might be modifying the same content at the same time. Means for synchronization and possibly also merging changes need to be in place, or handled by some other way (i.e. explicitly locking the object prior to edit).

When the virtual space is authored and edited, overall, the complexity of the virtual space need to be understood. Even though one single manipulation to the virtual space might be small, and result in as a small 3D object, it still might be the key contributing component in the overall space complexity which makes the virtual space unusable for a mobile device. Even if the aforementioned virtual space services have their own flavors of implementing the client-server architecture, as described in Fig. 2, none of the services really address the paradigm of varying client device user base. Whatever information is stored in the asset database will be distributed to the requesting client, regardless of the information type or its complexity.

### *B. The problem of content distribution*

The virtual space client first initiates the connection to the virtual space server. Handshakes and authentications are performed, as required by the given virtual space service. If the client is allowed to pass, the next step is to share information for the client about the structure of the virtual space and its assets that the client needs for the execution of the space. The execution includes both assets for rendering, but also the assets for programmable in-world logic (scripts).

The above mentioned procedure is a generic one, and common for all virtual space services. It follows a generic client-server type of network architecture [3]. The difference between the services mainly comes from the file formats chosen for the assets, and the protocol and strategy of their distribution. For example, realXtend-based virtual spaces distribute their geometric assets in Ogre3D format, whereas Open Wonderland uses Google Earth (KMZ) or Collada (DAE) file formats. In all cases, it is assumed that the client handles downloading of the issued assets properly, and can handle a scene graph for maintaining them in a local graphical simulation.

When the service targets only workstation and laptop users, the management of the assets and their complexity is rather straightforward, because the currently available hardware is quite harmonized. OpenGL graphics hardware acceleration with programmable graphics pipeline is available from all major hardware manufacturers, and hence nearly all computers in the market include a relatively good support for re-creating 3D graphics. However, in the mobile client and web-based client space, the situation is completely different. In mobile space, there are a number of chip-sets used, each with differing features and performance level. Some have fully programmable graphics hardware, others have only a fixed-point version of it. Some chip-sets have a pixel fill rate of millions of pixels per second, others only a fraction of that. Some chip-sets support only a certain set of texture formats, while other others something completely different.

In the mobile space, it is no longer possible just to create a generic version of any of the given assets, and then assume it works efficiently with all mobile devices. Because of the wide range of mobile devices in the market, one solution will always work well on one system, but badly on some other. Hence, there is a need for a system which is able to automatically do more than just serve a regular client-server type of a connection: a system, which is

always able to choose and adapt the 3D content to whoeve, or whatever kind of a device is requesting it.

Also, another view to the topic comes via the assets which are modified, and hence downloaded and updated to the 3D scene. There might be a requirement for periodically updating the new assets, which emphasizes even further the need that all asset updates and increments are handled tailored to the mobile device in question. This kind of dynamics can be practically handled only with an automated service, as described in the following section.

## IV. Architectural proposal

The defined problem can be addressed either at the creation phase or in the delivery phase. In the creation phase, the decision of asset format can be automatic, or it can be decided by the author. This kind of approach can work fairly well for a small set of target devices, but it will not work when the target devices' capabilities are not know, or their variation is widely heterogeneous. Hence, in this approach the created asset will never fully match to any of the target devices.

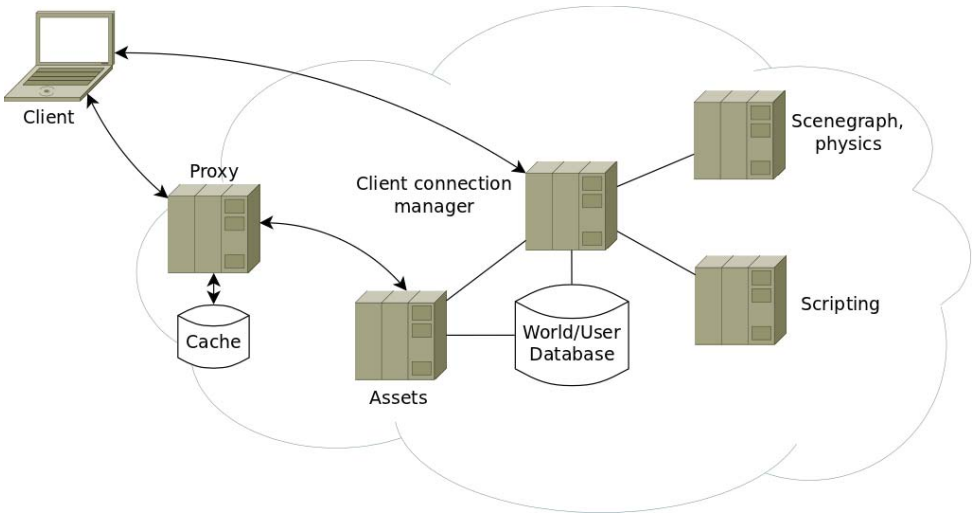


Fig. 3. Asset proxy based generic building blocks for collaborative virtual space service

