What is the best depth-map compression for Depth Image Based Rendering?

Jens Ogniewski and Per-Erik Forssén

Department of Electrical Engineering, Linköping University, 581 83 Linköping, Sweden
{jenso,perfo}@isy.liu.se
http://www.isy.liu.se/

Abstract. Many of the latest smart phones and tablets come with integrated depth sensors, that make depth-maps freely available, thus enabling new forms of applications like rendering from different view points. However, efficient compression exploiting the characteristics of depth-maps as well as the requirements of these new applications is still an open issue. In this paper, we evaluate different depth-map compression algorithms, with a focus on tree-based methods and view projection as application.
The contributions of this paper are the following: 1. extensions of existing geometric compression trees, 2. a comparison of a number of different trees, 3. a comparison of them to a state-of-the-art video coder, 4. an evaluation using ground-truth data that considers both depth-maps and predicted frames with arbitrary camera translation and rotation.
Despite our best efforts, and contrary to earlier results, current video depth-map compression outperforms tree-based methods in most cases. The reason for this is likely that previous evaluations focused on low-quality, low-resolution depth maps, while high-resolution depth (as needed in the DIBR setting) has been ignored up until now. We also demonstrate that PSNR on depth-maps is not always a good measure of their utility.

Keywords: Depth map compression, Quadtree, Triangle Tree, 3DVC, View Projection

1 Introduction

In recent years depth-map compression has become an important research issue, with the advent of commonly available depth sensors, like the Microsoft Kinect, which even are included in some modern mobile phones. The data generated by these sensors becomes more and more accurate, thus enabling new applications, like efficient algorithms to create astonishingly accurate geometric models from this type of data, e.g. [1]. Alas, video cameras with included depth sensors (so called RGB+D sensors) generate tremendous amounts of data, which only gets worse due to ever increasing image resolutions. Thus, efficient compression algorithms are increasingly important.
While compression of RGB images and videos are well understood, depth images have

different statistics and applications, and how to compress them is still an open research topic. Some of the first depth-map compression techniques used mesh based approaches, for example [2], which suggested a hierarchical decomposition of the depth-map to generate the mesh. Current methods include e.g. [3] which adapts a current video coder for depth-map compression, or geometric primitives [4], which uses a block based plane model. In this paper however we concentrate mainly on treebased methods.
Tree-based approaches have earlier be proven to be beneficial for depth-map compression. They allow alignment of the borders of the nodes with the edges of the depth-map, thus creating sharp edges in the compressed depth-map, a property that we try to enforce further with adaptive trees (see section 3). In contrast, most texture encoding techniques cause fuzzy edges in depth maps, and these will cause stray points when rendering from a different view. Also, tree-based methods usually encode plane equations and can thus recreate slanted areas with higher fidelity, less effected by quantization step effects typically found in texture encoding. Furthermore, tree-based methods describe the depth-map by a number of geometric primitives, which can be rendered very quickly, compared to point wise projection as is necessary when using image based methods to compress depth-maps. For all these reasons, trees are a natural choice for depth-map compression for *Depth based image rendering* (DBIR). Finally, tree-based approaches have previously shown superior compression performance, albeit only compared to still-coder or (now outdated) video compression schemes in low quality scenarios. However, high quality scenarios, which are more interesting for projection purposes, have not been examined further. Nor have comparisons been done with a current state-of-the-art video encoder, neither have the different tree-based approaches been compared to each other. Here, we compare quad trees[5][6][7][8], triangle trees[9][10][11], as well as own enhancements aimed at improving coding efficiency. We also compare with the 3DVC module [14] from the latest video standard (HEVC[17]). All this is done in a high-quality scenario, which is a requirement for good performance in DIBR.
There are different protocols for evaluation of depth-map compression quality. [12] and [13] suggest a specific distortion metric for depth-maps, based on the quality of warped views rather than the PSNR of the encoded depth-map itself. While the former considers a stereo scenario, the latter includes a scenario with a number of parallel camera positions, translated along the x axis. We find the basic idea promising, but consider a more flexible framework that allows for arbitrary camera translation and rotation. [18] examines the influence of different warping techniques on the visual quality. Here, we present the PSNR values of the compressed depth-maps, as well as PSNR and Multi-Scale SSIM [16] results of projections using the compressed depth-maps, compared to projections using the ground-truth values. We use our own, state-of-the-art projection algorithm [15] for that. For our evaluation, we use the depth extension of the Sintel datasets [19] (see also figure 1), which provide ground-truth for RGB, depth and camera poses. Thus, we are certain that any errors and artifacts are introduced by the algorithms themselves rather than noise or inaccurate input data. While the results for the different projections are interesting by themselves, it should be pointed out they could easily be extended to similar applications (like e.g. construction of models from depth data).
The rest of the paper is organized as follows: Section 2 describes the Sintel dataset as well as some small modifications we did to increase the accuracy of our evaluation.

