

Progressive Lower Trees of Wavelet Coefficients: Efficient Spatial and SNR Scalable Coding of 3D Models

Marcos Avilés, Francisco Morán, and Narciso García

Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid,
E.T.S. Ing. Telecomunicación — 28040 Madrid, Spain
{mar, fmb, ngs}@gti.ssr.upm.es

Abstract. We perform an in-depth analysis of current state-of-the-art wavelet-based 3D model coding techniques and then present a new one that outperforms them in terms of compression efficiency and, more importantly, provides full spatial and SNR scalability: PLTW (Progressive Lower Tree Wavelet) coding. As all SNR scalable bit-streams, ours can be used in heterogeneous networks with a wide range of terminals, both in terms of processing power and bandwidth. But because of being spatially scalable, the PLTW bit-stream does not impose on the less powerful terminals the need of building detail trees as deep as required by the maximum LOD, because the wavelet coefficients are sent on a per-LOD basis, thus achieving a “local” SNR scalability within a “global” spatial scalability. In particular, we show that our technique provides a substantial advantage over the only similar one in a current ISO standard (MPEG-4), and thus suggest that PLTW be considered for its future versions.

1 Introduction

Traditional coding algorithms have only focussed on efficient compression, with the sole objective of optimizing data size for a given reconstruction quality. However, due to the growth of the Internet and networking technology, users with different processing capabilities and network bandwidth can easily communicate. As a result, efficient compression alone is not enough. A new challenge appears: providing a single flexible bit-stream that can be consumed by multiple users with different terminal capabilities and network resources. Scalable coding is the response to this challenge. A scalable coder produces a bit-stream containing embedded subsets, each of which represents increasingly better versions of the original data, either in terms of resolution or distortion. Different parts of the bit-stream can then be selected and decoded by the terminal to meet certain requirements. Low performance terminals will only decode a small portion of the bit-stream and reconstruct a low quality and/or resolution version of the source, whereas higher performance decoders will take advantage of the reception of bigger portions to achieve higher quality and/or resolution.

Two different types of scalability can be typically applied to 3D geometry coding, exactly as in the better known case of image coding: SNR (Signal to Noise Ratio) scalability and spatial scalability. SNR scalability refers to the possibility of having different terminals decode the 3D model (or image) with the same spatial resolution

but with different fidelity. On the other hand, spatial scalability is the feature in the bit-stream that allows to decode the 3D model (or image) with different spatial resolutions, i.e., number of vertices/facets (or pixels).

Both scalability types are desirable, and so is progressiveness in the bit-stream, especially in the case of streaming scenarios where bandwidth is a limiting factor and it is important to be able to receive and display a coarse version of the media as soon as possible, and later refine it gradually, as more data are transmitted. However, for terminals with low processing power, both in terms of CPU and RAM, and with low resolution screens, as is the case of cell phones (and to a lesser extent that of PDAs), perhaps the most important kind of scalability is the spatial one. There is little point in encoding a 3D mesh so that may be progressively refined to have 100000 triangles if the cell phone that must render it can barely handle hundreds. And there is even less point in doing so if the decoding process that must take place prior to the rendering one will completely eat up all the cell phone resources. And even less if, anyway, nobody will be able to tell the difference between a 100 triangle mesh and a 1000 triangle one when rendered on a screen of 200x200 pixels!

In this paper we review the scalability of current state-of-the-art WSS (Wavelet Subdivision Surface) coders [3] and then present a new algorithm that outperforms previous techniques in terms of compression efficiency and, more importantly, provides full spatial and SNR scalability. We have chosen the WSS modelling/coding paradigm for its adequacy to approximating and manipulating 3D surfaces at different LODs (Levels Of Detail). The successive control meshes yielded along the subdivision process, usually called LODs themselves, are pyramidally nested, and inherently define a multi-resolution model of the limit 3D surface, which is most appropriate to address spatial scalability — without forgetting the SNR scalability, which comes naturally from using wavelet coefficients.

