Fast decompression for web-based view-dependent 3D rendering

Federico Ponchio*
Visual Computing Lab, ISTI-CNR, Pisa

Matteo Dellepiane†
Visual Computing Lab, ISTI-CNR, Pisa

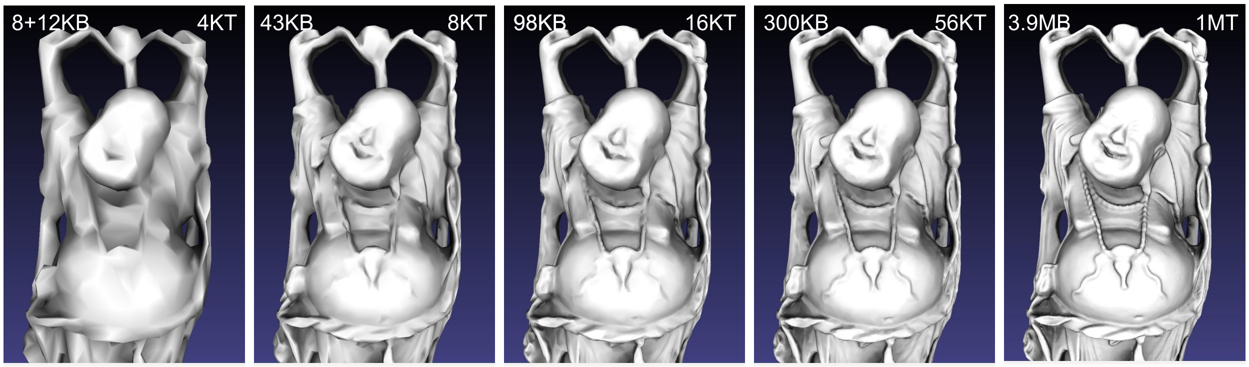


Figure 1: *Progressive refinement of the Happy Buddha: on the upper left corner the size downloaded, on the upper right corner the number of triangles in the refined model. The header and index amount to 8KB*

Abstract

Efficient transmission of 3D data to Web clients and mobile applications remains a challenge due to limited bandwidth. Most of the research focus in the context of mesh compression has been on improving compression ratio. However, in this context the use of Javascript on the Web and low power CPUS in mobile applications led to critical computational costs. Progressive decoding improves the user experience by providing a simplified version of the model that refines with time, and it's able to mask latency. Current approaches do so at very poor compression rates or at additional computational cost. The need for better performing algorithms is especially evident with this class of methods where Limper [Limper et al. 2013b] demonstrated how decoding time becomes a limiting factor even at moderately low bandwidths. In this paper we present a novel multi-resolution WebGL based rendering algorithm which combines progressive loading, view-dependent resolution and mesh compression, providing high frame rates and a decoding speed of million of triangles per second in Javascript. This method is parallelizable, robust to non-manifold meshes, and scalable to very large models.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms;

Keywords: Multi-resolution, web visualization, 3D compression

1 Introduction

Limited bandwidth and increasing model sizes pose a challenge in the transmission of 3D data to Web clients and mobile applications. Mesh compression is a viable approach to minimize transmission time, and most research focus in this field has been on optimizing compression ratio.

Unfortunately, limited bandwidth often pairs with limited computational power, either because of Javascript environment or low CPU power mobile devices, to the point that for most algorithms decoding time becomes the bottleneck even at moderately low bandwidth. Acceptable rates can be regained reducing compression ratio (for example forfeiting connectivity compression) or using less sophisticate entropy compression algorithms.

A different approach makes use of progressive reconstruction algorithms, which improve the user experience by providing a simplified version of the model that refines while the remaining part of the model is being downloaded. The model converges very quickly at the beginning of the download, and only the details require the full model. However this class of algorithms performs even worse in terms of decoding time (as shown in Limper [Limper et al. 2013b]) or in terms of compression ratio.

Another desirable feature, especially for very large models, is view-dependent resolution: this allows to prioritize the download, decode

*e-mail:federico.ponchio@isti.cnr.it
†e-mail:matteo.dellepiane@isti.cnr.it

a specific part of the model and vary resolution of the rendered geometry to maintain a constant screen resolution. This is obtained by maximizing quality at a given frame rate.

In this paper we present a novel multi-resolution WebGL based rendering algorithm which combines progressive loading, view-dependent resolution and a mesh compression providing good rates and a decoding speed of million of triangles per second in Javascript. This method is can handle non-manifold meshes, and it is also scalable to deal with very large models.

The method is based on a class of multiresolution structures [Cignoni et al. 2004; Cignoni et al. 2005] where the "primitive" of the multiresolution becamesis a patch made of thousands of triangles. The processing required to traverse this structure becomes a fraction of triangle based multiresolution algorithms, and allows "batch" operation on the patches: moving data from disk or network to GPU RAM, rendering, and decompression.

In section 3 we describe the improvement made on the multiresolution structure and the how the compression algorithm was designed to optimize decoding time while maintaining a good compression ratio. In section 4 we compare it with existing web solutions for mesh compression and progressive visualization. It represents a solid alternative to current methods, providing a practical mean to handle 3D models on the web.

2 Related Work

This paper is related to several topics in the field of Computer Graphics. Among them, the main are: web-based 3D rendering, progressive and multiresolution rendering approaches, and fast decompression methods for 3D models.
While a complete overview of all these subjects goes well beyond the scope of the paper, in the next subsections we provide a short description of the state of the art, trying to focus on the aspects which are more related to the proposed approach.

2.1 Web-based 3D rendering

Three-dimensional content has always been considered as part of the multimedia family. Nevertheless, especially when talking about web visualization, its role with respect to images and videos has always been a minor one. Visualization of 3D components was initially devoted to external components, such as Java applets or ActiveX controls [Mic 2013].