sleeping2 **alley2** **temple2** **bamboo1** **mountain1**

Fig. 1. The first frames of the Sintel sequences used in this evaluation.

Section 3 describes the different tree-based methods, section 4 the used encoding methods. In section 5 we present a short introduction to our projection algorithm (for more detail the reader is referred to our earlier paper [15]). Section 6 presents the evaluation and the results, while section 7 closes with some concluding remarks.

2 The Sintel Dataset

For the evaluation, we use the 2015 depth extension of the Sintel datasets [19]. This provides ground-truth data for both depth and camera poses. This is necessary to ensure that all errors or artifacts are introduced by the algorithms themselves rather than by noisy or erroneous input data. The Sintel data set provides two different texture streams for each sequence: *clean* without any after-effects (like lightning and blur) as well as *final* with the after-effects included. Due to the very nature of these effects, they remove fine detail from the sequences, and thus the differences between the different projections (using ground-truth depth or any of the compressed depth maps) are more pronounced in the clean sequences. Therefore we only used the clean sequences in our evaluation.
The sequences we chose were *sleeping2*, *alley2*, *temple2*, *bamboo1* and *mountain1* (see figure 1), since they contain only low to moderate amounts of moving objects (which are currently not handled by our projection algorithm), but moderate to high camera movement/rotation. Thus, they represent cases where the inaccuracies introduced by depth-map coding are noticeable.
To adapt the sequences to our evaluation, and further increase the accuracy of the results, we did the following changes to the sequences:

1. We removed the last 4 rows of all images and depth-data, to make the height divisible by 16. This is done to increase the efficiency of the reference video coder, which would otherwise have to resort to use very small blocks, which would lead to an increased bitrate.
2. The depth map of the first frame of the sleeping 2 sequence contains some pixels with enormously high values. These are probably introduced by pixels which were never written to during the rendering of the scene. Since these pixels would dominate both the encoding and the evaluation, we clamp them to the second highest value found in this depth-map.

3 Tree-based Methods

The trees are divided into inner nodes and leaves, where only the inner nodes have children (which might either be leaves or inner nodes, too), while only the leaves contain

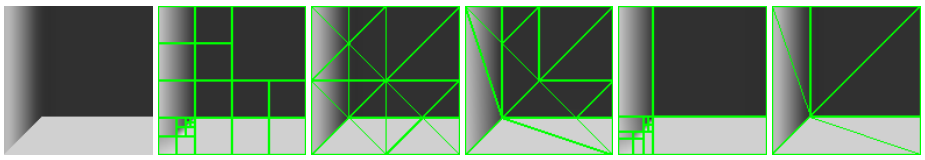

Fig. 2. Example for tree-based encoding, from left to right:
Input, quadtree, isosceles trees, triangle trees, adaptive quadtree, adaptive triangle trees

information describing the actual depth values, normally in the form of one plane equation per leaf. The geometric primitives which are typically chosen for tree-based coding are triangles and quads.
Triangles are typically split in one of their sides, thus each inner node has 2 children (and therefore triangular trees are also called binary trees). In the case of quads, they are split in each of their four sides, thus leading to four children. Therefore, given the same number of leaves, triangle trees will have more inner nodes and thus need more bits to encode the actual tree structure. On the other hand, quads are only able to model vertical and horizontal edges correctly; for depth maps containing edges with arbitrary direction a high number of quads is needed to describe them with adequate accuracy. Triangles on the other hand can also model slanted edges correctly (if the angle coincides with the angle found in the triangles). They thus offer slightly more flexibility.
Normally only triangles are used that are isosceles and right triangles, and they are always split in the middle of their long side, thus creating two children that are isosceles and right triangles as well. This has however two disadvantages:

1. This requires that the depth-map can be divided into a few, comparably big triangles that are both isosceles and right triangles. If the biggest possible triangles are too small, the reached compression will be quite small as well since it is not possible to merge neighboring triangles that can be approximated by the same (or nearly the same) plane equation.
2. We can only approximate edges in the depth-map that are horizontal, vertical or diagonal. Although this is better than in the case of quad trees, we would like a more flexible method.

