Evaluating 3D Thumbnails for Virtual Object Galleries

Max Limper[1,2] Florian Brandherm[1] Dieter W. Fellner[1,2,3] Arjan Kuijper[1,2] *

[1] Fraunhofer IGD [2] TU Darmstadt [3] TU Graz

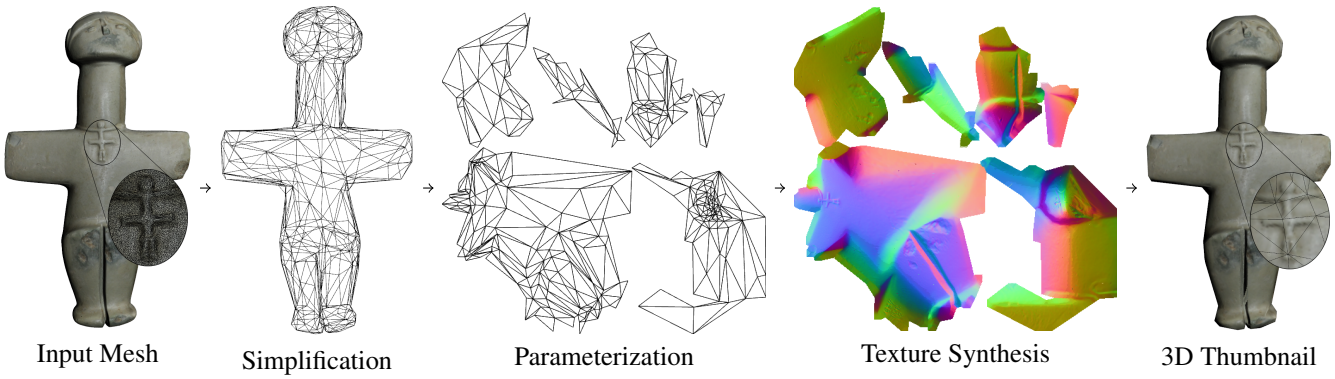

Figure 1: *Overview over our automatic 3D thumbnail generation pipeline. We simplify the geometry of the input mesh. We then parameterize the simplified geometry in order to synthesize textures. These textures are then used to map original, high-resolution surface attributes onto the 3D thumbnail. Finally, mesh data is converted into a compact delivery format. By comparing against image-based 3D previews, we evaluate the applicability of this approach, with regards to file size and visual error, in the context of virtual object galleries on the Web.*

Abstract

Virtual 3D object galleries on the Web nowadays often use real-time, interactive 3D graphics. However, this does usually still not hold for their preview images, sometimes referred to as thumbnails. We provide a technical analysis on the applicability of so-called 3D thumbnails within the context virtual 3D object galleries. Like a 2D thumbnail for an image, a 3D thumbnail acts as a compact preview for a real 3D model. In contrast to an image series, however, it enables a wider variety of interaction methods and rendering effects. By performing a case study, we show that such true 3D representations are, under certain circumstances, even able to outperform 2D image series in terms of bandwidth consumption. We thus present a complete pipeline for generating compact 3D thumbnails for given meshes in a fully automatic fashion.

CR Categories: I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture I.3.2 [Computer Graphics]: Distributed/network graphics

Keywords: WebGL, Texture Mapping, Thumbnail, Compression, 3D Formats

1 Introduction

Presenting a collection of 3D objects online has become a very common use case for 3D Web technology. Popular examples include online shops, Web portals for 3D printing, or virtual museums. Furthermore, the increasing popularity of low-cost digitization devices raises demand for Web-based visualization of high-resolution, scanned artifacts. To enable a fast overview over a large collection of 3D objects, most applications employ so-called *thumbnails*: like for image collections, small preview images provide a rough overview of the database content. Such a thumbnail often consists of a single image. Moreover, some Web pages use animated image series to create the illusion of a 3D rendering. When connected to the user's input, these image series can even be used to mimic a true 3D viewer, which, however, offers only a limited amount of freedom for navigation and interaction. Typically, interaction is limited to turntable-like rotations around the up-axis, for instance. This is due to the fact that, in order to serve as a preview (or: thumbnail), these image series should be able to be loaded very fast, and hence they should not consume too much bandwidth.

Within this paper we investigate the possibility of using true 3D representations as interactive previews for the content of 3D object galleries on the Web. Our contributions can be summarized as follows:

- We present a fully automatic pipeline for converting a high-resolution 3D model to a compact 3D representation, entitled *3D thumbnail*.
- We evaluate the file size of 3D thumbnails against the file size of comparable image series, answering the important question if fast, true 3D previews are feasible in terms of bandwidth consumption.
- We provide a brief discussion of the advantages and disadvantages of both methods.

We limit our case study to a scenario where the thumbnails serve as previews for the original, full-quality view onto the data. This full-quality view is assumed to be a 3D visualization, realized via WebGL or a comparable high-performance 3D graphics API for the Web (such as Stage3D, for instance). One typical scenario could be, for instance, an online exhibition of digitized artifacts, for example from the domain of cultural heritage. While our experiments are

*E-Mail for all authors:{firstname.lastname}@igd.fraunhofer.de

performed with the explicit use case of preview images for object galleries in mind, we believe that our technical analysis is generally interesting when it comes to the question whether a full 3D viewer should be used for a given application, or not.

