Wavelet-based hierarchical heightmap compression method

Michal Lašan
Supervised by: Martin Kahoun

Charles University in Prague
Faculty of Mathematics and Physics
Prague / Czech Republic

Abstract

In this paper, we present a wavelet-based method for lossy compression of heightmap terrain data. It keeps the reconstructed data within a certain absolute per-sample error bound adjustable by the user. This method accepts blocks of float samples of dimensions $2^n x 2^n$ at the input, for which it can perform progressive mip-maps decompression. The compression of a 256x256 block takes about 30 ms and the decompression about 1 ms. Thanks to these attributes, the method can be used in a real-time planet renderer. It is able to achieve the compression ratio of 37:1 on the whole Earth 90m/sample terrain dataset transformed and separated into square blocks, while respecting the maximum error of 5m.

Keywords: heightmap, lossy compression, wavelet, lifting, guaranteed maximum error bound, real-time planet rendering, mip-map, progressive decompression, transformation, quantization, filtering

1 Introduction

Real-time rendering of the whole Earth requires working with large terrain data, their storage and distribution. A multiresolution (LOD-ing[1]) approach is essential in order to reach reasonable frame rates. In literature, a survey paper summarizing the most common multiresolution rendering methods exists [7]. Some of them also contain data compression.

For example, in C-BDAM [5] and P-BDAM [2], the compression takes place in the refinement of a node of the LOD hierarchy. The values inside a child node are predicted from the parent node as accurately as possible. The differences between these predictions and the real values are called residuals. These residuals are then quantized and compressed by an entropy codec - thus, the compression is lossy. Both these methods use a slight modification of the wavelet lifting scheme, ensuring that the error of the reconstructed data is kept within a maximum error bound adjustable by the user [8].

Another paper [6] describes a method based on the same principle - the residuals required to refine a square node of the terrain hierarchy are compressed. The computation of the residuals is based on the JPEG2000 standard, which is again a wavelet scheme. However, this method does not support an arbitrary maximum error adjustable by the user and its rendering pipeline does not handle visual artifacts between adjacent nodes of different LODs.

In practice, many applications handle the real-time rendering well with LOD schemes tailored to their needs. In such cases, a compression method tied to a concrete LOD scheme (which is the case of the mentioned methods) is not feasible. This method handles only the compression, so it can be used as a plug & play component in an existing real-time renderer. Its only job is to compress a block of terrain height samples sized $2^n x 2^n$ and to provide fast progressive decompression of its mip-maps, while respecting the maximum error bound at every mip-map. The source code of the method is written modularly, so that any representation of the height samples can be compressed - doubles, floats or even arbitrary structures. It is inspired by C-BDAM - the compression method is extracted from the LOD scheme and simplified.

As a case study we have implemented this method as a plugin into an application, which transforms the heights on the planet surface into 256x256 blocks of 32-bit float samples in the unit of meters, which are then stored separately and during the run fetched into a quadtree-based LOD hierarchy. The mip-maps of the blocks are used while looking at them from a side.

This approach introduces heavy redundancy of the data - a block corresponding to a certain quadtree node contains simplified blocks of its children and all these blocks are stored separately. To the contrary, in C-BDAM only the residuals needed to reconstruct the children from the parent node are stored.[2] However, the reason why this approach is used is that the user can navigate to any area almost immediately - only the data needed for the scene has to be fetched, without having to reconstruct it by traversing from the root. Moreover, this approach enables the user to flexibly extend the terrain data by high-resolution insets. The mentioned redundancy of the data emphasizes the need for as efficient compression method as possi-

[1] LOD is the abbreviation of level of detail - degradation of quality of the displayed data with the growing distance in order to optimize the rendering

[2] The LOD structure in C-BDAM is not a quadtree, though

ble, doing only what is required - providing the mip-maps while respecting the maximum-error bound of the samples inside each one of them.

In Section 2, we briefly describe the basic theory of wavelets and link C-BDAM and this method to it, in Section 3, we briefly describe the basic outline of the method. In Section 4, we describe the details of the method. In Section 5, we compare the core algorithm of this method to the algorithm of C-BDAM. We present the results in Section 6 and then discuss them in Section 7.

2 The introduction to wavelets

Basically, two generations of wavelets exist - the first one which is based on computation with dilated and translated wavelet function [1] and the second one which is based on filter banks - high-pass and low-pass filtering [3]. It has been proven that these two approaches are computationally equivalent [4].

