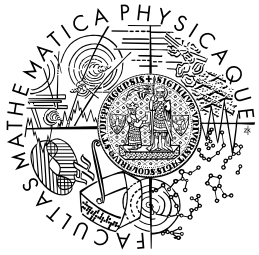

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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:	Computer science
Study branch:	Software Systems

Prague 2016

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 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

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

Title: Height map compression techniques

Author: Bc. Michal Lašan

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Martin Kahoun, Department of Software and Computer Science Education

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 $2^n \times 2^n$ as an input, for which it should be able to perform progressive decompression of mip-maps (lower-resolution representations). It should keep the reconstructed data within a certain maximum per-sample error bound for each mip-map level. This bound should be in the unit of meters and 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 2 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

Hereby, I would like to thank my supervisor for providing valuable feedback on the thesis. I would also like to thank my family and friends who supported me during my studies.

Contents

Introduction

A planetary renderer based on real data has to work with huge amounts of information. This includes its storage and distribution to both the developers working on the application and the users of it. Data compression can be very beneficial in this situation — it can greatly reduce all the storage demands. This results in decreased network and disk utilization — the data needed to transmit via network become smaller, so the transmittion is quicker. Similarly for the rendering, the data which have to be fetched from disk into RAM are smaller which decreases the transfer time, often being one of the major real-time performance bottlenecks.

In this kind of renderer, terrain information forms a significant subset of the utilized data. These are basically the heights of the terrain at all places of the planet. To keep the size of the data reasonable, the terrain height is usually discretely sampled in a regular square grid of some density (eg. 90m per sample). Each one of these height samples is a number — either double (8 bytes) or float (4 bytes) or short (2 bytes) — the accuracy of which is sufficient enough for the given purpose. This number stands for the height at the location of the sample in meters. This thesis focuses on compression of this kind of data represented by float samples.

Classical image compression techniques (eg. png, jpeg) [25, 20] are not suitable for this purpose, because they are designed to compress four 1-byte channels per pixel. They benefit from patterns and similarities of the channel values among adjacent pixels. Putting a 4-byte float height value into four channels directly does not result in anything meaningful from which these methods might benefit — even with a slight change of the height value, all the four channels, into which it has been put, can change significantly. This is why they do not achieve good compression ratios when used this way. Of course, this might be sorted out by a more sophisticated translation of the height value into the channels, but this approach would still not be able to control the maximum absolute height error of the compressed data which is a very important feature for simulations.

For example, if the compression flattenned a narrow but high peak, inside the simulation it would be possible to choose a flight trajectory through this peak which would result in a collision in the real world. Simply put, it is important to ensure that the compressed data differ minimally from the real data. Even though JPEG 2000 offers a maximum deviation control, it is only able to control the per-channel maximum error which is impossible to translate into the height error, as a single height value would be decomposed into four channels. Thus, heightmap compression requires a different approach — the heights have to be handled directly as float values and similarity or closeness of adjacent height values has to be exploited.

Generally, every application designed to render huge scenes must implement some kind of multiresolution approach in order to keep the rendering frame rates reasonable. This is called level of detail (LOD), i.e., degradation of quality of the displayed data with the growing distance in order to optimize the rendering. A planetary renderer certainly belongs to this group of applications, so among the other things, it must somehow ensure that the further the terrain is from

the camera, the less detailed it is. With the growing distance, the amount of observable detail decreases anyway, so displaying it would be just an unnecessary performance burden.

Such LOD-ing approach is almost always implemented by a multiresolution tree hierarchy of nodes [23]. Usually, all these nodes have the same form — triangles, squares, multipolygons, etc. The leaf (bottom-level) nodes of this hierarchy are the partitions of the original terrain data and the other non-leaf nodes are gradual simplifications of the leaf nodes — the higher the level a node belongs to (the further it is from the leaf level), the coarser representation of data it presents. This hierarchy gives a very good basis for the subsequent multiresolution rendering of scene — given a camera position, we determine which nodes will be rendered on the basis of their distance from the camera, using an appropriate metrics. This choice forms a cut in the hierarchy, with its lowest point at the finest node — the one closest to the camera. The height of this cut increases with growing distance from the camera.

To prepare a terrain LOD hierarchy, several transformations of the input terrain data are required — projecting it from the geographic coordinates into the space of the LOD hierarchy followed by splitting it into its basic partitions forming the leaf nodes of the hierarchy, which are then iteratively grouped into coarser parent nodes in order to build the whole multiresolution hierarchy. The nodes of this hierarchy from subsequent levels might either be stored completely independently from each other or dependently — for the children of a certain node we remember just the data which are required to reconstruct the children from their parent. The independent approach introduces significant data redundancy — every non-leaf level of the hierarchy stores independently again the same terrain data as the following finer level, just in a coarser form. This results in the greater size of the resulting data hierarchy prepared for rendering. On the other hand, this approach makes the rendering quicker — in order to fetch a certain node, we do not have to fetch the data of all its ancestors. In practice, both approaches are used. The application which served as a testbed for our compression method uses the first approach — every node of its LOD hierarchy is a square, stored completely independenly from the others. The method designed by us should be able to compress such a square, not interfering with the LOD hierarchy of the application at all — not trying to remove the redundancy caused by independent storage of its nodes in any way, because, at it has been said, it is beneficial in some aspects.

More precisely, the aim of this thesis is to review the existing heightmap compression methods and either find or come up with a method which is able to compress a regular square of float terrain samples the dimension of which is 2^n as efficiently as possible, while enabling subsequent real-time progressive decompression of its data from the coarsest to the finest mip-map. Mip-mapping is a standard method for multiresolution representations of textures. Mip-maps are gradually smaller and coarser squares with the dimensions $2^{n-1..0}$ representing the same data [17]. Usually, every pixel inside a certain mip-map is the average of the four corresponding child pixels inside the previous larger mip-map. This averaging can be performed in the domain of colors, but in our case, it should be performed in the domain of heights. The decompression should be as fast as possible. We should explore the possibility to perform the decompression

on the GPU mainly in order to save RAM. We should keep in mind that from among the mentioned optimization objectives, the resulting size of compressed data is the most important one. The maximum absolute 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 renderer using this method will handle this.

The mip-maps which the decompression should be able to provide are in fact a form of level of detail technique too, 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.

In Chapter 1, we list the methods which contain heightmap compression and form a conclusion to get inspired by the compression used inside one of them named C-BDAM in order to build our own method. In Chapter 2, we briefly describe the core component of compression in many of the mentioned papers, including C-BDAM — the wavelet transform. Following this, we compare the basic wavelet approaches to the compression inside C-BDAM and our method. Lastly, we briefly describe the basic methods of lossless compression which are often utilized in wavelet compression. In Chapter 3, we briefly describe the basic outline of our 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 Chapter 6. In Chapter **??**, we discuss these results along with the possibility how the decompression could be put onto the GPU in order to spare RAM. So far, we have not found a way how this could be done.

1. Related work

A survey paper by Pajarola et al. [23] summarizes the best known multiresolution terrain rendering methods. All these methods also handle the rendering supported by their own LOD-ing hierarchies. We only looked at those of them which contain data compression as a sub-task. Some of them are designed to render just a flat area [13, 22, 9, 18], others are able to render the whole planet [4, 12]. In the rest of this chapter, we will briefly describe the methods which contain terrain data compression

C-BDAM[1] [13] and P-BDAM[2] [4] by Gobbetti et al. turned out to be the most promising ones after comparing with the rest of the methods. Both these methods handle the LOD rendering too and perform wavelet-based [3] 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 [26, 19]. Both these methods are able to compute the residuals in the way which ensures that the error of the reconstructed data are 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 [27]. 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. It achieved the ratio of 64:1 on the whole-planet data with 90 m resolution and 16 m maximum height deviation — 58 GB before the compression and 869.9 MB after it.

Wavelet tiled pyramids by Ricardo Olanda [22] is another 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. However, this method is not able to reconstruct the data within a certain maximum-error bound. Due to this fact, we cannot judge its compression ratios, as they are not connected with any maximum deviation. Besides, the visual artifacts between adjacent nodes of different LODs are not handled by its rendering pipeline, but it is not our concern anyway.

Apart from that, we found several methods dedicated to compressing triangulated terrain representations [**?**, 18, 12]. Two of them [9, 18] perform decompression on the GPU and are also able to keep the compressed data within a certain

[1] Compressed Batched Dynamic Adaptive Meshes
[2] Planet Sized Batched Dynamic Adaptive Meshes

maximum deviation adjustable by the user. However, they add more complexity to the task as they allow for greater irregularity of their input data. Unlike the previous methods which work with regularly distributed height samples, the triangle meshes these methods compress can be irregular which introduces the need to handle more spatial information. Due to this, the compression ratios they achieve are all way lower than the ones of the previous methods. Our method is expected to work with just regularly distributed height samples — square mip-maps, so this additional generalization is unnecessary — it only results in worse efficiency and greater complexity compared to the methods which exploit this regularity well.

HFPaC [10] by Durdevic and Tartalja is the only method we found which focuses solely on heightmap compression, which looked promising at first. Moreover, it is GPU-based which can bring greater performance. However, it has several severe limitations. It is near-lossless, but the maximum deviation of the reconstructed data cannot be controlled by the user. Quite to the contrary, it varies on the basis of both the characteristics of the input and the settings of its internal parameters, such as the size of its compression block. Even though the authors of the method observed that this deviation is relatively small most of the time (up to 2 meters), this is undoubtedly not exactly what we want, as our method should be able to support any arbitrary maximum error set by the user. In addition, this method is only able to provide exactly three layers of resolution of the compressed data, but our method should be able to provide a larger number of mip-maps which varies depending on the dimension of the input. Additional modifications of such a complex method in order to make it support progressive mip-mapping would be complicated, if not impossible, and would most probably decrease its efficiency.

As terrain exhibits some fractal characteristics, fractal compression [24, 1] seems as a natural starting point for any heighmap compression technique. However, methods of this family are quite complicated and designed to compress the data without any possibility of progressive decompression. More precisely, it might be possible to introduce the progression there, but to our best knowledge, no one has found the way yet and we did not see any straightforward option to do this either. The same statement holds for the arbitrary maximum per-sample deviation controlled by the user. Additionally, the most important aspect of data beneficial for this family of methods is its regularity or at least certain repeating. This can be best exploited when compressing huge amounts of data. However, our method should be able to compress and decompress just one small square of height samples completely independently from the others and we cannot benefit from much repetition on such a small area. Moreover, it happens in many places that terrain is not fully fractal, mostly due to erosion — the Earth contains a lot of flat or near-flat areas (deserts, basins, etc.). In these cases, the usage of these methods would not be very beneficial. Thus, fractal compression is an interesting area of research, but not very suitable for our purpose.

At the end, we decided to rather stick to the more promising options — the methods which support multiresolution approach by definition. When designing our method, we decided to start off the ideas behind the compression inside C-BDAM. The main reasons are that C-BDAM achieves the best compression ratios and is able to respect a certain maximum-error bound adjustable

by the user which is exactly what we needed, too. As we already mentioned in the beginning, the size of the compressed data was the most important constraint, which is why we chose to follow the best compression ratio. The compression inside C-BDAM is relatively simple which makes it easy to understand and is also fast when implemented. However, C-BDAM contains features which we did not need, so we omitted them and just isolated the compression and tailored it for our needs.

C-BDAM performs the compression inside the refinement of a node of its LOD hierarchy. Due to the request that our method should be able to decompress the mip-maps progressively, it seemed convenient to put the compression in our method inside the refiniment of a mip-map. This way, only the residuals needed to reconstruct the following finer mip-map would have to be fetched in order to perform a single step of refinement. So, instead of a LOD node in C-BDAM, we put a mip-map in our method. Additionally, we significantly simplified the compression equations performed in C-BDAM in order to increase the efficiency and speed of our method while still being able to satisfy the required maximum absolute error bound constraint. However, let it be repeated again that a LOD node in C-BDAM is actually the basic component of its multiresolution rendering hierarchy, whereas a mip-map in our method not, it is only a side component of the LOD hierarchy of the application into which our method should be plugged. The main component of the multiresolution rendering hierarchy of this application is the square which should be compressed by this method, so our method lies outside this hierarchy. However, it is not hard to utilize the compression in C-BDAM inside our method, because the principle remains the same in both cases — we refine a set of height samples to a more detailed one.