The rest of this paper is organized as follows. Section 2 reviews the SPIHT algorithm [6] and analyses both its scalability and the extra benefits of entropy coding its output. In Section 3, we present our PLTW coder and describe the proposed modifications to the LTW algorithm [5] to achieve both SNR and spatial scalability. Section 4 compares the rate-distortion performance of the proposed coder with other SPIHT-based coders. Finally, Section 5 concludes our presentation.

2 Analysis of SPIHT-Based Coders

For almost a decade already, the SPIHT algorithm [6] has been the reference against which to compare other coding techniques based on the wavelet transform. It was originally designed to code scalar wavelet coefficients, but this has been no obstacle for extending it to handle 3D coefficients, such as the ones resulting from colour images or 3D surfaces modelled thanks to WSSs [2][3][4]. Typically, a coordinate transform is applied to the coefficients coming from the RGB pixels or XYZ detail vectors, so that their three components are decorrelated, and then the SPIHT algorithm is run on each of the three transformed components, yielding three independent bit-streams whose bits are interleaved in some sensible way.

2.1 Scalability

Due to the coefficient ordering and bit-plane encoding achieved by the SPIHT algorithm, the resulting bit-stream is inherently SNR scalable: every bit contributes to reduce the reconstruction error as much as possible for the number of already read bits. This feature allows the decoder to simply stop reading from the bit-stream at any point and have the best possible reconstructed model for that number of decoded bits. Typical rate-distortion curves for a geometry coder based on the SPIHT algorithm are shown in Fig. 1. The horizontal axis reflects the number of bits per vertex while the vertical one shows the PSNR, defined as $PSNR = 20 \log_{10}(bbox/d)$, where $bbox$ is the bounding box diagonal, and d is the L^2 distance between the original and reconstructed meshes, measured with MESH [1].

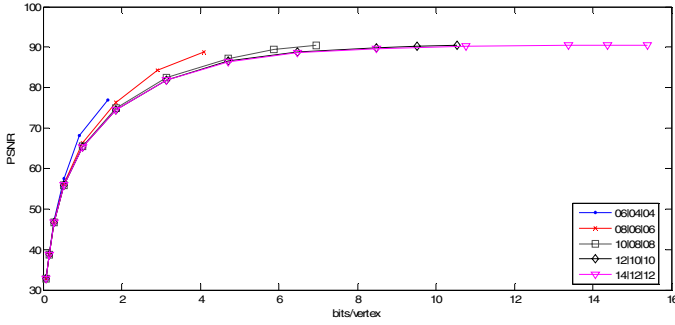


Fig. 1. Rate-distortion curves for different quantization schemes

Besides, as the generated bit-stream is fully embedded, it can also be truncated at a given size in case the target file size is a restriction. However, one might wonder whether it is better to cut the (long) bit-stream produced by running the coder over finely quantized coefficients, or to consume the whole (shorter) bit-stream resulting from coarsely quantized coefficients. Fig. 1 shows the rate distortion curves obtained for bit-streams produced with different sets of quantization values. In all the cases, coefficients have been expressed using local frames to decorrelate energy and save bits by quantizing each of the tangential components four times more coarsely (i.e., with two bits less) than the normal one [2][3]. The three values in each of the entries of the legend reflect the number of bits assigned to the normal and two tangential components, respectively: $nbits_N|nbits_{T1}|nbits_{T2}$. It can be seen how, for a given target size S , the best option is to choose the values $nbits_N$, $nbits_{T1}$ and $nbits_{T2}$ (note that there is only one variable if one sets $nbits_{T1} = nbits_{T2} = nbits_N - 2$) so that the resulting bit-stream is slightly larger than S , and then truncate it to S . In fact, quantization sets of high values (e.g., 14/12/12) hardly contribute to lessen the reconstruction error compared to sets with lower values (08/06/06 or 10/08/08), even though the generated bit-stream is substantially larger.