We will describe the second generation of discrete wavelet transform which is relevant for this work. The basic step of such transform is called lifting - a decomposition of the input signal samples the count of which is a power of two into two equally sized parts - low-pass (the low frequency information) and high-pass (the high frequency information). This step is then recursively applied to the produced low-pass part until its length is 1. The opposite of lifting is reconstruction - enriching low-pass samples by high-pass information to obtain twice as detailed set of samples. It is the exact inverse of lifting. This transform is widely used for compression of data which can be achieved by quantizing the produced high-pass parts (often called residuals).

The lifting is performed in the following way: the input samples x_k are split into the even ones: $x_{2k} = x_e$ and the odd ones: $x_{2k+1} = x_o$. Then two operators are introduced: the prediction operator P which is used to produce the final high-pass part d (residuals) from x_o and the update operator U which is used to produce the final low-pass part s from x_e.

The prediction-first methods firstly apply the prediction operator and then the update operator:

$$d = x_o - P(x_e)$$
$$s = x_e + U(d)$$

The reconstruction is then the exact inverse:

$$x_e = s - U(d)$$
$$x_o = d + P(x_e)$$

To the contrary, the update-first methods firstly apply the update operator and then the prediction operator:

$$s = x_e + U(x_o)$$
$$d = x_o - P(s)$$

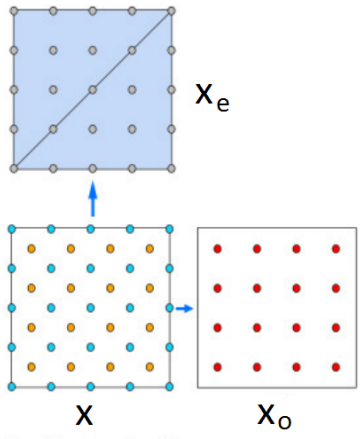

Figure 1: Lifting in C-BDAM - the samples x are separated into the even ones x_e which will become s - low-pass and the odd ones x_o which will become d - high-pass.
Source: C-BDAM [5] (edited)

The reconstruction then looks like this:

$$x_o = d + P(s)$$
$$x_e = s - U(x_o)$$

C-BDAM uses a slight variation of update-first lifting when constructing the coarser LOD s from the finer LOD x. It uses not only x_o, but the whole x as the input to the first update. Moreover, the computation of s cannot be written as the summation of the product of U and x_e anymore, because x_e is multiplied inside $U(x)$:

$$s = U(x)$$
$$d = x_o - P(s)$$

The corresponding reconstruction is:

$$x_o = d + P(s)$$
$$x_e = U^{-1}(x)$$

Besides, the samples x are regularly distributed in the plane, so the decomposition into x_e and x_o depends on the position of the samples, no longer on their indices (Fig. 1). However, the count of x_e is still half the count of x. Note that if the residuals d were simply quantized after lifting, each step of the reconstruction would make the maximum absolute deviation from the original values larger. To ensure the maximum-error bound at each level, the residuals computed during the lifting are corrected according to the actual values in another top-bottom pass which then turns out to be identical to the reconstruction (decompression).

The proposed method uses the same main lifting principle as C-BDAM (update-first and use the whole x as the input of U), but introduces several differences: the count of x_e and thus s is not half the count of x, but one fourth of

it, as each four neighboring pixels inside x are collapsed into one inside s (Fig. 2). In addition, the lifting is not complete, as no prediction and computation of residuals is performed there. These are let to be performed in the second top-bottom pass where also the maximum-error bound is ensured. Just like in C-BDAM, this pass is identical to the reconstruction of the data. The prediction operator is applied more times in the reconstruction which is explained in Section 4. The reasons for all these differences are explained in Sec. 5.

3 The outline of the method

Here is how the compression works. We perform two passes on the input heightmap. In the first bottom-top pass, we compute the target mip-maps - from the largest one to the smallest one. In the second top-bottom pass, we construct the compressed mip-maps from the smallest one to the largest one with respect to the target mip-maps in order to preserve the maximum-error bound. For each constructed mip-map, we store the residuals needed to produce it from the previous decompressed mip-map.

In more detail, given the input square block of float height samples $\boldsymbol{L_n}$ sized $2^n x 2^n$, n mip-maps $\boldsymbol{L_{o\ldots n-1}}$ are constructed from it. The dimension of $\boldsymbol{L_i}$ is half the dimension of $\boldsymbol{L_{i+1}}$ and $\boldsymbol{L_i}$ is computed from $\boldsymbol{L_{i+1}}$ by averaging of pixels - see the details in the next section.