Thus, we introduce here the notion of *adaptive trees*. In an adaptive quad tree, both the split-positions in x and in y direction can be selected freely, i.e. they do not need to split the sides exactly in half as is normally done in quad trees. It would of course be possible to split all four sides of the quad freely, however this would mean that we have to encode double the amount of data, which therefore does not seem to be beneficial.
For triangle trees, we allow them to be split in any of their sides, as well as an arbitrary split position. Thus, we end up with 5 different trees:

1. *Quadtrees*, which are always split in the middle of their sides.
2. *Adaptive quadtrees*, where the splits are on an arbitrary position of each side. However, the split position of the two horizontal sides has to be the same, as well as the split position of the two vertical sides has to be the same.

3. *Isoscele Trees*, which are triangle trees that are always split in the middle of their longest side.
4. *Triangle Trees*, which are triangle trees that are split in the middle of one of their sides.
5. *Adaptive Triangle Trees*, which are triangle trees split in an arbitrary position in any of their sides.

Examples for the different trees can be found in figure 2.
In most tree-based methods the image is divided into a number of trees first. However, here we omit this and encode the whole image directly in one tree in case of the quadtrees, and 4 in case of the triangle trees. This enables a more flexible division of the depth map, and thus a smaller bitrate. Using only one (resp. four) tree(s) means that we might have to transmit a couple of more bits to describe additional splits. On the other hand, the data needed to describe these splits is very small, compared to the data used for the plane equation. Thus, depending on the input data we can actually save data with our approach. The standard approach might lead to a situation where several neighboring trees contain only one leaf with the same plane equation (or at least nearly the same). Using only one tree, these could be merged to one leaf instead, thus avoiding sending the same plane equation several times and therefore saving data rate, even if using a prediction scheme for the encoding of the plane equations.
For the actual modeling of the depth-values, we use one plane equation per leaf, as is usually done. We tried different ways to express this plane equation, and finally settled for a differential form (i.e. describing the plane by the depth in the middle point as well as how the depth changes in both x and y directions), since no other form offered a higher accuracy or encoding gains, but this form simplifies the computations.

4 Depth-map Encoding

The search for an efficient encoding is done using the following algorithm:
We calculate all possible splits available in the tree. That even includes evaluating splitting each leaf in each of its sides (in case of the triangle trees) and each split position (in case of the adaptive trees). We then choose the split that leads to the highest increase in PSNR. For efficiency, we save the calculated PSNR improvements, to avoid recalculating them in later steps. This step is repeated until a given bitrate is reached.
In case of the triangle and isoscele trees, we first have to divide the image in a number of triangles. We choose 4, with the exact middle point as the common point of all 4 triangles. For the adaptive triangle tree, this point was allowed to be placed on an arbitrary position. We only split one of the trees in each step of our algorithm.
In some cases large leaves will not be split since this only lead to a comparably small change. To counter this, we introduced a *look-ahead* functionality: rather than calculating the PSNR after just one split, we calculate the PSNR after the leaf is split n number of times. For adaptive trees this could be done in two different ways:

1. Calculate each split for itself.
2. Choose one certain split, than calculate all possible consecutive splits which could result from this certain split, before evaluating the next split.