Nevertheless, although the SPIHT coder is fully SNR scalable, it does not support spatial scalability and does not yield a bit-stream that can be easily parsed according to a given maximum resolution (i.e., number of subdivisions or LODs) imposed by

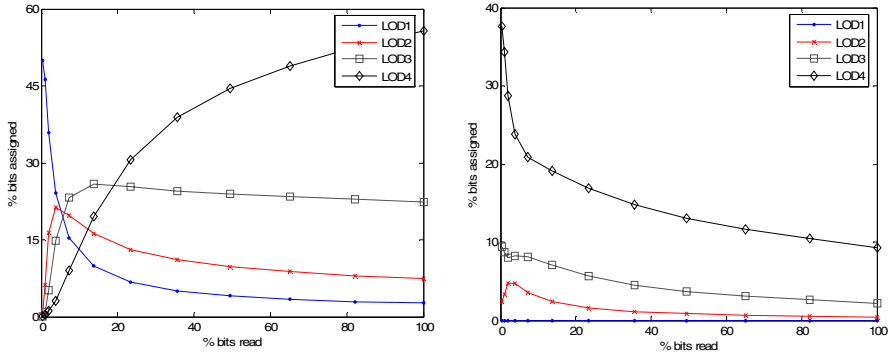


Fig. 2. Bit-stream distribution among different LOD's for the venus model: coefficient (left) and navigation (right) bits assigned to each LOD

the decoder. The reason for this is that the sorting pass of the SPIHT algorithm groups together coefficients with similar magnitude, independently of the LOD they belong to, and then outputs their bits interleaved. The wavelet transform decomposes the initial model into bands with details of diminishing energy, which makes very likely that coefficients from the root or upper branches of the detail tree are sent before those of the leaves. However, Fig. 2 proves how that assumption is not always true by showing the distribution among levels of the read bits for each part of the bit-stream. For this bit-to-level assignment, we have distinguished two types of bits: bits that can be directly attributed to a single coefficient; and “navigation” bits gathering information from multiple coefficients (which are the descendants of some other), which are used to guide the algorithm through the different stages. The assignment of the first ones is straightforward and it is depicted in Fig. 2 (left). For the second ones, we have decided to split each bit between all the coefficients involved and impute them only a fraction of that bit. For instance, let us call n_{subdiv} the total number of subdivisions (in Fig. 2, $n_{subdiv} = 4$), and consider one of the navigation bits that must be output to reflect the significance, with respect to the current bit-plane, of the descendants of a given coefficient of LOD $n_{subdiv} - 2$. As such a coefficient has (in general) four sons and sixteen grandsons, $4/20$ bits will be imputed to LOD $n_{subdiv} - 1$ and $16/20$ to LOD n_{subdiv} . The fraction of navigation bits vs. the fraction of read bits is plotted in Fig. 2 (right). As it is shown, at any stage of the bit-stream, the bit contribution comes from all levels, and even at its beginning, where almost all the coefficient bits come from lower LODs, the amount of navigation bits from LOD 4 is especially high — which is not at all desirable if a terminal can only handle, say, LOD 2, as it may already be obvious, but explained further in Section 4.

2.2 Entropy Coding

Even though the SPIHT algorithm is very efficient at exploiting the correlations between the detail of a parent and those of its descendants, its output can be further compressed by using entropy coding. In this respect, arithmetic coding [8] is probably the best choice, as it allows to use a fractional number of bits per symbol, although at the expense of larger (de)coding complexity.

As already stated in the original SPIHT paper [6], there is little gain by entropy coding coefficient signs or bits output during the refinement pass. However, significance bits are not uniformly distributed: for early bit-planes, most coefficients are insignificant, resulting in zeroes, whereas for later bit-planes, many coefficients become significant, resulting in ones. Besides, there is also a statistical dependence between the significance of a coefficient and those of its descendants, and between spatially close coefficients.

We have then implemented arithmetic coding of the coefficients using two approaches. The first one consists in using an adaptive model for each type of bit (significance, sign, refinement and so on) sent to the output, reflecting the fact that some symbols tend to be grouped. The second approach takes also advantage of the spatial correlation between coefficients by grouping the significance bits of the children of a detail and coding them as a single symbol [6].