The second choice is more favorable, since it would ideally allow the author to create only one type of an asset, and then the system would decide in which format it will be actually delivered to the client device. Transformations would be done automatically, and their quality would be matched to the client device capabilities. This naturally requires that the client device needs to identify itself properly, because otherwise the transformation would not work efficiently.

Different assets naturally require different means of optimization. 3D geometry (meshes) can be optimized in terms of the number of vertices and faces, for example. Image files can be stored in the type formats especially supported by the device, and/or their size can be optimized for the client video memory size. Surface materials can be simplified not to include expensive shading effects, but instead less detailed versions of them, which will optimize the duty cycles of graphics processing unit (GPU) rendering, and hence, save power. All of the optimization methods can target to optimize graphics in terms of complexity, but the target of the optimization can also vary. Energy efficiency could be one of the most important targets,

in which all of the assets are optimized in a way the client device maximizes its ability to save energy during the process.

Having the asset manipulation implemented in the delivery phase, will lead the network design to favor proxy-based networks [3] [13]. In this approach, all relevant traffic is translated through a proxy component, which is able to transform it for the client. The proxy would be aware of the device capabilities and it would act accordingly. As the asset translation is not a free operation in terms of processing time, and will always take an amount of time to complete, it is wise to include a caching mechanism to the proxy. Similar assets would be stored for later use, to minimize the need to re-translate assets, which have already been translated with a similar client device type. There are several generic web caching solutions to achieve this, such as the widely used Squid3. Fig. 3 presents the modified version of the generic network architecture, which adds in a proxy and a cache component to the asset download path.

Adding a proxy component to the asset delivery path will increase latency to the overall process. The latency is two folded; first, one extra hop in the network and a cache fetch adds a slight systematic delay and second, if the asset needs to be translated it will cause a processing time delay. It is acknowledged that these will have an impact on the user experience, but for this paper they were ruled out of the scope and instead posponed for future research. In general it is anticipated that the asset translation will happen only once and cached, and hence with multitude of devices the effect can be neglegted. Additional network delay needs to be verified from the user experience perspective later.

## V. Test setup

As a proof of concept, i.e. a setup, was implemented using a laptop PC computer and realXtend client/server software. The setup follows the implementation proposal presented in Fig. 3. In the client device, a realXtend client software was run, which was configured to do all HTTP asset requests via an external proxy component. During the tests, it identified itself as a regular laptop, a tablet device, or a mobile device; hence, mimicking different device types accessing the virtual space service.

The proxy consisted of two parts. The first was the caching part, which was implemented using Squid3, and the second part was translation part, which was responsible of the asset translation. The latter part was a custom implementation and it was making sure all the assets got translated according to the device type. The rest of the architecture was covered with the unmodified realXtend server.

In the proof of concept test, all image files were manipulated to favor the requesting device. Meshes, material definitions and scripts were left unmodified. Those are planned for a future test, after this proof of concept. The virtual space used for the pilot tests was Cheasepeake bay, a joint effort by the Immersive Education initiative and the realXtend association. In terms of image assets, the structure of the selected virtual space was as follows:

TABLE I
IMAGE DISTRIBUTION OF THE VIRTUAL SPACE UNDER TEST

| Image format | Amount | Size on Disk [kB] |
|---|---|---|
| PNG | 98 | 17772 |
| JPG | 82 | 13656 |
| TGA | 1 | 1380 |
| Total: | 181 | 32808 |

There were four different methods of manipulation defined for the test. The methods included different operations in them, like scaling and re-compression of images, and they were chosen to highlight the best choice for each device type. With the defined methods, it was assumed to get enough measurement data to later make a decision of a proper optimization method for each device type target.

