Efficient Image Distribution on the Web
Instant Texturing for Collaborative Visualization of Virtual Environments

Michael Englert, Yvonne Jung, Jonas Etzold, Marcel Klomann and Paul Grimm
Fulda University of Applied Sciences, Marquardstr. 35, 36039 Fulda, Germany

Keywords: HTML5, WebGL, GPU Decoding, Mobile Devices, Progressive Image Transmission, Collaboration.

Abstract: In this paper, we present a browser-based Web 3D application that allows an instant distribution of image data even over mobile networks as well as textured rendering of large image collections on mobile devices with restricted processing power. Applications utilizing a lot of image data require an adaptive technology to build responsive user interfaces. This applies especially for the use in mobile networks. Furthermore the up- and download of the massive amount of image data should be transmitted in a progressive manner to get an instant feedback. While people are used to instant reaction of web applications and do not care about the amount of data that has to be transferred, the instantaneous display of imperfect content that gets continuously refined is state of the art for many application areas on the web. However, standard 2D image transmission technologies are usually inappropriate within a 3D context. In 2D, image size as well as resolution are often set during the authoring phase, whereas in 3D applications size and displayed resolution of textured 3D objects depend on the virtual camera. Our *GPUII* approach (GPU-based Image Interlacing) follows a client-server architecture, which allows an instant distribution of new data while also reducing the CPU load and network traffic.

1 INTRODUCTION

Today, the combination and integration of real world information into virtual environments is a well-known task in Augmented and Virtual Reality research. The increasing performance of mobile devices and wearables like Google Glass additionally promote this research area. Applications like Google Streetview (Anguelov et al., 2010), as well as Microsoft Photosynth or Photo Tourism (Snavely et al., 2006) already use a huge amount of real world image data to reconstruct 3D geometry directly from digital photos and show that this type of application is getting more and more important.

Another important fact is the increasing availability of 3D web applications. Web applications can be a good alternative to native apps because of their great deployment possibilities even on mobile devices. Standards like HTML5 and WebGL as well as 3D frameworks like Three.js (Cabello, 2013) or X3DOM (Behr et al., 2010) facilitate the development of such applications and additionally provide an easy integration of 3D real-time graphics into web pages without installing further plugins. To be able to cope with the requirements of modern client-side web apps not only new web standards were developed but also the performance of JavaScript engines did increase enormously the last few years. Furthermore, browser vendors and the W3C accomplished more features through their API specifications, which e.g. allow direct access to the devices' media hardware like the camera or other sensors.

Building on these APIs, the collaborative construction planning system recently proposed by (Etzold et al., 2014b) clearly shows that complex Mixed Reality applications can be developed using only web standard technologies. It combines CAD planning data with real world information by placing real photos and annotations within the virtual 3D scenes generated from the CAD blueprints. Therefore, the authors are using the camera access and other web technologies to visualize the combined data via hardware supported 3D rendering. Furthermore, a server-based collaboration component allows all involved people, like workers on the building site as well as supervisors or even investors to discuss about the current state or problems during the construction phase.

Here, not only static scenes are used, but new image material is integrated into the 3D scene at the original camera pose by using the sensor information (such as GPS, gyroscope, etc.) of the mobile device, whenever new photos are captured (Etzold et al.,

Englert M., Jung Y., Etzold J., Klomann M. and Grimm P..
Efficient Image Distribution on the Web - Instant Texturing for Collaborative Visualization of Virtual Environments.
DOI: 10.5220/0005345704780485
In *Proceedings of the 10th International Conference on Computer Graphics Theory and Applications* (GRAPP-2015), pages 478-485
ISBN: 978-989-758-087-1

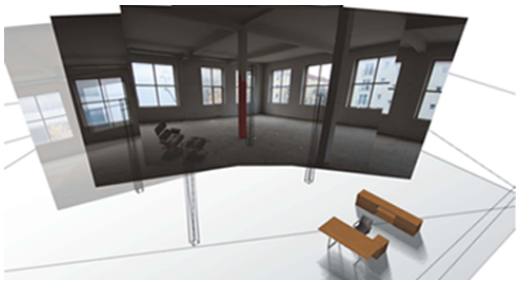

Figure 1: Unordered photo collection that represents parts of a room or building and reorganization within a 3d scene.