After some initial efforts for standardization [Raggett 1995; Don Brutzmann 2007], the proposal of WebGL standard [Khronos Group 2009b], which is a mapping of OpenGL|ES 2.0 [Khronos Group 2009a] specifications in JavaScript, brought a major change. Several actions related to the use of advanced 3D graphics has been proposed since then. For a general survey, please refer to the survey by Evans [Evans et al. 2014b]. Since the use of OpenGL commands needs advanced programming skills, there have been several actions to provide an "interface" between them and the creation of web pages. We could subdivide the proposed systems between *declarative* approaches [Jankowski et al. 2013], like X3DOM [Behr et al. 2009] or XML3D [Sons et al. 2010], and *imperative* approaches, like Three.js [Dirksen 2013], SpiderGL [Di Benedetto et al. 2010] and WebGLU [DeLillo 2009]. The main difference between the groups is that the first ones rely on the concept of *scenegraph*, hence a scene has to be defined in all its elements, while the second ones provide a more direct interface with the basic commands. Other systems provide a sort of hybrid approach, where a very simplified scene has to be defined.
Evans [Evans et al. 2014b] points out in his survey that declarative approaches had a major impact in the research community, while imperative approaches were mainly used in the programming community.
More in general, given the fact that the amount of data that needs to be sent to the webpage can be quite big, several efforts about a better organization of generic streamable formats [Limper et al. 2014a; Sutter et al. 2014] has been proposed. Nevertheless, when complex 3D data have to be streamed, these structures are not flexible enough to handle them.
In order to face this problem, in the last three years some progressive compression methods ad hoc for 3D streaming have been developed. Gobbetti et al. [Gobbetti et al. 2012] proposed a quad-based multi-resolution format. Behr et al. [Limper et al. 2013a] transmit different quantization levels of the geometry using a set of nested GPU-friendly buffers. Lavouè et al. [Lavoué et al. 2013] proposed an adaptation for the Web (reduced decompression time at the cost of a low compression ratio) of a previous progressive algorithm [Lee et al. 2012]. Other research has been also conducted to handle other types of data, like point clouds [Evans et al. 2014a], which may present different types of issues to face with. Please refer to next subsections for a more in-depth analysis.

2.2 Progressive and Multi-resolution methods

An important feature for user experience when rendering over slow connections or compressed models is progressiveness: the possibility to temporarily display an approximated version of the model and to refine it while downloading or processing the rest of the data.

The simplest (and widely used) strategy is to use a a discrete set of increasing resolution models (usually known as Level Of Detail, LOD). The main drawback with this approach is the abrupt change in detail each time a model is replaced.

A change of paradigm was brought by progressive meshes, introduced by Hoppe [Hoppe 1996]. These meshes encode the sequence of operations of a edge collapse simplification algorithm. This sequence is traversed in reverse, so that each collapse becomes a split, and the mesh is refined until the original resolution. An advantage of progressive techniques is the much more smooth transition resolution changes, and the possibility to combine it with selective refining or view-dependent multiresolution, but this high granularity was achieved at the cost of low compression rates: about 37 bpv with 10 bit vertex quantization.

A large number of progressive techniques were later developed, but as noted in [Limper et al. 2013b], Table 1, the research focus, however, was on rate-distortion performances and speed was mostly neglected. Latest algorithms still run below 200KTs in CPU.

Mobile and web application would be really too slow using these methods. As a compromise, pop buffers [Limper et al. 2013a] propose a method to progressively transmit geometry and connectivity, while completely avoiding compression.

Another desirable feature, especially for large models, is view-dependent loading and visualization. Most multiresolution algorithms were made obsolete by the increased relative performances of GPU over CPU around the first years of 2000. It simply became inefficient to operate on the mesh at the level of the single triangle. Several works [Yoon et al. 2004; Sander and Mitchell 2005; Cignoni et al. 2004; Cignoni et al. 2003] achieved much better performances by increasing the granularity of the multiresolution to a few thousand triangles.

The main problem when increasing the granularity is ensuring boundary consistency between patches at different resolution: Yoon

[Yoon et al. 2004] and Sander [Sander and Mitchell 2005] both employ a hierarchical spatial subdivision, but while the first simply disallow simplification of most boundary edges, which results in scalability problems, the second relies relies on global, spatial GPU geomorphing to ensure that progressive meshes patch simplification is consistent between adjacent blocks. The works by Cignoni [Cignoni et al. 2003; Cignoni et al. 2004] rely instead on a non hierarchical volumetric subdivision and a boundary preserving patch simplification strategy that guarantees coherence between different resolutions while at the same time ensures no boundary persists for more than one level. While not progressive in a strict sense, given current rendering speed, the density of triangles on screen is so high that popping effects are not noticeable.

Compression comes as a natural extension to this family of multiresolution algorithms: each patch can be compressed independently from the others as long as the boundary still matches with neighboring patches. a wavelet based compression was developed in [Gobbetti et al. 2006] for terrains, a 1D Haar wavelet version in [Rodríguez et al. 2013] for generic meshes on a mobile application. A comprehensive account of compression algorithms and the convergence with view-dependent rendering of large datasets can be found on a recent survey from Maglo et al.[Maglo et al. 2015].