Because there were three device profiles each with four manipulation methods, it resulted in twelve separate test cases to compare. The methods were defined as:

- Scaling filter. In this implementation, the original images were scaled down in resolution depending on which device type was requesting it. For a laptop, there was no reduction. For a tablet device, the size (width and height) was reduced by two powers of two (i.e. divided by 4), and for a mobile device by four powers of two (i.e. divided by 16). Powers of two were used because that is the normal reduction method in mipmapping techniques, but also because OpenGL ES 2 specification has mandatory support textures, of which dimensions are powers of two.
- Scaling filter + JPEG quality reduction. This method was essentially the same as the previous one, but in addition, all the incoming JPG images were re-compressed with the compression factor of 35. The factor was passed to Python Imaging Library (PIL), which executed the compression.
- Scaling filter + ETC1 transform. In this method, all incoming images were translated into compressed texture format ETC1, and scaled according to the first option. End results were additionally stored into ZIP archive to gain further lossless compression. ETC1 image format was chosen as one test subject, because is a widely supported format in mobile device graphics hardware [14], [15].
- Scaling filter + DDS transform. In this method, all incoming images were translated into compressed texture format DXT1 and stored in DDS container, and scaled according to the first option. End result was stored into a ZIP archive.

Each of the mentioned implementations was run when the laptop PC was pretending to be either a laptop, a tablet device or a mobile device. Achieved compression ratios are presented in Fig. 4. The scale of the figure is logarithmic for easier interpretation of the results. A laptop was used for all of the test cases, simply to speed up the testing process.

## VI. Test results

In Fig. 4 it can be seen that scaling the resolution down for tablet and mobile profiles will have a huge impact on the final asset sizes. In general the sizes of the original textures files varied somewhat, but averaged around 512 times 512 pixels. Hence, for tablet profile the average texture size became 128 times 128 pixels and for the mobile phone profile it was 32 times 32 pixels. Another note from the figure is favoring the compressed image formats ETC1 and DDS/DXT1. These formats are more efficient to render, since the GPU does not need to un-compress them during the rasterization process. Table II summarizes all of the numerical results.

TABLE II
Summary of downloaded asset sizes [kB]

| Profile: | Scaling | Scale+JPG | ETC1+ZIP | DDS+ZIP |
|---|---|---|---|---|
| Laptop | 32808 | 24004 | 15596 | 26800 |
| Tablet | 3232 | 2860 | 1648 | 2468 |
| Mobile | 864 | 828 | 740 | 800 |