2014a). As a consequence, the updated scene data including the newly taken images first has to be uploaded to the server and then the data has to be distributed to all connected clients in order to get a discussion base. In a short time a lot of photos are integrated into the virtual scene this way (see figure 1). However, since the authors are using only standard textures and do not focus on a more advanced streaming approach, connected users have to spend a lot of time to receive new image data, especially when using slow mobile networks.

Along with the improving quality of images (e.g., captured by digital cameras), their sizes and resolution also increase, which is why the integration of several images into the applications requires a lot of data transmission that can be very problematic while using mobile devices and their often strictly limited bandwidth on mobile networks. While people are used to instant reaction of web applications (responsive user interfaces) and do not care about the amount of data that has to be transferred, the transmission of this image data can take a lot of time, which can lead to a frustrating user experience. Users are getting bored, if the system shows no reaction after a few seconds and in the worst case they even leave the page.

Besides using smaller images, a good and elegant approach to improve the application's startup and runtime behavior during synchronization with other clients is to employ progressive image transmission (PIT) strategies (Chee, 1999). Instead of transferring a single entity the images are divided into smaller parts, which are transmitted consecutively and can also be used as preview. The first levels are rough approximations while the quality increases with each additional detail level. The instantaneous display of imperfect content that gets continuously refined is state of the art for many application areas on the web. While the PNG and JPEG standards have become a well-established method for progressive delivery of 2D image data, there was no focus on suitable methods for progressive transmission of textures in interactive 3D web applications.

Also browsers still only support the parallelization of working processes in a rather limited way, so accessing the raw byte stream including the decoding process during loading phase substantially impacts the interactive framerate (Herzig et al., 2013). Other possibilities have to be developed that better deal with the extremely limited processing power of mobile devices and concurrently transmit the image data in a progressive way to support an instant feedback.

To overcome these problems, our proposed web application shows how a huge amount of real world image data can be efficiently distributed to an arbitrary number of connected clients using different devices and bandwidths. Our GPUII method for texture streaming and level-of-detail (LOD) has a minimal overall data rate, easy and fast en- and decoding with small first previews. In addition, we derive a control function to specify the maximum refinement levels of each texture based on the required rendering quality and the current camera state. For load balancing between CPU and GPU during the decoding phase our approach is optimized for mobile devices with less processing power and allows an instant texturing in 3D web applications using the roughest approximation given by the first refinement level.

2 RELATED WORK

There exist different progressive technologies for 2D image transmission schemes, which are shortly discussed in order to select and transfer the best practice as far as possible to a new progressive texture transmission approach that allows instant texturing of virtual 3D objects and fits well with the limited processing power of mobile devices.

While all web browsers support image formats like PNG, JPEG, and GIF, different progressive strategies to download them are *natively* implemented and intermediate download results cannot be accessed from within JavaScript/ WebGL. Though all browsers present several previews of the image during transmission, their efficiencies vary. In this regard, the Adam7 interlacing scheme of the PNG file format (Costello, 2003) allows a very fast first impression of the image due to the simplicity of the algorithm. Adam7 is a 2-dimensional interlacing scheme. Its encoding strategy consists in transposing the pixel order from a sequential to a 2-dimensional distribution by a 8×8 pixel pattern with exactly seven steps. Within these seven steps, the resolution increases by a factor of two with respect to the previous step.

Utilizing the original Adam7 interlacing for textures in 3D scenes imposes several problems that have

to be resolved, as already discussed in (Herzig et al., 2013). On the one hand, it is not possible to get access to the exact preview steps, and on the other, web browsers do not allow accessing this stream in an efficient way. The byte stream has to be converted using JavaScript, which could be rather time-consuming. The Adam7 interlacing scheme strictly uses seven (preview) steps. This can be problematic, if very big images should be transferred, as the first version already could be too big for slow mobile networks. Although in 3D scenes often not the whole quality of a texture is required, for instance because of a larger distance to the virtual camera, in the original Adam7 method the stream cannot be paused if enough data is already visualized. Furthermore, the Adam7 technique is only implemented for the PNG image format, and JPEG compression cannot be exploited efficiently this way.