2 Previous Work

Mesh Simplification and Detail Preservation. A wide variety of techniques have been proposed to reduce the complexity of a 3D mesh, while approximating its original shape or appearance as closely as possible. The most popular one is probably the quadric-based edge collapse technique proposed by Garland and Heckbert [Garland and Heckbert 1997]. Cignoni et al. have shown how the surface details of a mesh can be preserved on a simplified version by encoding them in textures [Cignoni et al. 1999]. They propose to adopt the texture sampling density based on an estimated screen-space footprint of the rendered object. However, the proposed texture parameterization method includes many discontinuities, and is not optimized with regards to any metric. Sander et al. have improved the approach of Cignoni et al. by showing that a normal-shooting approach can produce better approximations than closest-point sampling [Sander et al. 2000]. The method relies on a good parameterization of the simplified mesh.
One popular approach for parameterizing a mesh (also available in the open-source modeling tool *Blender*, for example) is the *Least-Squares Conformal Maps* (LSCM) method. [Lévy et al. 2002]. It provides a least-squares approximation of a parameterization that preserves local angles within the mesh. *Angle-based flattening* is another method that aims to achieve a conformal mapping [Sheffer and de Sturler 2001]. There are other parameterizations methods that optimize by different criteria, such as geometric stretch or signal stretch [Sander et al. 2001; Tewari et al. 2004]. The interested reader is referred to the survey of mesh parameterization methods presented by Hormann et. al. [Hormann et al. 2007].
An alternative approach was presented by Cohen et al. [Cohen et al. 1998]. Their approach was to limit the allowed texture distortion of a textured high-resolution mesh during simplification. Within the context of 3D thumbnail generation, an interesting property of their approach is the ability to measure the deviation of the simplified version of the mesh in screen space, given a specific display resolution.
In general, our concept of a 3D thumbnail is closely related to the idea of using a coarse Level-of-Detail representation (see [Luebke et al. 2002] for an introduction), obtained via mesh simplification. However, the special property of a 3D thumbnail is that it is optimized for a specific target viewport size, and that the file size of the compressed result should be as small as possible.

Thumbnails as Previews for 3D data bases. The term *3D Thumbnail* was introduced by Chiang et al.[Chiang et al. 2010; Chiang and Kuo 2012]. They transform meshes into a low-dimensional thumbnail-descriptor that captures the main features of the mesh. From that, a 3D preview can be generated and rendered. These 3D thumbnails differ from our 3D thumbnails, as we try to approximate not only the shape, but the entire appearance (including surface details) in a 3D preview.
Besides mesh-based 3D representations, there are also image-based approaches, using pre-rendered views of a 3D object, as well as video-based methods. A recent introduction to these topics can be found in the work of Lipski et al., for instance [Lipski et al. 2015]. As previews for today's Web galleries, simple animated or interactive 2D image series, realized using JavaScript libraries such as *jQuery reel*[1], are still the most popular image-based method in use.

[1] http://test.vostrel.net/jquery.reel/example/index.html

The *3DNP* (3D - No Plugins) technology by Thorsten Schlüter[2] is an open-source system to render and display 360-degree views of 3D objects inside a browser. These views are simply represented as images, and interaction and navigation around the object is realized via JavaScript. The viewer enables the user to rotate the model around two different axes, and it is currently being used by the popular 3D printing portal *Shapeways*[3].

Mesh Compression for the Web. Alliez and Gotsman, as well as Peng et al., presented surveys on different mesh compression methods [Alliez and Gotsman 2003; Peng et al. 2005]. Recently, Maglo et al. have presented another state-of-the-art paper, focusing on latest developments within the past decade [Maglo et al. 2015]. While most mesh compression techniques traditionally optimize the rate-distortion ratio of the resulting compressed representation, typical real-world formats for the Web rely on much simpler binary formats, in order to allow for fast decoding [Limper et al. 2013]. A popular method (for instance, used by the glTF format, as proposed by the Khronos group[4]) is to send binary buffers, which can then directly be uploaded to the GPU (cp. also [Behr et al. 2012; Limper et al. 2014; Sutter et al. 2014]).

3 3D Thumbnails

Within this paper, we define a *3D Thumbnail* as a true 3D preview representation for the original mesh, which is significantly smaller in file size than the original model. Moreover, it is specifically designed for being displayed within a viewport of a fixed, small size. This way, a 3D thumbnail serves a similar purpose as a 2D image thumbnail: to be loaded much faster, and at the same time, to provide the best possible insight about the large-scale structure of the original object. One could also state that the simplified version should be visually as close as possible to the original mesh, by preserving overall shape, texture and surface details, given a fixed threshold for the resulting file size.
A common method to meet the goal of an optimum approximation, using a given file size budget, is the use of textures for representing surface signals (cp., e.g., [Cohen et al. 1998; Cignoni et al. 1999]). Those textures are applied to a geometrically simplified version of the original mesh. In the following, we describe our pipeline for generating a 3D thumbnail, which follows this general concept. A schematic overview is also shown in Fig. 1.

