Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Bc. Michal Lašan

Height map compression techniques

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Martin Kahoun

Study programme: Informatics

Specialization: Software Systems

Prague 2016

Dedication.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In .... date .... signature of the author

Název práce: Komprese výškových map

Autor: Michal Lašan

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: Mgr. Martin Kahoun, Univerzita Karlova v Praze

Abstrakt:

Klíčová slova:

Title: Height map compression techniques

Author: Michal Lašan

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Martin Kahoun, Charles University in Prague

Abstract: The goal of this thesis is to design a suitable method for lossy compression of heightmap terrain data. This method should accept blocks of float samples of dimensions $2^n x 2^n$ at the input, for which it should be able to perform progressive mip-maps (progressive lower-resolution representations) decompression. For every mip-map, it should keep the reconstructed data within a certain maximum absolute per-sample error bound in the unit of meters adjustable by the user. Given these constraints, it should be as efficient as possible. Our method is inspired by the second generation of progressive wavelet-based compression scheme modified to satisfy the maximum-error constraint. We simplified this scheme by factoring out unnecessary computations in order to improve the efficiency. Our method can compress a 256x256 block in about 30 ms and decompress it in about 1 ms. Thanks to these attributes, the method can be used in a real-time planet renderer. It achieves 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, mip-map, guaranteed maximum error bound

Contents

1. Introduction

In the beginning of this chapter, we clearly state the aim of this thesis and then briefly describe and summarize the works most related to the topic. In the end, we present and explain our decision how to solve the assignment which we made on the basis of this literature.

The aim of this thesis is to either find or come up with a method which solves the task as well as possible. The task is just to compress a regular square of float terrain samples as efficiently as possible, while enabling subsequent real-time progressive decompression of its data from the coarsest to the finest mip-map. The decompression should be as fast as possible. The maximum per-sample deviation of the compressed data must be controllable by the user. No rendering of data has to be handled, as it is supposed that the application using this method will handle this. Knowing these requirements, we started researching the available literature. However, we did not find any method which solves exactly this task while doing nothing additional which just decreases the required efficiency for our purpose. The point is, that most works related to heightmap compression are also able to render the compressed data. Many times, the compression is built into their multiresolution (LOD-ing[1]) rendering pipeline. Of course, for a terrain renderer, it is crucial to implement some form of multiresolution rendering in order to reach reasonable frame rates, but this is not what our method should handle.

Our method should just be plugged into an already existing multiresolution rendering engine, the node of the LOD hierarchy of which is a regular square of float height samples stored completely independently from the other nodes. It must be stated that our method should not interfere in any way with how this engine traverses this hierarchy in order to render a scene. The mip-maps which the decompression should be able to provide are in fact a form of level of detail technique, but it must be made clear that the ability to provide simplified multiresolution representations of every square of the LOD hierarchy is just a tiny part of the multiresolution rendering pipeline, a LODing terrain engine can certainly not be build solely on this ability. The application using the desired method should be able to traverse its multiresolution hierarchy of square nodes on itself in order to render a scene. After selecting which squares should be displayed (the lower the distance, the more detailed the displayed square), the application then should decide for each of the squares which mip-map of it will be displayed. Thus, the mip-maps present a less significant LOD concept inside the greater LOD concept - the multiresolution squares hierarchy. The mip-map selection can be based on the screen-space area of the square in order to reduce the terrain aliasing. For example, when looking at a certain terrain square from a side, a coarser mip-map of it should be chosen than in case we look at it from the top.

As we already stated, we did not find any paper which solves just the task of terrain data compression without solving its rendering. Many methods are able to compress multiresolution hierarchies prepared to be used in rendering which

[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

introduces unnecessary overhead for us. Thus, to find out how the terrain height data can be compressed while respecting a maximum-error bound constraint, we had to look for the suitable compression inside the methods the scope of which is broader. We started with a survey paper summarizing the best known multiresolution terrain rendering methods [7]. All these methods also handle the rendering supported by their own LODing hierarchies. However, some of them contain data compression as a sub-task. Some of them are designed to render just a flat area, others are able to render the whole planet. For our purpose, the methods rendering a flat portion of terrain seemed sufficient to get to know, the scope of both groups of methods is larger than required anyway, but we dug through all of them with the aim to find the most efficient compression inside them. We did not limit our search only to the methods referenced by this survey paper, we also searched recursively by references from the already discovered papers and, of course, on the internet. In the rest of this section, we will briefly describe the methods which contain terrain data compression

The first methods we came accross are C-BDAM[2] [5] and P-BDAM[3] [2]. Both these methods handle the LOD rendering too and perform the data compression in the refinement of a node of their LOD hierarchy. Once the values of a certain node are known, they are used to predict the values of its children as accurately as possible. After that, the differences between these predictions and the real values are computed. These are called residuals. With the help of them, the real values can be restored with absolute accuracy. However, the residuals are then quantized to achieve better compression ratio which means that the compression is lossy. Then, they are losslessly compressed by an entropy codec. Both these methods are able to compute the residuals in the way which ensures that the error of the reconstructed data is kept within a maximum error bound adjustable by the user in every node of their LOD hierarchy. This can be achieved by a slight modification of the second-generation wavelet lifting scheme [9]. C-BDAM is designed to render just a flat portion of terrain, whereas P-BDAM is just C-BDAM modified to be able to render a whole planet. These modifications do not include any improvements to the efficiency of the compression, so, from our point of view, it is suffiecient to know just C-BDAM from these two methods. What makes C-BDAM the most interesting are two aspects: the outstanding compression ratio achieved and the ability to respect a certain user-set maximum per-sample error bound (the ratio of 64:1 on the whole planet with 16m maximum deviation).

Another paper [6] describes a method for rendering a flat portion of terrain. This method contains data compression based on the same principle - the residuals needed to reconstruct the children of a square node of the terrain LOD hierarchy are compressed. The computation of residuals is based on the wavelet-based JPEG2000 standard. This method is not able to reconstruct the data within a certain maximum-error bound which makes it less interesting to us. Besides, the visual artifacts between adjacent nodes of different LODs are not handled by its rendering pipeline, but it needs not bother us anyway.

After summarizing the available literature, we decided get insipred by the compression inside C-BDAM and tailor it for our needs to create out own method.

[2] Compressed Batched Dynamic Adaptive Meshes

[3] Planet Sized Batched Dynamic Adaptive Meshes

Instead of LOD node in C-BDAM, we put a mip-map in our method. The compression in our method takes place in the transition from a coarser mip-map to the finer one, analogically to the transition from a coarser LOD node to the finer one in C-BDAM. Note the crucial difference that We significantly simplified the compression equations performed in C-BDAM in order to increase the efficiency and speed of this method, while still being able to satisfy the required maximum absolute error bound constraint.