2.3 Fast Decompression of 3D models

Given that decompression speed is a key factor in order to be able to use compressed mesh, there's been some effort by the community to provide solutions.
Gumhold and Straßer [Gumhold and Straßer 1998] developed a connectivity only compression algorithm that was able to decompress at 800KTs in 1998. Pajarola and Rossignac in [Pajarola and Rossignac 2000], in 2000, reported 26KTs for a progressive compression algorithm, and developed a high-performance Huffman decoding identifying entropy compression as a possible bottleneck. Finally, Isenburg and Gumhold in 2003 developed a streaming approach to compression of gigantic meshes reaching an impressive decompression speed of 2MTs.

3 Method

Our multiresolution algorithm builds upon the methods described on [Cignoni et al. 2004; Cignoni et al. 2005], which is recapped in section 3.1 for completeness. In our solution we adopt a improved partition strategy (see section 3.2), and, more importantly, a novel compression scheme (section 3.3) tailored around the need for decompression speed.

3.1 Batched Multiresolution

The model is split into a set of small meshes at different resolutions that can be assembled to create a seamless mesh simply traversing a tree which encodes the dependencies between each patch, using the estimated screen error to select the resolution needed in each part of the model. To build this collection of patches we need a sequence of non-hierarchical volume partitions (V-partition) of the the model; non hierarchical means essentially that no boundary is preserved between partitions at different levels of the hierarchy.

The data structure is composed of a fixed size header describing the attributes of the models, an small index which contains the tree structure of the patches and the position of each patch in the file, and the patches themselves. We use HTTP Range requests to download header and index, ArrayBuffers to parse this structures into Javascript; the patches are then download prioritizing highest screen error. Figure 2 shows an example of a model before and after view-dependent refinement.

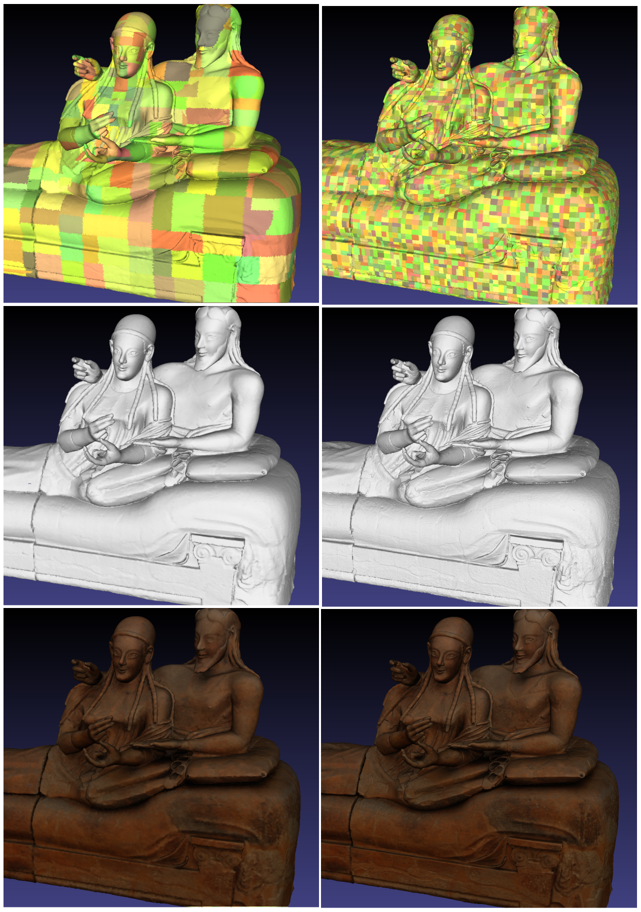


Figure 2: *First column: before refinement. Second column: after refinement. From top to bottom: a visual representation of the geometric patches representing the model, the model with pure geometry, the model with color information.*

The rendering requires the traversal of the patch tree, which is usually quite small since each patch is in the range of 16-32K vertices, computing the approximated screen space error in pixel from the bounding sphere and the quadric error (or any other error metric) during simplification. The traversal is stopped whenever our triangle budget is reached, the error target is met or the required patches are still not available.

Since the rendering can start when the first patch is downloaded and the model is refined as soon as some patch is available, this is effectively a progressive visualization albeit with higher granularity. On the other hand, this structure is view dependent and thus able to cope with very large models, on the order of hundreds of millions of triangles.

3.2 Partition

Cignoni et al [Cignoni et al. 2005] showed that any non-hierarchical sequence of volume partitions can be the base of a patch based multiresolution structure. Good partition strategy minimize boundaries thus generating compact cells. In addition, it allows streaming construction and generates well balanced trees even when the distribution of the model triangles is very irregular. The Voronoi structure, while optimal for boundary minimization and balance, is not suitable for streaming leading to long processing times. On the other hand the regular spatial subdivision used in [Cignoni et al. 2004] might generate unbalanced trees for very irregular models. This may impact on adaptivity.

In our solution each volume partition is defined by the leaves of a KD-tree built on the triangles of the model; to ensure the non hierarchical condition, the split ratio in the KD-tree alternates between 0.4 and 0.6 instead of the usual 0.5. This choice allows for streaming processing of the model and good adaptivity. As a bonus, the very regular shape of the patches (see figure 2) may be useful when adding texture support.

3.3 Mesh Compression

Our multiresolution algorithm imposes a set of constrains to mesh compression:

- each patch needs to be encoded independently from the other, so the method must be efficient and fast even on small meshes
- boundary vertices, replicated on neighboring patches, need to remain consistent through compression
- non manifold models must be supported

It would be possible to exploit the redundancy of the data due to the fact that the same surface is present in patches at different levels of resolution. We choose not to do so in order to keep the compression stage independent of the simplification algorithm used and to simplify parallel decompression of the patches (we would have to keep track of and enforce dependencies otherwise).

3.3.1 Connectivity compression

We modified the algorithm presented in [Floriani et al. 1998], to support non manifold meshes and surfaces with handles or holes.