In the second top-bottom pass, with $\boldsymbol{L_{o..n}}$ at hand, we compute $\boldsymbol{L^\bullet_{o..n}}$ - the final decompressed mip-maps. $\boldsymbol{L_i}$ and $\boldsymbol{L^\bullet_i}$ are of the same size and the maximum absolute deviation between their corresponding samples is not greater than $\boldsymbol{D}$ - the parameter set by the user. We will denote this by:

$$maxdev(\boldsymbol{L_i}, \boldsymbol{L^\bullet_i}) \leq \boldsymbol{D},$$

where

$$maxdev(A,B) = \arg\max_{x,y} |A[x][y] - B[x][y]|$$

We construct these mip-maps with the help of a uniform quantizer Q_D respecting this error bound:

$$maxdev(\boldsymbol{Q_D}(x), x) \leq \boldsymbol{D},$$

where x is an arbitrary float sample or block of samples.

$\boldsymbol{L^\bullet_o}$ contains just the quantized sole sample of $\boldsymbol{L_o}$

$$\boldsymbol{L^\bullet_o} = \boldsymbol{Q_D}(\boldsymbol{L_o}),$$

and $\boldsymbol{L^\bullet_{i+1}}$ is computed from $\boldsymbol{L^\bullet_i}$ by quantizing the differences between the target values $\boldsymbol{L_{i+1}}$ and the predictions from $\boldsymbol{L^\bullet_i}$:

$$\begin{aligned} \boldsymbol{E_{i+1}} &= \boldsymbol{L_{i+1}} - P(\boldsymbol{L^\bullet_i}) \\ \boldsymbol{E^\bullet_{i+1}} &= Q_D(\boldsymbol{E_{i+1}}) \\ \boldsymbol{L^\bullet_{i+1}} &= P(\boldsymbol{L^\bullet_i}) + \boldsymbol{E^\bullet_{i+1}} \end{aligned} \quad (1)$$

where P is a prediction operator, $\boldsymbol{E_{i+1}}$ are the differences and $\boldsymbol{E^\bullet_{i+1}}$ are the quantized differences. Note that thanks to the fact that the quantizer keeps the maximum absolute error within the bound $\boldsymbol{D}$ and the residuals $\boldsymbol{E_{i+1}}$ are computed with respect to the target mip-map $\boldsymbol{L_{i+1}}$, $maxdev(\boldsymbol{L^\bullet_{i+1}}, \boldsymbol{L_{i+1}}) \leq \boldsymbol{D}$, no matter what values are in $\boldsymbol{L^\bullet_i}$ and what is the form of the prediction operator P. All the quantized residuals $\boldsymbol{E^\bullet_{o..n}}$ are then compressed with the help of an entropy codec (Zlib) and saved ($\boldsymbol{E^\bullet_o} = \boldsymbol{L^\bullet_o}$), so the more accurate P is, the better the compression ratio is. The details of the prediction operator used in this method are described in the next section.

During the decompression, the quantized residuals are read and used to progressively reconstruct the mip-map levels $\boldsymbol{L^\bullet_{o..n}}$ (eq. 1).

4 Details of the method

In this section, we describe the details of the method, namely how the target mip-maps are constructed and what the prediction operator P looks like.

The down-sampling of the mip-maps can be performed by any form of averaging. As we saw in the previous chapter, the maximum absolute error does not depend on how the mip-maps look, as long as they contain valid values. However, the way the mip-maps are constructed affects the compression ratio. Moreover, various mip-map constructions produce different visual artifacts. In terms of the visual artifacts, the best way to down-sample a mip-map is to just average the four neighboring pixels [2n, 2n+1], [2n, 2n+1] at $\boldsymbol{L_{i+1}}$ into [n][n] at $\boldsymbol{L_i}$.

In the previous section, we made a simplification claiming that a decompressed mip-map $\boldsymbol{L^\bullet_{i+1}}$ is constructed from the previous $\boldsymbol{L^\bullet_i}$ in just one step (eq. 1). We did that in order to emphasize the fact, that $maxdev(\boldsymbol{L^\bullet_{i+1}}, \boldsymbol{L_{i+1}}) < \boldsymbol{D}$. In fact, three such steps happen. Nevertheless, the residuals are checked after each of these steps and all the predictions are made from the decompressed values, so the maximum error bound is still kept. So, when we construct the following decompressed mip-map, every pixel p from $\boldsymbol{L^\bullet_i}$ is substituted by four pixels in $\boldsymbol{L^\bullet_{i+1}}$ as shown in Fig. 2.