2. The preliminaries

This chapter consists of three 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 the compression inside 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 a bit more, see Section 4.2 for the details. Finally, in Section 2.3, we describe two most common approaches to lossless data compression which are both used in Zlib[1] — a free compression library which we use in the implementation of our method.

2.1 The second-generation wavelets

Basically, there are two generations of wavelets. The first generation uses dilated and translated wavelet function [3] for computation. The second one uses filter banks to perform high-pass and low-pass filtering [5]. The computational equivalency of these two approaches has been proven [8]. For this work, the second generation of discrete wavelet transform methods is more 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. We will base this description on 1-dimensional input data. We will also briefly describe how the transition of the principle to 2-dimensional data, such as images, can be performed.

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 size of both of them is half the size of the input. The lifting is firstly applied on the input set of signal samples and then is recursively applied to the low-pass part produced by 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-up 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-up pass, the highest tier has the width of 1 (Fig. 2.1).

To reconstruct the input signal samples, all we need is the highest tier low-pass information together with all high-pass tiers (Fig. 2.1). This can be done in the subsequent top-down pass which builds the low-pass pyramid from the top to the bottom. Producing the lower tier from the previous one and its associated high-pass information is called the reconstruction which is the exact inverse of

[1] http://www.zlib.net/

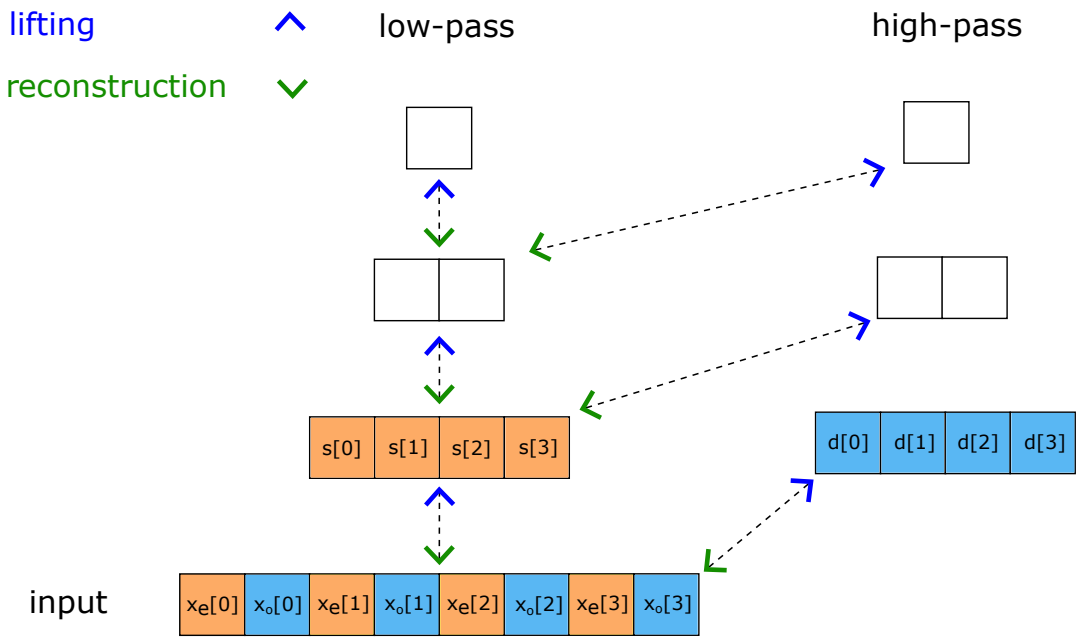

Figure 2.1: An example of recursive wavelet decomposition of 1-dimensional data of samples. On the left side, there is a pyramid of low-pass tiers, the bottom tier of which is the input itself. On the right side, there is the corresponding pyramid of high-pass tiers. The dashed lines represent both the decomposition (lifting) inside the first bottom-up pass (when following the blue arrows) and the inverse compositions (reconstruction) inside the second top-down pass (when following the green arrows). On the input samples, it is also marked which samples form the smaller low-pass part (the orange ones) and the smaller high-pass part (the blue ones). The orange ones are the even ones, as the indexing of samples starts at zero.

lifting.

Now, let us briefly describe how this principle can be extended to 2-dimensional image data. Basically, this extension introduces just one change to the described principle — when the lifting is applied to an image with dimension 2^n, it produces four equally-sized outputs with dimension 2^{n-1}: one low-pass part and three high-pass parts (Fig. 2.2). Each of these high-pass parts contains different information — for example, the first one might contain vertical edges, the second one horizontal edges and the third one diagonal edges. During the subsequent reconstruction, all these four parts are used to produce the larger low-pass part with twice better resolution.

What makes these decompositions and backward compositions useful is the fact that the top-down 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. 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 [25].

In the following lines, we will describe the lifting and reconstruction steps in 1-dimensional domain 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 (high-pass filter) and the update operator U (low-pass filter). 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:

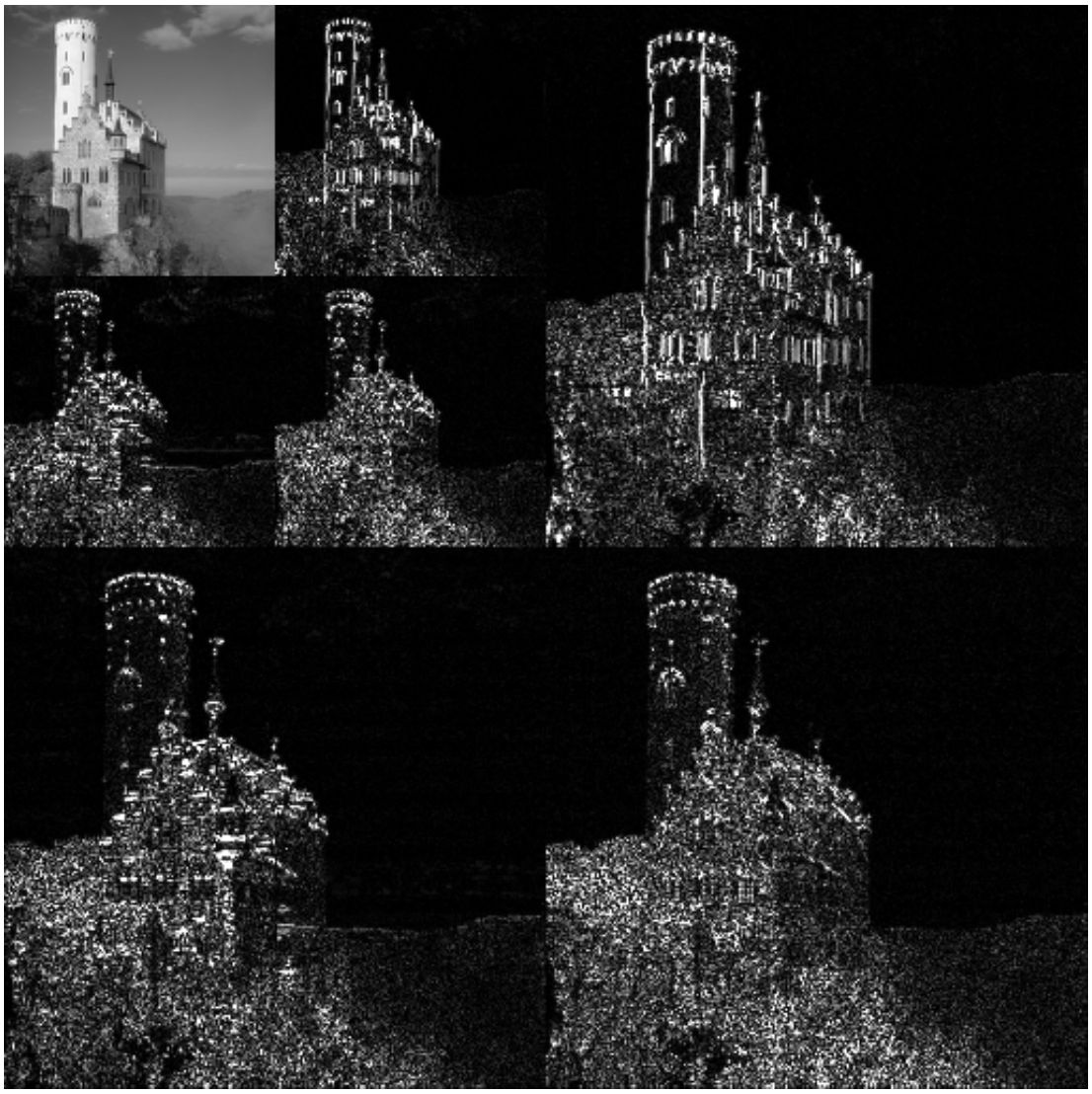

Figure 2.2: An example of recursive wavelet decomposition of 2-dimensional black-and-white image. The sub-image at the top-left corner stands for the low-pass part after two iterations of lifting applied to the input. The original input is the same as this sub-image, except for four times better resolution. The neighbors of this low-pass part are its corresponding high-pass parts. The top-right one of them contains vertical edges, the bottom-left contains horizontal edges and the bottom-right one contains diagonal edges. From these four top-left sub-images, twice as detailed low-pass part can be reconstructed. This can be then merged with the remaining three largest high-pass parts to obtain the original input.
Source: Wikimedia Commons [7]

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

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 Comparing the wavelets to C-BDAM and our method

In this section, we will briefly describe how the compression inside C-BDAM and our method differ from the basic second-generation wavelet scheme. Even though C-BDAM works with 2D data, its lifting still has just two outputs, unlike in the case of images. It is like it because of the spatial arrangment of its height samples. In one iteration of lifting, a half of samples is omitted (Fig. 2.3). Its lifting 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. 2.3). 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 1D lifting.

Note that if the residuals d were simply quantized after lifting and used in the reconstructions inside the second top-down pass, each step of the reconstruction would increase the maximum absolute deviation from the original low-pass values produced in the first bottom-up 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-down

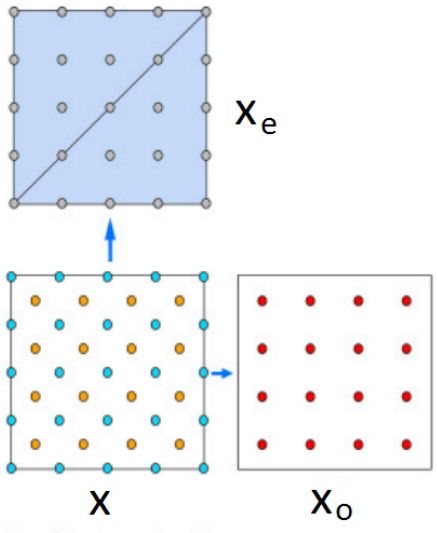

Figure 2.3: Lifting in C-BDAM — the samples x are split into the even ones (x_e) which will become low-pass (s) and the odd ones (x_o) which will become high-pass (d)
Source: C-BDAM [13] (edited)

pass. Basically, each reconstructed value is compared to its target value and when it is too far from it, its residual is shifted by a multiple of the quantization interval, so that the resulting value is close enough to the target value. Thanks to the selection of the quantization interval which respects the set maximum-error bound, it is always possible to find such a shift. The following reconstruction (decompression) is again the same top-down pass, except for the fact that just the corrected residuals are read, decompressed and used to progressively reconstruct the data.

The compression of the method proposed in this thesis performs two consecutive passes on the input heightmap — a bottom-up pass followed by a top-down pass. These passes are analogic to the passes of the described second-generation wavelet methods. 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, just like in the 2D example of images, as each four neighboring pixels of x are collapsed into one inside s (Fig. 4.2).