Table 1 shows the efficiency gain obtained by using arithmetic coding. There is a benefit of around 6% when the first approach (row “AC”) is followed and around 8% when the more complex modelling (row “AC++”) is considered. In a typical asymmetric scenario where one encoder runs as an off-line process on a much more powerful machine than the ones used by the many decoders, so encoder complexity is not as big an issue as decoder complexity is. The ultimate decision of whether to use entropy coding or not is then a compromise between compression and decoder complexity. Nevertheless, notice that the second approach is hardly preferable to the first one, as it only provides a very slight compression improvement, whereas the complexity of its decoder (especially in terms of memory) is considerably larger.

Table 1. Effect of arithmetic coding of the SPIHT bit-stream for some models

model	venus	bunny	horse	dinosaur
no AC	1414885	646111	662048	1466663
AC	1326293	603238	621358	1381779
AC++	1298064	590795	607414	1353311

3 Proposed Technique: Progressive Lower Tree Wavelet Coder

The proposed technique builds upon the work described in [5] for still image coding, which represents a low complexity alternative to traditional wavelet coders, like SPIHT [6] and EBCOT [7], with a higher compression performance and featuring spatial scalability. As a counterpart, it does not provide full SNR scalability due to the fixed, sequential traversal of the coefficients. To achieve both spatial and SNR scalability, we have introduced bit-plane encoding of the coefficients for each LOD.

3.1 Algorithm

In this algorithm, the quantization process is performed by following two strategies: a scalar uniform quantization of the coefficients with *nbits* bits is applied prior to the encoding itself; later on, once the algorithm is executed, the *rplanes* least significant

```

0. INITIALIZATION
  output nbits
  output rplanes
1. CALCULATE LABELS
  scan the bottom-most detail level in blocks B of coefficients within the same tree
  for each  $d \in B$ 
    if  $d < 2^{rplanes} \forall d \in B$ 
       $d := LOWER\_COMPONENT \forall d \in B$ 
    else
      for each  $d \in B$ 
        if  $d < 2^{rplanes}$ 
           $d := LOWER$ 
  scan the rest of levels from bottom to top in blocks B of coefficients within the tree (top
  level details will be arranged in groups of 4)
  for each  $d \in B$ 
    if  $d < 2^{rplanes} \wedge d' = LOWER\_COMPONENT \forall d \in B, \forall d' \in O(d)$ 
       $d := LOWER\_COMPONENT \forall d \in B$ 
    else
      for each  $d \in B$ 
        if  $d < 2^{rplanes} \wedge d' = LOWER\_COMPONENT \forall d' \in O(d)$ 
           $d := LOWER$ 
        if  $d < 2^{rplanes} \wedge d' \neq LOWER\_COMPONENT \forall d' \in O(d)$ 
           $d := ISOLATED\_LOWER$ 
2. OUTPUT COEFFICIENTS
  scan the detail levels from top to bottom
   $LIC := LSC := \emptyset$ 
  Add all the details not labeled as LOWER_COMPONENT to the LIC
  output nbb (number of bits needed to code the largest coefficient in the LOD)
  set  $n := nbb$ 
  while  $n > rplanes$ 
    for each  $d \in LIC$ 
      output  $b := n^{th}$  bit of  $d$ 
      if  $b = 1$ 
        output  $sgn(d)$ 
        if  $O(d) \neq \emptyset$ 
          output  $b := (d' \neq LOWER\_COMPONENT) \forall d' \in O(d)$ 
          move  $d$  to the LSC
      for each  $d \in LSC$  (excluding entries just appended in this same pass)
        output  $b := n^{th}$  bit of  $d$ 
      for each  $d \in LIC$ 
        output the label of  $d$  (can only be LOWER or ISOLATED_LOWER)
    decrement  $n$  by one

```

Fig. 3. Proposed encoding algorithm

bit-planes of the coefficient are removed. As it can be seen in Fig. 3, the final algorithm has two main stages.