3.1 Mesh Simplification

In order to decrease the size of a 3D model to a level that can be used for a 3D thumbnail, the geometry must be simplified aggressively. The simplification has to be performed once, during a preprocessing step, and the mesh is simplified to a very low number of vertices (cp. Table 1). Because of this, we chose good shape approximation over simplification speed. We used the OpenMesh[5] library's implementation of the quadric edge-collapse algorithm [Garland and Heckbert 1997], while forbidding the orientation of faces to be flipped.
The target number of vertices, required for a simplified mesh to resemble the shape of the original one closely enough, depends on the original's shape. Also, the notion of *closely enough* may be subject to the viewer's opinion. Ideally, the number of vertices of the simplified mesh is determined by an error threshold, either in screen space or in 3D object space.

[2] http://www.thoro.de/page/3dnp-introduction-en
[3] http://www.shapeways.com
[4] http://gltf.gl/
[5] http://www.openmesh.org

3.2 Mesh Parameterization

We encode surface details (in our test setup: diffuse color and normals) of an original 3D model in 2D textures. Thus, the simplified mesh needs a parameterization. Even if the original model is parameterized and textured, we cannot reuse any textures because the simplification would result in a very distorted parameterization. Furthermore, many high-resolution models are not parameterized and only have per-vertex colors and normals, for example.

Cutting In order to generate a contiguous 2D parameterization of a simplified mesh, some edges have to be cut apart such that the topology of the mesh meets the requirements of the used parameterization method. In our case, we need the mesh to be of disk topology, since we chose the LSCM algorithm for parameterizing it. We can then segment the mesh first and build a texture atlas afterwards.
To achieve this, we used a simpler variant of the segmentation algorithm presented by Lévy et al. [Lévy et al. 2002]. First we find the 5% of edges with the largest dihedral angle and mark them as features. Then, we compute the minimal distance to any feature edge or boundary edge for each vertex, edge and face. Beginning from the local maxima of these distances, we grow segments by iteratively adding triangles at the segment boundaries. Finally, we separate the segments by cutting the mesh at the segment boundaries, resulting in multiple segments which are each of disk topology.
The advantage of such a segmentation over cutting the mesh until it has the required topology is that the parameterization can be done for each segment in parallel. Furthermore, smaller segments are more likely to be parameterized without overlapping themselves, which would require further processing.

Parameterization During this step, we map every segment of the simplified 3D mesh onto a planar (2D) domain. In principle, any parameterization algorithm could be used here, we decided to use the LSCM algorithm. An alternative parameterization could, for example, be angle-based flattening [Sheffer and de Sturler 2001]. The resulting parameterizations for all segments must be scaled in texture space, such that their relative size in texture space corresponds to their relative surface area in 3D object space. This ensures a consistent level of texture detail across all segments.

Packing We create a texture atlas from the parameterizations of all segments by packing them into a square. This is done with a simple packing algorithm based on bounding boxes. The drawback of this approach is the amount of unused texture space inside these boxes. Therefore, more elaborate packing algorithms have the potential to increase the texture utilization significantly. One example for such an algorithm would be the packing algorithm that was introduced by the authors of the LSCM method [Lévy et al. 2002]. A more recent alternative would be the approach that was taken by Nöll and Stricker, where the parameterizations are packed under the assumption that texture content is repeated during sampling [Nöll and Stricker 2011].

3.3 Texture Synthesis

At this stage of the pipeline, we transfer the surface attributes (diffuse color, normals) to textures. In order to do so, a mapping must be established between the simplified mesh and the original mesh. A naive mapping would be a nearest-point mapping. However, it has been shown that visually better results can be obtained by a normal shooting approach (see [Sander et al. 2000]), which we have adapted. For every texel of a synthesized texture, the corresponding point on the simplified mesh is determined. We then search, in forward and backward direction, along a ray which originates from this point, following the direction of the interpolated normal, to find the closest point of the original mesh. We furthermore require that the original mesh's face intersected by the ray has the same orientation as the corresponding face on the simplified mesh. Once that intersection point is found, its interpolated surface attributes are sampled and assigned to the texel. To reduce interpolation artifacts, we add a margin around all sampled texel regions, repeating texture content from the borders of the respective islands.
For a given simplified mesh with a given parameterization, the ideal resolution of the synthesized textures depends on the viewport resolution of the resulting 3D thumbnail

3.4 Conversion to a Delivery Format

Finally, the simplified mesh and the texture maps have to be converted to a delivery format which is suited for the Web. In order to yield the smallest possible file sizes, we have to compress the geometry and textures. We also have to choose file formats that can be natively decoded by Web browsers in order to keep loading times to a minimum. Diffuse textures can be compressed lossily, with little visual difference, as JPEG images. The normal textures, however, cannot be easily stored as JPEG without a visible loss of quality. Thus, we save them as PNG images.