Additionally, the lifting is not complete, because the prediction operator is not applied there and the computation of residuals is not performed there either. The correct residuals which already ensure the satisfaction of the maximum-error bound constraint are computed directly in the second top-down pass, also utilizing the prediction operator. Similarly to C-BDAM, the computations inside this pass are identical to the subsequent decompression of the data. During the reconstruction, the residuals are read and used to refine the previously decompressed data. See the high-level description of the method in Chapter 3 for more details. 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.

2.3 Lossless compression

Lossless compression decreases the data size without any information loss. It is often used to compress quantized residuals — the output of wavelet transform. In this section, we briefly describe two main approaches to such compression — Huffman coding [16] and LZ77 dictionary compression [28]. We do this because both these approaches are used in Zlib — a publicly available lossless compression library which we use in our method to losslessly compress the quantized residuals. We chose it because it achieved the best compression ratios on our data from among all publicly available lossless compression libraries. It is also quite fast. It firstly applies LZ77 to the data and then Huffman coding. Both these approaches basically profit from the redundancy of data, but in different ways.

LZ77 compression

This method is able to detect duplicite subsequences inside the input which are relatively close to each other. It is very suitable for binary data. It reads the input sequentially during the compression. During this, it keeps track of the last N characters. We call this track the sliding window, with N being its length. In practice, N is relatively large — 32KB in Zlib.

We start at the beginning of the input stream with an empty window. Then, being at a certain position, we examine the data after the window for the longest continuous sequence beginning right after window which can also be found somewhere inside the window. When we have found it, we just write a triple (D, L, C) to the output, with D beingh the distance of the start of the matching sequence found inside the window to the end of the window, L being the length of the sequence and C being the character in the stream right after this sequence. Of course, it might happen that no such sequence is found — especially at the very start — then both D and L will be 0. Note that L might be greater than D — in this case, when we reach the end of the window while trying to find the longest match, we start again D steps back from the window's end. After outputting the triple (D, L, C), we move the sliding window $L+1$ steps forward. Note that this procedure works even for the initial state — the empty sliding window. In this case, (0, 0) is output, followed by the first character of the input stream. In this particular situation, (0, 0) can even be ommitted, as it has to always be written at the beginning. Then, we repeat the same matching of sequences again, as long as there is something to read. When we have reached the end of the stream, we write a special end character at the place of C.

The decompression then works inversely as follows — it start with an empty sliding window and then it starts reading all the following triples (D, L, C). After reading each triple, it moves D steps back from the end of the sliding window and copies L following characters from the current position to the end of the decompressed stream. If L is greater than D, it goes back D steps from the window's end, or equally, it starts traversing the output stream right after the window's end, as in both cases, the same characters will be found (the first character which has been written after the window was the one located D steps back from the window's end). After writing all L characters to the decompressed stream, it writes C to the output stream, pushes the sliding window forward by

$L + 1$ steps and continues again with the following triple. When it encounters the special end character, it reads no more triples.

Huffman coding

Huffman coding is a form of prefix coding. It is usually used to compress data composed of characters. It might not be very suitable for arbitrary binary data, as it would most probably have to partition them into certain characters (bytes) in order to work. Thus, if the binary data are actually composed of float numbers or something else distinct from characters, Huffman coding might not work very well and LZ77 is definitely a better choice. However, in Zlib, Huffman coding is used to compress the data produced by LZ77 — the triples consisting of two natural numbers and one character — which can be easily interpreted as characters.

This algorithm starts by assigning a certain weight to each of the characters contained in the data. The more often the character occurs, the larger its weight should be. Then, according to these weights, it builds a binary Huffman tree, with the characters being the leaf nodes of it. From each non-leaf node of the tree, two edges go — they are marked as "0" and "1". The marks of the edges through which we have to traverse from the root of the tree to a certain character form the code of the character in the compressed stream. Ideally, the greater the weight of a certain character, the closer to the root it should be, so the shorter its code is.

With the tree built, we can easily compress the data. Instead of each character, we write its code. The decompression is then simple once we have the tree at hand — at the beggining of the compressed stream, we are at the root of the tree. Every following bit in the stream than says which edge towards a leaf should we take from the current position inside the tree. Once we get to a leaf, we output its corresponding character to the decompressed stream and return to the root of the tree.

3. The outline of the method

In this chapter, we will briefly describe how the compression works. The details of it are described in the following Chapter 4. Basically, two consecutive passes are performed on the input heightmap — a bottom-up pass followed by a top-down pass (Fig. 3.1). These passes are analogic to the passes of the described second-generation wavelet methods.

The bottom-up 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. Every smaller mip-map is computed from the previous larger one. These mip-maps will be used just as a temporary reference during compression and are not stored in any way.

The top-down pass performs the compression of these reference mip-maps. During this process, it already constructs the mip-maps exactly how they will look after the decompression, from the smallest one to the largest one. Thus, the terms "compressed" and "decompressed" are equivalent when used with mip-maps or heights. This trick ensures that the maximum deviation is respected without any further corrections, because the data needed to progressively construct these mip-maps (residuals) are computed against the reference mip-maps directly on the basis of the mip-maps which will be available during the decompression. The compression computes these residuals from the smallest mip-map to the largest one, quantizes them, losslessly encodes them and stores them (Fig. 3.1).

The first compressed mip-map is just the sole quantized value of the smallest reference mip-map. Then, each following mip-map is firstly predicted from the previous compressed one. These predictions are then subtracted from the corresponding same-sized target mip-map to obtain the residuals. These residuals are then quantized as much as possible so that the maximum deviation of the current compressed mip-map is still respected. Then, they are losslessly encoded and stored (Fig. 3.2). On one hand, the quantization causes the information loss, but on the other hand, it decreases the variability of residuals which makes the output of the subsequent lossless compression smaller.

During the decompression, the fetched and decompressed residuals are just added to the predictions to obtain the finer mip-map (Fig. 3.3). Note that this method is not lossless only due to the quantization. If we omit this quantization, the method becomes lossless. This happens when the maximum height deviation is set to 0 by the user.

More formally, the first pass is given the input square block of float height samples $\boldsymbol{L_n}$ sized $2^n \times 2^n$ 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 $\boldsymbol{L_{i+1}}$. Just like it is common in mip-maps $\boldsymbol{L_i}$ is computed from $\boldsymbol{L_{i+1}}$ by straightforward averaging of pixels — see the details in the following chapter.

The second top-down 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

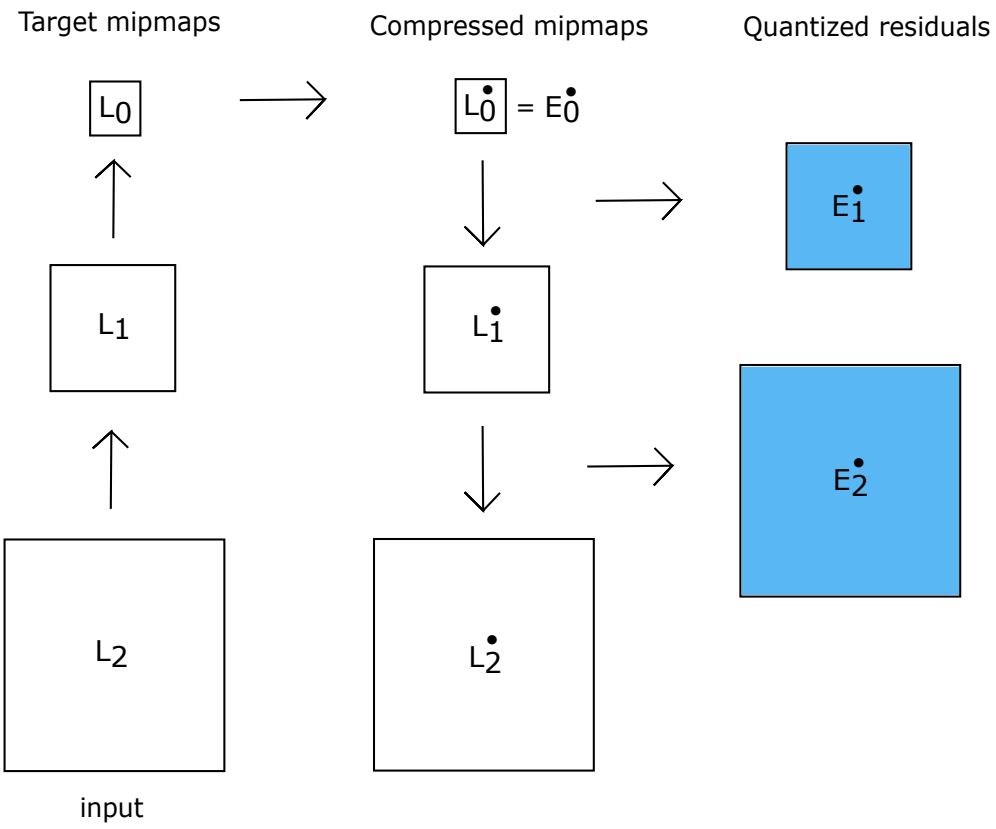

Figure 3.1: The main schema of compression of the method. On the left, the target mip-maps $\boldsymbol{L_n}$ are constructed in the first bottom-up pass. On the right, the compressed mip-maps $\boldsymbol{L_n^\bullet}$ are constructed with respect to the target mip-maps in the second top-bottom pass. The residuals $\boldsymbol{E_n^\bullet}$ needed to progressively reconstruct these mip-maps marked by blue color are losslessly encoded and stored.

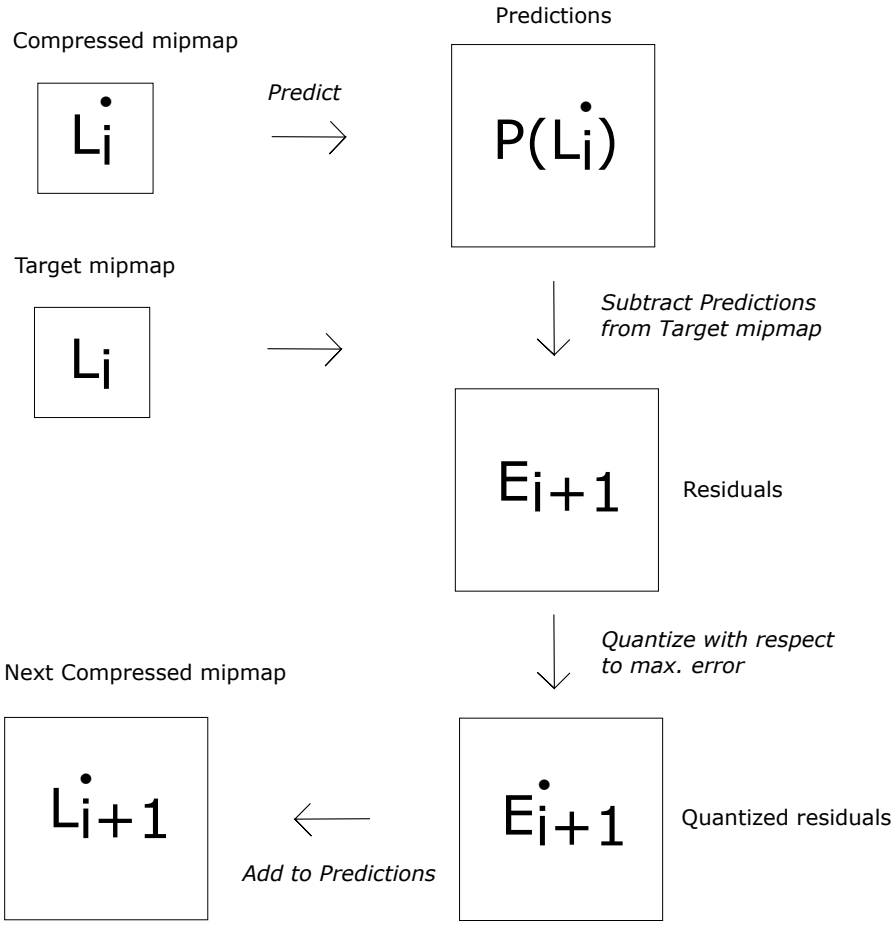

Figure 3.2: The diagram of construction of larger compressed mip-map $\boldsymbol{L}^\bullet_{i+1}$ from the smaller $\boldsymbol{L}^\bullet_i$ during the off-line compression. Only the quantized residuals $\boldsymbol{E}^\bullet_{i+1}$ are losslessly compressed and stored. They are sufficient to reconstruct $\boldsymbol{L}^\bullet_{i+1}$ from $\boldsymbol{L}^\bullet_i$.