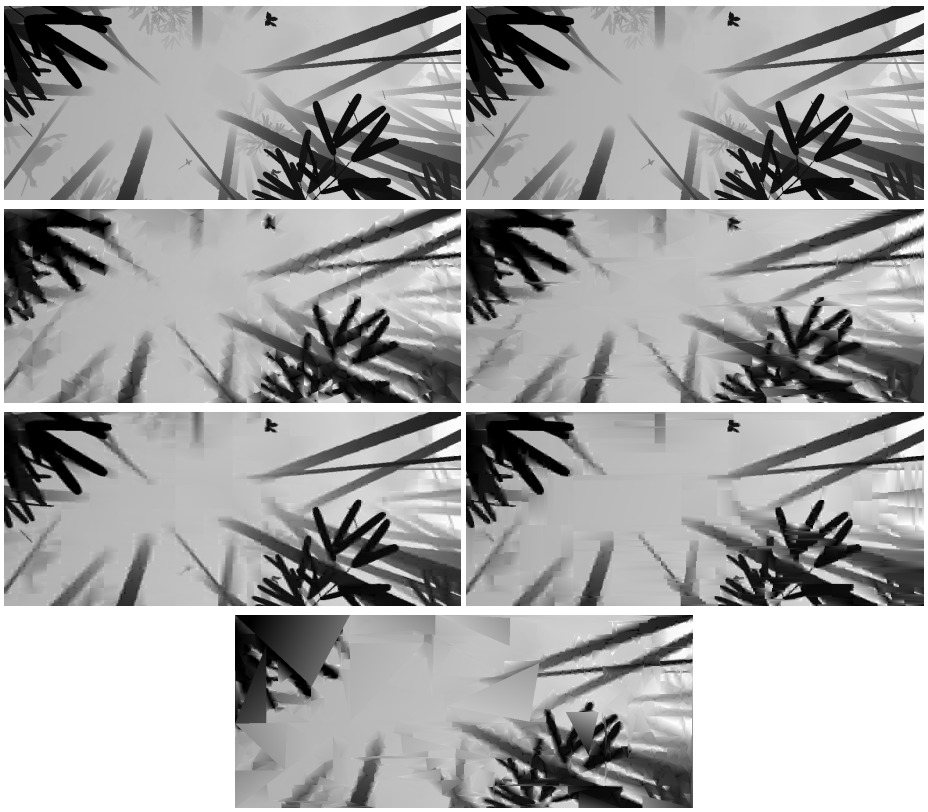

Fig. 3. Depth-map of the first image from the Sintel Bamboo 1 sequence, as used for the warping. a) (top left) input depth-map b) (top right) result using HEVC/3DVC compression, c) (middle-up, left) using isoscele trees, d) (middle-up, right) using triangle trees, e) (middle-down, left) using quad tree-compression, f) (middle-down, right) using triangle tree-compression and g] (bottom) using adaptive triangle tree-compression.

While it would of course be preferably to evaluate all possible combinations, this would increase the runtime exponentially and is therefore not feasible. On the other hand, this means that the look-ahead functionality improved the results for the adaptive trees only marginally. However, it improved the reached quality significantly for the non-adaptive trees.

After the tree division was done, the final tree was encoded using the following approach: Different streams were created for the split information and the plane equations. In the case of adaptive trees, an additional stream was added containing information where exactly the splits occur. This was done to separate the different data, which have different stochastic properties. In a real world application, the arithmetic encoder needs decoding trees to be able to decode these streams, which could either be included directly in the encoded-video-sequence, or general ones might be applied, based on the statistics derived from a high number of different input sequences. Here, we emulated