We need face-face topology for compression and this is computed as follows: we create an array containing three edges for each triangle, and sort it so that edges sharing the same vertices will be consecutive (independently of the order of the edges). The edges are then paired taking orientation into account, and all non paired edges are marked as boundary. Non manifold meshes will simply force the creation of some artificial boundaries.

The encoding process starts with a triangle and expands iteratively adding triangles. The processed region is always homeomorphic to a disk and if the region meets already considered triangles, we consider the common vertices as duplicated. The boundary of the already processed (encoded or decoded) region is stored as a doubly linked list of oriented edges (*active edges*), The list is actually implemented as an array for performances reasons. A queue keeps track and prioritize the *active edges*.

The first triangle adds three active edges to the list; iteratively an edge is extracted from the queue and, if not marked as processed, the following codes are emitted (see Figure 3):

SKIP if the edge is a boundary edge, or the adjacent triangle has already been encoded; the edge is marked as processed.

LEFT or **RIGHT** if the adjacent triangle shares two edges with the boundary; The two edges are marked as processed, a new edge added to the queue and its boundary adjacencies adjusted.

VERTEX if the adjacent triangle shares only one edge with the boundary, in this case the edges is marked as processed and two new edges added to the queue. If vertex of the new triangle opposing the edge was never encountered before its position is estimated using parallelogram prediction and the difference encoded, otherwise its index is encoded (in literature this case is often referred as a “split”). This is a key difference with [Floriani et al. 1998], where in the second case a SKIP code would be emitted, to keep the encoded region simple.

If the mesh is composed of several connected components, the process is restarted for each component.

The order in which the active edges are processed is important as we would like to minimize the number of VERTEX split operations, and generate a vertex-cache-friendly triangle order. To do so, we simply prioritize the right edges in the VERTEX operation, so that the encoding proceeds in 'spirals'. If the mesh is not homeomorphic to a disk, some split operations are required. This strategy reduces the number of splits to less than 1% in our examples, incurring in an average of 0.2 bpv cost.

This algorithm is certainly not optimal in term of bitrate, but it is extremely simple, linear in the number of triangles and robust to non-manifold meshes; as we will see in the results, speed is more important than bitrate.

3.3.2 Geometry and vertex attribute compression

To ensure consistency between boundary vertices of adjacent patches, we adopt a global quantization grid for coordinates, normals and colors. The global grid step for vertex position quantization is chosen automatically based on the quadric errors during the simplification step in construction.

Geometry and vertex attributes are encoded as differences to a predicted value. The distribution of these values exhibit a bias which we can exploit to minimize the number of bits necessary to encode them. Our strategy is based on the assumption that most of the bias is concentrated on the position of highest bit (the log_2 of the value) of these value while the subsequent bits are mostly random. We simply store in an array, which is later entropy coded, the number of bits necessary to encode the value; the subsequent bits are stored in an uncompressed bitstream. In this way we need to decode a single symbol, from a limited alphabet, and read a few bits from a bitstream to decode a difference.

Each new vertex position, result of a VERTEX code, is estimated using a simple parallelogram predictor, and the differences with the actual position encoded as above. Color information is first converted into YCbCr color space and quantized, we encode the difference with one of the corner of the edge processed when emitting the VERTEX code. Normals vector are estimated using the decode mesh position and connectivity, and differences encoded as usual.

3.4 Entropy coding

We have shown how to convert connectivity, geometry and attributes into a stream of symbols and bits. It is worth compressing the symbol stream due to the biased probability distribution of the symbols.

Entropy decoding is the speed bottleneck in many mesh decompression methods, often due to the main goal of minimizing bit per vertex. Pajarola and Rossignac [Pajarola and Rossignac 2000] developed a high-performance Huffman decoding algorithm in order to overcome this problem. The main advantage of this method is that it reduces the decoding phase to a couple of table lookups. Arithmetic coding, for example, outperforms Huffman in term of compression rate, but exhibits lower speed. A problem with this approach is the initialization time required to create the, possibly very large, decoding tables. It is then not suitable for decoding

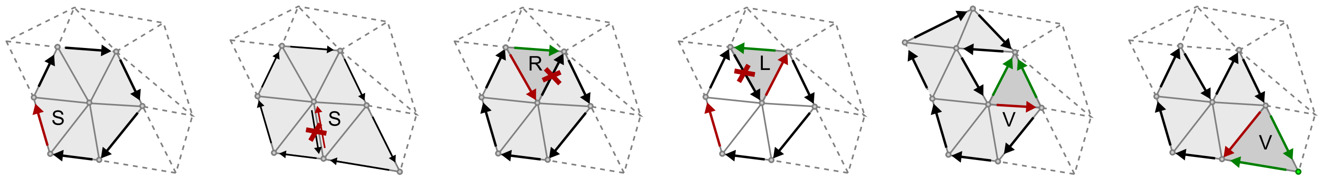


Figure 3: *The four decompression codes: black arrows represent the front, the red arrow the current edge, in green the new edges added to the front.*

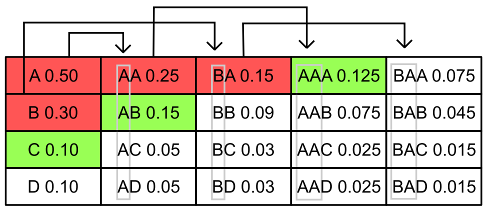
small meshes where the construction time would dominate over the decoding time.