Already Chee (Chee, 1999) classified approaches to transmit images in a progressive way into four different categories: successive approximation, multi-stage residual coding, transmission sequence based coding, and hierarchical coding. Some approaches only target an efficient transfer, deal with massive amounts of pixels, and visualize them in a 2D context, like Deep Zoom Images (DZI) (Kopf et al., 2007), or they require a sophisticated encoding. Following the classification scheme of (Chee, 1999), DZI can be seen as hierarchical coding strategy, because of its quadtree-based layout.

The pyramid transmission scheme of (Herzig et al., 2013) uses a quadtree to divide and transfer images as well as 3D data. The authors show that such a strategy can be utilized as well in a 3D context, to access 3D terrain data progressively. However, their method still exhibits some disadvantages, as esp. the data transfer lacks efficiency, because all pixels of the first preview are transferred twice when transmitting the next level, and a third time on the next coarser level, etc. Other approaches to stream images progressively try to extract the regions of interest and transmit the preferred image parts as soon as possible like (Hu et al., 2004), or esp. (Lim et al., 2010), who already exploits a quadtree-based method to select and transmit the preferred image parts first to quickly get better previews.

Furthermore, other authors adapt images on the server for different devices to better exploit the sometimes rather limited bandwidth (Wilcox, 2014), since for example on mobile devices with small screens only low resolution images need to be visualized. This way, images can automatically be provided for every required size by a PHP script (instead of the typically manual process of preparing images in various sizes as needed for different media queries/ screen sizes in responsive web design), but they cannot be transmitted progressively.

Especially for texture compression in hardware-accelerated 3D graphics, the DXT formats (Pat Brown et al., 2013) where developed. In contrast to image compression algorithms like JPEG they have a fixed data compression rate and require only one memory access per texel for decompression. To overcome artifacts arising with e.g. normal maps, (Munkberg et al., 2006) later outlined several techniques to improve quality also for normal map compression (cp. (van Waveren and Castano, 2008) for a more detailed discussion). However, all these methods are not yet available in WebGL and only aim at reducing storage memory and bandwidth, but do not provide any means for progressive texture transmission over the web.

Since the broad support of WebGL (Khronos Group, 2014) with its hardware-accelerated 3D rendering in all major web browsers along with 3D web frameworks like X3DOM (Behr et al., 2010) or three.js (Cabello, 2013) for simplifying the development of 3D web apps, adaptive methods not only for images but also for textures in 3D scenes gained importance. Thus, in (Schwartz et al., 2013b) and (Schwartz et al., 2013a) recently a shader-based algorithm that supports progressive transmission as well as access to the dataset within a 3D context was presented in order to stream a BTF (bidirectional texture function) to allow for photorealistic rendering using WebGL, though encoding is still very specialized.

3 TEXTURE REPRESENTATION

Mixed Reality (MR) applications like the web-based support and collaboration tool presented in (Etzold et al., 2014b) and (Etzold et al., 2014a) for construction planning and supervising scenarios use a lot of images to reconstruct virtual worlds combined with real world data. The tool combines classical CAD planning data with photo collections representing temporal snapshots of the associated construction site, which can be integrated into the already existing 3D scene during runtime. After arranging new photos within the scene using sensor data and other MR methods they have to be distributed to all connected clients. A scene can obtain hundred or even more images (figure 1 should give an idea of this use case). On every single start of the application all data has to be transferred via network before it can be used on the client, which can be rather time consuming.

Most of the time clients are connected by mobile devices combined with their strictly limited band-

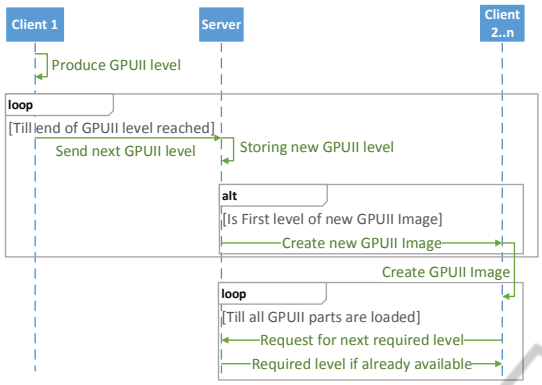

Figure 2: Sequence diagram for upload and distribution of a newly captured and positioned photo using GPUII levels.

width due to mobile networks. If someone wants to share a photo he or she has to upload it. Unfortunately, the upload is usually even much slower than the download, so one has to wait a long time. After the upload the image also has to be downloaded by all other connected (mobile) clients. But there are more challenges to solve when using mobile devices. They are not as powerful as a modern desktop PC or notebook. Thus, encoding and decoding has to be very cheap and efficient.