Geometry is ideally stored in a binary format that is compact in file size and, at the same time, not introducing any decoding overhead. This aspect is especially crucial for 3D thumbnails, since they already serve as previews, which means they should be loaded as fast as possible. We have used X3DOM's BinaryGeometry format [Behr et al. 2012]. The main reason for this was the fact that, with the *aopt* tool, the InstantReality framework [6] already provides a powerful tool for optimizing data for delivery on the Web, using X3DOM. Although the SRC format can also be written by this tool, its compressed version can currently not be handled by X3DOM, otherwise this would have been a more state-of-the-art alternative [Limper et al. 2014]. Further interesting technologies that provide compact delivery of 3D mesh data for the Web (but are not available in X3DOM yet) are glTF[7] and Blast [Sutter et al. 2014]. For generating the X3DOM Binary Geometry files, we have used a compression setting that creates 16 bit vertex positions and texture coordinates. The resulting binary files have then been zipped for delivery.

4 Comparison with 2D image series

To evaluate the performance of a 3D thumbnail, we have created a test setup where an animated series of 2D images serves as a comparison. Like for a 3D thumbnail, the main aim of such an image series is to provide the user with an impression about the overall 3D structure of the object, by showing views onto the object from different angles.

A great advantage of 2D image series, which is worth to be noted at this point, is that one can basically display objects at any degree of realism, without any significant difference in application performance. The images could, for example, show photographs, or high-quality path-traced renderings. However, we have focused our experiments on a use case where the full-resolution view onto the data is realized as a real-time 3D visualization (e.g., based on WebGL). Therefore, we found it a natural assumption that the image-based previews should also be generated using the same rendering tech-

[6] http://www.instantreality.org
[7] https://github.com/KhronosGroup/glTF

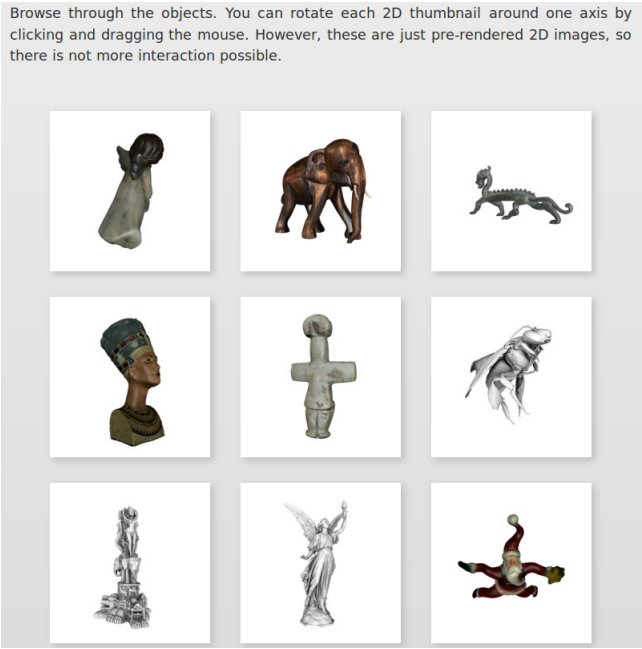

Figure 2: *Test Web page, showing previews for our test models (here: image series, both versions of the Web page are available for testing as part of the supplemental material). From top left to bottom right: Angel, Elephant, Dragon, Nofretete, Cruciform, Bee, Thai Statue, Lucy, Santa.*

Scene	Original		Simplified	
	#vertices	#tris	#vertices	#tris
Angel	500,355	1,000,000	588	688
Elephant	115,318	230,636	2,337	2,994
Dragon	125,000	250,000	1,526	1,972
Nofretete	220,474	440,297	684	786
Cruciform	192,183	284,361	510	563
Bee	8,473,793	16,946,880	4,099	5,294
Thai Statue	4,999,996	1,000,000	2,579	3,008
Lucy	14,027,872	28,055,742	1,219	1,318
Santa	75,781	151,558	856	996

Table 1: *Test models used for our experiments.*

niques, so that the user's first impression gets as close as possible to the original, full-resolution view.

4.1 Test Setup

To perform a comparison between 2D image series and 3D thumbnails, in terms of file size as well as of preview quality, we have first selected several scanned meshes as test cases. An example Web page, showing a gallery with preview images for all of our nine test models, is shown in Fig. 2. As can be seen in Table 1, our test models include very small meshes, as well as gigantic ones, consisting of many millions of primitives. Furthermore, we used meshes with very simple geometry, such as the Nofretete bust, as well as meshes with highly complex geometric structures, such as the eulaema bee. For the meshes that didn't have any colors, we created grayscale ambient occlusion maps instead.