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

2. The wavelets

This chapter consists of two sections. In the first one (2.1), we will briefly and formally describe the main principle and usage of second generation wavelet transformation methods which are relevant for this thesis. In the second one (2.2), we will compare C-BDAM and our method to these methods. Even though C-BDAM is based on the same principle, it differs from these methods a bit, so we will describe the basic differences. Then we will perform the same basic comparison with our proposed method. Our method differs from the described wavelet scheme and C-BDAM a bit more which will be clarified in that section.

2.1 The introduction to second-generation wavelets

Basically, there are two generations of wavelets. The first generation uses dilated and translated wavelet function [1] for computation. The second one uses filter banks to perform high-pass and low-pass filtering [3]. The computational equivalency of these two approaches has been proven [4].

For this work, the second generation of discrete wavelet transform methods is most relevant, so we will briefly describe it in this section in order to give the reader an idea of the wavelet concept which is referred to in many places of this thesis. The second wavelet generation is much easier to understand than the first one, so it is possible to describe its basic idea in a few pages.

Every method of this generation consists of just several subsequent applications of lifting onto the input. The lifting is the basic step of the method. It splits the set of its input signal samples into two parts - low-pass (the low frequency information) and high-pass (residuals, the high frequency information). The lifting is firstly applied on the input set of signal samples and then is recursively applied to the low-pass part produced in the previous iteration until the length of the latest low-pass part is 1. In order to make this recursion possible, the count of samples of the original input of the method must be a certain power of two. If the length of the input is 2^n, the method performs n iterations of lifting. The described successive application of lifting on smaller and smaller input is called the bottom-top pass. We can imagine this as building a pyramid of low-pass outputs the first tier of which is the input itself and every following higher tier is the low-pass output of lifting applied to the tier right below. Every tier is half the width of the previous one and after the bottom-top pass, the highest tier has the width of 1.

After this bottom-top pass, we can perform the inverse top-bottom pass. This pass does not know how the produced pyramid looks, it only knows its highest tier, sized 1. However, it is supposed to be able to progressively reconstruct the whole pyramid from the top to the bottom, only utilizing the knowledge of the high-pass information. Producing a certain tier from the previos one is called the reconstruction which is the exact opposite of lifting.

At this point, you might ask what all these decompositions and backward compositions are good for. What makes them interesting is the fact that the bottom-top pass just needs the high-pass information (residuals) to fully reconstruct the input. This information tends to be sparse and input-dependent - the smoother the input, the less high-pass information it contains. If we compress it well, we can save much storage space. Thus, if we want to store a set of samples the count of which is a power of two in as little space as possible, we will not store the samples directly, but we will store just the compressed residuals produced by the successive iterations of lifting applied to the input. If we are not required to accurately reconstruct the input, we can even decimate (quantize) the residuals. Because this information often contains just details, its careful decimation does not deform the reconstruction much and ensures better compression ratio. One more interesting fact is that the residuals bound to lower tiers of the pyramid carry finer details than those bound to the higher ones. Thanks to this, the more-detailed (larger) sets of residuals can be compressed more aggressively than the less-detailed (smaller) ones. This is called progressive compression and it is used for example in JPEG standard [8].

In the following lines, we will describe the lifting and reconstruction steps more formally. Let us say that the lifting is given the input samples x_k. It splits them into the even ones: $x_{2k} = x_e$ and the odd ones: $x_{2k+1} = x_o$. This splitting is not yet based on any frequency properties of the samples, it is based just on their order. However, these two sets of samples will subsequently be modified, so that the even ones will contain the low-pass information and the odd ones will become the residuals - the high-pass information. This will be performed with the help of two operators: the prediction operator P and the update operator U. P will be used to produce the residuals d from x_o and U will be used to produce the low-pass part s from x_e.

Up to this point, just the common properties of the second-generation methods have been described. Now will come the differences between them. The only thing they differ in is the way they perform lifting and reconstruction. The way the lifting step is performed clearly determines the way how the reconstruction is performed, as the reconstruction must be the exact inverse of lifting. The lifting step varies in the order in which the operators P and U are applied. According to this, the methods can be split into two main groups - the prediction-first ones and the update-first ones.

In the prediction-first methods, the prediction is applied first:

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

The reconstruction must be the exact inverse:

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

In the update-first methods, the update operator is applied first:

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

Here is how the reconstruction looks then:

$$x_o = d + P(s)$$

$$x_e = s - U(x_o)$$

2.2 Comparisons between the wavelets, C-BDAM and our method

In this section, we will describe how C-BDAM and our method differ from the basic second-generation wavelet scheme and from each other. The lifting inside C-BDAM is a slight variation of the update-first approach. The main difference is that the input to the first update is not only x_o, but the whole x. In addition, the computation of s is not the summation of the product of x_e and U anymore, because inside $U(x)$, x_e is multiplied:

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

The inverse reconstruction is then:

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

Moreover, the samples x are regularly distributed in the plane, so the spliting into x_o and x_e no longer depends on the indices of the samples, but on their positions instead (Fig. 2.1). Nevertheless, this is just a formal difference which has no effect on the computation. The size of x_o and x_e is still half the size of x which is crucial to keep the original form of lifting. Note that if the residuals d were simply quantized after lifting and used in the reconstructions inside the second top-bottom pass, each step of the reconstruction would increase the maximum absolute deviation from the original low-pass values produced on the first bottom-top pass. To ensure that the reconstructed values are within the maximum-error bound from their corresponding values produced in the first pass at each tier, the residuals computed in the first pass are slightly corrected according to the actual values in another additional top-bottom pass which then turns out to be identical to the reconstruction (decompression), except for the fact that in the following decompression, just the corrected residuals are used to progressively reconstruct the data.

The method proposed in this thesis shares the same main lifting principle with C-BDAM - it is update-first and uses the whole x as the input to U, but has several differences: the size of x_e and thus s is not half the size of x, but one fourth of it instead, as each four neighboring pixels of x are collapsed into one inside s (Fig. 4.1). Additionally, the lifting is not complete, because the prediction operator is not applied there and the computation of residuals is not performed there either. In the lifting of C-BDAM, just temporary approximate residuals are computed and they are corrected in the subsequent top-bottom pass, whereas in our method, the correct residuals which already ensure the satisfaction of the maximum-error bound constraint are computed directly in the second top-bottom pass, also utilizing the prediction operator. Similarly to C-BDAM, the computations inside this pass are identical to the reconstruction of the data, except for the fact that during the reconstruction, the residuals are not computed

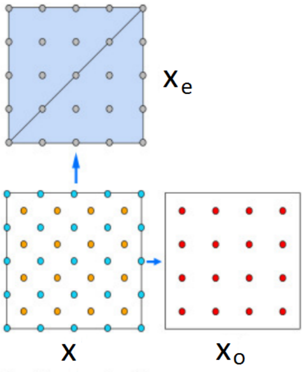

Figure 2.1: Lifting in C-BDAM - the samples x are split into the even ones (x_e) which will become low-pass (s) and the odd ones (x_o) which will become high-pass (d)
Source: C-BDAM [5] (edited)

anymore. Additionally, the prediction operator is applied multiple times in one step of the reconstruction which is explained in Chapter 4. The rationale behind all these differences is explained in Chapter. 5.