In this section, we therefore present our approach for progressive image transmission to allow instant textured rendering in distributed, web-based 3D applications using mobile devices, which we call *GPUII* (GPU-based Image Interlacing). The goal of our work is to allow an efficient texture transmission for 3D web applications. For instant response a first refinement level should be delivered very fast, which results in a small preview image. The quality of the transmitted textures should be controlled dynamically by their size and displayed resolution in the final rendering. The amount of transmitted data should be minimal, which means, that each refinement level is integrated into the next one, so that no pixels are transmitted twice. We do not focus or optimize for a specific image format in order to support a broad range of applications and to allow the usage of all commonly supported image formats of a browser exploiting their specific advantages. To allow using massive amounts of images and to minimize the requirements for the web browser the proposed approach should be lightweight regarding CPU and memory resources, and should also scale well for all server jobs.

3.1 Encoding Scheme

We use a technique that exploits a 2-dimensional interlacing to split original images, independent of their original format, into a subset of preview images. The result is an image set with progressively increasing resolution, where the original image format only serves as container format. Instead of utilizing a static count of subimages the number of images is calculated based on the original image size. We split the original image until a minimum size (e.g. 128×128 pixels) is reached, which can be dynamically defined. In case of a 512×512 pixel image we produce a subset of only five images. If an image with a resolution of 4096×4096 pixels shall be used, we encode the image into a subset of eleven preview steps.

Before creating all GPUII levels, we scale the original image to the best matching power-of-two (POT) representation to provide better performance and MipMap support to prevent flickering during camera movement. After this the count of image levels for the resulting *GPUII* data has to be calculated. This is done in dividing width and height by two in an alternating manner, till one of both dimensions reaches the defined minimum resolution. After scaling and calculation of the iterations the actual encoding can be started. The encoding scheme, we exploit in our application is oriented on the Adam7 interlacing scheme (Costello, 2003). In contrast to the original implementation there is no restriction to exactly seven steps. Thus, better adaption on large images are possible, which results in predictable loading times of the first preview independent of the image size. In addition, refinement control is possible.

To provide interactive framerates during encoding we use worker threads to produce all image levels. Through workers, browsers allow parallel processing. Functions can run as long as needed without affecting the framerate or user interface of the application. During encoding a CPU-based approach is no problem, because only one image has to be converted simultaneously. Decoding is in our application a little bit more tricky, because a lot of images have to be decoded concurrently.

3.2 Decoding Scheme

During the encoding step, the original image was splitted into a set of preview images with progressively increasing resolution. The number of subimages is varying and stored with the following naming scheme: $[1].[format] - [n].[format]$. To decode the image correctly, the previews are loaded in a chronological manner starting from preview number 1 to n. After loading one or several levels of the image, it has to be decoded before presentation, where the highly parallelized computing power of the GPU is used. Moreover, every image pixel is transmitted only once

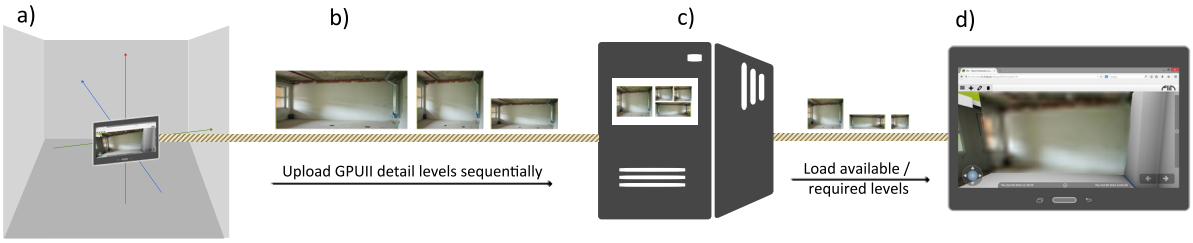

Figure 3: Distribution of a) newly captured and arranged photo with b) upstream of the levels of GPUII dataset, c) storage of the GPUII dataset on the server and distribution to connected clients d) using progressive download with different bandwidth.

during download, which is ensured through our encoding strategy.