this step instead.
Most of the compression gain of modern coding schemes is reached by applying arithmetic coding, and thus for an accurate comparison, an arithmetic coder needs to be added to the compression of the trees. However, developing such an encoder is unfortunately time consuming and therefore out of scope for this project. Instead, we approximate the outcome of the arithmetic coding by calculating the minimum number of bits that are required to encode the sequences. The possible symbols that were not included in our actual tree were added with a very low probability for this calculation. This emulation gives quite accurate results since modern arithmetic coder can actually reach (or at least come very close to) this minimum. Also, since prediction did not improve our results, it is highly unlikely that context-sensitive encoding could improve on this either. Finally, if the trees do not perform better than other approaches using this theoretical optimal compression, they will never perform better, and thus there is no need to actually develop an arithmetic coder for them.
Firstly, the tree structure was encoded. For the isoscele and the quadtrees, this was done by using a 1 to describe an inner node, a 0 to describe a leaf. For the (adaptive) triangle trees another symbol was added which describes in which side it was split. It was assumed that each side has the same probability to be split.
Secondly, for the adaptive trees the split position were encoded, and for the adaptive triangle tree, we add the position of the common point of the 4 trees to this encoding stream.
We tried different value ranges for this, and settled to use the difference of the actual position to the middle of the side. For large nodes, it is more likely that they are split close to the middle than towards the edges. For small nodes on the other hand their sides will be very small and thus no split far away from the middle can occur. Thus, using the difference to the middle creates a distribution where values around zero are much more likely, and thus enables a more efficient encoding. We were even able to wittness this during our experiments.
Finally, the plane equations were encoded using 8 bits for each of the 3 values (depth in the center, depth change in x and depth change in y direction, both calculated at the edge of the segment the leaf is describing). We tried to use different schemes to predict these values from surrounding other leaves, which however did not lead to an improved bitrate. If two leaves have very similar plane equations, in all likelihood their parent node will not have been split in the first place. Also, if the plane equation of two neighboring leaves are very different, this might lead in an increase of the number of symbols needed to encode the tree (if the values range from -1 to 1, and both extremes are discovered in neighboring leaves, prediction might require increasing the symbol range to a range from -2 to 2).

5 View Projection

For the projection needed in our evaluation, we use our own, state-of-the-art projection algorithm, described in [15]. It consists of a flexible framework integrating a multitude of different methods, both state-of-the-art methods as well as own contributions,

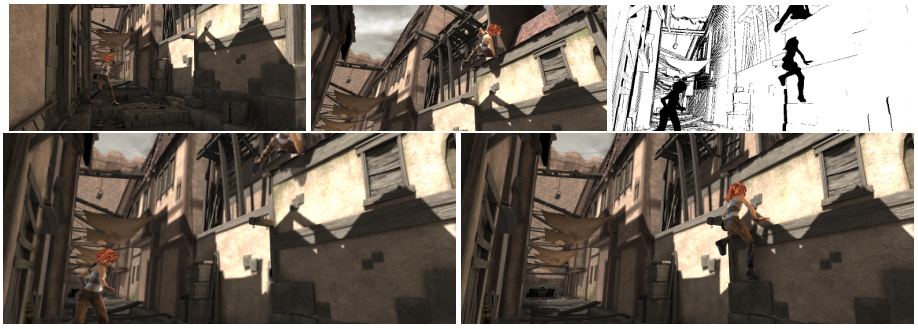

Fig. 4. Example of the projection process:
Top row: input images, image 1 (left), image 50 (middle) and mask-image (right)
Bottom row: combined projection using ground-truth depth (left), and ground-truth frame 34 (right)

which were mainly introduced to counter artifacts we discovered during this work. This framework was then finely tuned using an exhaustive search for the optimal parameters and methods. In the end, it is a forward warping projection scheme, which employs an internal upscale (by both 3 in width and height; Gaussian filtering is used for the down-scale), and agglomerative clustering to merge different candidate pixels and projections from different view points, among other things. Hole-filling is provided by a hierarchical approach using scale pyramids for a rough estimate and cross-bilateral filtering for refinement, which can be seen as an enhancement of *hierarchical hole-filling* (HFF) [20].

Table 1. Depth map quality of the different approaches measured in PSNR for the different sequences. An average value of the two frames of each sequence is shown.

method	Sequence				
	Alley	Bamboo	Mountain	Sleeping	Temple
HEVC	49.07	45.53	47.49	46.05	49.51
Quad tree	32.93	25.56	31.39	32.28	30.55
Isoscele tree	29.01	21.47	28.32	28.32	26.91
Triangle tree	29.60	21.39	28.71	29.25	27.29
Adpative triangle tree	29.17	19.06	27.27	26.69	27.09
Adaptive quad tree	32.7	23.86	31.96	31.38	31.63