3. The outline of the method

In this chapter, we will briefly describe how the compression works. Basically, two consecutive passes are performed on the input heightmap. These passes are analogic to the passes of the described second-generation wavelet methods. The first bottom-top pass computes the target mip-maps - from the largest one to the smallest one. Those will be the mip-maps, against which the accuracy of reconstruction will be measured. The largest mip-map is the input itself. The second top-bottom pass constructs the compressed mip-maps from the smallest one to the largest one with respect to the target mip-maps in order to ensure that the maximum deviation of every compressed mip-map from its corresponding target mip-map is within the maximum error bound set by the user. The smallest of these mip-maps is just the suitably quantized sole value of the corresponding target mip-map. It is directly stored as first. The values of each following compressed mip-map are predicted from its previous compressed mip-map. For these mip-maps, we store just the residuals which are added to the predictions to satisfy the maximum deviation constraint.

More formally, the first pass is given the input square block of float height samples $\boldsymbol{L_n}$ sized $2^n x 2^n$ and produces n mip-maps $\boldsymbol{L_{n-1..0}}$ from it, one by one. The dimension of $\boldsymbol{L_i}$ is half the dimension of $\boldsymbol{L_{i+1}}$. Generally, $\boldsymbol{L_i}$ can be computed from $\boldsymbol{L_{i+1}}$ by any form of averaging of pixels - see the details in the following chapter.

The second top-bottom pass has already $\boldsymbol{L_{0..n}}$ available and computes $\boldsymbol{L^{\bullet}_{0..n}}$ - the compressed mip-maps. The dimension of $\boldsymbol{L_i}$ and $\boldsymbol{L^{\bullet}_i}$ is the same. The computation ensures that the maximum absolute deviation between their corresponding samples is not greater than $\boldsymbol{D}$ - the parameter set by the user. This will be denoted 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 will achieve this with the help of the uniform quantizer Q_D the quantization step of which is set to the maximum value which still respects this error bound:

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

where x is an arbitrary float sample or block of samples. The quantizing step of this quantizer is $2\boldsymbol{D} - 1$ in case $\boldsymbol{D} \geq 0.5$ and $2\boldsymbol{D}$ otherwise.

As we already mentioned, $\boldsymbol{L^{\bullet}_0}$ is just the quantized sole value of $\boldsymbol{L_0}$:

$$\boldsymbol{L^{\bullet}_0} = Q_D(\boldsymbol{L_0})$$

Thanks to the fact that the quantizer respects the maximum-error bound $\boldsymbol{D}$, $maxdev(\boldsymbol{L_0}, \boldsymbol{L^{\bullet}_0}) \leq \boldsymbol{D}$.

Then, the values of every following $\boldsymbol{L^{\bullet}_{i+1}}$ are predicted from the values of the previous $\boldsymbol{L^{\bullet}_i}$. The raw differences between the target values and the predicted values are denoted as $\boldsymbol{E_{i+1}}$ (the residuals). With the help of them and the predictions from $\boldsymbol{L^{\bullet}_i}$, we would be able to accurately reconstruct the target mip-map $\boldsymbol{L_{i+1}}$. However, these residuals are then quantized with the uniform quantizer Q_D to $\boldsymbol{E^{\bullet}_{i+1}}$. With the help of the quantized residuals, we are no longer able to accurately reconstruct $\boldsymbol{L_{i+1}}$, but thanks to the fact that the used quantizer keeps the maximum absolute error within the bound $\boldsymbol{D}$, we can guarantee that the reconstructed $\boldsymbol{L^{\bullet}_{i+1}}$ will satisfy the maximum-error constraint: $maxdev(\boldsymbol{L^{\bullet}_{i+1}}, \boldsymbol{L_{i+1}}) \leq \boldsymbol{D}$. Here is how we construct $\boldsymbol{L^{\bullet}_{i+1}}$:

$$\begin{aligned}
\boldsymbol{E_{i+1}} &= \boldsymbol{L_{i+1}} - P(\boldsymbol{L^{\bullet}_{i}}) \\
\boldsymbol{E^{\bullet}_{i+1}} &= Q_D(\boldsymbol{E_{i+1}}) \\
\boldsymbol{L^{\bullet}_{i+1}} &= P(\boldsymbol{L^{\bullet}_{i}}) + \boldsymbol{E^{\bullet}_{i+1}}
\end{aligned} \tag{3.1}$$

Thanks to the fact that the residuals $\boldsymbol{E_{i+1}}$ are computed with respect to the target mip-map $\boldsymbol{L_{i+1}}$, the maximum-error constraint is satisfied, no matter what values are in $\boldsymbol{L^{\bullet}_{i}}$ and what the prediction operator P looks like. At the end, the quantized residuals $\boldsymbol{E^{\bullet}_{0..n}}$ are compressed with the help of an entropy codec (Zlib) and stored ($\boldsymbol{E^{\bullet}_{0}} = \boldsymbol{L^{\bullet}_{0}}$). The order of their storage is from $\boldsymbol{E^{\bullet}_{0}}$ to $\boldsymbol{E^{\bullet}_{n}}$, so that progressive decompression is possible. The more accurate P is, the smaller the residuals are, thus the higher the compression ratio is. The details of the prediction operator used in this method are described in the following chapter. The higher $\boldsymbol{D}$ is, the less entropy there is among the residuals, thus the higher the compression ratio is, but the lower the reconstruction quality is.

The real-time decompression then just reads the stored quantized residuals and decompresses them with the help of the same entropy codec. Thanks to the fact that the residuals of a smaller mip-map are stored before the residuals of a larger mip-map, progressive decompression of mip-maps $\boldsymbol{L^{\bullet}_{0..n}}$ is possible, utilizing the same principle of producing predictions from the previous reconstructed mip-map and adding residuals to them (eq. 3.1). Of course, in order for this to work, the prediction operator must be identical to the one used in the compression.

4. Details of the method

In this chapter, the method is described in more detail. Unlike the previous outline, this description should be sufficient enough for the reader to implement this method by themselves. In Section 4.1, it will be said how exactly we construct the target mip-maps during the first bottom-top pass and what alternative constructions we also considered. In Section 4.2, we will explain what is the form of P - the prediction operator - and how exactly it is applied in the second top-bottom pass in order to compute the residuals needed to reconstruct a finer mip-map from the coarser one and also perform the reconstruction with the help of these residuals. Note that this method does not use an update operator, see Chapter 5 for the explanation and details.