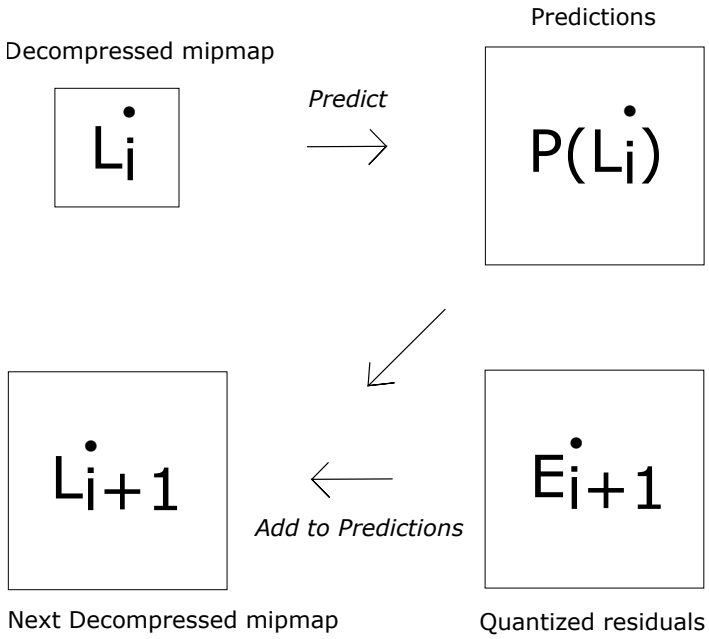

Figure 3.3: The diagram of construction of larger decompressed mip-map $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ from the smaller $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$ during the real-time decompression. The quantized residuals $\boldsymbol{E}^{\bullet}_{\boldsymbol{i+1}}$ are read and decompressed. Then, they are added to the predictions from the smaller mip-map.

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, $\boldsymbol{L^{\bullet}_{o}}$ is just the quantized sole value of $\boldsymbol{L_o}$:

$$\boldsymbol{L^{\bullet}_{o}} = Q_D(\boldsymbol{L_o})$$

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

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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ will satisfy the maximum-error constraint: $maxdev(\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}, \boldsymbol{L}_{\boldsymbol{i+1}}) \leq \boldsymbol{D}$. Here is how we construct $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$:

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

Thanks to the fact that the residuals $\boldsymbol{E}_{\boldsymbol{i+1}}$ are computed with respect to the target mip-map $\boldsymbol{L}_{\boldsymbol{i+1}}$, the maximum-error constraint is satisfied, no matter what values are in $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$ and what the prediction operator P looks like. At the end, the quantized residuals $\boldsymbol{E}^{\bullet}_{\boldsymbol{0..n}}$ are losslessly compressed with the help of Zlib and stored ($\boldsymbol{E}^{\bullet}_{\mathbf{0}} = \boldsymbol{L}^{\bullet}_{\mathbf{0}}$). The order of their storage is from $\boldsymbol{E}^{\bullet}_{\mathbf{0}}$ to $\boldsymbol{E}^{\bullet}_{\boldsymbol{n}}$, 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 lossless 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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{0..n}}$ 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. Section 4.1 details the construction of the mip-maps inside the first bottom-up pass and what alternative constructions we also considered. In Section 4.2, we explain what is the form of P — the prediction operator — and how exactly it is applied in the second top-down 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-up pass

As mentioned in the previous chapter, in the first — bottom-up — pass, we just construct the target mip-maps one by one, from the largest one — the input itself — to the smallest one, 1 pixel in size. 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 and contains four times less pixels. 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 less different the heights between the neighboring mip-maps are, the lower the residuals of the transition between them are, thus the higher the compression ratio is. Additionally, as mentioned in the introduction, 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 $\boldsymbol{L_{i+1}}$:

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

For a comparison, in transition to a coarser LOD in C-BDAM [13], 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. Thus, in order to minimize the visual artifacts, as they really affect the user experience, we decided to use the described averaging scheme (eq. 4.1, Fig. 4.1).

4.2 Top-down pass

This pass is performed in the offline compression after the first bottom-up pass, in which case it computes the residuals $\boldsymbol{E}^{\bullet}_{\mathbf{0}..\boldsymbol{n}}$ 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 mip-maps. Let us describe it in detail, 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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ from the smaller $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$ 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 $maxdev(\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}, \boldsymbol{L_{i+1}}) < \boldsymbol{D}$ 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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ from $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$, 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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ 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 reconstructed 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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ from $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$, we can imagine it as every pixel p from $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$ being substituted by four pixels a, b, c, d in $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$ as depicted in Fig. 4.2. 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 $\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}$, we use a simple prediction operator $P_a(\boldsymbol{L}^{\bullet}_{\boldsymbol{i}}) = p$. We compute the residuals $\boldsymbol{E_a}$ and $\boldsymbol{E}^{\bullet}_{\boldsymbol{a}}$ 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

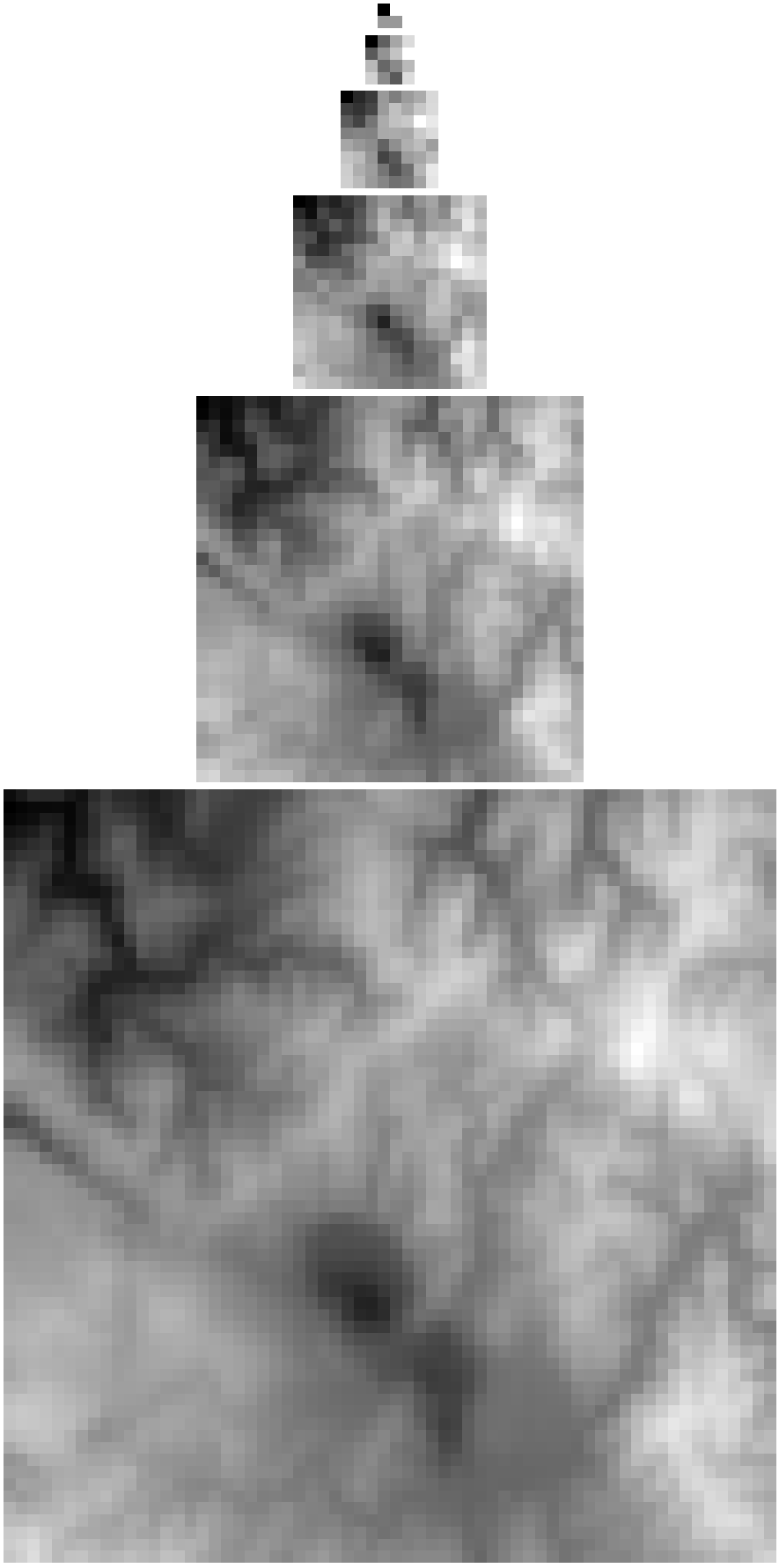

Figure 4.1: An example of the averaging used to produce mip-maps; each mip-map is produced from the previous larger one by averaging of each four neighboring pixels. The brighter a pixel, the higher point it represents.

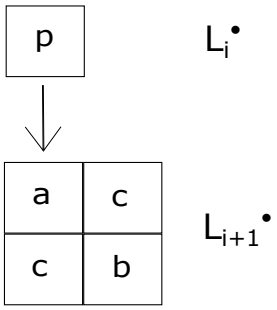

Figure 4.2: Substituting the pixel p in $\boldsymbol{L_i^\bullet}$ with four pixels in $\boldsymbol{L_{i+1}^\bullet}$

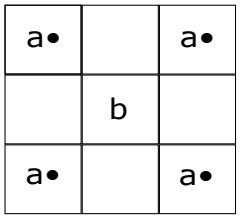

Figure 4.3: The prediction operator of b — P_b — averages the compressed heights at all the displayed $a\bullet$.

Chapter 3.

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

In the second step, we compute the pixels labeled b. We do not predict them from $\boldsymbol{L_i^\bullet}$ anymore, but from the already available pixels $a\bullet$ inside $\boldsymbol{L_{i+1}^\bullet}$. 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. 4.3. 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 $\boldsymbol{E_b^\bullet}$, 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).

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

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

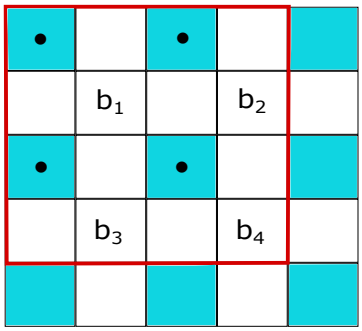

Figure 4.4: Handling of border cases in the computation of P_b — the red line represents the border. From among all the points which can be reached by the filter (marked as blue), only the interior ones (additionally marked with a dot) are averaged.

its LOD hierarchy, the filter does not pick the samples behind the node's border (Fig. 2.3). 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 clipping — only including the valid interior values in the resulting average and completely ignoring the imaginary values behind the mip-map borders (Fig. 4.4). 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 some borders and just 1 at some corners. This way, it is always ensured that the filter does not make any values up, unlike the possible alternative of some mirror extension of values behind the borders.

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. 4.5). 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. 4.6).