During the first step, the coefficient trees are scanned from the leaves (bottom) to the root (top) to try and grow subtrees of what we call “irrelevant” coefficients, i.e.,

lower than $2^{rplanes}$. A node whose descendants (represented as $O(d)$ in Fig. 3) are all irrelevant is labelled as *LOWER_COMPONENT*. To start, all coefficients of the bottom-most level (i.e., LOD n_{subdiv}) sharing the same parent (of LOD $n_{subdiv} - 1$) are checked for irrelevance, and hopefully labelled all as *LOWER_COMPONENT*. Later on, if an upper level node N is irrelevant and all its sons are labelled as *LOWER_COMPONENT*, then N is a good candidate to be tagged itself as *LOWER_COMPONENT*, yielding a new, larger lower-tree, providing that all its siblings are also candidates to be tagged as *LOWER_COMPONENT*. However, if any of the sons of a node N is relevant, the lower-tree cannot continue growing upwards. In that case, those irrelevant sons of N having all their sons labelled as *LOWER_COMPONENT* will be tagged as *LOWER*, while other irrelevant sons of N will be tagged as *ISOLATED_LOWER* (see Fig. 3).

Notice that some nodes are not labelled at all, and that no extra space is needed to store these four labels (\emptyset , *LOWER*, *LOWER_COMPONENT*, *ISOLATED_LOWER*), since the lower $rplanes$ bits of the coefficient are not coded, which allows to encode them as codewords in the range $[0, 3]$ as long as $rplanes \geq 2$.

In the second stage, which is our main contribution, the coefficients are sequentially scanned in LODs starting from the top-most one and moving downwards. Coefficients are bit-plane encoded with the help of two lists, *LIC* (List of Insignificant Coefficients) and *LSC* (List of Significant Coefficients), used to keep track of their significance with respect to the current bit-plane. In each bit-plane pass n , all the coefficients in the *LIC* are tested: those having the n^{th} most significant bit set are moved to the *LSC*. Then, the coefficients in the *LSC* are processed sequentially and the n^{th} bit of each of them is output. The last step before moving to the next LOD consists in encoding the labels of the remaining coefficients, which are still kept in the *LIC*. These symbols do not contribute to lower the reconstruction error as they only provide information for the next LODs, and could even be skipped if the decoder will not decode further LODs.

3.2 Scalability

The hierarchical traversal of the coefficients, scanned in LODs, naturally produces a spatially scalable bit-stream. This way, the decoder first receives all the coefficients corresponding to a LOD and, only when it has finished reading them, it proceeds (if it has enough resources) with those from the next. Besides, with the introduction of bit-plane encoding, bits from each LOD are ordered in such a way that the first to arrive are the ones that contribute more to lower the reconstruction error, while bits from negligible coefficients arrive last. Fig. 4 shows renderings of the bunny model at different stages of the decoding process. First, LOD 1 is progressively reconstructed (second row). Once all coefficients of LOD 1 have been decoded, the mesh is subdivided and details in LOD 2 are processed (third row). LOD 3 (fourth row) and forthcoming levels are sequentially decoded until the whole bit-stream is read.

Fig. 5 shows the rate distortion curves for both the original LTW coder and our technique. The leap between LODs is clearly reflected in the curve discontinuities, which are the result of applying a subdivision scheme with a smoothing stage, such as