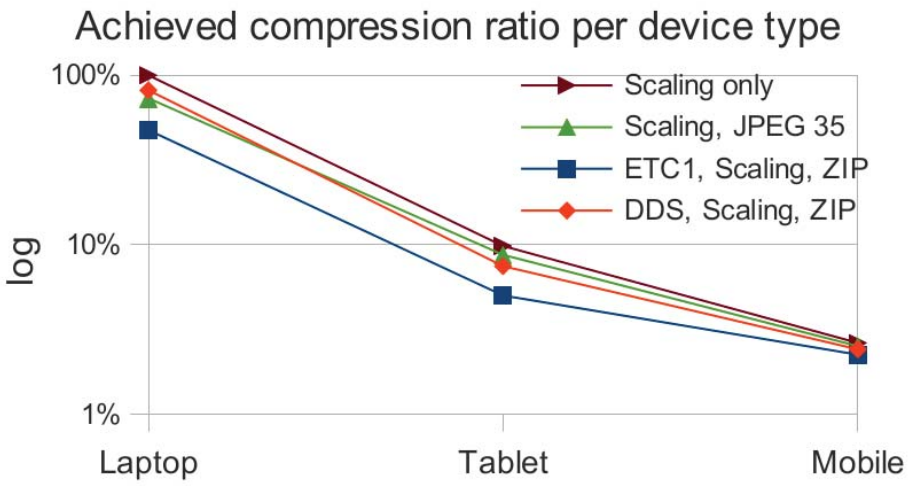


Fig. 4. Summary of achieved compression ratios for each device profile

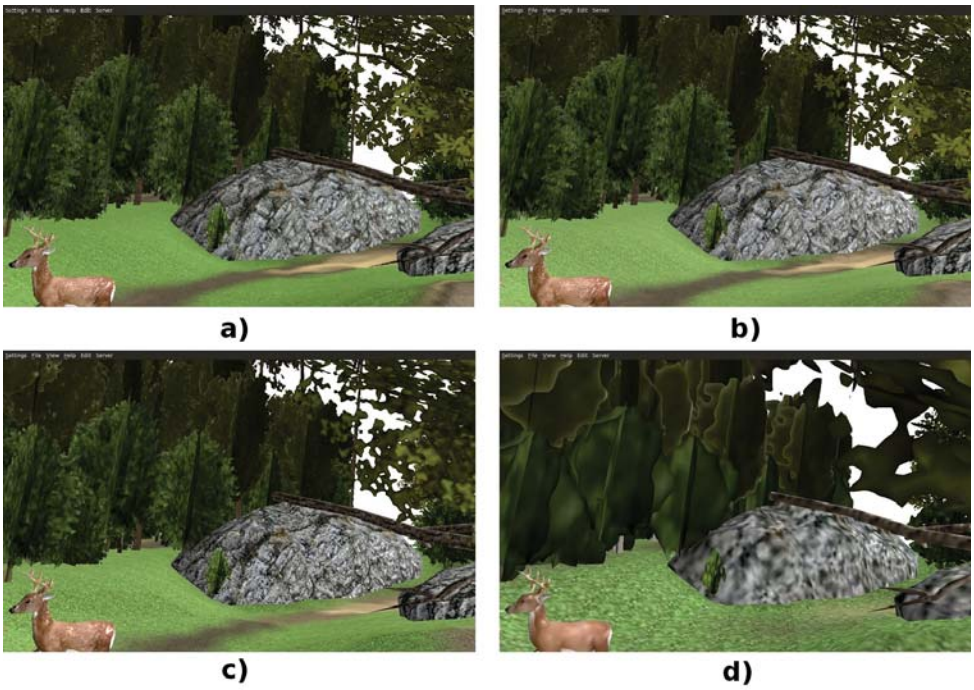


Fig. 5. Sample screen shots of four different optimization methods

Compared to the original size of images, the tablet profile is able to shrink the original data up to 95% (ETC1+ZIP), whereas the mobile profile is able to achieve 98% (ETC1+ZIP) reduction. The majority of the compression factor comes from the fact that the original image resolution is heavily downgraded; hence, resulting in the fact that the output files are much smaller. A variance can be seen between different methods for each device type, where the ETC1+ZIP method seems to be the most efficient in all categories. This is promising, since being the smallest, it favors download sizes, but it could also be the most efficient

way to render the content as well. However, such high compression ratios are not necessarily needed. Instead, these methods prove a tool-set which enable compression capabilities. Further research would include development of intelligence in a proxy, which can adaptively select the best compression solution for each asset, and for each device type. It can be expected that sometimes it is best to save bandwidth, but other times the best thing is to preserve the original quality. In image quality tests, the results of detail reduction are visible. In Fig. 5a, there is a view of the original virtual space, without detail reduction. As a comparison, Fig. 5b shows the same view without scaling down the images, but re-compressing them into DDS/DXT1. Fig. 5c is a snapshot of a tablet view with DDS images, and Fig. 5d is the mobile version with DDS images.

Between 5a and 5b, there is hardly a noticeable difference. Fig. 5c starts already show artifacts in the background trees and foreground ground path. Fig. 5d is clearly already different compared to the rest of figures. However, regardless of the 98% compression ratio, the image looks very recognizable still. However, it must be taken into account that the underlying geometry was neither reduced nor compressed. This means that the original shape of the meshes is kept intact, and therefore the objects are holding their original visual structure. The situation might be different when applying geometry reduction together with image reduction.

## VII. Conclusion

In this paper, a problem of content handling with collaborative and extensible virtual spaces was presented. Collaboration in the virtual spaces means both contribution and consumption of the virtual space content. Collaborative virtual spaces allow authors to create content, which eventually may be difficult to handle by heterogeneous mobile devices intending to consume the data. Authors of the content cannot, and they should not try to address, all the different device types individually when the content is being created. Instead, a system level solution is needed, which enables all authors to make a contribution, whereas the system makes sure that content is optimized for each different device type requesting it. Workstation computers can handle higher quality and more complex content, whereas mobile devices are much more conservative in terms of the available processing power.

A proxy-based network architecture was presented, and a proof of concept test was executed with it. The proof of concept test had content optimization methods for images data types and for three different device types: laptops, tablets and mobile phones. All the tests were run with a laptop computer hardware, which made requests to network pretending of being one of the above mentioned device types. The results show how the amount of downloaded data could be reduced up to 98% compared to the size of the original assets. However, such a large compression ratio is not always needed, and there is room for intelligence in the proxy, which can be used to determine proper levels of compression for each device type and content in question. Also, a brief comparison of image quality was made between the device types showing what kind of visual changes were experienced in the proof of concept test.