The first one of them, labeled a, is predicted directly from $\boldsymbol{L^\bullet_i}$ by a simple prediction operator $P_a(\boldsymbol{L^\bullet_i}) = p$. Following this, the residuals $\boldsymbol{E_a}$ and $\boldsymbol{E^\bullet_a}$ are computed according to the target value a_t in $\boldsymbol{L_{i+1}}$ and a is assigned the final value $a\bullet$ (eq. 2). It holds that $maxdev(a\bullet, a_t) \leq \boldsymbol{D}$.

$$\begin{aligned} \boldsymbol{E_a} &= a_t - p \\ \boldsymbol{E^\bullet_a} &= Q_D(\boldsymbol{E_a}) \\ a\bullet &= p + \boldsymbol{E^\bullet_a} \end{aligned} \quad (2)$$

The second one of them, labeled b, is predicted from the pixels $a\bullet$ in $\boldsymbol{L^\bullet_{i+1}}$ by the straight-oriented order 2

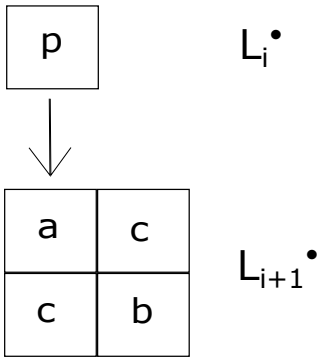

Figure 2: Substitution of a pixel p from $\boldsymbol{L_i^\bullet}$ by four children in $\boldsymbol{L_{i+1}^\bullet}$

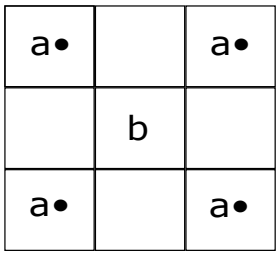

Figure 3: The prediction of b - $P_b(\boldsymbol{L_{i+1}^\bullet})$ - is the average of all the displayed $a\bullet$

Neville interpolating filter (fig. 3), just like the border values are predicted in C-BDAM. A similar computation of residuals $\boldsymbol{E_b}$ and $\boldsymbol{E_b^\bullet}$ then follows according to the target value b_t from $\boldsymbol{L_{i+1}}$ and b is assigned the final value $b\bullet$ (eq. 3).

$$\begin{aligned} \boldsymbol{E_b} &= b_t - P_b(\boldsymbol{L_{i+1}^\bullet}) \\ \boldsymbol{E_b^\bullet} &= Q_D(\boldsymbol{E_b}) \\ b\bullet &= P_b(\boldsymbol{L_{i+1}^\bullet}) + \boldsymbol{E_b^\bullet} \end{aligned} \tag{3}$$

The cases when the filter comes out of the image are handled by a specific mirror extension (fig. 4). Unlike C-BDAM, where the order 4 Neville filter is used for the interior values, in this method, the order 2 filter is used even for the interior values in order to increase the speed. As an additional optimization, the interpolation with the order 2 filter can be easily cached during horizontal traversal. Moreover, using the order 4 filter made the compression ratio slightly better - probably because it predicts hills and walleys more accurately, but made the quality of the reconstructed heightmap worse - it produced sharper artifacts on the borders of smooth gradient terrain blocks (Fig. 5) and near sharp terrain changes (Fig. 6).

The reason for these artifacts is that while the predictions are close enough to the real terrain (their quantized residuals are zeroes), the reconstructed values might be

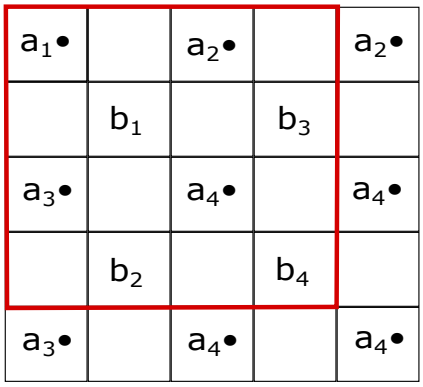

Figure 4: Handling of border cases in the computation of $P_b(\boldsymbol{L_{i+1}^\bullet})$ - the red line represents the border.