Figure 3 gives an overview of how our test setup is organized. After creating the 3D thumbnail representations, we render them from different points of view, in a similar way how the image series are

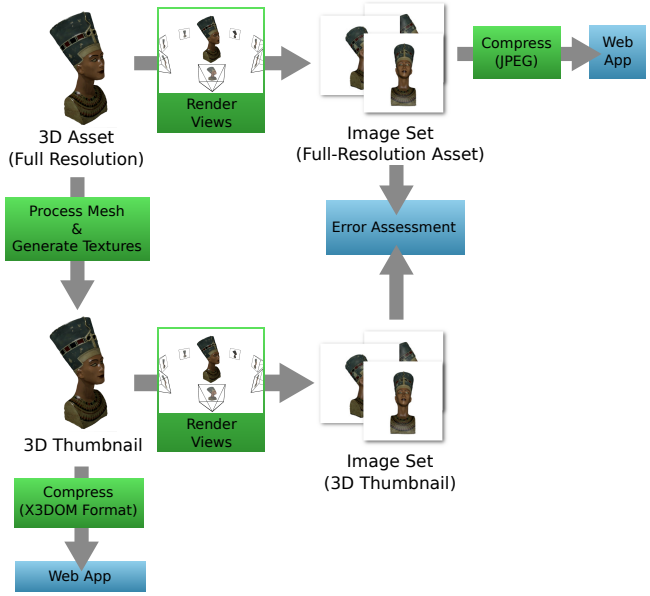

Figure 3: *Test setup used for our experiments. On the one hand, 3D assets are simplified and parameterized to store surface details in textures, which leads to true 3D thumbnail representations. On the other hand, we create image series by rendering the asset from different views. To compare the quality of both approaches, we create an image series using the 3D thumbnail and compute the mean square error over the resulting images.*

created from the full-resolution models. Since both approaches should serve as previews of a Web-based real-time rendering of their full-resolution versions, we have used X3DOM to generate the image sets. By using X3DOMs *getScreenshot* function, and by programatically triggering a click onto an HTML anchor element, downloading all rendered views of a particular object was easily possible. The rendered views are, in both cases, stored in a lossless format (PNG) and then compared to estimate a mean square error over all pixels, for each view onto each model (see Table 4). To avoid aliasing artifacts inside the image series, occuring with small viewports and high-resolution mesh data, the image series have been rendered using a larger viewport (800×800 pixels) and then sampled down to their final resolution (200×200 pixels).

4.2 Deciding over Variable Testing Parameters

Having decided to use previews of 200×200 pixels for all of our nine test meshes for our experiments, there were still a couple of test parameters left, which had to be configured. In the following, we briefly summarize how we decided about the most important parameters within the test setup.

Image Formats. To provide a fair comparison of the resulting file sizes of both approaches (see Table 3), we have converted the image series to JPEG format, using ImageMagick's *mogrify* tool with a 90% quality setting. Similarly, we have stored the diffuse color texture of each 3D thumbnail as a JPEG image, using the same quality setting as for the image series. Normal maps, however, cannot be compressed as JPEG images without accepting a significant loss of quality. We therefore stored all of the normal maps using the loss-free PNG format.

Scene	#images (360 deg. rotation)	resolution
reel: Phone	10	200×200
reel: Vase	12	210×186
reel: Car 2	20	200×200
reel: Teapot	24	160×120
reel: Car 1	35	276×126
reel: Arrow	36	130×60
WebRotate 360: Shoe	36	400×264
YouSpin: Gun	70	569×491
3DNP: FRITZ!Box	252	300×300

Table 2: *Resolution (pixels) and number of images per image series for some turntable-like 360 degree preview examples from the Web.*

Number of Images per Image Series. When comparing a 3D thumbnail with an image series in terms of interactivity and file size, a crucial question is: What is the typical number of images in such a case? To answer this question, we have considered multiple examples from the Web, which are shown in Table 2. Some numbers are taken from the public demo page of the popular *jQuery reel* JavaScript library for animated image series, as well as from the public company pages of *Web Rotate 360*[8] and *YouSpin*[9], and from the 3DNP demo page.
As can be seen from Table 2, typical numbers vary between 10 and 252, while the large value of 70 seems to be an outlier and only occurs for the high-quality YouSpin Gun demo, using a large size of 569×491 pixels, and showing only one object on the entire page. The 3DNP demo uses significantly more images than the other ones, since it also provides the user with an additional degree of freedom for interaction. However, the large amount of images leads to an overall file size of over 2 MB. Therefore it is already significantly larger than the other examples, and also larger than all of our 3D thumbnails, hence we limited ourselves to a more meaningful comparison against turntable-like image series with one degree of freedom. At least from the examples we considered, values within a range of 10 to 35 seemed to be typical for 200×200 pixel viewports, therefore we have used configurations with 8, 16 and 32 images, for each of our test models, throughout our experiments.

Amount of Mesh Simplification. When simplifying the original meshes for generating 3D thumbnails, the question arises to which amount this simplification should be performed. Ideally, we would use a method that takes the visible error on the image plane into account, since we already know the resolution of the target viewport for our 3D thumbnails.
However, in our prototypical pipeline, we have simply picked this resolution for our nine test meshes manually. Compared to Cohen et al. (see [Cohen et al. 1998]), we have used an orthogonal approach: instead of deciding about a visual error, and letting this error determine the final file size, we have simplified all meshes until the resulting 3D thumbnail representations were of a size that is comparable to the size of the corresponding image series. With this fixed mesh sizes, we have then assessed the visual error. Table 1 shows an overview of the amount of triangles and vertices for the simplified meshes.