Unlike Huffman and other variable-length codes, Tunstall code [Tunstall 1967] maps a variable number of source symbols to a fixed number of bits. Since in decompression the input blocks consists of a fixed number of bits and the output is a variable number of symbols, Tunstall is slightly less efficient than Huffman, especially where the bit size of the input block is small. The decoding step is very similar to the high-performance Huffman algorithm, as it consists in a lookup table and a sequence of symbols for each entry, but the table size is only determined by the word size, and a fast method to generate it described in [Baer 2009].

Given an entropic source of M symbols, to generate an optimal encoding table for a word size of N bits, we need to generate 2^N symbol sequences that have a frequency as close as possible to 2^{-N}, allows to encode every possible input (it is complete) and no sequence is a prefix of any other sequence (it is proper).

Tunstall optimal strategy starts with the M symbols as initial sequences, removes the most frequent sequence A and replaces it with M sequences concatenating A with every symbol until we reach 2^N sequences. The algorithm most time consuming step is to find the most probable sequence.

If we use a matrix where the first column contains the sorted symbol in order of probability, and at each step we replace the sequence with highest probability with M sequences adding a new column, we can observe that this table is sorted both in columns and rows (see Figure 4). This allows to select the next sequence by keeping each row in a queue and using a priority queue to keep track of which queue has the highest front element.



A 0.50	AA 0.25	BA 0.15	AAA 0.125	BAA 0.075
B 0.30	AB 0.15	BB 0.09	AAB 0.075	BAB 0.045
C 0.10	AC 0.05	BC 0.03	AAC 0.025	BAC 0.015
D 0.10	AD 0.05	BD 0.03	AAD 0.025	BAD 0.015

Figure 4: *First four steps in construction of a Tunstall code with four symbols, the sequences A, B, AA, BA are replaced with a new column, beside each sequence, its probability is shown. In green the candidates for the next expansion.*

To initialize the decoding table the symbol frequencies needs to be transmitted in advance.

Finally, an important advantage of variable-to-fixed coding is that the compressed stream is random accessible: decoding can start at any block. This makes it especially suited for parallel decompression in particular GPU decompression. Unfortunately, current limitations in the capabilities of WebGL do not allow for such an implementation.

4 Results

The C++ and Javascript implementation is freely available at http://vcg.isti.cnr.it/nexus under GPL licence.

Our implementation has been successfully tested on major browsers on a variety of platform, from desktop machines to low end cell phones. The results we report here were measured on an iCore5 3.1Gh, using Chrome 41. Timings taken other browsers (e.g.Firefox) where comparable.

The multiresolution model construction is a preprocessing operation, and the bottleneck is the quadric simplification algorithm that runs at about 60K triangles per second per core. Compression time is negligible at about 1M triangles per second.

4.1 Entropy Compression: Comparison

We tested, both in C++ and Javascript, compression rates and decompression speed of:

- our implementation of Tunstall coding (T)
- Huffman coding (H), in the high-performance version of Pajarola [Pajarola and Rossignac 2000] (our implementation, C++ only)
- available implementations of LZMA
  in C++: http://www.7-zip.org/sdk.html
  and Javascript: https://code.google.com/p/js-lzma/
- lz-string, a LZW based Javascript implementation
  http://pieroxy.net/blog/pages/lz-string/index.html

	C++			Javascript		
symbols	T	H	LZMA	T	LZMA	LZW
4	1058	520	1066	201	19	55
9	369	212	170	145	10	23
13	423	168	95	150	6	20
17	359	136	77	163	6	19
22	332	98	67	180	6	17

Table 1: *Decompression speed in million of output symbols per second for Poisson distribution of 32K sequences*

The results are presented in Table 1, the lenght of 32K has been chosen since it is typical in our application.

Huffman and Tunstall are very similar in term of decompression speed, the difference is mainly in the time required to generate the decoding tables which are much larger for Huffman, especially when increasing the number of symbols. We tested also other probability distributions and found little difference in terms of speed.

LZMA and LZW avoid this startup cost, however their more complex and adaptive dictionary management allows them to outperform Huffman and Tunstall in term of decompression speed only for very small runs (and very small dictionaries). In terms of compression ratio, Huffman and LZMA performed quite close to the theoretical minimum, while Tunstall was about 10% worse.

We did not implement Huffman in Javascript, as we are confident the result would be very similar. On the other hand the numbers for LZMA change dramatically. Lz-string serves as a comparison, as a better library, optimized for Javascript. The poor LZMA performances in Javascript help explain the relatively slow performances of CTM in Limper [Limper et al. 2013b].

4.2 Mesh Compression: Comparison

We used the Happy Budda model (in Figure 1), to compare compression ratio and decompression speed with OpenCTM (CTM) [Geelnard] Pop buffers (POP)[Limper et al. 2013b], P3DW [Lavoué et al. 2013], WebGL-loader (CHUN) [Chun 2012]. We compare our multiresolution (OUR) and, to test single resolution performances of our compression approach, a version (FLAT) which loads only the highest resolution level of the model. In each case the model has been quantized at 11 bit for coordinates and 8 bit for normals, and includes colors.

	FLAT	OUR	CTM	CHUN	POP	P3DW
MB	1.9	3.9	3.5	2.8	15	4.5
bpv	28	57	51	41	220	66
full	0.4	0.9	5.3	0.06	0.5	10

Table 2: *Statistics for the Happy Buddha: model size in megabytes, bit per vertex and time in seconds required to fully decompress the model.*

Our decompression Javascript implementation can decode about 1-3 million triangles per second with normals and colors in a single thread, on a desktop machine and 0.5 MT/s on a iPhone Five. Performances are somewhat degraded when the code is run during streaming visualization.