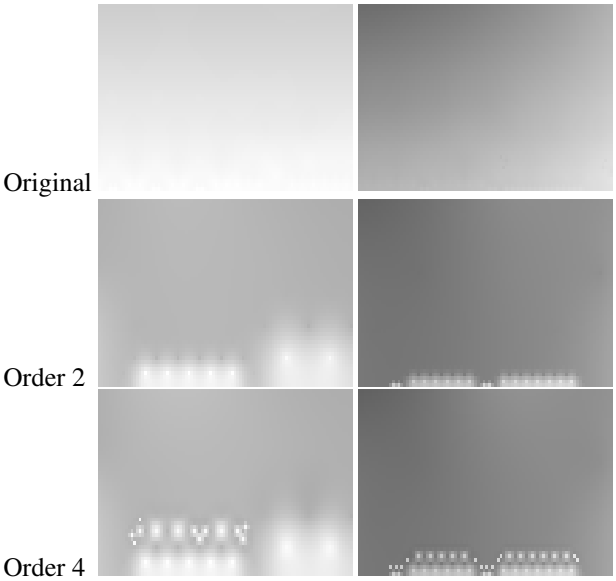

Figure 5: Two examples of different artifacts caused by order 2 and order 4 filters at the border of smooth gradient terrain - the first row shows the target heightmaps, the second row shows the same heightmaps compressed with the order 2 filter, the third row with the order 4 filter.

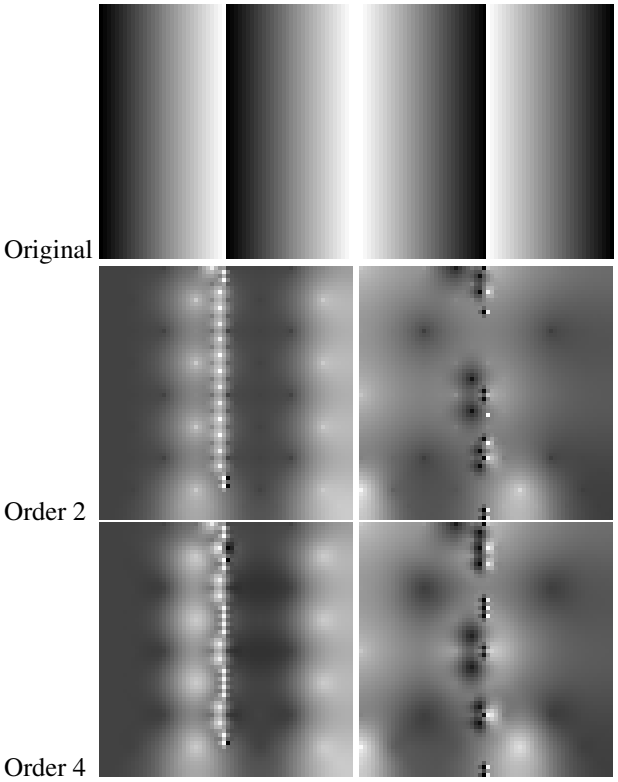

Figure 6: Two examples of different artifacts caused by order 2 and order 4 filters at a sharp terrain change - the first row shows the target heightmaps, the second row shows the same heightmaps compressed with the order 2 filter, the third row with the order 4 filter. The values in the original images range from 0 to 16 and the maximum deviation of the compression is 9.

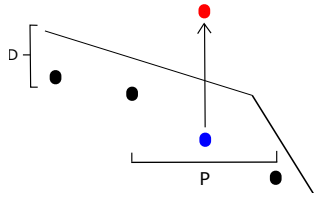

Figure 7: The illustration of how an artifact occurs - the black predictions are within the maximum-error bound D, so they are equal to the reconstructed values, but the blue one is not. Because a uniform quantizer is used, the blue prediction is shifted by $2D-1$ to the top, creating an artifact.

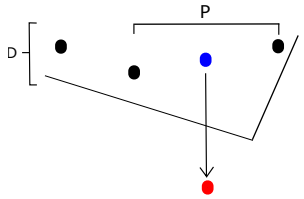

Figure 8: Another illustration of an artifact - the black predictions are within the maximum-error bound D, but the blue one is not. The blue prediction is shifted by $2D-1$ to the bottom, creating an artifact.

systematically above/under the terrain. But as soon as one prediction is a bit further from the terrain than those at the adjacent pixels, its residual is quantized to a non-zero value and the reconstructed value might flip to the other side of the terrain, producing a visual artifact. This often happens when smooth terrain is followed by a sharp change. The prediction operator might then predict different values near this change, as it reaches out to the area behind the change (Fig. 7, 8). This spike is then propagated to the next levels, but still within the maximum error bound. The mirroring at the borders produces such sharp changes, too, in a different, more complex way. However, some better form of mirroring might sort this out.

While the prediction of the order 2 filter is the average of the four neighboring values, the prediction of the order 4 filter tends to differ from the neighboring values more, so this filter has the tendency to produce more disturbing artifacts.