6 Evaluation

We choose the depth-map of the first and the last frame from each of the test sequences to be encoded. Thus, the projection evaluation could be done by using a combined projection from these two frames to each other frame in the sequence.
Since the depth-maps are in a floating point format, they first need to be converted to an intermediate gray scale image to make it possible to encode them with HEVC/3DVC[14]. The lowest value in each depth-map was mapped to a luminance of -0.5 in the intermediate image, the highest value to 255.5 (rounding was performed always towards 0). This mapping was done linearly. During earlier experiments we also tested an inverse mapping, and found that it leads to similar or slightly worse results. We used the standard settings of the encoder to allow for easier comparison with our experiments. After encoding, the luminance of the resulting images was transferred back to the original floating point range.
Note that video encoder typically reach much higher compression even for still images than most still image coders like e.g. JPEG2000, since these normally do not take correlation between the different blocks into account, while video coders tend to predict blocks from other blocks contained in the same (or other) image(s).
We then encoded the original depth-maps with all different tree methods, aiming at a similar bit rate as reached by the HEVC/3DVC reference encoder. The resulting quality levels are given in table 1 (an average of the two encoded frames from each sequence is given), and example depth-maps are shown in figure 3. The average bitrates per frame were 23480 for the alley sequence, 99924 for bamboo, 17208 for mountain, 29960 for sleeping and 36264 for the temple sequence.
For the projection evaluation, we did a combined projection from the first and the last image of each sequence to each other image of the same sequence. To reach higher accuracy, we mask part of the projected images that block regions with holes caused by disocclusion and regions containing moving objects. For that, we created mask images beforehand, by projecting every point of the input frames to the target frame, and rounding the obtained x- and y-positions both up and down. To exclude moving objects from the masks, the depth of the projected point is compared to the depth of the ground-truth image for each pixel of this 2×2 region, and only the pixels for which this difference was below a predetermined threshold were then set in the mask. An example of the projection process is shown in 4. The resulting PSNR and MS SSIM values of the projected images compared to the groundtruth images are given in figure 5a) and figure 5b) respectively.
In terms of PSNR/Multiscale SSIM for the warped images, the depth maps encoded with the reference HEVC/3DVC performed best by far, with the exception of the mountain sequence. Simple quad-trees performed surprisingly well as well. On first glance, the PSNR values of the compressed depth-maps correspond well to the PSNR and MS SSIM results reached by the projected images. However, this is not the case for the mountain sequence, neither for the sleeping sequence if comparing quad- and adaptive quad-tree. This means that the PSNR of a depth-map is not in all cases a good indicator for the visual quality that can be reached by images that are results of image warping using its depth values, and is thus not reliable as a performance measure in this case.
The reason why the adaptive methods performed worse than their non-adaptive coun-

terparts lies in the encoding algorithm. While the look-ahead functionality works very well in the non-adaptive cases, it did not perform much better for the adaptive cases, sometimes even worse. The reason is that a split with a certain split position might be beneficial for the exact lookahead-level, but not for a level that is larger. Using an extreme large look-ahead level, or evaluating all possible combinations might solve the problem, but is however not feasible from a computational viewpoint.

7 Conclusion & Future Work

We have presented an evaluation of the most important tree-based compression methods, and introduced modifications aimed at increasing their quality and coding performance. However, the latest standard video encoder for depth-maps (HEVC/3DVC) outperformed the tree-based methods in all but one of the tested sequences in our high-quality evaluation setting.

All tree-based approaches presented in this paper could potentially be improved if a better encoder was found (especially the adaptive ones). As exhaustive search is out of the question due to computational constraints this would require designing a better, novel parameter search strategy. Even if such a strategy could be found it is highly questionable if the performance could be raised to the level of HEVC/3DVC. Thus, while it still might be preferable to use tree-based approaches in narrow, very specialized cases, such as the low-quality setting previously explored, HEVC/3DVC will outperform them at least in the high quality setting that is needed for accurate DIBR. It might however be interesting to consider other recent compression schemes, e.g. [21].

While high PSNR values of compressed depth-maps often coincides with a good projection performance, we have found cases where they do not. Thus, the PSNR of a depth-map is not a good predictor for the quality of projection using this depth-map. To further improve depth-map compression for DIBR purposes a better error metric should therefore be developed. One solution is of course to directly measure the projection performance. However, using one or even several high quality projections might not always be possible due to the increased computational complexity.