An important comparison is with [Rodríguez et al. 2013], which employs the same multiresolution batched strategy. For their mobile multiresolution application they reports compression rates of 45-50 bpv on large colored meshes (which should be compared to our 28bpv). The difference is probably mostly due to the different connectivity encoding which, in their case, requires 20bpv against our 4 or 5bpv. It is difficult to compare the speed of the two decompression approaches since they run natively in C# on an iPhone4 while we run in Javascript on the same platform. Our implementation speed is still, if a bit faster than their 50KTS [1], at about 60KTs. The difference is probably due their more sophisticate (and slow) arithmetic encoding.

C++ decompression speed is of course faster, reaching 9MTs, including colors and normals, and 16MTs for just position and connectivity. The speed reported in [Floriani et al. 1998] of 35KTs for just the connectivity, as they mention, is due to the dynamic memory allocation in their implementation.

4.3 Streaming and Rendering

Loading the geometry through the Range HTTP request requires an increased number of HTTP calls: one for each patch, or 30-60 calls every million of triangles. This does not really impact over performances: the overhead is quite small (about 400 bytes per call) and pipelining (the process of enqueueing requests and responses between browser and server) ensures full utilization of the available bandwidth. Random access is really necessary only to fully exploit the view-dependent characteristics of the multiresolution structure: the code could be easily modified to load the model with a single call if a higher number of HTTP calls was problematic on certain web hosting architectures.

In the demo page (http://web3d.duckdns.org) it is possible to compare the performances of our method w.r.t. existing solutions in the case of a slow connection. Moreover, very complex geometries are also available for further testing. As an example, in Figure 5 we show our system rendering the Portalada, a 180M triangles model at 30fps. The triangle budget has been fixed at 1M triangles and the streaming requires 2-3 seconds to reach full resolution on a good connection. The original model is 3.6GB, while the compressed multiresolution model is 838MB.

Figure 6 shows two examples where the method deals with non-optimal geometries. On the left side, a model exhibiting strong topological artifacts. On the right side, a model with very unbalanced data density. In both cases, the method is able to deal with the issues and provide an accurate and reliable rendering.

5 Conclusion

The method proposed in this paper provides good compression ratio, progressive visualization, fast decoding and view dependent rendering. It proves effective in a wide range of bandwidth availability, computing power and rendering capabilities. Moreover, it is able to handle models of arbitrary size. This means that also very complex geometries can be now explored in real time with average connections speeds.

Many mesh compression algorithms for mobile and web application do not employ topological connectivity compression often because it is believed to be excessively complex or slow and limited to manifold meshes. We prove that, if implemented correctly, this is not the case, and the choice of the entropy compression algorithm can play a much more important role.

5.1 Future improvements

An important limitation of the current implementation is the lack of texture support. Adding UV coordinates to the multiresolution structure and supporting them in compression is a trivial task, but dealing with simplification and providing multiresolution textures is much more difficult. We plan to tackle this problem in the near future, with an approach similar to Texture Mapping Progressive Meshes [Sander et al. 2001].

Point clouds are currently supported, with a z-index vertex compression strategy, we are working on improving the presentation. While the multiresolution structure is not really needed, using it has the advantage of working on an single framework.

Acknowledgements

The research leading to these results was funded by EU FP7 project ICT Harvest4D (http://www.harvest4d.org/ , GA n. 323567) and EU INFRA Project Ariadne (GA n. 313193, http://www.ariadne-infrastructure.eu/).

[1] The number is extrapolated from the decoding time of a large mesh given in their paper