The remaining pixels labeled c are predicted from the pixels $a\bullet$ and $b\bullet$ in $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ by the diagonally-oriented order 2 Neville interpolating filter (fig. 9). The computation of residuals $\boldsymbol{E_c}$ and $\boldsymbol{E^{\bullet}_c}$ then follows according to the target value c_t from $\boldsymbol{L_{i+1}}$ and c is assigned the final value $c\bullet$ (eq. 3).

$$\boldsymbol{E_c} = c_t - P_c(\boldsymbol{L^{\bullet}_{i+1}})$$

$$\boldsymbol{E^{\bullet}_c} = Q_D(\boldsymbol{E_c})$$

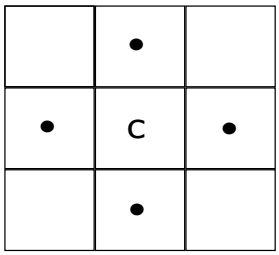

Figure 9: The prediction of c - $P_c(\boldsymbol{L}^{\bullet}_{i+1})$ - is the average of all the pixels marked with a dot - •.

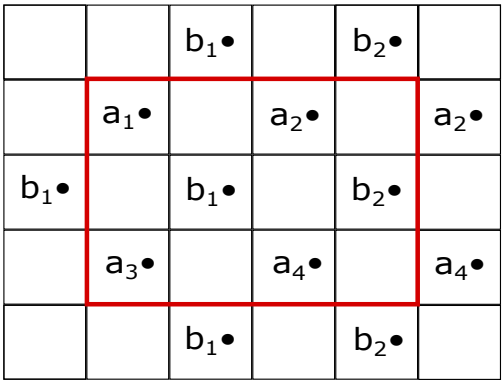

Figure 10: Handling of border cases in the computation of $P_c(\boldsymbol{L}^{\bullet}_{i+1})$ - the red line represents the border.

$$c\bullet = P_c(\boldsymbol{L}^{\bullet}_{i+1}) + \boldsymbol{E}^{\bullet}_{c} \tag{4}$$

The cases when the filter comes out of the image are handled by a specific mirror extension (fig. 10). For the same reasons as in the prediction of b pixels, the order 2 filter is used for the prediction of all c pixels - both interior and exterior ones. Similarly, the interpolation with such filter can be cached during diagonal traversal.

The residuals $\boldsymbol{E}^{\bullet}_{a}$, $\boldsymbol{E}^{\bullet}_{b}$ and $\boldsymbol{E}^{\bullet}_{c}$ are then encoded by an entropy codec and stored. The decompression is done in a similar manner with the only difference that the residuals are not computed anymore, but just decoded and read. So, we substitute every pixel from $\boldsymbol{L}^{\bullet}_{i}$ by four pixels in $\boldsymbol{L}^{\bullet}_{i+1}$, the value of which is computed in three passes of prediction followed by adding the read residual (the last lines of eq. 2, 3, 4).

5 Functional comparison to C-BDAM and wavelets

As we already mentioned, C-BDAM contains the whole rendering pipeline, whereas our method does not. However, it can be compared to C-BDAM in terms of how lifting is performed. As we already mentioned in the end of Section 2, C-BDAM omits a half of the samples while constructing a coarser LOD, whereas our method omits three fourths of the samples. This is spatially equivalent to two steps of lifting in C-BDAM (Fig. 1). The first step removes the pixels b and the second step removes the pixels c as seen in Fig. 2. Nevertheless, this equality is only spatial.

In our method, an analogy of the update operator of lifting is used to construct $\boldsymbol{L_i}$ from $\boldsymbol{L_{i+1}}$ (the averaging of four neighboring pixels - Sec. 4). However, the lifting is not complete in our method as it does not contain the prediction operator - no residuals are computed there yet. In C-BDAM, also a prediction operator is used in the lifting to produce intermediate residuals. However, using just these residuals would not guarantee any maximum error bound, so C-BDAM makes another top-bottom pass to correct the residuals against the real values of samples produced in the first bottom-top pass. To make this correction fit into the original wavelet framework, several intricate computations need to be performed, including division, which is quite a large performance hit.

Our vision was that once it is needed to perform an extra top-bottom pass to correct the residuals so that the maximum error bound is guaranteed, it is not neccessary to compute any temporary values of the residuals during the lifting steps (the construction of the LOD pyramid). This is why we perform just an analogy of the update (the averaging of pixels) in the update-first scheme and let the following top-bottom pass compute suitable values of residuals. This is obviosly a major deviation from the wavelet scheme. In the top-bottom pass, we just predict the values in the finer LOD as accurately as possible, but these predictions have no linkage to the previous bottom-top pass, as they have not been used there at all. Then we directly compute the residuals with respect to the original values computed in the first bottom-top pass at the corresponding levels.