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. 4.7). The last reason is that the order 4 filter produces sharper artifacts which is described in

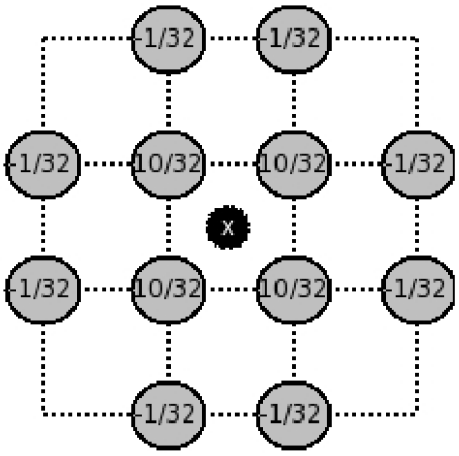

Figure 4.5: The order 4 Neville interpolating filter as used in C-BDAM. Centered at the black point marked with a cross, it calculates a weighted sum of the grey points, the number on each of which stands for its weight inside the sum.
Source: C-BDAM [13]

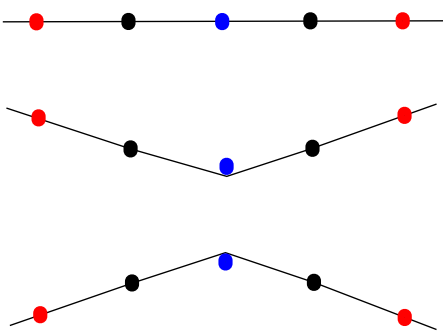

Figure 4.6: A 2D intuition into the function of order 4 filter illustrated on three examples. The black points are the ones with a positive weight inside the filter, the red ones have negative weights. The blue points represent the resulting predictions. The top example shows how the filter behaves on flat terrain — thanks to the fact that the overal sum of weights is 0, the prediction lies exactly on the terrain. The other two examples show sharp terrain, with the predicted point located near the terrain extreme — the bottom of valley in the center example and the top of hill in the lower example. Thanks to the fact that the red points have negative weights, the resulting predictions can get close to the actual terrain in these cases. Of course, these examples are quite synthetic and the 2D view is a significant simplification, but the principle remains the same in practice: the order 4 filter is able to capture the trend (derivation) of terrain between its border points (the ones with the negative weights) and the interior points (the ones with the positive weights).

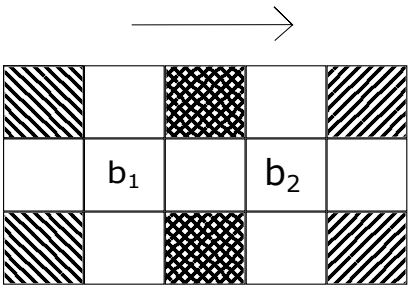

Figure 4.7: The caching of P_b during a horizontal traversal computing predictions of all b pixels. The arrow above the grid shows the direction of traversal — left to right, so that b_1 is predicted as first, b_2 as second and the traversal might continue further to the right. The pixels with top-left to bottom-right strips are averaged to predict b_1, whereas the pixels with bottom-left to top-right strips are averaged to predict b_2. Two pixels are inside both groups — they are stripped in both ways. Thus, the sum of them can be cached during the prediction of b_1 and reused inside the prediction of b_2, sparing one summation per prediction this way.

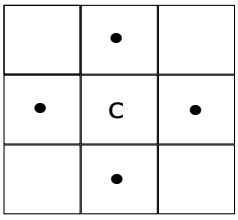

Figure 4.8: The prediction operator of c — P_c — averages the compressed heights at all the pixels marked with a dot — •. Both $a\bullet$ and $b\bullet$ are among these pixels.

Section 6.1 in more detail.

Finally, in the third step of compression, we compute the pixels labeled c. We predict them from the already available compressed values of both $a\bullet$ and $b\bullet$ inside $\boldsymbol{L}^{\bullet}_{\boldsymbol{i+1}}$. 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. 4.8). 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).

$$\begin{aligned} \boldsymbol{E_c} &= c_t - P_c(\boldsymbol{L^{\bullet}_{i+1}}) \\ \boldsymbol{E^{\bullet}_c} &= Q_D(\boldsymbol{E_c}) \\ c\bullet &= P_c(\boldsymbol{L^{\bullet}_{i+1}}) + \boldsymbol{E^{\bullet}_c} \end{aligned} \tag{4.4}$$

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. 4.9). Similarly to the predictions of b pixels,

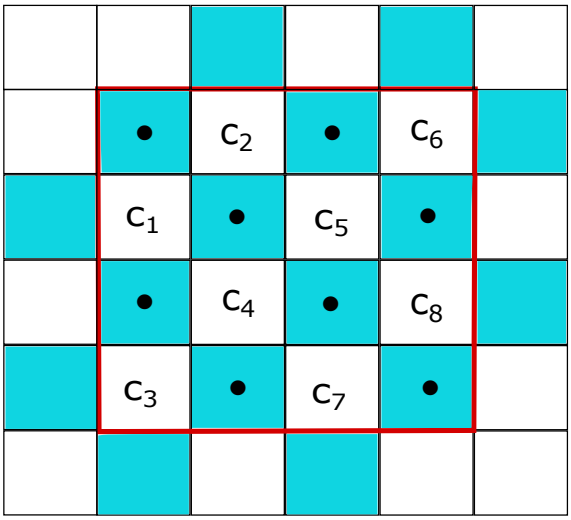

Figure 4.9: Handling of border cases in the computation of P_c — the red line represents the border. From among all the points which can be reached by the filter (marked as blue), only the interior ones (additionally marked with a dot) are averaged.

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. 4.10).

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 losslessly compress them and store them. We firstly compress and store $\boldsymbol{E^\bullet_0} = Q_D(\boldsymbol{L_0})$, then $\boldsymbol{E^\bullet_1}$, up to $\boldsymbol{E^\bullet_n}$. Thanks to this organization, when we want to run-time decompress any $\boldsymbol{L^\bullet_i}$, we will be required to read just the starting continuous block of the compressed data $\boldsymbol{E^\bullet_{0..i}}$. 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, they are just read and decompressed. Thus, with $\boldsymbol{L^\bullet_i}$ available, we obtain $\boldsymbol{L^\bullet_{i+1}}$ by substituting every pixel labeled p inside $\boldsymbol{L^\bullet_i}$ by four neighboring pixels labeled a, b, c in $\boldsymbol{L^\bullet_{i+1}}$ (Fig. 4.2), the heights of which will then be computed in three steps. At each of these steps, we will just predict the heights

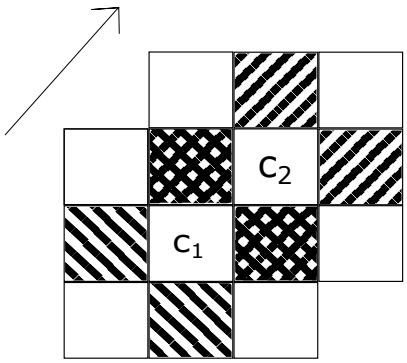

Figure 4.10: The caching of P_c during a digonal traversal computing predictions of all c pixels. The arrow above the grid shows the direction of traversal — bottom-left to top-right, so that c_1 is predicted as first, c_2 as second and the traversal might continue further to the top-right. The pixels with top-left to bottom-right strips are averaged to predict c_1, whereas the pixels with bottom-left to top-right strips are averaged to predict c_2. Two pixels are inside both groups — they are stripped in both ways. Thus, the sum of them can be cached during the prediction of c_1 and reused inside the prediction of c_2, sparing one summation per prediction this way.

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).

The lossless compression of quantized residuals is performed in two steps — packing followed by lossless compression by Zlib. During packing, we divide all the residuals with their quantization interval. Then we shift them up by the minimum amount which ensures that they are all non-negative. Then we crop the number of bits of each residual to the number of bits of the largest value. Obviously, we also store the information needed to perform the inverse of packing in the reconstruction — the quantization interval and the upper-shift amount. Only after this packing are the residuals losslessly encoded by Zlib. The packing proved to be useful, as it additionally increased the compression ratio. The decompression is just the inverse — lossless decompression by Zlib followed by the unpacking — expanding the residuals back into floats, shifting them down by the shifting amount and multiplying them with the quantization interval.

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 [13] in more detail. The main difference which we already mentioned is that unlike in C-BDAM where the whole rendering pipeline is contained, our method only focuses on compression. Nevertheless, 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. At 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.

In this aspect, C-BDAM is similar to 1D wavelet, whereas our method is more similar to 2D wavelet, apart from the fact that we construct just one common high-pass output, not three. However, our high-pass is as large as four high-passes of 2D image wavelet, so the total count of high-pass samples is larger in our method. It is like that mainly because of the maximum-error bound constraint — it is without any doubt that when reconstructing a mip-map, we need to have a full individual control over each value inside it. We think that this control is possible only if the high-pass used to reconstruct this mip-map contains at least as many pixels as the reconstructed mip-map.

Another difference from classical image wavelet is that our high-pass information cannot be separated into three groups according to the feature they capture (eg. horizontal edges, vertical edges, diagonal edges), even though it could be done so quite simply — by using different prediction operators (see the previous Sec. 4.2). For example, to predict the heights of b pixels at the top-right (Fig. 4.2), we could use a simpler interpolation filter instead of P_b which would simply average the left and the right neighbour of the pixel. This way, the corresponding residuals would capture vertical edges. Similarly, we could predict the bottom-left b pixels as the average of their top and bottom neighbors and their residuals would capture the horizontal edges. Finally, to predict c pixels, we could average their top-left and bottom-right neighbors to get diagonal-edge residuals. However, the described prediction operators are most of the time not as accurate as the ones we used — they utilize less information for the prediction (two pixels instead of four), so it is very probable that the residuals would be overally a bit larger which would result in larger size of compressed data.

As we already said in Section 2.2, it cannot be really claimed that our method performs the lifting — during the first bottom-up 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-BDAM. 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 mip-maps from the coarsest LOD to the finest one in the subsequent top-down 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-up pass. This correction is done in another top-down 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-up 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-down 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-up pass. This is the reason why we use just an analogy of the update operator in the first bottom-up pass to produce the target mip-maps 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-down 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-down 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-up 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-up pass. Then we compute the final residuals directly with respect to the target values calculated in the first bottom-up 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 con-

form 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-down 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 mip-map, 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-up 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-down 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 implemented this method both as standalone and as a plugin for a real-time planet renderer as we already mentioned in the introduction. We did so in order to evaluate the usability of the method in practice on real data. With this method used by the renderer, we compressed and then real-time viewed the whole-Earth height data with 90 m span between height samples (SRTM[1] [11]). The uncompressed dataset is 29 GB large and each sample of it is a short (2 bytes). 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 260 GB. This hierarchy already contained transformed float samples separated into independent blocks. When we applied our compression method to them, we managed to compress these data down to 7 GB with the maximum deviation of the compression set to 5 m. This yields the compression ratio of 37:1.

For a comparison, C-BDAM [13] achieved the compression ratio of 64:1 on the same 90 m-resolution SRTM dataset with no redundancy with the maximum error bound set to 16 m. 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 redundancy 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. However, in their case, this was not 28 GB, but twice as much — 58 GB, as they converted each sample to a float before the compression. The final size of the compressed data prepared for rendering in C-BDAM was just 870 MB. For a comparison with the mentioned ratio of our method, in case of C-BDAM, it is fair to accept the ratio of 64:1 based on the float input, as our method had the float input, too, even though differently pre-processed.

The most accurate comparison of our method with C-BDAM we could perform was to compress the SRTM dataset alone with the maximum deviation set to 16 m. We did so, the final size of the compressed data varied depending on the dimension of the squares into which we partitioned the terrain (Tab. 6.1). At first, it decreased with growing dimension but then it started growing, probably because the overhead (the total area covered by the squares outside the dataset) overweighted the advantage of larger squares (greater redundancy exploitation). The smallest size we achieved was 1.03 GB for the dimension equal to 1024 which yield the compression ratio of 56:1 when compared to the size of the dataset converted to floats (58 GB), similarly as in the case of C-BDAM. This is quite a worse ratio, probably caused by the fact that C-BDAM does not compress the data by independent blocks, but it builds a hierarchy of residuals atop them, thanks to which it can profit of heights similarity on much coarser levels over much larger areas. On the other hand, our method has the advantage of modularity and independence on the target usage.