The first preview can be displayed directly without any decoding effort. All consecutively loaded $n-1$ image levels are integrated into the already existing image, following the algorithm sketched in (Englert et al., 2014) – here, cp. figure 2 for a visualization. After the second level of the image has been loaded, the new data and the currently visualized texture are both sent to the shader. Additionally, a combination pattern is required. The shader itself is applied to a view-aligned quad of the targeted image size during an offscreen rendering pass using two FBOs (Framebuffer Objects). The result of the rendering pass is written to an FBO and serves in the following frames as the new and finer representation of the surface texture. This process is repeated in a ping-pong'ing fashion until the final texture resolution, depending on the factors outlined in section 3.3, is reached.

For a combination operation we exploit two patterns that are alternately applied. A pattern is a small texture of 2×2 px that is used as lookup texture and that describes how to combine the pixels of two images into an intermediate result. Which pattern will be used first is computed from the number of previews of the decoding dataset. An even number of texture previews are combined line by line, whereas in the other case the combination is done column by column. The colors black and white are used as index and always define the texture whose pixel should be placed on the resulting image. Obviously, more complex patterns are possible this way. To implement this approach on the GPU, two textures that should be combined are required including a pattern texture along with a scaling vector $R_{x,y}$ as uniform variable for scaling the texture coordinates accordingly.

$$R_{x,y} = \begin{cases} \left(\frac{r_x(p_1) \cdot 2^{\lfloor 0.5n \rfloor}}{2}, \frac{r_y(p_1) \cdot 2^{\lfloor 0.5(n-1) \rfloor}}{2} \right), 2 \nmid m \\ \left(\frac{r_x(p_1) \cdot 2^{\lfloor 0.5(n-1) \rfloor}}{2}, \frac{r_y(p_1) \cdot 2^{\lfloor 0.5n \rfloor}}{2} \right), 2 \mid m \end{cases} \quad (1)$$

Here, $R_{x,y}$ defines how often the pattern has to be repeated in x and y direction (or s and t respectively). If for instance two textures, both of resolution 64×64 should be combined and the number of desired previews is odd, then the used vector would be $\binom{64}{32}$. The exact scaling factor is obtained via equation 1, where $r_{x,y}$ specifies the resolution in x, y directions, n the current preview number, m the preview count, and p_1 the first preview image. The name *GPU-based Image Interlacing* thus denotes the basic idea of our algorithm. The consecutively ordered image levels are downloaded progressively and are combined in an interlaced manner on the GPU.

3.3 Data Distribution

To distribute photos our application consists of two parts. First, the image has to be uploaded to the server, followed by the distribution to all other clients. Figures 2 and 3 visualize and additionally explain the entire streaming process.

Upload of New GPUII Dataset. Once we have created a new GPUII dataset on the client, the image data has to be transferred to the server and distributed to all connected clients. All levels of the GPUII dataset are transferred in a chronological manner and stored by the server using the previously explained naming scheme. The distribution to all other connected clients starts immediately when the first level is available. The server sends a message to all clients to create a new GPUII dataset with a position and URL. They directly start the download of the first level and then follow the rules explained next.

Download of All Required GPUII Datasets. Textured objects in a 3D scene often cover only a small part of the viewport. Hence, a more sophisticated download method is necessary that also takes properties of the 3D scene, like camera position, etc., into account, to decide which preview has to be loaded next and, if multiple textures are used, how to sort them for importance. We define importance using the following criteria with decreasing priority:

- Visibility
- Necessity of next refinement level
- Distance to camera
- Currently loaded preview step

All images are sorted along these criteria. In addition to the visibility of each image, it has to be decided if further preview steps are needed. This information is determined by calculating the screen space size of the image and relating it to its pixel density (similar to mipmapping). The advantage of this control function is that not in every case a better resolution is required. All images that are visible and require a finer resolution level are registered to the system for sorting along camera distance, size, and priority.