References

1. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D. Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon, A.: KinectFusion: Real-time dense surface mapping and tracking. Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality, pp. 127–136 (2011).
2. Sung-Yeol Kim, Yo-Sung Ho: Mesh-Based Depth Coding for 3D Video using Hierarchical Decomposition of Depth Maps. Proceedings of IEEE International Conference on Image Processing, pp. 117–120 (2007).
3. Jingjing Fu, Dan Miao, Weiren Yu, Shiqi Wang, Yan Lu, Shipeng Li: Kinect-Like Depth Data Compression. IEEE Transactions on Multimedia 15, pp. 1340–1352 (2013).
4. Merkle, P., Múller, K., Marpe, D., Wiegand, T.: Depth Intra Coding for 3D Video based on Geometric Primitives. IEEE Transactions on Circuits and Systems for Video Technology 99 (2015).

5. Bing-Bing Chai, Sethuraman, S., Sawhney, H.S, Hatrack, P.: Depth map compression for real-time view-based rendering. Pattern Recognition Letters 25, pp 755–766 (2004).
6. Morvan, Y., Farin, D., de With, P.H.N.: Multiview video coding using depth based 3D warping. Proceedings of IEEE International Conference on Image Processing 5, pp 105–108 (2007).
7. Colleu, T., Pateux, S., Morinc, L., Labit, C.: A polygon soup representation for multiview coding. Journal of Visual Communication and Image Representation 21, pp. 561–576 (2010).
8. Sandberg, D., Forssén, P.E., Ogniewski, J.: Model-Based Video Coding Using Colour and Depth Cameras. Proceedings of International Conference on Digital Image Computing Techniques and Applications, pp. 158–163 (2011).
9. Bing-Bing Chai, Sethuraman, S., Sawhney, H.S.: A depth map representation for real-time transmission and view-based rendering of a dynamic 3D scene. Proceedings of First International Symposium on 3D Data Processing Visualization and Transmission, pp. 107–114 (2002).
10. Sarkis, M., Waqar Zia, Diepold, K.: Fast Depth Map Compression and Meshing with Compressed Tritree. Lecture Notes in Computer Science Volume 5995, pp. 555–566 (2009).
11. Byung Tae Oh, Lee, J., Du-Sik Park: Binary tree decomposition depth coding for 3D video applications. Proceedings of IEEE International Conference on Multimedia and Expo, pp 1-6 (2011).
12. Byung Tae Oh, Lee, J., Du-Sik Park: Depth Map Coding Based on Synthesized View Distortion Function. IEEE Journal of Selected Topics in Signal Processing 5, pp. 1344–1352 (2011).
13. Li Wang, Lu Yu: Rate-distortion Optimization for Depth Map Coding with Distortion Estimation of Synthesized View. Proceedings of IEEE International Symposium on Circuits and Systems, pp 17–20 (2013).
14. Müller, K., Schwarz, H., Marpe, D., Bartnik, C., Bosse, S., Brust, H., Hinz, T., Lakshman, H., Merkle, P., Rhee, F.H., Tech, G., Winken, M., Wiegand, T.: 3D High-Efficiency Video Coding for Multi-View Video and Depth Data. IEEE Transactions on Image Processing 22, pp. 2266–2278 (2013).
15. Ogniewski, J., Forssén, P.E.: Pushing the Limits for View Prediction in Video Coding. 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (2017).
16. Zhou Wang, Simoncelli, E.P., Bovik, A.C.: Multi-scale Structural Similarity for Image Quality Assessment. 37th IEEE Asilomar Conference on Signals, Systems and Computers (2003).
17. Sullivan, G.J., Ohm, J.R., Woo-Jin Han, Wiegand, T.: Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology 22, pp 1649–1668 (2012).
18. Iyer, K.N., Maiti, K., Navathe, B., Kannan, H., Sharma, A.: Multiview video coding using depth based 3D warping. Proceedings of IEEE International Conference on Multimedia and Expo, pp. 1108–1113 (2010).
19. Butler, D.J., Wulff, J., Stanley, G.B., Black, M.J.: A naturalistic open source movie for optical flow evaluation. Proceedings of European Conference on Computer Vision, pp. 611–625 (2012).
20. Solh, M., Al Regib, G.: Hierarchical Hole-Filling(HHF): Depth image based rendering without depth map filtering for 3D-TV. IEEE International Workshop on Multimedia and Signal Processing (2010).
21. Yung-Hsuan Chao, Ortega, A., Wei Hu, Gene Cheung: Edge-adaptive depth map coding with lifting transform on graphs. Picture Coding Symposium (2015).