4.1 Bottom-top pass

As we already said, in the first bottom-top pass, we just construct the target mip-maps one by one, from the largest one - the input itself - to the smallest one, sized 1. At each step, we construct a smaller mip-map from the last constructed one. The dimension of the new mip-map is half the dimension of the last one, in other words, it is half as detailed. Generally, we can build the new mip-map by any form of averaging of pixels of the larger mip-map. In the previous chapter, we explained that the maximum absolute error of the reconstruction is not dependent on how the mip-maps look, as long as they contain valid values (no infinities, NaNs). However, the appearance of the mip-maps affects the compression ratio. The closer the neighboring mip-maps are to each other, the lower the residuals of the transition from the smaller one to the larger one are, thus the higher the compression ratio is. Additionally, as described in Chapter **??**, inside the renderer in which the method has been applied, the mip-maps of a certain terrain square are carefully selected, so that aliasing is minimized. This decision is based on the area of the square projected to the screen. This means that while looking at a certain square from above, its finest mip-map is displayed and during a fixed-radius circular traversal around it up to the point when we look at it from a side, we will be gradually displaying coarser (less detailed) mip-maps of the square. This means that if the mip-maps are significantly different from each other, disturbing visual artifacts might occur during this traversal. The best way how to minimize these artifacts is to use the simplest form of averaging of heights when producing a lower-resolution mip-map where the height at every pixel of the smaller mip-map $\boldsymbol{L_i}$ will be the average of the heights of the four corresponding neighboring pixels inside the larger mip-map $\boldsymbol{L_{i+1}}$:

$$\boldsymbol{L_i}[m][n] = \frac{\sum\limits_{om=0}^{1} \sum\limits_{on=0}^{1} \boldsymbol{L_{i+1}}[2m+om][2n+on]}{4} \tag{4.1}$$

For a comparison, in transition to a coarser LOD in C-BDAM, a different form of heights averaging is utilized. It properly conforms to the standard lifting scheme - it uses the update operator to produce the coarser LOD. This averaging is even parametrized by one coeffiecient named subsampling weight the value of which can span from 0 to 1. To the contrary, our method does not use the update operator there which is explained in Chapter 5. However, we tried to use a similar averaging of pixels inspired by the one performed in the update operator in C-BDAM. With the subsampling weight well set, we achieved a slightly better compression ratio, but the mentioned visual artifacts were more disturbing. At last, we decided to minimize the visual artifacts, as they really affect the user experience, and stick to the described simple averaging (eq. 4.1).

4.2 Top-bottom pass

This pass is performed in the offline compression after the first bottom-top pass, in which case it computes the residuals $\boldsymbol{E^\bullet_{0..n}}$ needed to completely progressively reconstruct the compressed multiresolution representation of the input $\boldsymbol{L_n}$. During the following real-time decompression, this is the only pass which is performed, with the only difference that it no longer computes the residuals, but it just reads them and uses them to reconstruct the data. Let us describe it detailly, so that it becomes clear how this pass is implemented.

In the previous chapter describing the outline of the method, we claimed that we construct a larger compressed mip-map $\boldsymbol{L^\bullet_{i+1}}$ from the smaller $\boldsymbol{L^\bullet_i}$ in just one step (eq. 3.1). This is not exacly true, it is a simplification which we made for several reasons: to give the reader a simple high-level idea of the method, to make the fact that $maxdev(\boldsymbol{L^\bullet_{i+1}}, \boldsymbol{L_{i+1}}) < \boldsymbol{D}$ easier to see and to make it clear that the way the target mip-maps $\boldsymbol{L_{n-1..0}}$ look has no effect on this constraint. However, the truth is that to get $\boldsymbol{L^\bullet_{i+1}}$ from $\boldsymbol{L^\bullet_i}$, the prediction operator P is applied consecutively three times. Its form is different at each of these applications which reflects the fact that before each application, different height values are known - the later the application, the more values are known. After each of these applications, the residuals are computed during the compression and added to the already computed values during both the compression and decompression. Nevertheless, the two main principles which ensure that the maximum error bound between $\boldsymbol{L_{i+1}}$ and $\boldsymbol{L^\bullet_{i+1}}$ is kept remain unchanged - during the compression, the residuals are still computed against the target mip-map $\boldsymbol{L_{i+1}}$ after each application of P and all predictions are made from the compressed values which ensures that both the compression and the decompression add the residuals to the same values. Hence, let us explain how these three steps are performed.

When constructing the larger compressed mip-map $\boldsymbol{L^\bullet_{i+1}}$ from $\boldsymbol{L^\bullet_i}$, we can imagine it as every pixel p from $\boldsymbol{L^\bullet_i}$ being substituted by four pixels a, b, c, d in $\boldsymbol{L^\bullet_{i+1}}$ as depicted in Fig. 4.1. This substitution is the exact inverse of the one performed in the first bottom-top pass described in the previous section ???. We will apply the prediction operator and subsequent residuals computation and addition three times in order to compute the values of the four pixels and the residuals needed to reconstruct them from p during the decompression.

In the first of the three steps, we compute the pixels labeled a. To predict them from their corresponding p pixels inside $\boldsymbol{L^\bullet_i}$, we use a simple prediction operator $P_a(\boldsymbol{L^\bullet_i}) = p$. We compute the residuals $\boldsymbol{E_a}$ and $\boldsymbol{E^\bullet_a}$ with respect to the target value a_t in $\boldsymbol{L_{i+1}}$ and then assign a the final value $a\bullet$ (eq. 4.2, recall that Q_D is a uniform quantizer respecting the maximum deviation D: $maxdev(v, Q_D(v)) < D$. It is clear that $maxdev(a\bullet, a_t) \leq \boldsymbol{D}$. The explanation would be the same as in

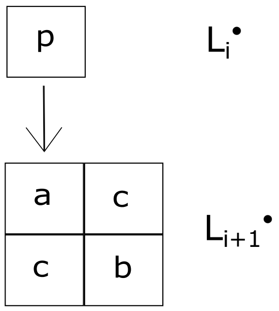

Figure 4.1: Substituting the pixel p in $\boldsymbol{L}_{\boldsymbol{i}}^{\bullet}$ with four pixels in $\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet}$

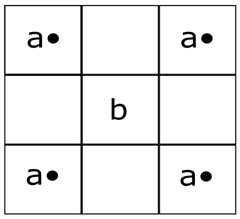

Figure 4.2: The prediction operator of b - $P_b(\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet})$ - averages the compressed heights at all the displayed $a\bullet$.