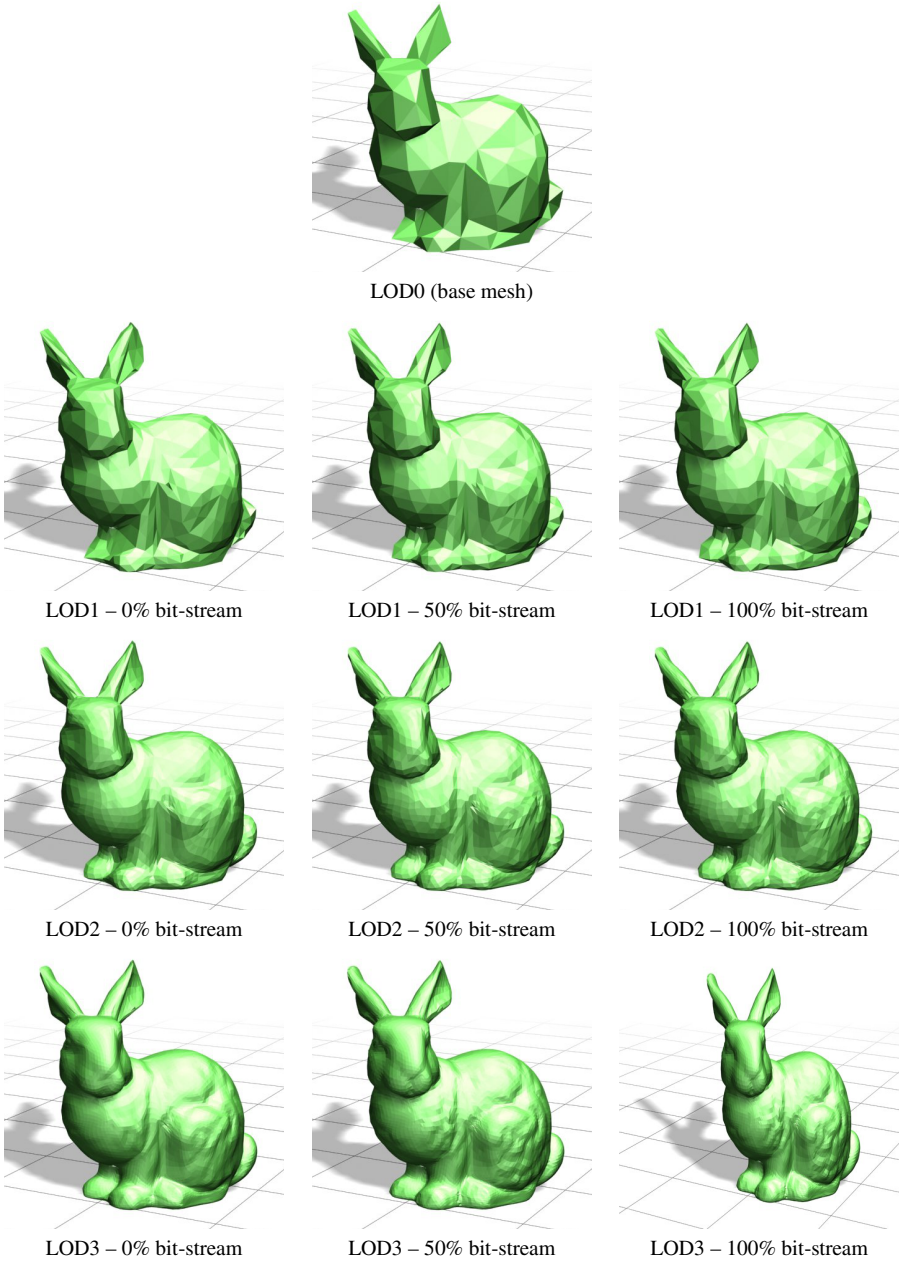


Fig. 4. Progressive reconstruction of the bunny model from a PLTW bit-stream

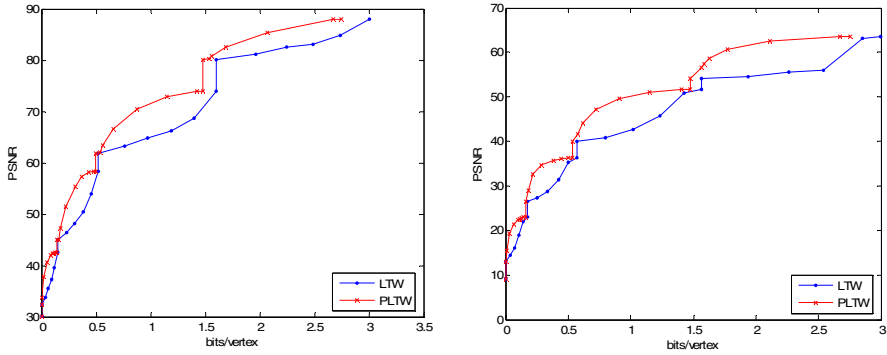


Fig. 5. Scalability of the original LTW algorithm and our PLTW technique for the venus (left) and bunny (right) models

Butterfly or Loop. This means that just by moving from a given LOD n to LOD $n + 1$, the reconstruction error is reduced (this effect will not occur for Midpoint subdivision, which only consists of an upsampling stage). Because of the sequential scanning of the coefficients in the LTW algorithm, the distortion curves between levels are arbitrarily shaped, although monotonically increasing, as they depend on the also arbitrary distribution of coefficients. In contrast, the curves for the PLTW coder demonstrate how bits with larger contribution are sent first whereas unimportant ones are output at the end. Note that the whole curve is not globally smooth, as in the SPIHT algorithm (see Fig. 1), because the bit-plane encoding is only performed on a per-level basis, while the SPIHT algorithm considers coefficients from all levels at the same time.