In Table 6.2, you can see the dependence of the speeds of compression and decompression on the dimension of the input square. In Table 6.3, you can see the dependence of the compression ratio on the maximum deviation, all mea-

[1] http://www2.jpl.nasa.gov/srtm/

Dimension	Size
128	1.31 GB
256	1.12 GB
512	1.05 GB
1024	1.03 GB
2048	1.04 GB
4096	1.07 GB
8192	1.11 GB

Table 6.1: The size of the compressed SRTM dataset with 16 m maximum deviation for various dimension of squares into which the dataset has been partitioned.

Dimension	Compression	Decompression
128	? ms	? ms
256	? ms	? ms
512	? ms	? ms
1024	? ms	? ms
2048	? ms	? ms
4096	? ms	? ms
8192	? ms	? ms

Table 6.2: The compression/decompression speeds for various dimensions of the input mipmap.

sured on a reference heightmap (Fig. 6.1). In Attachment A, we use this image to demonstrate how the result of the compression changes depending on the compression ratio. In Figure 6.2, you can see the output of compression applied to a synthetic spiky terrain with a fixed maximum deviation.

6.1 The compression artifacts

This section is dedicated to analyzing the artifacts visible in the terrain compressed by this method. We will give some grayscale example visualisations of

Deviation	Ratio
0	1:1.06
1	4.02:1
2	6.42:1
5	13.68:1
10	23.84:1
15	37.27:1
20	53.02:1

Table 6.3: The compression ratio for various maximum deviations — tested on a reference image which can be seen in Attachment A.

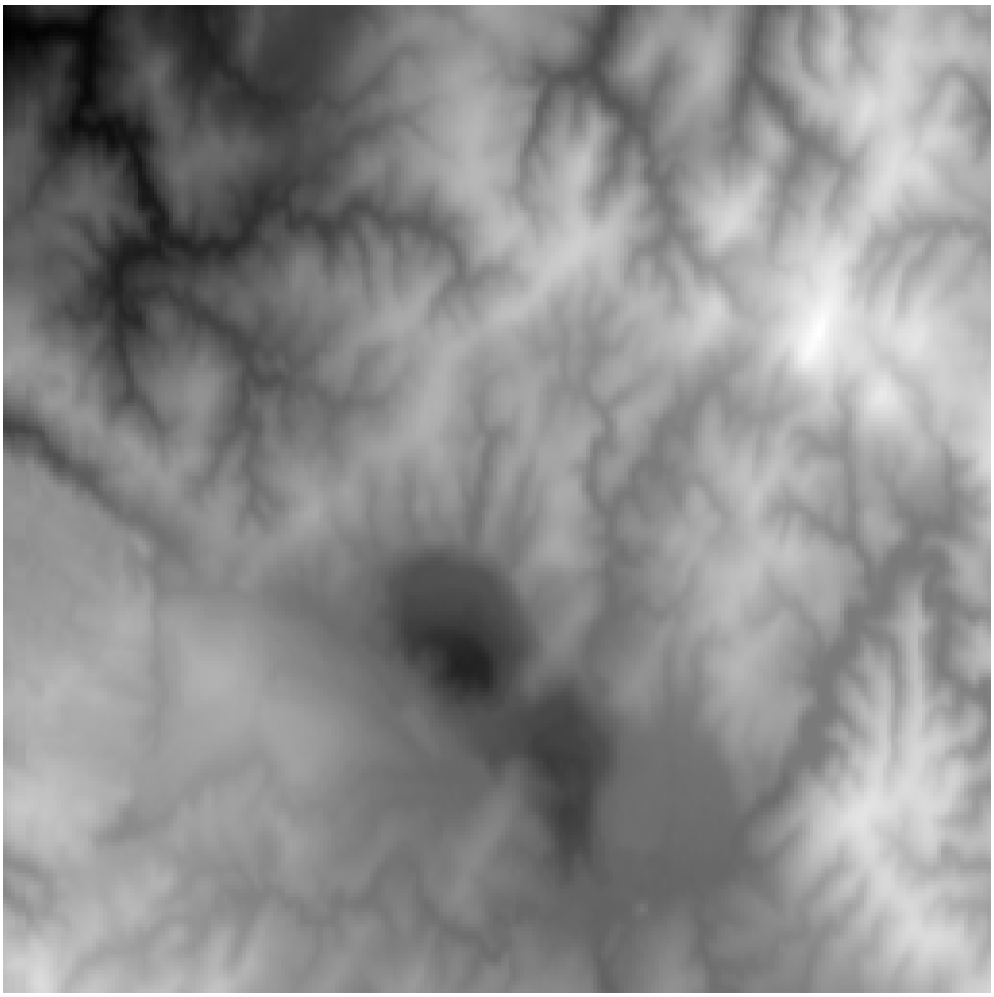

Figure 6.1: The reference heightmap on which we measured the compression ratios for various deviations and which we use in Attachment A to demonstrate the outputs of compression for various maximum deviation settings.

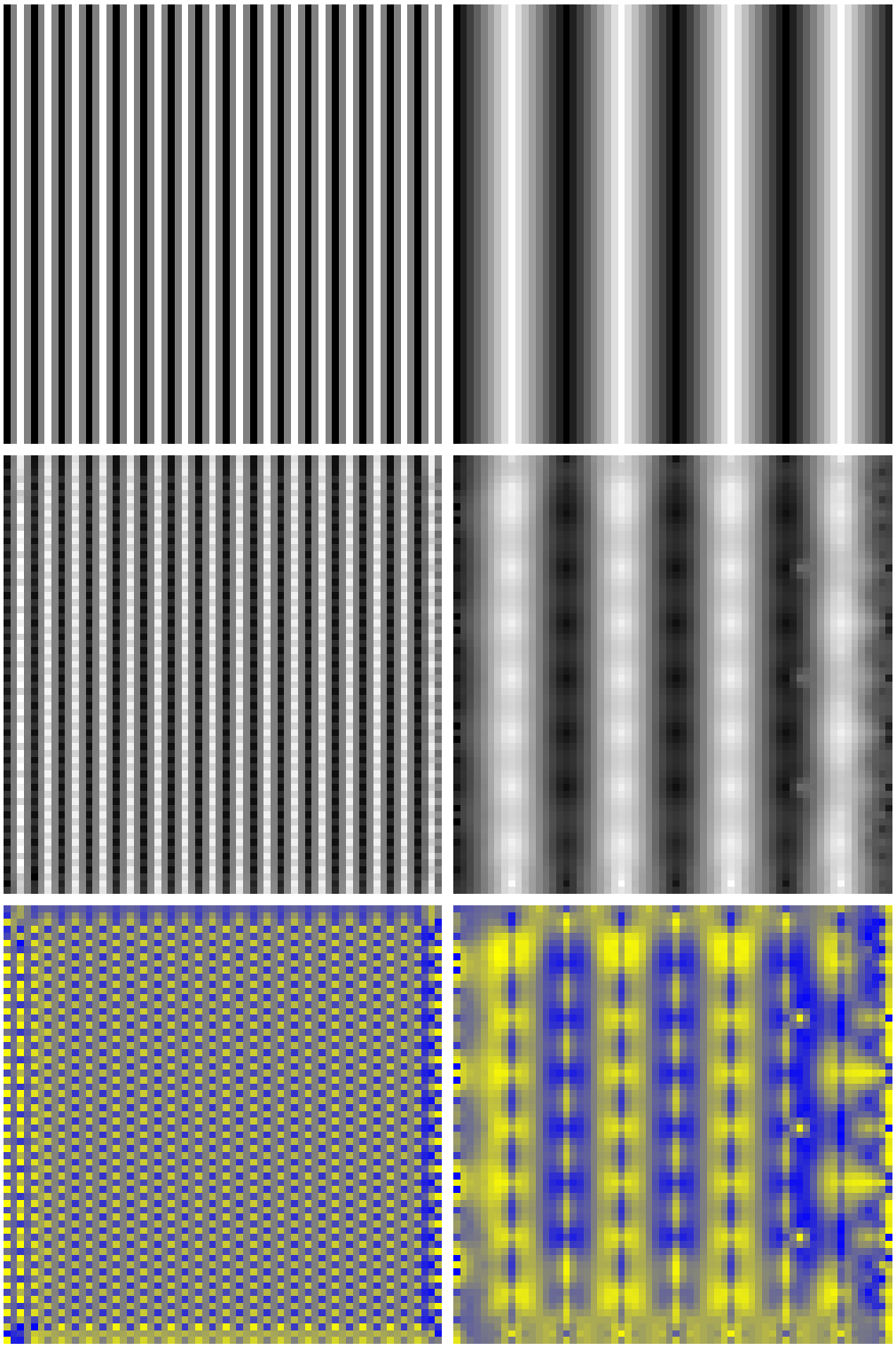

Figure 6.2: Two synthetic test images of size 64x64, each one containing spiky terrain with the heights ranging from -16 to 16. On the left, the longitude of spikes is 4, on the right, it is 16. From the top to the bottom — the original, compressed with the maximum deviation of 5, the difference between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

how they look, explain why they usually occur and discuss various ways how they can be reduced.

Because this method is designed with the maximum height error as the only measurement of quality of the compressed data, we cannot guarantee that the resulting terrain will look smooth and will not contain any sharp peaks within the maximum-error bound. Due to the character of the method, such artifacts indeed occur. They cannot be completely avoided but we can reduce them by a suitable choice of the prediction filters, setting a lower maximum deviation or by applying the median filter to the output of the decompression if it is not a problem for any reason (eg. decreased performance or increased maximum deviation).

As we already explained in Section 4.2, we use Neville interpolating filter of order 2 for all pixels, both near the mip-map edges and in its interior. We also mentioned that C-BDAM only uses the order 2 filter for the pixels near the borders and the order 4 filter (Fig. 4.5) for the rest of them. We 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 valleys (Fig. 4.6) — but worsened the quality of compression by producing more significant artifacts. The most probable cause of this is that the order 4 filter is more sensitive to terrain changes — it scans larger area to capture the trends in terrain. We have found some general situations in which the artifacts are particularly disturbing — near smooth terrain's borders (Fig. 6.3) and near sharp terrain transitions (Fig. 6.4).

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. 6.5).

This difference might then cause the difference in 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. 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.