Chapter 3.

$$\begin{aligned} \boldsymbol{E_a} &= a_t - p \\ \boldsymbol{E}_{\boldsymbol{a}}^{\bullet} &= Q_D(\boldsymbol{E_a}) \\ a\bullet &= p + \boldsymbol{E}_{\boldsymbol{a}}^{\bullet} \end{aligned} \tag{4.2}$$

In the second of the three steps, we compute the pixels labeled b. We do not predict them from $\boldsymbol{L}_{\boldsymbol{i}}^{\bullet}$ anymore, but from the already available pixels $a\bullet$ inside $\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet}$. The prediction operator P_b used for this has now the form of a straight-oriented Neville interpolating filter of order 2. All it does is that when it is requested to predict the height at some pixel, it just averages the heights of its certain four neighboring pixels as depicted in Fig. 4.2. It is easy to see that as long as it is requested to predict the height at pixels b, it always averages only the already known $a\bullet$ pixels. This is the same prediction operator as the one used in C-BDAM to predict the heights of the samples located at the border of a LOD hierarchy node. Once the predictions of b pixels are known, we perform an analogic computation of residuals $\boldsymbol{E_b}$ and their quantizations $\boldsymbol{E}_{\boldsymbol{b}}^{\bullet}$, again with respect to the corresponding target values b_t in $\boldsymbol{L_{i+1}}$. Finally, we assign b its final value $b\bullet$ (eq. 4.3).

$$\begin{aligned} \boldsymbol{E_b} &= b_t - P_b(\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet}) \\ \boldsymbol{E}_{\boldsymbol{b}}^{\bullet} &= Q_D(\boldsymbol{E_b}) \\ b\bullet &= P_b(\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet}) + \boldsymbol{E}_{\boldsymbol{b}}^{\bullet} \end{aligned} \tag{4.3}$$

The reason why C-BDAM uses the order 2 Neville interpolating filter at the borders is that thanks to the way the samples are organized inside a node of its LOD hierarchy, the filter does not pick the samples behind the node's border. We can view the mip-map in our method as an analogy to the node in C-BDAM. However, the spatial organization of mip-map samples in our method differs from the organization of samples inside a LOD node in C-BDAM, so unlike in C-BDAM, in our method it might happen that this interpolating filter comes out of the underlying mip-map. We handle this by only including the valid interior values in the resulting average and completely ignoring the imaginary values behind the mip-map borders. Thus, when computing a certain prediction, we count how many times the filter has hit the interior of mip-map and divide the sum of the valid interior heights with this number of hits. Most of the times, the number of hits will be 4, but it will be 2 at the borders and just 1 at the corners. This way, it is always ensured that the filter does not make any data up, unlike the possible alternative of some mirror extension of data behind the borders. ??? Comparison with mirror extension ???

C-BDAM uses the larger order 4 Neville interpolating filter to predict the heights at the interior of a node of its LOD hierarchy. This filter covers larger area - it samples twelve points instead of four. In addition, it does not compute the average of these points, but their weighted sum. Just like in the case of simple averaging, the sum of the weights is 1. The difference is that the four closest points have a certain positive weight, whereas the remaining eight further points have a different negative weight, the absolute value of which is lower than the first weight (Fig. ???). The property with the lower absolute value indicates that the points which are further affect the result less. The fact that their weights are negative basically means that the valleys and hills are predicted better (Fig. ???).

Unlike C-BDAM, this method uses the smaller order 2 filter even for the interior samples. Let us explain the reasons why we decided to do so. The first reason is the increase of speed. The order 2 filter only averages four values, whereas the order 4 filter averages 12 values. Moreover, the subsequent averaging performed by the order 2 filter can easily be cached during the horizontal traversal which is an additional reduction of the computation overhead (Fig. ???). We also tried using the order 4 filter with various weights settings for the interior values, too. This slightly increased the compression ratio - probably because this filter is better at predicting hills and walleys - but worsened the quality of compression by producing more significant artifacts near smooth terrain's borders (Fig. 4.3) and sharp terrain transitions (Fig. 4.4). The most probable cause of this is that the predictions made by the order 4 filter tend to differ from the neighboring heights more. This emphasises the artifacts.

Generally, the reason why these artifacts occur is that as long as the predictions are close enough to the target mip-map and their quantized residuals are equal to zero, the compressed values might remain above/under the terrain for a long time, but only until one prediction gets a bit further from the target terrain. As soon as it happens, its associated residual will be quantized to a certain non-zero value which will result in the reconstructed value being flipped to the opposite side of the real terrain which produces a visual artifact. It is not a coincidence that this often occurs near a sharp change in the terrain. The predictions produced by the averaging filter get a bit different from the adjacent ones near this change, because at these places, the filter reaches out to the area behind the change (Fig. 4.5). This difference might then cause the difference in

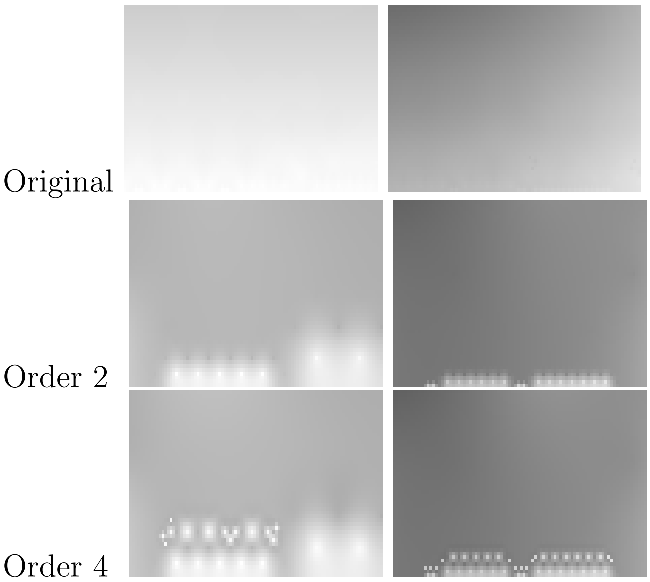

Figure 4.3: Two examples of the difference between artifacts caused by order 2 and order 4 filters near smooth terrain's border - in the first row there are the target heightmaps, in the second, there are the same heightmaps compressed using the order 2 filter, in the third row, the heightmaps compressed with the order 4 filter.

residuals - the quantized residuals further from this change might be all zeroes, whereas the residual near this change not, causing a spike to occur. This spike will then get propagated to the following compressed mip-map levels. The only thing that is guaranteed is that the maximum error bound is still satisfied. ???(musim overit, clipping je nasadeny len v Bohemke, ale je to lepsi napad ako mirroring, tusim, ze to aj zlepsilo kompresny pomer, no nezistoval som pri nom, ake su artefakty)The clipping performed by the predicting filter near the mip-map borders creates the effect similar to a sharp terrain change, too, in a bit different way - by the sole fact that the terrain behind the border no longer follows its trend up to the border (rising, for example), but is practically mirrored behind the border (following the example, falling), because instead of reaching out to the non-existing values out of the mip-map, the existing ones are used.???

In the last of the three steps, we compute the pixels labeled c. We predict them from the already available compressed values of both $a\bullet$ and $b\bullet$ inside $\boldsymbol{L^\bullet_{i+1}}$. The prediction operator P_c used for it has now the form of a diagonally-oriented Neville interpolating filter of order 2. It is the same as the previously applied P_b, except for its different orientation - relatively to P_b, it is rotated by 45 degrees and averages the 4-connected neighbors of the point of application (Fig. 4.6). Once the predictions of heights at c pixels are known, we perform an analogic computation of residuals $\boldsymbol{E_c}$ and their quantizations $\boldsymbol{E^\bullet_c}$, again with respect to the corresponding target values c_t inside $\boldsymbol{L_{i+1}}$. At last, we assign every pixel c its final value $c\bullet$ (eq. 4.4).