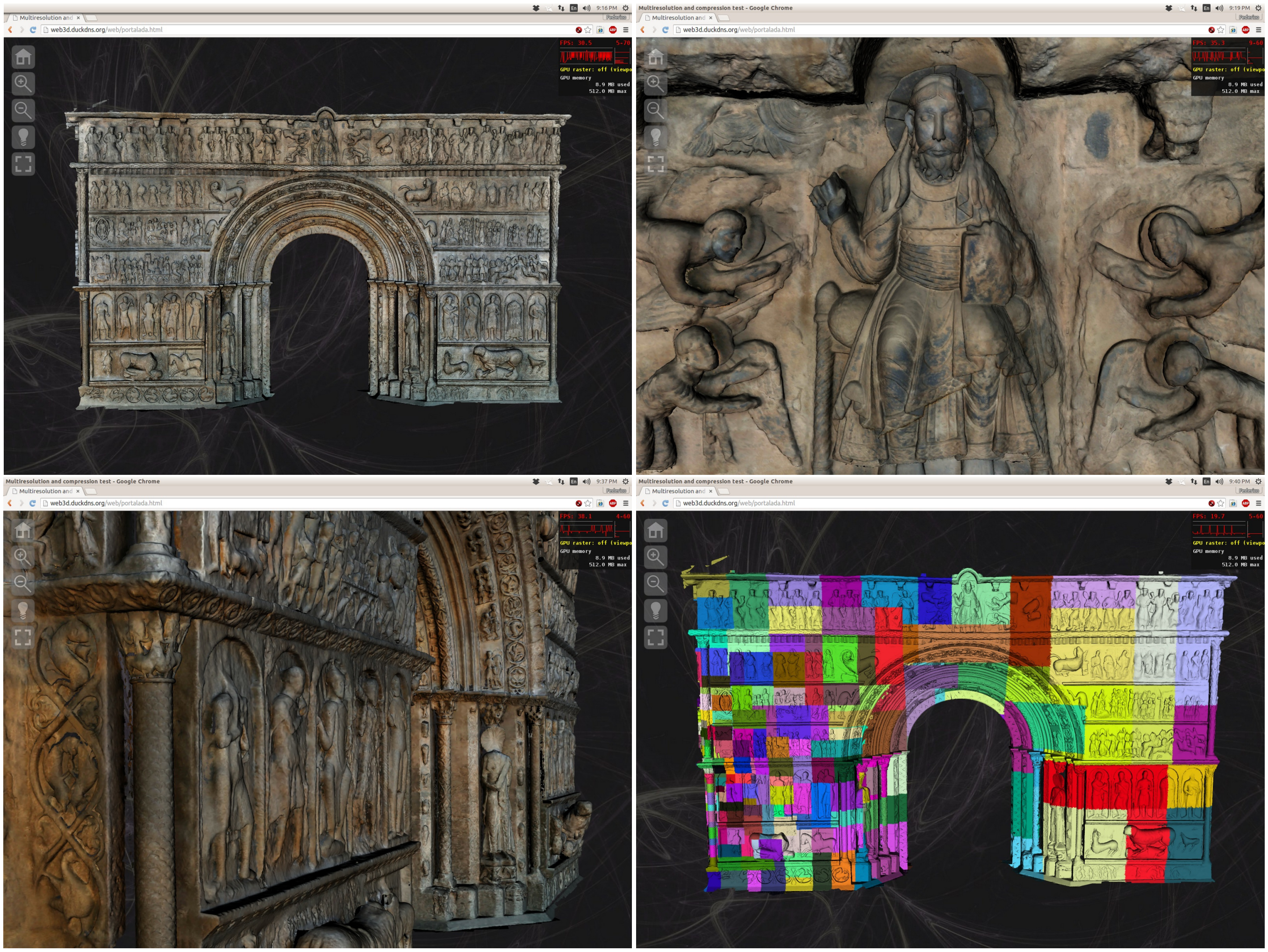


Figure 5: *Portalada rendered in a browser: top left: the full model, top right: a detail of the figure above the arch, middle right: the resolution of the model as seen from the middle left view point (without frustum culling)*

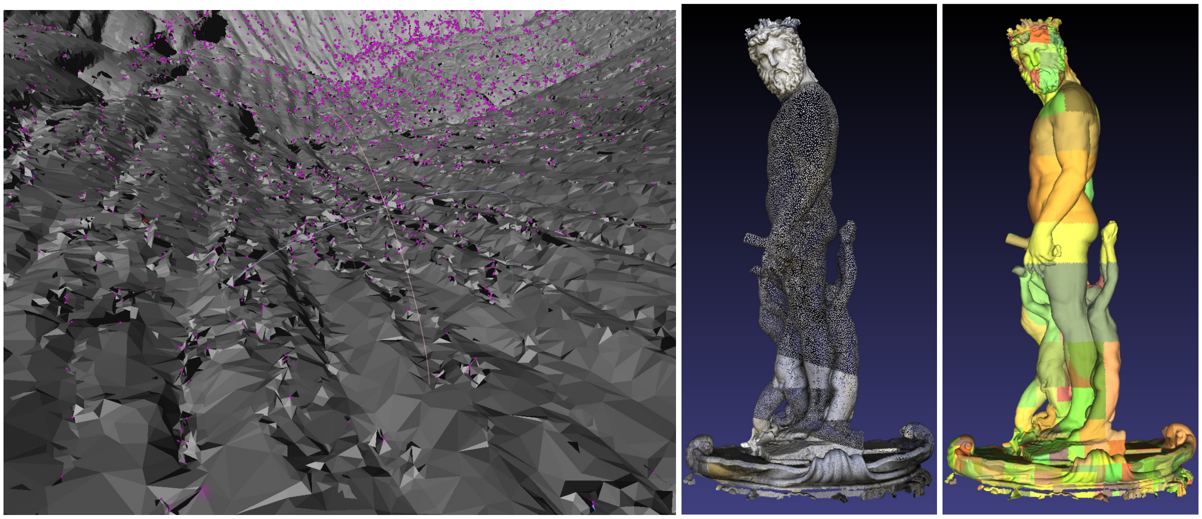


Figure 6: *Left: a model with severe topological issues. Right: a model with very imbalanced vertex distribution*

References

ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 195–202.

BAER, M. 2009. Efficient implementation of the generalized tunstall code generation algorithm. In *Proceedings of the 2009 IEEE International Conference on Symposium on Information Theory - Volume 1*, IEEE Press, Piscataway, NJ, USA, ISIT'09, 199–203.

BALSA RODRIGUEZ, M., GOBBETTI, E., MARTON, F., AND TINTI, A. 2013. Compression-domain seamless multiresolution visualization of gigantic meshes on mobile devices. In *Proc. ACM Web3D International Symposium*, New York, NY, USA, ACM Press, 99–107.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '09, 127–135.

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 *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '12, 17–25.

BLUME, A., CHUN, W., KOGAN, D., KOKKEVIS, V., WEBER, N., PETTERSON, R. W., AND ZEIGER, R. 2011. Google body: 3d human anatomy in the browser. In *ACM SIGGRAPH 2011 Talks*, ACM, New York, NY, USA, SIGGRAPH '11, 19:1–19:1.

CHUN, W. 2012. Webgl models: end-to-end. In *OpenGL Insights*, P. Cozzi and C. Riccio, Eds. CRC Press, 431452. `http://www.openglinsights.com/`.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. Batching meshes for high performance terrain visualization. In *Second Annual Conference of Eurographics Italian Chapter*, Eurographics Association, Milano (ITALY), Conference Series.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3, 796–803.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2005. Batched multi triangulation. In *Proceedings IEEE Visualization*, IEEE Computer Society Press, Conference held in Minneapolis, MI, USA, 207–214.

DELILLO, B., 2009. WebGLU: A utility library for working with WebGL . `http://webglu.sourceforge.org/`.