We also tried to apply progressive quantization to the residuals in order to reduce the artifacts in the reconstructed terrain. In Sec. 2.1, we mentioned that progressive quantization degrades the coarser residuals less than the finer (more detailed) ones. In our context, it would mean that the maximum error of $\boldsymbol{L}_i^{\bullet}$

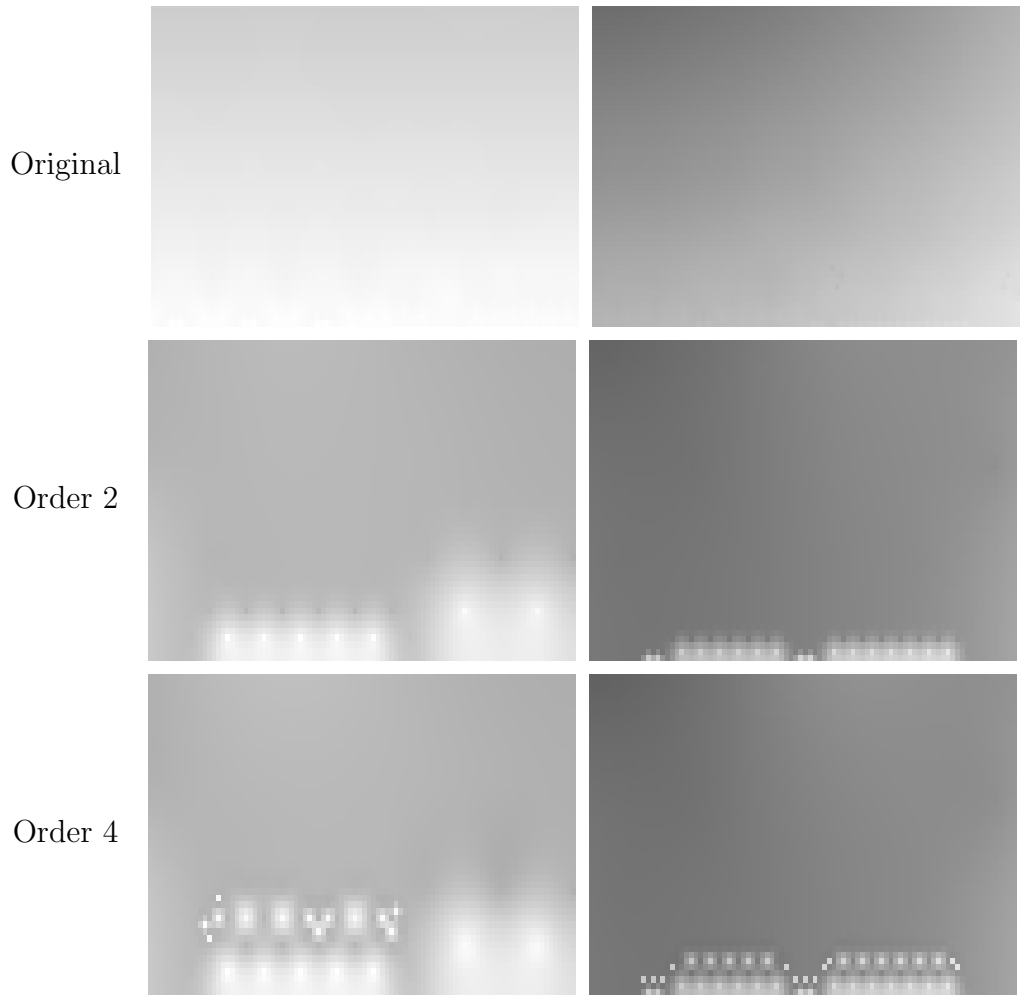

Figure 6.3: Two synthetic examples of the difference between artifacts caused by order 2 and order 4 filters near smooth terrain's border — in the first row there are the target heightmaps, in the second, there are the same heightmaps compressed using the order 2 filter, in the third row, the heightmaps compressed with the order 4 filter.

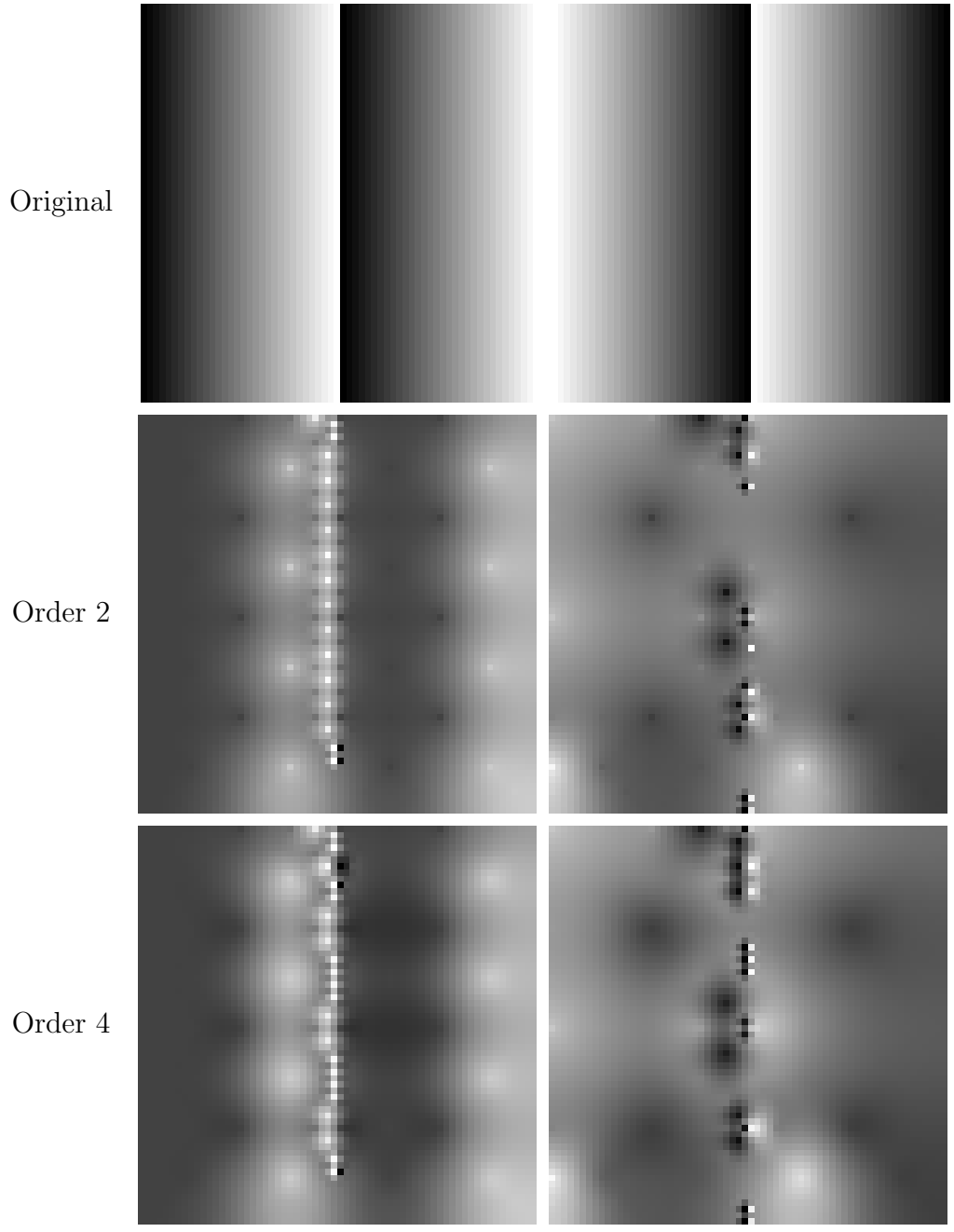

Figure 6.4: Two synthetic examples of the difference between artifacts caused by order 2 and order 4 filters near a sharp terrain change — in the first row there are the target heightmaps, in the second row, the same heightmaps compressed using the order 2 filter, in the third row, the heightmaps compressed 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.

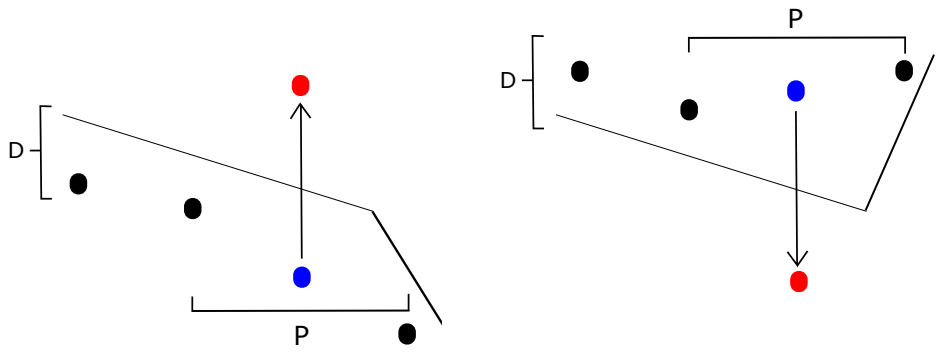

Figure 6.5: Two illustrations of how artifacts can occur near sharp terrain changes — the black dots stand for the predictions which are still within the maximum-error bound D from the target terrain, the blue dots represent the predictions which are just slightly further from the terrain, because their 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.

(the reconstructed data) equal to the quantization interval of $\boldsymbol{E}_i^\bullet$ (the quantized residuals) would grow with i up to D at the finest level. To compute D_i — the maximum error of $\boldsymbol{L}_i^\bullet$, we used the following equation: $D_i = D * (\frac{i}{n})^k$, where k, a natural number, is a parameter of this approach. The larger it is, the less degraded are the coarser levels, but the maximum deviation of the finer one is still D.

However, more careful decimation of coarser high-pass information only helps reduce large-scale artifacts which is not the case of our artifacts — they are mainly caused by differences between neighboring pixels which happens due to the fact that the residuals are cropped to respect the maximum deviation. Indeed, our experiments with progressive quantization did not help reduce the artifacts at all — progressive quantization does not tackle the problem described in the previous paragraph. Quite to the contrary, it makes the artifacts in the finest level more regularly distributed which is a bit more disturbing. This is probably because there is less variance in deviation propagated from the coarser levels which exposes the regularity of the prediction filters inside the finest level more (Fig. 6.6). Additionally, the progressive quantization increased the size of data.

Finally, we found only one way how to smoothen the artifacts — to post-process the decompressed data. This post-processing does not happen during the compression, but after the decompression. In the image processing, the median filter [14] is very suitable for "salt and pepper" noise reduction, whereas our artifacts often resemble this kind of noise (the bottom of Fig. **??**). This filter substitutes the height of each pixel with the median of its height together with the heights of all its neighbors. The size of the neighborhood can vary, but we used the simplest alternative — 1, so that the median of nine values is calculated (Fig. 6.7). However, generally, this filter can increase the maximum deviation of the decompressed data by any amount. For example, imagine a very high pixel surrounded by very low pixels. In this extreme case, the high pixel will be aligned

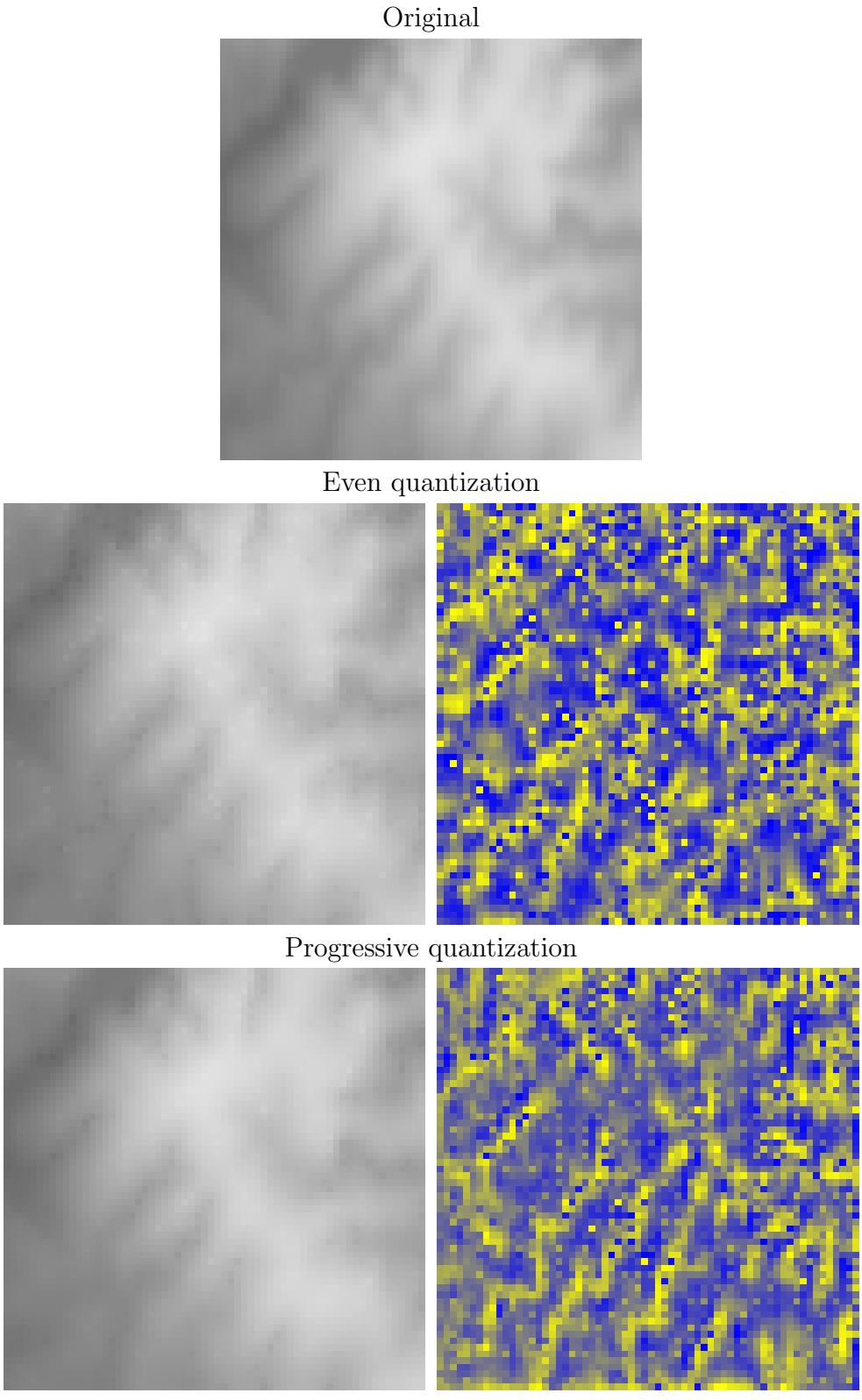

Figure 6.6: Comparison of artifacts at the finest mip-map level produced by the even quantization (as described in Chap. 4) and those produced by the progressive quantization with k set to 5, both with 10 m max. error at the finest level. On the top there is the original, in the remaining two rows there are the compressed versions on the left and their differences with the original on the right, in which the yellow color means 9.5m and the blue color means -9.5m.