Texture Resolution. One critical parameter, with regards to the resulting file sizes of the 3D thumbnails, is texture resolution (cp. Fig. 4). We therefore had to decide about the texture resolutions that should be used for normal maps and diffuse textures. In the case of 3D thumbnails, where the size of the target viewport can assumed to be known, we can choose our texture size according to the viewport dimensions. In an ideal case, where the texture parameterization is assumed to be regular, without significant stretch along the 3D surface, and where we assume to look at a flat surface with a projected size that is equal to the viewport dimensions, the ideal texture resolution is identical to the viewport size. Since the size of our previews is known to be 200×200 pixels, and assuming that users will only zoom into the scene to a limited amount, we identified textures at resolutions of 128×128 pixels, 256×256 pixels, and 512×512 pixels as a suitable set of candidate configurations.

[8] http://www.webrotate360.com/360-product-viewer.html
[9] http://www.youspin.co/youspin/demo/360-spin/

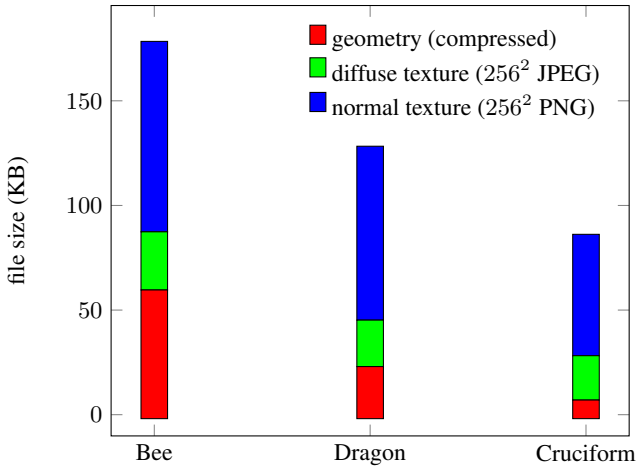

Figure 4: *This graph shows by example how the file sizes of a 3D thumbnail are composed of the geometry files (X3DOM BinaryGeometry), the diffuse texture (JPEG) and the normal texture (PNG). The size of the normal texture becomes more significant as the size of the geometry decreases, but it is the largest part in all of the three cases. This bottleneck becomes even more apparent when texture size increases.*

4.3 Comparing File Sizes

Table 3 shows an overview of the resulting file sizes, using different numbers of images for the image series, and different texture resolutions for the 3D thumbnails. As can be seen, both representations produce files within a comparable range of size. For each of the test meshes, the 3D thumbnail representations become even larger than the image series with 32 images, as soon as a texture of 512×512 pixels is used. This is due to the fact that bandwidth consumption of the texture images, especially for the normal map, is usually the dominant factor for the 3D thumbnails, as can also be seen in Fig. 4: even for the rather complex Bee mesh, the normal texture already consumes more bandwidth than the compressed geometry.

For the image series, the elephant mesh produces the largest file sizes. We think that one of the most important reasons why the image series of the elephant mesh consumes so much space is the fact that the rendered, centered elephant mesh covers large parts of the viewport, from all possible angles. In contrast, the thin, elongated dragon model has a much smaller average screenspace footprint, and hence there is less information in the overall image, which in turn leads to significantly smaller file sizes.

4.4 Comparing Visual Quality

Besides the pure file size, an important criterion for assessing the quality of 3D thumbnails is the visual error that is introduced by simplifying the original data. To measure the approximation qual-

Model	8 images	16 images	32 images	3D thumbnail (128^2)	3D thumbnail (256^2)	3D thumbnail (512^2)
Angel	34.4	68.9	138.0	35.6	90.3	262.8
Elephant	55.0	110.3	**221.0**	67.1	133.6	359.9
Dragon	28.1	57.0	114.1	55.4	127.3	363.4
Nofretete	43.9	88.1	176.0	37.7	96.2	281.8
Cruciform	**30.1**	61.0	121.9	**32.8**	85.2	253.4
Bee	35.2	70.8	141.0	96.1	177.5	**456.6**
Thai Statue	36.5	72.9	145.8	71.6	157.1	455.5
Lucy	31.5	63.5	127.0	50.1	123.4	368.4
Santa	36.6	74.0	147.7	40.1	97	288.3

Table 3: *File size (in KB) of the image series and 3D thumbnail representations (including geometry and textures) for our test models. For the 3D thumbnails, texture sizes are indicated in brackets. The largest and smallest value for each of both categories, images and 3D thumbnails, are printed in bold type.*

Model	128×128	256×256	512×512
Angel	5.55	5.28	5.11
Cruciform	4.70	4.40	4.32
Dragon	7.06	6.54	6.11
Elephant	10.58	10.44	10.35
Nofretete	4.96	3.96	3.45
Bee	40.7	33.7	25.1
Thai Statue	39.9	31.1	23.4
Lucy	34.4	29.8	22.6
Santa	10.17	10.15	10.13