To prevent that always the same image is preferred by the distance criterion, distance is ordered by concentric circles around the virtual camera to group images with similar distance. In a last sorting step, the images of the nearest circle are ordered by their current preview step, where the images with the smallest preview step are preferred to get a consistent look.

4 RESULTS AND APPLICATIONS

In this section we discuss the main benefits of our approach and also present some additional use cases.

4.1 Results

Using GPUII instead of standard image textures has several advantages for applications, which are using huge amounts of image data on low-end hardware like mobile devices together with mobile networks.

Adaption. The advantage of adaption can be splitted into three areas. First the download can be adapted to the bandwidth. In case of using mobile networks the download of the GPUII preview parts can be restricted. Using one step less than the entire amount of GPUII preview versions can reduce the download amount up to 50%. Furthermore the download can be adapted to the storage size of the presentation device. Mobile devices are not only restricted in processing power, but also in their memory. In our application a lot of images should be visualized at the same time what often leads in reaching the memory limit. In case of extremely storage limitations the download amount for each GPUII image can also be restricted as previously explained. Additionally to the restriction caused from bandwidth and device limitations our approach adapts to the camera position within the scene and the size of the viewport. This can efficiently

Figure 4: a) Difference image (enhanced contrast for better visibility) of b) original PNG texture (4096×4096 px, 8.5 MB) and c) corresponding 4096×4096 px GPUII dataset (1 MB) adaptively rendered on 1024×768 px viewport.

reduce the data download amount. Often images in 3D scenes are rendered very small due to their distance to the virtual camera. And when presenting them in fullscreen they also require only the size of the viewport that is often less than image resolutions of currently available digital cameras. Utilizing our proposed *GPUII* method offers cost-savings of about 80% in situations with big viewport but small image sizes. A comparison of an original PNG image with a *GPUII* visualization is shown in figure 4, where a) shows the slightly exaggerated difference image.

First Preview. Another important aspect that has to be taken into account is the transmission efficiency of the first preview of the image. Additionally, the first preview should provide enough data to get a good first impression of what will be shown later. Therefore, a trade-off between minimal resolution and download speed has to be considered. Adam7 is limited in its variability and always uses statically seven steps to reorder the pixels. This always leads to $\frac{1}{64}th$ of the number of pixels in the first representation, and both, high resolution as well as low resolution images, are restructured in the same way. This soon leads to rather large first previews with increasing source image sizes. Our proposed method in contrast exploits a dynamic amount of preview steps and allows specifying a minimum resolution in x- and y-direction, which prevents a further subdivision of the source image. In case of small source resolutions the dataset thus consists of a small number of previews, while high resolution source images instead affect in an opposing manner: the number of previews increases. The size of the first preview is nearly identically using arbitrary image resolutions on the source file. For *GPUII* a minimum size of 128×128 px seems most reasonable – in our tests we found this being a good trade-off between data size and preview quality.

Data Distribution. A fast distribution of new photos is another important aspect. While uploading standard images, the server has to wait until all data of the image is available before distributing it to all

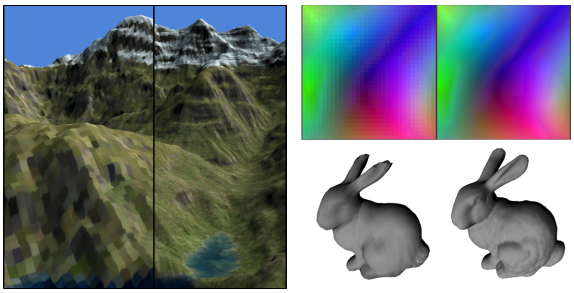

Figure 5: Using GPUII to transmit terrain data (left) or geometry images as proposed in (Gu et al., 2002) (right).

connected clients. Calculating GPUII images on the client before uploading, the distribution can be accelerated efficiently. Because of ever increasing image resolutions of digital cameras the image resolution can be reduced before uploading it, adapted to the bandwidth. Furthermore, because of using the images as textures in WebGL, they should be power of two. After transmitting the first preview level of GPUII, which requires a transfer of only a few KB, the distribution can be started some time earlier. This leads to instant previews on all connected clients directly after starting an image upload, independent from supported data rates of the network.