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

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

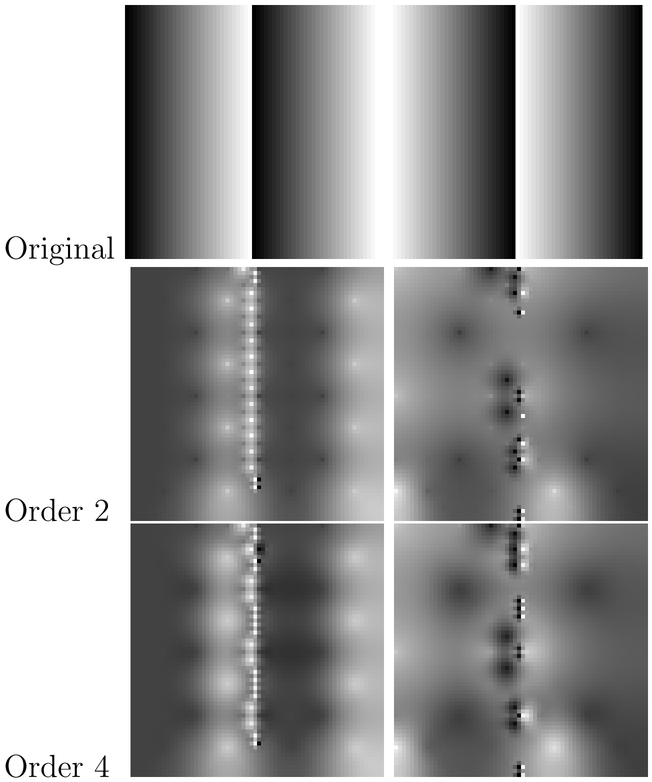

Figure 4.4: Two examples of the difference between artifacts caused by order 2 and order 4 filters near a sharp terrain change - in the first row there are the target heightmaps, in the second row, the same heightmaps compressed using the order 2 filter, in the third row, the heightmaps compressed with the order 4 filter. The span of the values in the original images is from 0 to 16 and the maximum absolute deviation (D) of compression is set to 9.

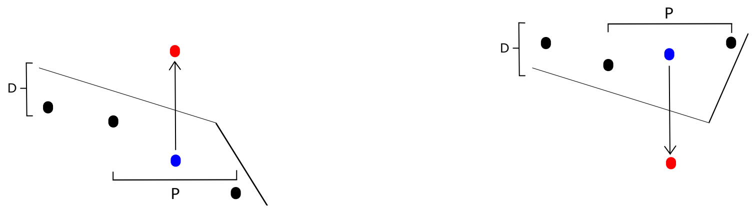

Figure 4.5: Two illustrations of how artifacts can occur near sharp terrain changes - the black dots stand for the predictions which are still within the maximum-error bound D from the target terrain, the blue dots represent the predictions which are just slightly further from the terrain, because their filters span to the area behind the change. Due to the fact that a uniform quantizer with the step of $2D - 1$ is applied to the residuals, the residuals added to the blue predictions will cause them to be shifted by $2D - 1$ to the top (the image on the left) or to the bottom (the image on the right), creating sharp peaks in the reconstructed values - the artifacts.

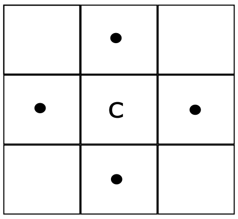

Figure 4.6: The prediction operator of c - $P_c(\boldsymbol{L^\bullet_{i+1}})$ - averages the compressed heights at all the pixels marked with a dot - $\bullet$. Both $a\bullet$ and $b\bullet$ are among these pixels.

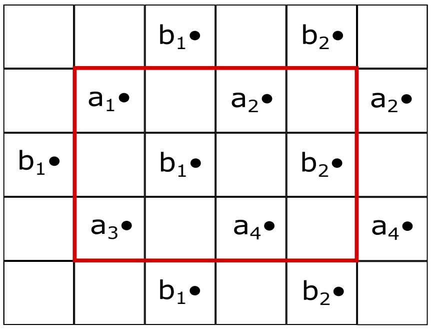

Figure 4.7: (!!!aktualizuj)Handling of border cases in the computation of $P_c(\boldsymbol{L^\bullet_{i+1}})$ - the red line represents the border.

$$c\bullet = P_c(\boldsymbol{L^\bullet_{i+1}}) + \boldsymbol{E^\bullet_c} \tag{4.4}$$

Just like in the case of P_b, P_c can reach out behind the mip-map borders, too. We handled these situations exacly the same way - we only average the heights which are inside the mip-map (Fig. 4.7). Similarly to the predictions of b pixels, we use the order 2 filter to predict the heights at all c pixels - even the interior ones. The subsequent predictions of neighboring c pixels can again be cached to spare some computations. However, the traversal with P_c must now be diagonal in order to make such caching possible (Fig. ???).

Now, let us sum up what we have already done in a few sentences. We have performed all the three subsequent applications of the prediction operator in different forms - P_a, P_b and P_c on the already computed compressed heights in order to obtain the predictions of the yet unknown heights. After each of these applications, we calculated the differences between the predictions and the target values located at the target mip-map at the same places, obtaining the raw residuals $\boldsymbol{E_a}$, $\boldsymbol{E_b}$ and $\boldsymbol{E_c}$. Then we quantized these residuals with Q_D - the uniform quantizer respecting the maximum deviation D which ensures that when the quantized residuals are added back to the predictions, all these summations will be within the deviation D from the corresponding target heights. We called the quantized residuals $\boldsymbol{E^\bullet_a}$, $\boldsymbol{E^\bullet_b}$ and $\boldsymbol{E^\bullet_c}$. Together, they form $\boldsymbol{E^\bullet_{i+1}}$ - all the residuals required to reconstruct the larger compressed mip-map $\boldsymbol{L^\bullet_{i+1}}$ from the previous compressed $\boldsymbol{L^\bullet_i}$.

Finally, with all the quantized residuals computed, we encode them losslesly using an entropy codec and store them. We firstly store $\boldsymbol{E^\bullet_0} = Q_D(\boldsymbol{L_0})$, then $\boldsymbol{E^\bullet_1}$, up to $\boldsymbol{E}_{\boldsymbol{n}}^{\bullet}$. Thanks to this organization, when we want to run-time decompress any $\boldsymbol{L}_{\boldsymbol{i}}^{\bullet}$, we will be required to read just the starting continuous block of the compressed data $\boldsymbol{E}_{\mathbf{0}..\boldsymbol{i}}^{\bullet}$. This is called the progressive decompression. The decompression itself is performed in a similar way. The only difference is that the quantized residuals are no longer calculated, but just read from the compressed data and decoded. Thus, with $\boldsymbol{L}_{\boldsymbol{i}}^{\bullet}$ available, we obtain $\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet}$ by substituting every pixel labeled p inside $\boldsymbol{L}_{\boldsymbol{i}}^{\bullet}$ by four neighboring pixels labeled a, b, c in $\boldsymbol{L}_{\boldsymbol{i+1}}^{\bullet}$ (Fig. 4.1), the heights of which will then be computed in three steps. At each of these steps, we will just predict the heights of the pixels with the relevant prediction operator. This will be followed only by adding the read and decompressed residuals to the predictions (the last lines of eq. 4.2, 4.3, 4.4).