Table 4: *Mean square error, multiplied by* 10^4 *for readability, for different texture resolutions. The error was averaged over 32 views of each model.*

ity of the 3D thumbnails, we have rendered them from 32 different points of view, in a similar fashion as for the creation of the image series (cp. Fig. 3). We have then, for each model and texture resolution, computed the mean square error (MSE) over all pixels of the 32 views. Our results are summarized in Table 4. As can be seen in the table, increasing the texture resolution always reduced the error. However, the magnitude of this decrease was not similar for all test models. The geometrically rather simple Nofretete bust, for example, showed a higher sensitivity to varying texture resolution than the more complex elephant mesh.
An interesting finding is that the variation of the MSE between the different meshes is far more significant than its variation among different versions of the same mesh at varying texture resolutions. There might be two different reasons for this: first the geometric simplification, and second the texture parameterization. The LSCM algorithm which we used only optimizes the conformality criterion, and it may demand post-processing with an optimization that takes into account geometric stretch or signal stretch [Lévy et al. 2002; Sander et al. 2001; Tewari et al. 2004]. We therefore believe that an ideal Thumbnail generation algorithm would take both into account, geometric simplification and distortion of attribute textures (cp. [Cohen et al. 1998]).

The error introduced by the texture resolution, as well as by parameterization and geometric simplification is visualized for the Nofretete bust in Fig. 5. The largest errors occur at the silhouette, due to the geometric simplification, as well as in regions of high texture detail.

4.5 Comparing User Experience

Finally, an important question remains: how different is the user experience for both approaches? To provide an optimum answer to this question, an extensive user study would need to be conducted. While this is out of the scope of this paper, we have investigated different ways of interacting with the 3D thumbnails, as can also be seen in the accompanying video and demo application: the user can navigate around the model, using X3DOMs *Turntable* navigation mode. Besides smooth rotation around arbitrary axes, zooming is also possible. Furthermore, the user can switch between the standard rendering mode and two others. The first additional rendering mode uses a uniform white base color for the shaded surface, instead of using the diffuse texture. The other one directly visualizes the normal map, which allows for a better inspection of small-scale surface structures. Clearly, more interaction possibilities can be imagined, such as moving the position of the light source with the mouse.

At this point, we can state that 3D thumbnails offer significantly more interaction possibilities than 2D image series of a comparable file size.

5 Conclusion

The results of our case study highlight different constraints, regarding the use of 3D thumbnails. Reasons for preferring 2D image series as previews, rather than using 3D thumbnails, can be summarized as follows:

- In contrast to real-time 3D renderings, generated inside the client's browser, 2D images have the advantage that they can display an object at any degree of realism.
- Especially when a low number of images is used, image series are more compact than 3D thumbnails.
- 2D images are independent from the geometric complexity of the object, while 3D thumbnails require a high vertex budget and sophisticated parameterization methods for objects with complex shape and topology.
- 2D images are comparably easy to generate, while generating a 3D thumbnail requires the application of several advanced mesh processing techniques.

For many common applications, these constraints do not apply. One example application is given as part of the supplemental material, and also shown in the corresponding video of this paper. In general, reasons for preferring 3D thumbnails over 2D image series are the following:

- 3D thumbnails provide the user with a lot of interaction possibilities. Besides smooth rotations and zooming, rendering effects, such as changes in lighting or switching between different rendering modes, can be realized.

Figure 5: *Visual comparison of two* 200×200 *pixel views. Left: Full-resolution mesh (440,297 triangles, vertex colors). Center: Simplified 3D thumbnail with* 256×256 *pixel texture (786 triangles). Right image: squared difference, multiplied by factor 8 for visualization purposes. Typical artifacts occur at the silhouette, due to reduced geometric fidelity, or at regions of fine texture detail, due to the small texture resolution.*

- For objects with simple shape and topology, 3D thumbnails provide great previews, even with low vertex budget.
- When smooth interaction (or interaction along multiple degrees of freedom) is required, 3D thumbnails are generally more compact than 2D image series.

Besides the already mentioned 3D galleries for the exhibition of digitized artifacts, other areas of application exist. In general, a 3D thumbnail can be used in every case where previews of a small, fixed size are employed. One use case that is potentially located outside the Web could, for instance, be a shop or inventory window in a game, where small 3D elements serve as previews for the actual 3D views onto different items. Such items could have been originally created by an artist as high-resolution meshes, for example by using sculpt tools. Due to the wide variety of possible use cases, and due to the general nature of the problem of creating compact 3D representations, our proposed 3D thumbnail generation pipeline can also be used for general 3D asset optimization.

Future Work includes several important improvements to the proposed pipeline. First, we need to use an expressive error measure for simplification, instead of manually deciding over the target number of vertices. Second, the texture size for the normal maps is the dominant factor for the overall size for a 3D thumbnail, and, at the same time, the texture quality is massively determined by the texture parameterization. We therefore believe that using more sophisticated algorithms for mesh segmentation, parameterization and packing are keys to better quality of the results. The use of algorithms for normal map compression also seems promising to mitigate this problem.