4.2 Additional Use Cases

Progressive Transmission of 3D Vertex Data. Having regular data eases progressive transmission, where esp. terrain data is very often arranged in a regular form. This type of data can be easily encoded in images and streamed with our approach. Figure 5 exemplarily shows some regular terrain data transmitted and rendered by using our *GPUII* method. The rendered terrain is visualized with the first version of both, the displacement data and the surface texture. Both textures are transmitted using PNG containers, because of the lossless decoding, although for the color texture alternatively JPEG is possible as image transport format, since here compression artifacts are usually not perceivable. Using geometry images (Gu et al., 2002), 3D models can be transformed into regular meshes. These meshes, or more precisely their vertices, can be arranged within images as regular RGB values depending on the topology. Therefore, this kind of meshes can be transferred progressively with *GPUII* using the same texture, which would not be possible with standard texturing.

Geometry Streaming and LOD. Furthermore, *GPUII* can be used together with progressively streamed 3D geometries (e.g., the PopGeometry presented in (Limper et al., 2013)), where the geometry data also gets streamed and builds up progressively in a LOD-like manner. However, for small models the texture data usually takes more time to transfer and thus the progressive 3D geometry will not have a texture until the transfer is completed. To enhance the quality, *GPUII* can be combined with the POP approach so that the visual perception is more consistent in that the geometry and texture data build up simultaneously, see (Englert et al., 2014). Note that smaller loading times are possible if for a certain camera distance not the full texture quality is needed. Thus, the texture refinement can stop at a lower level-of-detail (LOD), just like the POP buffer geometry does.

2D Image Viewer. *GPUII* can also be used for streaming texture data progressively when displaying the image in a pure 2D context. As a result, a preview of the corresponding image is shown almost immediately. The missing data to show the image in its full quality is transmitted progressively to further adjust the quality depending on the requirements. Therefore, images can be adapted to different screen sizes. Depending on the actual visual part of the image, the required data can thus be reduced to a minimum.

5 CONCLUSION AND FUTURE WORK

In this short paper, we have presented an adaptive bandwidth-optimized approach that allows instant image distribution and web-based textured rendering on strictly limited mobile devices and their closely linked mobile networks. Our method uses a simple encoding scheme, based on Adam7 interlacing, and a fast decoding algorithm that benefits from hardware acceleration by a GPU – even with WebGL's rather limited instruction set. Various image sizes of the previews are possible, including the possibility to specify the minimal resolution of the first preview, which enables us to generate first previews of nearly identical size in bytes, independent of the source image size. This allows us to get a more flexible handling of source images containing arbitrary resolutions.

Moreover, image interlacing schemes like Adam7 do not foresee pausing the download and carrying on if more data is required, though this can be very helpful to reduce the data amount to be transferred. Our *GPUII* approach however allows visualizing images in pure 2D applications in an adaptive manner, which is esp. useful for responsive web design, where an optimal viewing experience for a wide range of displays

has to be provided. Whereas in 2D the image size and resolution are defined during authoring, in 3D applications the size and displayed resolution of textured 3D objects depend on their world space positions and the viewpoint, which is updated every frame. So, not in every case the full texture quality is necessary, like for instance if an object is far away from the camera. Therefore, our approach can also be used as a new level-of-detail method on the texture level, independent from the geometric model representation.

To summarize, our proposed technique can be applied to stream surface textures for progressive meshes for consistent rendering, to load large amounts of images in a 3D scene, or to transmit regular geometry information, like e.g. displacement maps or other vertex information. In addition to the PNG format, all common image formats that are supported by browsers can be utilized as data transport containers for our *GPUII* textures. To ease usage we have integrated the proposed technique as special texture node in X3DOM. Moreover, in sec. 3 we have also shown an important application scenario, where our approach allows increasing the number of photos in the 3D scene by a factor of at least ten to twelve.

For future work, we would like to combine our method with a hierarchical approach to stream large regularly organized meshes (e.g., terrain data). Besides this, it would be interesting to natively implement our polyfill approach in the web browser for transparent and even more efficient usage.