5. Functional comparison to C-BDAM and wavelets

In this chapter, with the details of this method described, we will compare it with C-BDAM in more detail. The main difference which we already mentioned is that unlike in our method, in C-BDAM, the whole rendering pipeline is contained. However, we can compare how lifting is performed in these two methods. From the point of view of lifting, a mip-map level in our method is analogic to a node of LOD hierarchy of C-BDAM and the per-mip-map maximum absolute error bound constraint which our method has to keep is analogic to the same per-LOD-node constraint in C-BDAM. In the end of Section 2.2, we already mentioned that when constructing a coarser LOD node from the finer one, C-BDAM omits a half of the samples of the finer node. To the contrary, our method omits three fourths of the samples of the finer mip-map level when constructing a coarser one. With respect to the spatial arrangment of the samples, this transition is equivalent to two fine-to-coarser transitions in C-BDAM ???(nieco viac)(Fig. 2.1). In the first transition, the pixels labeled as b are removed and in the second one, the pixels labeled as c are removed, as depicted in Fig. 4.1. However, this equality remains only spatial, not computational.

As we already said in Section 2.2, it cannot be really claimed that our method performs the lifting - during the first bottom-top pass, it does not use any prediction or update operator. However, while slightly simplifying the reality, the averaging of four neighboring pixels described in the beginning of Section 4.2 which is used to construct $\boldsymbol{L_i}$ from $\boldsymbol{L_{i+1}}$ can be viewed as an analogy of applying the update operator of lifting in C-CDAM. The crucial difference is that in our method, the lifting is not complete as no prediction operator is applied there to produce the high-pass part - it computes no residuals yet. In C-BDAM, the lifting is complete - a prediction operator is applied there too in order to calculate intermediate quantized residuals. However, reconstructing the data from the coarsest LOD to the finest one in the subsequent top-bottom pass using just these residuals inside the inverse lifting equations would not ensure the required per-sample satisfaction of the maximum absolute error bound, even if these residuals have been quantized with a uniform quantizer respecting this error bound. The reason is that the calculations inside the proper lifting are much more intricate. After each step of the reconstruction, the maximum deviation from the target LODs would increase and would become uncontrollable.

This is exactly why C-BDAM corrects the intermediate residuals against the heights inside the target LODs computed in the first bottom-top pass. This correction is done in another top-bottom pass. It can be simply described as follows - we just reconstruct a certain finer LOD node from its coarser parent and compare it to its corresponding target LOD node produced in the first bottom-top pass. In the places where the reconstruction differs from the target more than the maximum-deviation constraint allows, the corresponding residuals are shifted within the quantization buckets, so that this constraint becomes satisfied. Because C-BDAM uses the same quantizer as our method - the uniform one, set to satisfy the maximum-error bound - such a shift is possible to find. To find the correct number of quantization buckets by which a residual should be shifted, some intricate computations must be performed, including division, which is undoubtedly a large performance hit. These computations are straightforwardly derived from the lifting equations.

After studying all these equations, we saw an opportunity for simplification there - once it is required to perform a top-bottom pass to correct the intermediate residuals in order to satisfy the maximum error bound constraint, we did not really see it as neccessary to calculate any temporary residuals inside the lifting of the bottom-top pass. This is the reason why we use just an analogy of the update operator in the first bottom-top pass to produce the target mipmaps and do not compute any high-pass information (residuals) there yet, and so do not utilize any prediction operator there. We just let the suitable values of residuals be computed in the following top-bottom pass. These values directly satisfy the maximum-deviation constraint and thanks to the fact that we do not need to ensure this with respect to any intricate lifting scheme, it becomes very simple, if not trivial, to compute them (sec. 4.2). Inside one reconstruction step of the following top-bottom pass, all we do is predict the yet unknown heights in the finer LOD with as much accuracy as possible. These predictions, however, are not linked to the previously performed bottom-top pass, because they have not been applied there at all. Thus, even though the prediction operator is applied three times inside one reconstruction step, we do not have to pay any atention for the equations resulting from it to be exacly inverse to the ones performed in the bottom-top pass. Then we compute the final residuals directly with respect to the target values calculated in the first bottom-top pass. This is undoubtedly a significant deviation from both C-BDAM and the standard second-generation wavelet scheme.

All in all, the way the residuals are computed in this method is a great simplification of the approach used in C-BDAM. Our approach does not even conform to the wavelet scheme of second generation - the lifting is incomplete and the reconstruction is not the inverse of lifting. However, our opinion is that without the per-level residuals correction in the subsequent top-bottom pass, it makes sense to respect this wavelet scheme, because it ensures computational equivalence with the wavelets of the first generation. But as soon there comes the need to correct the residuals at each level, we think that it starts to make no sense to still conform to this scheme, as these corrections destroy this equivalence at a glance - once a quantized residual is manually shifted, so that the resulting value gets closer to the target data, it can no longer be claimed that any of the subsequent reconstruction steps is the exact inverse to the corresponding lifting performed in the first bottom-top pass. Additionally, due to the deviation of C-BDAM from the normal update-first wavelet scheme of second generation which has already been discussed in Section 2.2, we question whether it is still computationally equivalent with the wavelets of the first generation even if it were not for the residuals quantization or cropping. These are the reasonss why we suppose that we can optimize the computations performed inside the second top-bottom pass without any cost. With respect to all the discussed matters, this method should be called wavelet-inspired rather than wavelet-based.

6. Results

We have plugged this method into the real-time planet renderer which we already mentioned in the introduction. With this method, we compressed and then real-time viewed the whole-Earth height data with 90m span between height samples (SRTM). The total size of this dataset is 58 GB. Due to the data redundancy of the LOD hierarchy of the renderer which is caused by the fact that it stores all LODs of terrain completely separately, the total size of the SRTM dataset converted to this hierarchy was 260GB. When we applied this compression method to every independent square node of this LOD hierarchy, we managed to compress this data down to 7GB with the maximum deviation of the compression set to 5m. This yield the compression ratio of 37:1. However, if we take the size of the original SRTM dataset as the original size, the compression ratio will drop to just about 8:1. It needs to be said that the first ratio is solely the credit of our method, whereas the second one is influenced by both the way the renderer handles the data and our method.

For a comparison, C-BDAM achieved the compression ratio of 64:1 on the same 90m-resolution SRTM dataset with the maximum error bound set to 16m. As we already mentioned, C-BDAM contains its own LOD rendering hierarchy without any redundancy - a finer LOD is constructed from the coarser one which is where the compression takes place. Thanks to this, there is no data redunduncy in the LOD hierarchy of C-BDAM, so the compression ratio of this method was evaluated with respect to the original size of the dataset which is 58 GB. The final size of the compressed data prepared for rendering in C-BDAM was just 870MB.