The proof of concept test indicated that there is a benefit in optimizing the 3D content for different device types. The next steps will include also an implementation of geometry simplification into the test setup, and measure a real mobile device performance and energy consumption improvements.

## Acknowledgment

The authors would like to thank the staff at the Intel and Nokia Joint Innovation Center for their help and assistance. This work has been carried out in the Tekes Chiru Project.

## REFERENCES


[1] Christophe Mouton, Kristian Sons, and Ian Grimstead, "Collaborative visualization: current systems and future trends", *Proceedings of the 16th International Conference on 3D Web Technology*, ACM, 2011, pp. 101-110.

[2] Mirko Suznjevic, Ognjen Dobrijevic, and Maja Matijasevic, "Hack, slash, and chat: a study of players behavior and communication in mmorpgs", *In Proceedings of the 8th Annual Workshop on Network and Systems Support for Games, NetGames 09*, pages 2:12:6, Piscataway, NJ, USA, November 23 - 24, 2009, IEEE Press.

[3] Chi Chi Hung and Lim Yan Hong, "Adaptive Proxy-Based Content Transformation Framework for the World Wide Web", *Proceedings of the The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region-Volume 2 - Volume 2*, IEEE Computer Society, 2000, p. 747.

[4] Uwe Rauschenbach and Heidrun Schumann, "Demand-driven image transmission with levels of detail and regions of interest", *Computers & Graphics*, 23(6):857866, dec 1999.

[5] G. Paravati, A. Sanna, F. Lamberti, and L. Ciminiera, "An adaptive control system to deliver interactive virtual environment content to handheld devices", *Mob.Netw.Appl.*, 16(3):385393, June 2011.

[6] Dieter Hildebrandt, Benjamin Hagedorn, and Jurgen Dollner, "Image-based strategies for interactive visualisation of complex 3d geovirtual environments on lightweight devices", *J.Locat.Based Serv.*, 5(2):100120, June 2011.

[7] Dieter Hildebrandt, Jan Klimke, Benjamin Hagedorn, and Jurgen Dllner, "Service-oriented interactive 3d visualization of massive 3d city models on thin clients", in *Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications*, COM.Geo 11, pages 6:16:1, New York, NY, USA, 2011, ACM.

[8] J. Vatjus-Anttila, S. Hickey, and E. Kuusela, "Methods and network architecture for modifying extensible virtual environment to support mobility", *In MindTrek11*, September 28, 2011.

[9] H. Rahimi, A. A. Nazari Shirehjini, and S. Shirmohammadi, "Activity-centric streaming of virtual environments and games to mobile devices", in *Haptic Audio Visual Environments and Games (HAVE)*, 2011 IEEE International Workshop on, page 45, oct. 2011.

[10] T. Alatalo, "An entity-component model for extensible virtual worlds", *Internet Computing, IEEE*, 15(5):30, sept.-oct. 2011.

[11] Huaiyu Liu, M. Bowman, R. Adams, J. Hurliman, and D. Lake, "Scaling virtual worlds: Simulation requirements and challenges", in *Simulation Conference (WSC), Proceedings of the 2010 Winter Scaling virtual worlds: Simulation requirements and challenges*, page 778, dec. 2010.

[12] U. Farooq and J. Glauert, "Scalable and consistent virtual worlds: An extension to the architecture of opensimulator", in *Computer Networks and Information Technology (ICCNIT), 2011 International Conference on Scalable and consistent virtual worlds: An extension to the architecture of OpenSimulator*, page 29, july 2011.

[13] Martin Mauve, Stefan Fischer, and Jorg Widmer, "A generic proxy system for networked computer games", in *Proceedings of the 1st workshop on Network and system support for games*, NetGames 02, pages 2528, New York, NY, USA, April 16 - 17, 2002. ACM.

[14] Jacob Strom and Tomas Akenine-Moller, "ipackman: high-quality, low-complexity texture compression for mobile phones", in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS 05, pages 6370, New York, NY, USA, 2005. ACM.

[15] Jacob Strom and Per Wennersten, "Lossless compression of already compressed textures", in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG 11, pages 177182, New York, NY, USA, 2011. ACM.