3.3 Entropy Coding

We have also analysed the gain obtained by entropy coding the PLTW bit-stream. Table 2 shows there is around a 46% benefit (42% in the worst case, for all the models we tested) when adding an arithmetic coder. Therefore, the PLTW algorithm is not as efficient as the SPIHT coder at exploiting relations between coefficients and the use of an entropy coder is almost required. However, the number of adaptive models needed in the arithmetic coder is still low, reducing the complexity and requirements of both the encoder and the decoder.

Table 2. Arithmetic coding of the PLTW bit-stream

model	venus	bunny	horse	dinosaur
no AC	1681701	825543	879696	1945907
AC	970636	441077	454124	1006442

4 PLTW vs. SPIHT

We now compare the PLTW coder with other SPIHT-based coders. Fig. 6 (top) shows the rate distortion curves for the PLTW coder and the arithmetic coded version of the

SPIHT algorithm (following the first of the approaches described in Subsection 2.2) for different LODs. LODs 1 and 2 are quickly reconstructed because their details are those that contribute more to lower the error. It seems clever to cut the stream or stop decoding after some point (e.g., 0,75 b/v for LOD 1 or 1,5 b/v for LOD 2) if only interested in coarser LODs, as the bits to come will hardly increase the PSNR. However, even in those cases, the decoder needs to build the whole detail tree (including finer LODs than those of interest) to be able to follow all the branches of the SPIHT algorithm. On the contrary, due to the spatial scalability of the PLTW coder, the decoder is able to stop decoding exactly at the desired LOD without allocating extra resources for further LODs — and even with a lower reconstruction error!

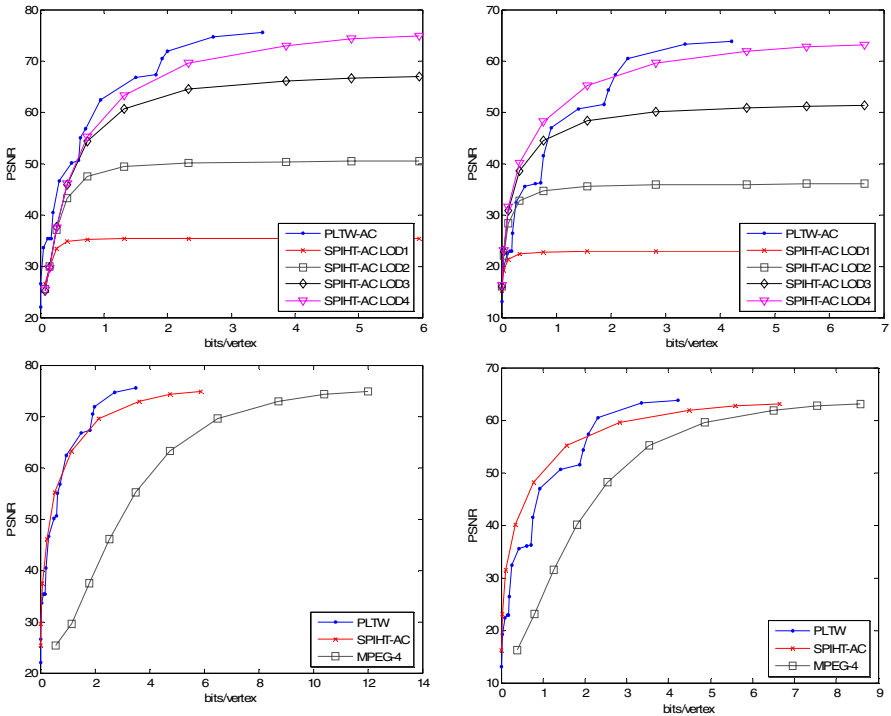


Fig. 6. PLTW vs. SPIHT: Per-LOD comparison (top) and overall compression (bottom) for the Max-Planck (left) and bunny (right) models