The most accurate comparison of our method with C-BDAM we could perform was compress the SRTM dataset prepared for rendering in the mentioned application by our method with the maximum deviation set to 16m. We did so, the final size of the data was ??? GB which yield the compression ratio of ??? when compared to the size of the dataset prepared for the renderer (260 GB) and the compression ratio of ??? when compared to the original size of the dataset (58 GB). Again, the first compression ratio is only the credit of our method, whereas the second one is also influenced by the way the data is handled by the renderer (the LODs redundancy, overlaps, etc.). However, in terms of functionality, C-BDAM is not analogic to our method, because our method cannot handle the rendering. It is only analogic to the renderer with our method plugged in - only this way, both compression and rendering are achieved. For this reason, the second compression ratio is the most relevant one for comparison, even though the implementation of the renderer is completely out of the scope of this thesis. Even though the compression ratio resulting from the usage of this renderer together with our method is worse than the one achieved by C-BDAM, the redundancy of the LOD hierarchy of the renderer has some benefits which have already been mentioned in the introduction - mainly much faster access to the data. Whereas this renderer is able to render a scene viewed from quite close to the ground quite quickly because it only needs to fetch the relevant LODs which are stored indepentendly, this is not the case of C-BDAM - to display such a scene, it needs to gradually reconstruct every ancestor LOD node of the nodes displayed in the scene which undoubtedly takes more time. However, we did not

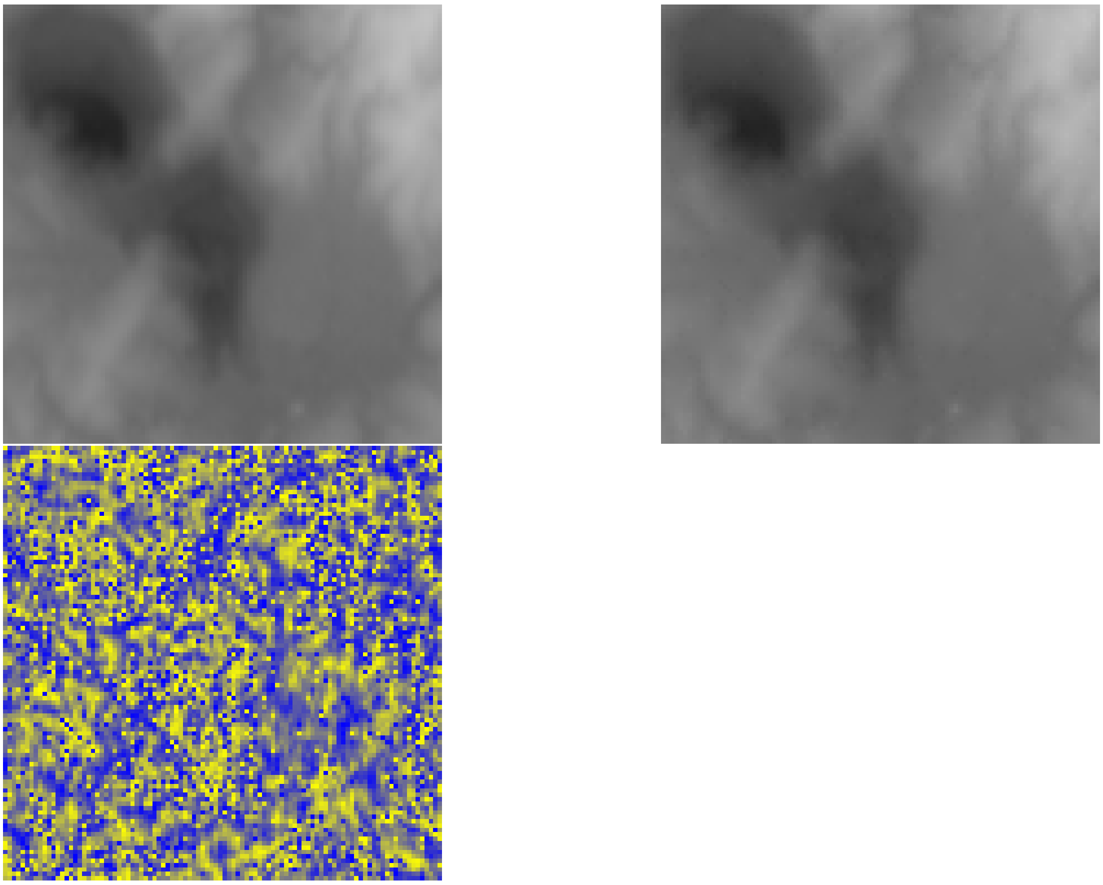

Figure 6.1: From the top to the bottom - the original terrain, the same terrain compressed with the maximum deviation of 5m, the difference between these two. The brighter the color, the greater the value. In the difference image, the yellow color means 4.5m, whereas the blue color means -4.5m.

measure these times.

Fig. 6.1 shows a part of a heightmap compressed by this method, along with the differences from the original.

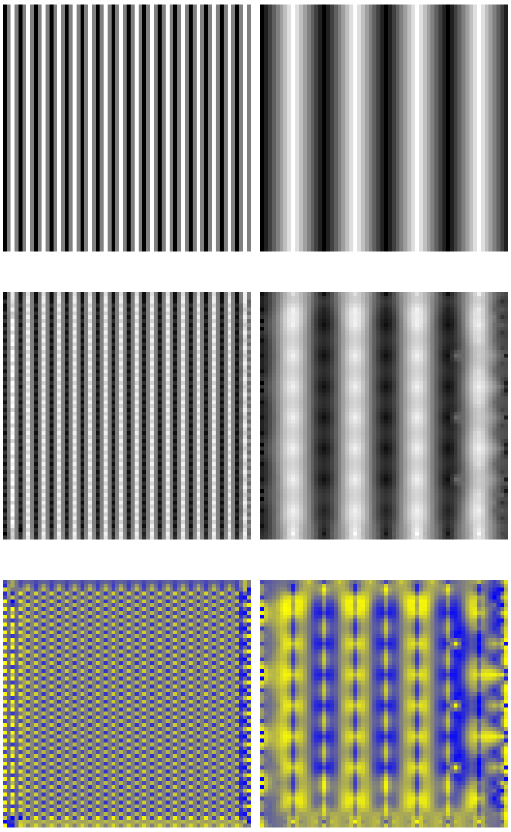

Figure 6.2: Two synthetic test images of size 64x64, each one containing spiky terrain with the heights ranging from -16 to 16. On the left, the longitude of spikes is 4, on the right, it is 16. From the top to the bottom - the original, compressed with the maximum deviation of 5, the difference between 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.

7. Conclusion

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.

Conclusion

[5]

Bibliography

[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.

[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] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, pages 30–44, 1991.

[9] 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.

List of Tables

List of Abbreviations

Attachments