REFERENCES

Anguelov, D., Dulong, C., Filip, D., Frueh, C., Lafon, S., Lyon, R., Ogale, A., Vincent, L., and Weaver, J. (2010). Google street view: Capturing the world at street level. *Computer*, 43.

Behr, J., Jung, Y., Keil, J., Drevensek, T., Eschler, P., Zöllner, M., and Fellner, D. W. (2010). A scalable architecture for the HTML5/ X3D integration model X3DOM. In *Proceedings Web3D '10*, pages 185–193, New York, USA. ACM Press.

Cabello, R. (2013). Three.js. http://threejs.org/.

Chee, Y.-K. (1999). Survey of progressive image transmission methods. *International Journal of Imaging Systems and Technology*, 10(1):3–19.

Costello, A. M. (2003). Portable network graphics (png) specification (second edition): Information technology – computer graphics and image processing.

Englert, M., Jung, Y., Klomann, M., Etzold, J., and Grimm, P. (2014). Instant texture transmission using bandwidth-optimized progressive interlacing images. In *Proceedings of 19th Intl. Conf. on 3D Web Technologies*, Web3D '14, New York, USA. ACM.

Etzold, J., Englert, M., Grimm, P., Jung, Y., and Klomann, M. (2014a). Mipos: Towards mobile image positioning in mixed reality web applications based on mobile sensors. In *Proceedings of 19th Intl. Conf. on 3D Web Technologies*, Web3D '14, pages 17–25, New York, USA. ACM.

Etzold, J., Grimm, P., Schweitzer, J., and Dörner, R. (2014b). karbon: a collaborative mr web application for communicationsupport in construction scenarios. In *CSCW Companion*, pages 9–12. ACM.

Gu, X., Gortler, S. J., and Hoppe, H. (2002). Geometry images. *ACM Trans. Graph.*, 21(3):355–361.

Herzig, P., Englert, M., Wagner, S., Jung, Y., and Bockholt, U. (2013). X3d-earthbrowser: Visualize our earth in your web browser. In *Proceedings Web3D 2013*, pages 139–142, New York, USA. ACM Press.

Hu, Y., Xie, X., Chen, Z., and Ma, W.-Y. (2004). Attention model based progressive image transmission. In *Multimedia and Expo, ICME '04*, volume 2, pages 1079–1082 Vol.2. IEEE.

Khronos Group (2014). Webgl specification. http://www.khronos.org/registry/webgl/specs/latest/1.0/.

Kopf, J., Uyttendaele, M., Deussen, O., and Cohen, M. F. (2007). Capturing and viewing gigapixel images. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA. ACM.

Lim, N.-K., Kim, D.-Y., and Lee, H. (2010). Interactive progressive image transmission for realtime applications. *Consumer Electronics, IEEE Transactions on*, 56(4):2438–2444.

Limper, M., Jung, Y., Behr, J., and Alexa, M. (2013). The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum*, 32(7):197–206.

Munkberg, J., Akenine-Möller, T., and Ström, J. (2006). High quality normal map compression. In *Proceedings Graphics Hardware '06*, pages 95–102, New York, NY, USA. ACM.

Pat Brown et al. (2013). GL_EXT_texture_compression_s3tc. http://www.opengl.org/registry/specs/EXT/texture_compression_s3tc.txt.

Schwartz, C., Ruiters, R., and Klein, R. (2013a). Level-of-detail streaming and rendering using bidirectional sparse virtual texture functions. *Comput. Graph. Forum*, pages 345–354.

Schwartz, C., Ruiters, R., Weinmann, M., and Klein, R. (2013b). Webgl-based streaming and presentation of objects with bidirectional texture functions. *Journal on Computing and Cultural Heritage (JOCCH)*, 6(3):11:1–11:21.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, pages 835–846, New York, NY, USA. ACM Press.

van Waveren, J. M. P. and Castano, I. (2008). Real-time normal map dxt compression. http://www.nvidia.de/object/real-time-normal-map-dxt-compression.html.

Wilcox, M. (2014). Adaptive images - deliver small images to small devices. http://adaptive-images.com/.