Fig. 6 (bottom) compares the overall compression performance of our technique with the SPIHT-AC algorithm and the WSS tool of MPEG-4's AFX (Animation Framework eXtension) [4], which also uses the SPIHT, but without AC. Except at very low rates, where the PLTW is still reconstructing upper LODs and does not benefit from the smoothing effect of subdivision (while its competitors do), our technique always results in higher PSNRs for the same bitrate. It is also noticeable how none of the SPIHT-based coders is able to reach the same PSNR as the PLTW coder even employing 160% (SPIHT-AC) or 330% (MPEG-4) of the bits used by PLTW for

the same quantization set of values. The poor results of the MPEG-4 coder are due to the overhead introduced to support view-dependent transmission of coefficient trees.

5 Conclusion

After an in-depth analysis of current state-of-the-art techniques for the hierarchical coding of 3D models with WSSs, we have proposed ours, the PLTW (Progressive Lower Tree Wavelet) coder, which achieves higher compression ratios and features both spatial and SNR scalability. Thanks to this, the same compact and scalable bit-stream can be used in heterogeneous networks with a wide range of terminals, both in terms of processing power and bandwidth. Our bit-stream does not impose on the less computationally powerful terminals the need of building detail trees as deep as required by the maximum LOD to follow the logical branching of the SPIHT algorithm, because the wavelet coefficients are sent on a per-LOD basis, thus achieving a “local” SNR scalability within a “global” spatial scalability.

Entropy coding has also been studied as a post-processing solution, and proven to provide an important benefit in compression efficiency, especially in the PLTW algorithm, compared to the additional complexity added to the decoder.

In particular, we have shown how our PLTW technique provides a substantial advantage over MPEG-4’s WSS tool, which is the only similar one in a current ISO standard. We therefore believe that our technique should be considered for future versions of MPEG-4’s AFX toolset, especially if one of its targets will be 3D Graphics applications for mobile terminals, in which the view-dependence offered by its current WSS tool would be much less important than the computational complexity and bandwidth savings offered by PLTW.

Finally, we would like to stress that, although we have developed our technique within the field of 3D model coding, it should yield the same excellent results when applied to image coding, or to any other problem addressable with the SPIHT algorithm.

Acknowledgments

This work has been partially supported by the 6th Framework Programme of the European Commission, within its research project FP6-IST-1-507926: On-Line GAMing (OLGA). Datasets are courtesy of Cyberware, Max-Planck Institut für Informatik and Stanford Computer Graphics Laboratory.

References

1. N. Aspert, D. Santa Cruz, and T. Ebrahimi, “MESH: Measuring Errors between Surfaces using the Hausdorff Distance,” *Proceedings of the IEEE International Conference on Multimedia and Expo*, 705–708, August 2002.
2. A. Khodakovsky, P. Schröder, and W. Sweldens, “Progressive Geometry Compression,” *Proceedings of the ACM SIGGRAPH Conference*, 271–278, July 2000.

3. F. Morán and N. García, "Comparison of Wavelet-Based Three-Dimensional Model Coding Techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, **14**-7, 937–949, July 2004.
4. MPEG: "ISO/IEC 14496 (MPEG-4) Part 16: Animation Framework eXtension (AFX)", ISO/IEC, April 2003 (International Standard status).
5. J. Oliver and M. P. Malumbres, "Fast and Efficient Spatial Scalable Image Compression using Wavelet Lower Trees," *Proceedings of the IEEE Data Compression Conference*, 133–142, March 2003.
6. A. Said and A. Pearlman, "A New, Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Transactions on Circuits and Systems for Video Technology*, **6**-3, 243–250, June 1996.
7. D. Taubman, "High Performance Scalable Image Compression with EBCOT," *IEEE Transactions on Image Processing*, **9**-7, 1158–1170, July 2000.
8. I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, **30**-6, 520–540, June 1987.