All in all, it can be said that the way the residuals are computed in this method is an extreme simplification of the way they are computed in C-BDAM. This way of computation does not even conform to the second-generation wavelet scheme - the lifting is not complete and the reconstruction is not the inverse of lifting. We think that without the residuals quantization or the per-level correction of residuals, respecting the wavelet scheme makes sense, as it ensures computational equivalency with the first-generation wavelets. However, in case the residuals need to be corrected at each level, we think that conforming to a wavelet scheme makes no sense, because this correction immediatelly destroys the mentioned equivalence - once a residual is cropped in order to get the resulting value closer to the actual data, it cannot be said that any of the following reconstruction is the inverse to the lifting performed before. Moreover, thanks to the mentioned deviation of C-BDAM from the classical update-first second-generation wavelet discussed in Section 2, we question its computational equivalency with the first-generation wavelet even with no residuals quantization or cropping performed. Because of this, we think that the computa-

tions made in the second top-bottom pass can be optimized this way without any cost. Thus, this method would probably better be called wavelet-inspired than wavelet-based.

6 Results

This method has been applied in the real-time planet renderer mentioned in the introduction on height data of the whole Earth with the resolution of 90m (SRTM). Due to the redundancy of data in the applied LOD hierarchy, the size of the original data was 260GB. With the maximum error bound set to 5m, the size of the compressed data is 7GB, which yields the compression ratio of 37:1.

For a comparison, C-BDAM reached the compression ratio of 64:1 on the same dataset, but with the maximum error bound set to 16m. Thanks to the fact that the LOD hierarchy of C-BDAM contains no redundancy, the size of the original data was just 29GB and the size of the compressed data just 870MB. Under such circumstances, only the comparison in terms of the compression ratio is relevant.

In Fig. 11, you can see a part of a heightmap compressed by this method, together with the differences from the original.

7 Conclusions

In this paper, we described a heightmap compression method designed to be a plugin into an existing real-time planet renderer with its own rendering pipeline. The method proved to be convenient for the purpose, providing fast decompression (only about 1ms per block of data). Its compression ratio is comparable to C-BDAM, which is the method with the best compression ratio among the methods for the terrain compression, which guarantee a maximum error bound adjustable by the user.