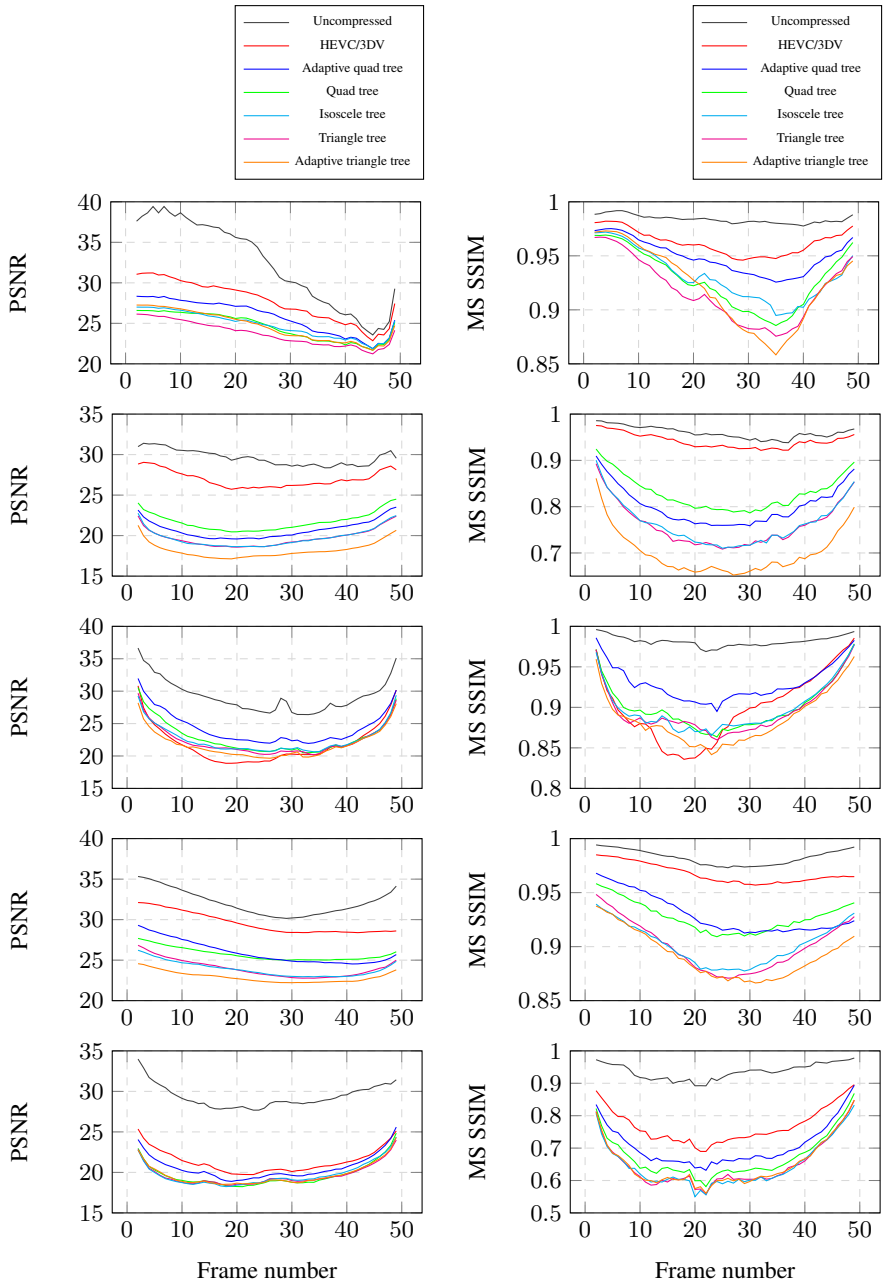

Fig. 5. Measured PSNR (a), left) and MS SSIM (b), right) between the warped images and the original images of the sequences:
From top to bottom: alley, bamboo, mountain, sleeping and temple.
Note that the curves are ordered according to their performance in the legend, the curve with the highest MS SSIM values is mentioned first.