Besides such technical contributions to the asset optimization pipeline, an important task for future research is the investigation of *usability* of 3D thumbnails, compared to other representations, such as 2D image series. This will require extensive user studies, as well as a full exploration of the potentials offered by the different preview methods. Part of such studies will also be a more sophisticated assessment of perceived visual quality of the previews, since the mean square error can not always be expected to be a good measure of what humans perceive as visual errors.

Finally, combining the advantages of both approaches also seems an interesting direction for future research. For example, transmitting an image series with G-Buffers instead of final renderings could be an interesting approach for achieving real-time rendering effects, such as dynamic changes of material and lighting.

6 Acknowledgements

The models *Angel, Dragon* and *Nofretete* are courtesy of Fraunhofer IGD, competence center for Cultural Heritage Digitization (CHD). The *Cruciform* model has been kindly provided by the Cyprus Institute, as part of FP7-funded EU project *V-MusT*. The *Bee* model is courtesy of the Smithsonian Institution (*www.3d.si.edu*). The models *Thai Statue* and *Lucy* are courtesy of the Stanford Computer Graphics Laboratory.

References

ALLIEZ, P., AND GOTSMAN, C. 2003. Recent advances in compression of 3D meshes. In *In Advances in Multiresolution for Geometric Modelling*, 3–26.

BEHR, J., JUNG, Y., FRANKE, T., AND STURM, T. 2012. Using images and explicit binary container for efficient and incremental delivery of declarative 3D scenes on the web. In *Proc. Web3D*, 17–25.

CHIANG, P.-Y., AND KUO, C.-C. J. 2012. Voxel-based shape decomposition for feature-preserving 3d thumbnail creation. *J. Vis. Comun. Image Represent. 23*, 1 (Jan.), 1–11.

CHIANG, P.-Y., KUO, M.-C., AND KUO, C.-C. 2010. Feature-preserving 3d thumbnail creation with voxel-based two-phase decomposition. In *Advances in Visual Computing*, G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung, R. Hammoud, M. Hussain, T. Kar-Han, R. Crawfis, D. Thalmann, D. Kao, and L. Avila, Eds., vol. 6453 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 108–119.

CIGNONI, P., MONTANI, C., ROCCHINI, C., SCOPIGNO, R., AND TARINI, M. 1999. Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer 15*, 10, 519–539.

COHEN, J., OLANO, M., AND MANOCHA, D. 1998. Appearance-preserving simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 115–122.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH*, 209–216.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH 2007 Courses*, ACM, New York, NY, USA, SIGGRAPH '07.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3 (July), 362–371.

LIMPER, M., WAGNER, S., STEIN, C., JUNG, Y., AND STORK, A. 2013. Fast delivery of 3d web content: A case study. In *Proc. Web3D*, ACM, New York, NY, USA, Web3D '13, 11–17.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. Src - a streamable format for generalized web-based 3d data transmission. In *Proc. Web3D*, 35–43.

LIPSKI, C., HILSMANN, A., DACHSBACHER, C., AND EISEMANN, M. 2015. Image- and video-based rendering. In *Digital Representations of the Real World: How to Capture, Model, and Render Visual Reality*, M. Magnor, O. Grau, O. Sorkine-Hornung, and C. Theobalt, Eds. CRC Press, Apr., 261–280.

LUEBKE, D., WATSON, B., COHEN, J. D., REDDY, M., AND VARSHNEY, A. 2002. *Level of Detail for 3D Graphics*. Elsevier Science Inc.

MAGLO, A., LAVOUÉ, G., DUPONT, F., AND HUDELOT, C. 2015. 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Comput. Surv. 47*, 3 (Feb.), 44:1–44:41.

NÖLL, T., AND STRICKER, D. 2011. Efficient packing of arbitrary shaped charts for automatic texture atlas generation. In *Computer Graphics Forum. Eurographics Symposium on Rendering (EGSR-11), June 27-29, Prague, Czech Republic*, Blackwell Publishing, Malden, US, 9600 Garsington Road, Oxford OX4 2DQ, UK, R. Ramamoorthi and E. Reinhard, Eds., vol. 30, The Eurographics Association.

PENG, J., KIM, C.-S., AND JAY KUO, C. C. 2005. Technologies for 3D mesh compression: A survey. *J. Vis. Comun. Image Represent.*, 688–733.

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. 2000. Silhouette clipping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 327–334.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 409–416.

SHEFFER, A., AND DE STURLER, E. 2001. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers 17*, 3, 326–337.

SUTTER, J., SONS, K., AND SLUSALLEK, P. 2014. Blast: A binary large structured transmission format for the web. In *Proc. Web3D*, 45–52.

TEWARI, G., SNYDER, J., SANDER, P. V., GORTLER, S. J., AND HOPPE, H. 2004. Signal-specialized parameterization for piecewise linear reconstruction. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, ACM, New York, NY, USA, SGP '04, 55–64.