DI BENEDETTO, M., PONCHIO, F., GANOVELLI, F., AND SCOPIGNO, R. 2010. Spidergl: a javascript 3d graphics library for next-generation www. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, New York, NY, USA, Web3D '10, 165–174.

DIRKSEN, J., Ed. 2013. *Learning Three.js: The JavaScript 3D Library for WebGL*. Packt Publishing.

DON BRUTZMANN, L. D. 2007. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann.

EVANS, A., AGENJO, J., AND BLAT, J. 2014. Web-based visualisation of on-set point cloud data. In *Proceedings of the 11th European Conference on Visual Media Production*, ACM, New York, NY, USA, CVMP '14.

EVANS, A., ROMEO, M., BAHREHMAND, A., AGENJO, J., AND BLAT, J. 2014. 3d graphics on the web: A survey. *Computers & Graphics 41*, 0, 43 – 61.

FLORIANI, L., PUPPO, E., AND MAGILLO, P. 1997. A formal approach to multiresolution hypersurface modeling. In *Geometric Modeling: Theory and Practice*, W. Strasser, R. Klein, and R. Rau, Eds., Focus on Computer Graphics. Springer Berlin Heidelberg, 302–323.

FLORIANI, L. D., MAGILLO, P., AND PUPPO, E. 1998. Compressing tins. In *In Proceedings of the 6th ACM Symposium on Advances in Geographic Information Systems*, 145–150.

GEELNARD, M. OpenCTM. `http://openctm.sourceforge.net/`.

GOBBETTI, E., MARTON, F., CIGNONI, P., BENEDETTO, M. D., AND GANOVELLI, F. 2006. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (September), 333–342. Proc. Eurographics 2006.

GOBBETTI, E., MARTON, F., RODRIGUEZ, M. B., GANOVELLI, F., AND DI BENEDETTO, M. 2012. Adaptive quad patches: An adaptive regular structure for web distribution and adaptive rendering of 3d models. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '12, 9–16.

GUMHOLD, S., AND STRASSER, W. 1998. Real time compression of triangle mesh connectivity. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 133–140.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 99–108.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 269–276.

ISENBURG, M., AND GUMHOLD, S. 2003. Out-of-core compression for gigantic polygon meshes. *ACM Trans. Graph. 22*, 3 (July), 935–942.

JANKOWSKI, J., RESSLER, S., SONS, K., JUNG, Y., BEHR, J., AND SLUSALLEK, P. 2013. Declarative integration of interactive 3d graphics into the world-wide web: Principles, current approaches, and research agenda. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 39–45.

KHRONOS GROUP, 2009. OpenGL ES - The Standard for Embedded Accelerated 3D Graphics.

KHRONOS GROUP, 2009. WebGL - OpenGL ES 2.0 for the Web.

KLEIN, S. T., AND SHAPIRA, D. 2011. On improving tunstall codes. *Inf. Process. Manage. 47*, 5 (Sept.), 777–785.

LAVOUÉ, G., CHEVALIER, L., AND DUPONT, F. 2013. Streaming compressed 3d data on the web using javascript and webgl. In

Proceedings of the 18th International Conference on 3D Web Technology, ACM, New York, NY, USA, Web3D '13, 19–27.

LEE, J., CHOE, S., AND LEE, S. 2010. Mesh geometry compression for mobile graphics. In *Proceedings of the 7th IEEE Conference on Consumer Communications and Networking Conference*, IEEE Press, Piscataway, NJ, USA, CCNC'10, 301–305.

LEE, H., LAVOU, G., AND DUPONT, F. 2012. Rate-distortion optimization for progressive compression of 3d mesh with color attributes. *The Visual Computer 28*, 2, 137–153.

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.

LIMPER, M., WAGNER, S., STEIN, C., JUNG, Y., AND STORK, A. 2013. Fast delivery of 3d web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology*, 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 *The 19th International Conference on Web3D Technology, Web3D '14, Vancouver, BC, Canada, August 8-10, 2014*, 35–43.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. Src - a streamable format for generalized web-based 3d data transmission. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, ACM, New York, NY, USA, Web3D '14, 35–43.

LIN, G., AND YU, T. P. Y. 2006. An improved vertex caching scheme for 3d mesh rendering. *IEEE Transactions on Visualization and Computer Graphics 12* (July), 640–648.

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.

2013. Microsoft ActiveX Controls. `http://msdn.microsoft.com/`.

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Squeeze: Fast and progressive decompression of triangle meshes. In *Proc. Computer Graphics Int'l (CGI 2000*, 173–182.

RAGGETT, D. 1995. Extending WWW to support platform independent virtual reality. *Technical Report*.

RODRÍGUEZ, M. B., GOBBETTI, E., MARTON, F., AND TINTI, A. 2013. Compression-domain seamless multiresolution visualization of gigantic triangle meshes on mobile devices. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 99–107.

SANDER, P. V., AND MITCHELL, J. L. 2005. Progressive buffers: View-dependent geometry and texture lod rendering. In *Proceedings of the Third Eurographics Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '05.

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.

SANDER, P. V., NEHAB, D., AND BARCZAK, J. 2007. Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. Graph. 26* (July).

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. Xml3d: Interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, New York, NY, USA, Web3D '10, 175–184.

SUTTER, J., SONS, K., AND SLUSALLEK, P. 2014. Blast: A binary large structured transmission format for the web. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, ACM, New York, NY, USA, Web3D '14, 45–52.

TUNSTALL, B. 1967. *Synthesis of Noiseless Compression Codes*. Georgia Institute of Technology.

YOON, S.-E., SALOMON, B., GAYLE, R., AND MANOCHA, D. 2004. Quick-vdr: interactive view-dependent rendering of massive models. In *Visualization, 2004. IEEE*, 131–138.