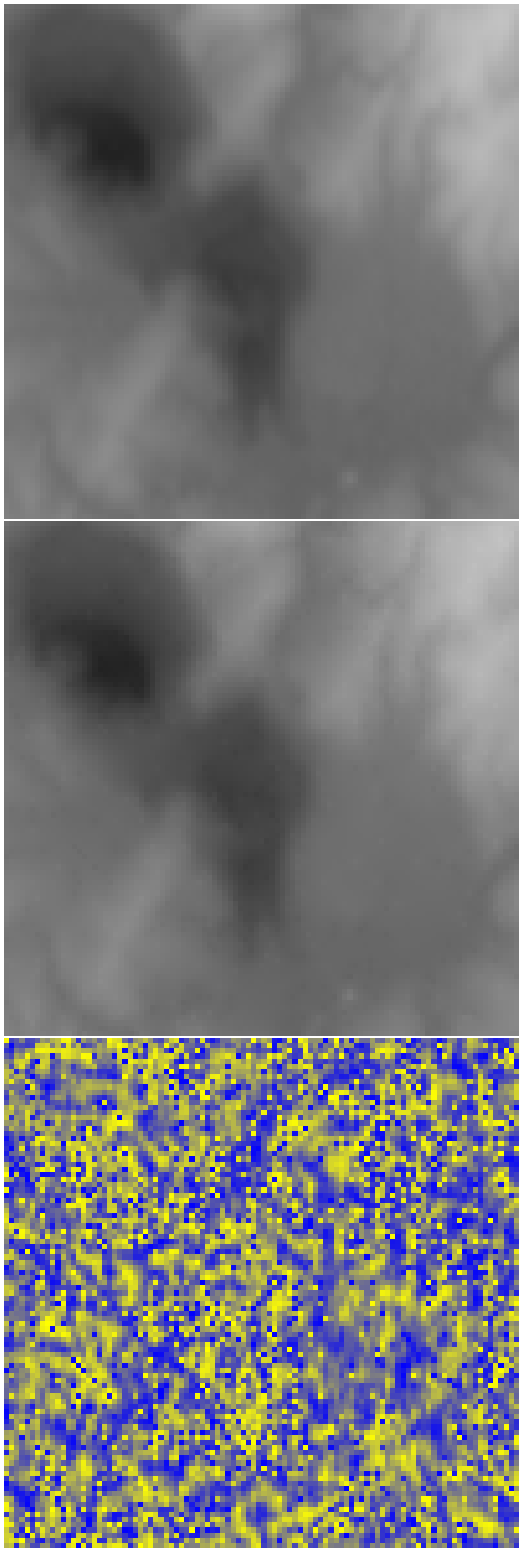

Figure 11: From the top to the bottom - the original terrain, the same terrain compressed with the maximum deviation of 5m, the difference between these two. The brighter the color, the greater the value. In the difference image, the yellow color means 4.5m, whereas the blue color means -4.5m.

References

[1] P. M. Bentley and J. T. E. McDonnell. Wavelet transforms: an introduction. *Electronics Communication Engineering Journal*, 6(4):175–186, Aug 1994.

[2] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Planet–sized batched dynamic adaptive meshes (p-bdam). In *Proceedings IEEE Visualization*, pages 147–155, Conference held in Seattle, WA, USA, October 2003. IEEE Computer Society Press.

[3] Roger L. Claypoole, Geoffrey M. Davis, Wim Sweldens, and Richard G. Baraniuk. Nonlinear wavelet transforms for image coding via lifting. *IEEE Trans. Image Processing*, 12:1449–1459, 2003.

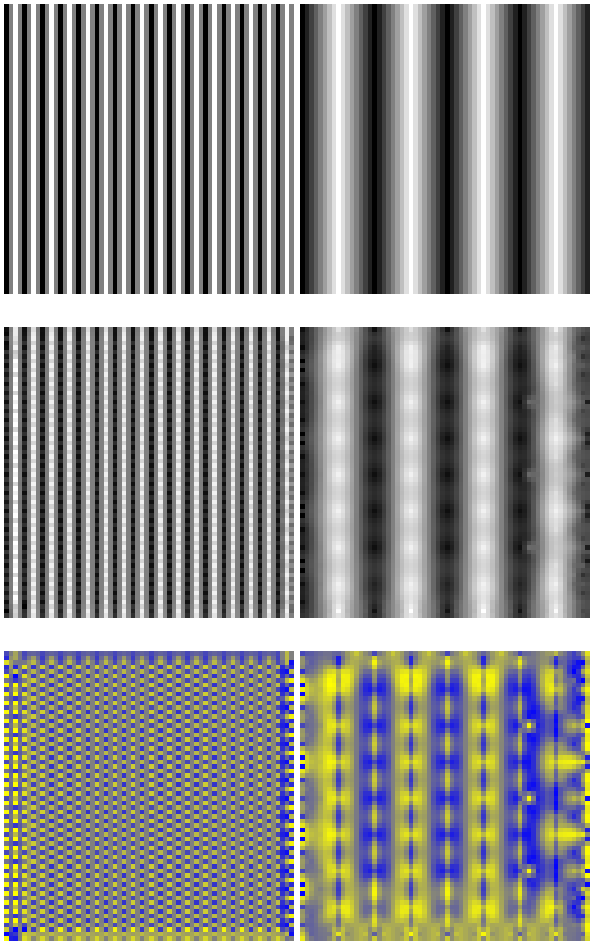

Figure 12: Two synthetic test images of size 64x64, each one containing spiky terrain with the heights ranging from -16 to 16. On the left, the longitude of spikes is 4, on the right, it is 16. From the top to the bottom - the original, compressed with the maximum deviation of 5, the difference between these two. The brighter the color, the greater the value. In the difference image, the yellow color means 4.5, whereas the blue color means -4.5.

[4] Ingrid Daubechies and Wim Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl*, 4:247–269, 1998.

[5] Enrico Gobbetti, Fabio Marton, Paolo Cignoni, Marco Di Benedetto, and Fabio Ganovelli. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum*, 25(3):333–342, September 2006. Proc. Eurographics 2006.

[6] Ricardo Olanda, Mariano Perez, Juan Manuel Orduna, and Silvia Rueda. Terrain data compression using wavelet-tiled pyramids for online 3d terrain visualization. *Int. J. Geogr. Inf. Sci.*, 28(2):407–425, February 2014.

[7] Renato Pajarola and Enrico Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605, 2007.

[8] Sehoon Yea and W.A. Pearlman. A wavelet-based two-stage near-lossless coder. *Image Processing, IEEE Transactions on*, 15(11):3488–3500, Nov 2006.