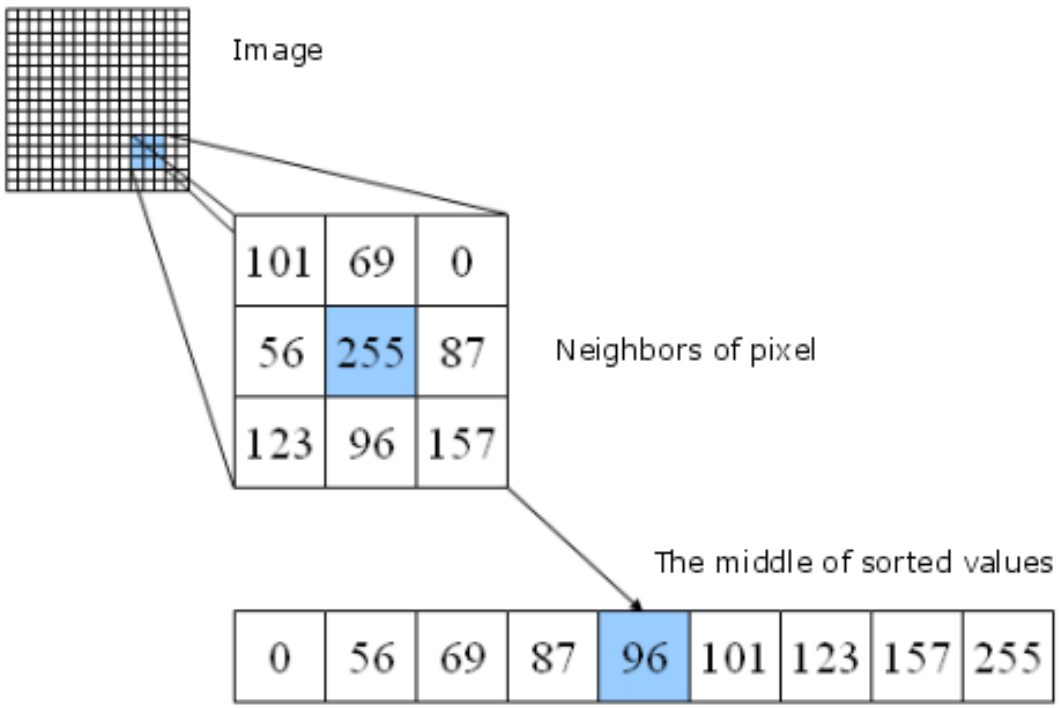

Figure 6.7: An example of basic median filtering. The value of the middle pixel (255) is substituted by the median of it together with all neighboring values (96). **Source:** Electronics and Computer Science University of Southampton, Computer Vision Demonstration Website [21] (edited)

to the height of one of the lower pixels. The higher the pixel was before, the larger the deviation caused by this filter will be.

To tackle this, we introduced a simple modification to the median filter. Before its application, we introduce another user parameter called D_M. It specifies the maximum additional deviation which the filtering by median can introduce. It behaves basically like a hard threshold applied to the output of the median filter. Once the filter has computed the new height of a certain pixel, we move the old height towards it by at most D_M. Even with this constraint did we manage to significantly reduce the artifacts (Fig. 6.8). However, filtering by median is quite an expensive operation and as we mentioned, it is applied after the decompression. Because the decompression is performed in the real-time when the method is used inside a renderer, and so it should be as fast as possible, we did not apply the filtering in that case.

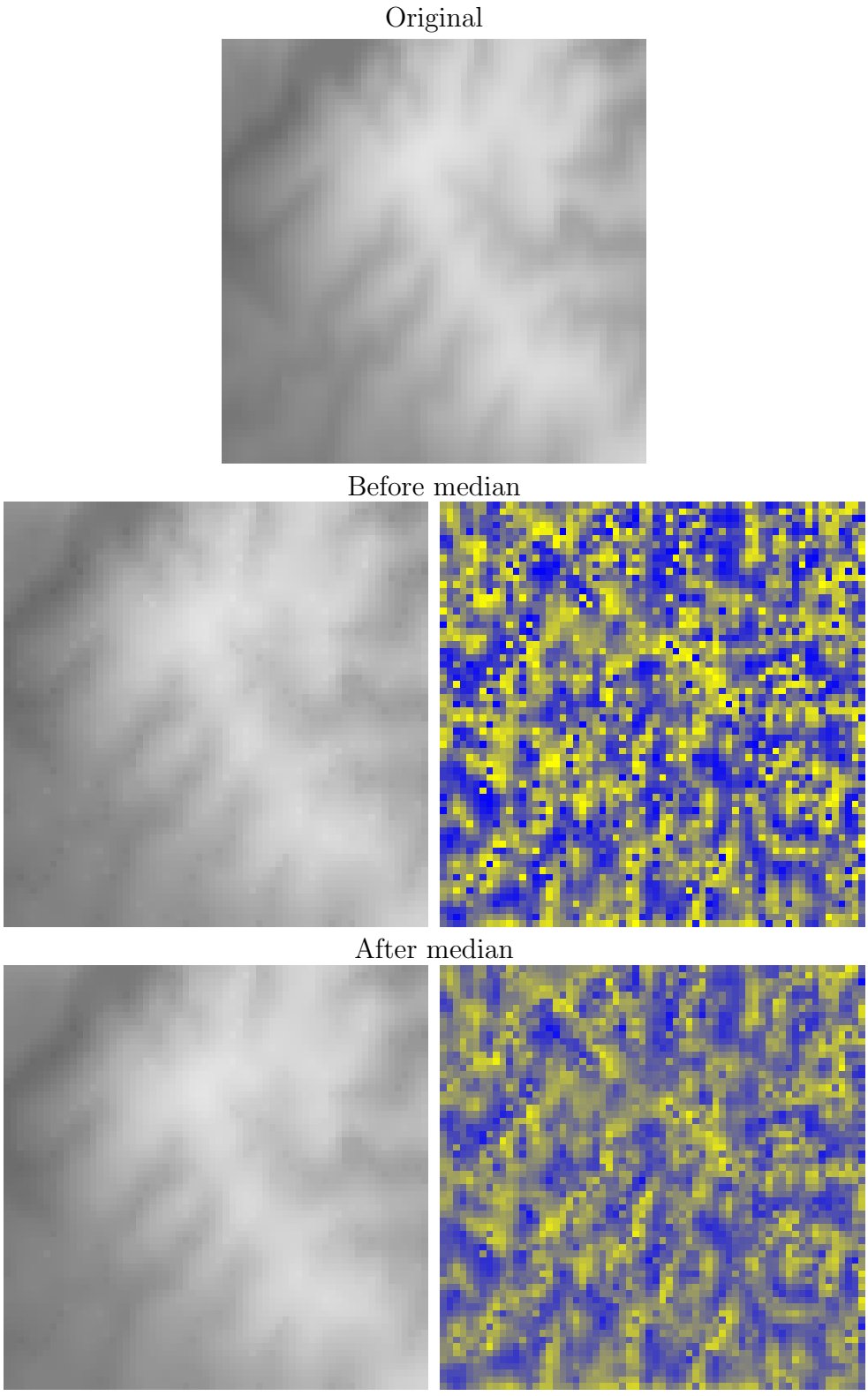

Figure 6.8: Comparison of artifacts at the finest mip-map level before and after the median filter applied. The maximum deviation of compression D is 10 m and the maximum deviation of the filter D_M is additional 5 m. On the top there is the original, in the remaining two rows there are the results on the left and their differences with the original on the right. In the upper difference image, the yellow color means 9.5 m and the blue color means -9.5 m, whereas in the lower one, the yellow means 14.5 m and the blue means -14.5 m.

Conclusion

The aim of this thesis was to design and implement a heightmap compression algorithm which could be, among other things, plugged into an existing real-time planet renderer. This algorithm should be able to compress a regular square block of height samples and progressively decompress it in the real-time, from the smallest mip-map to the largest one. Apart from this, the algorithm should not in any way interfere with the rendering pipeline of the application. After summarizing the available literature, we decided to write our own method inspired by the most promising compression approach used inside C-BDAM [13], the method for rendering compressed flat terrain. In our method, the decompression of one 256x256 square which is the size of square used in the application takes only about 1ms. The compression of it takes about 30 ms which is still bearable, as it does not have to be performed in the real time. The compression ratios of our method are comparable to the ones of C-BDAM which provides the best compression ratios among the studied methods.

We also investigated the possibility to perform the decompression on the GPU in order to spare RAM. However, the main priority was still to minimize the size of the compressed data which is why we followed the compression ratio in the first place. With respect to it, we designed our method and only after this we analyzed whether and how its decompression could be put onto the GPU. We saw two places where the decompression could already switch to the GPU:

1. before the decompression of residuals
2. right after the decompression of residuals

Both possibilities imply that the reconstruction of data (the predictions followed by the adding of residuals) would be performed on the GPU, whereas the first one implies that in addition, the lossless decoding of residuals would be performed there, too. Keep in mind that once the decompression has switched to the GPU, it cannot return from there. Otherwise, no RAM would be spared. The point of this switch is that we send the compressed data to the GPU and decompress it just there, so that some RAM is spared. If we subsequently fetched the decompressed data back from the GPU in any point, we would not save any RAM.

The first possibility would imply that the real-time decompression, as performed by Zlib, has to be put onto the GPU. Alternatively, a different lossless compression method could be picked, but Zlib achieves the best compression ratios from among the libraries we tested. As we already described in Sec. 2.3, Zlib contains both LZ77 and Huffman coding. There are works dedicated to the GPU implementation of LZ77 [15] and Huffman [2, 6], but they all require CUDA which is not used in many real-time renderers, not even in the one inside which we have tested our method. Moreover, these implementations do not take into account sharing of resources, so there is a possibility of a significant negative rendering performance hit.

The second possibility means less spared RAM, but still some, because, as we described in Sec. 4.2, before the residuals are losslessly compressed, they are packed — divided by their quantization interval, shifted, so they are all positive

and then skewed to the number of bits of the largest value. So, right after their decompression by Zlib, they are still a bit compressed. Theoretically, their unpacking could be done on the GPU. Then, the subsequent reconstruction of a larger mip-map from the smaller one would have to be performed on the GPU, too. The ability to do this surely depends on the implementation of the renderer. This is where we encountered a problem.

The renderer uses just one vertex buffer per each square. Then it sends a mip-map of the square to the GPU as a texture and according to the heights in this texture, it performs displacement of vertices of the buffer. Thus, if the unpacking of residuals and the progressive mip-maps reconstruction were all to be performed on the GPU, we would no longer send any textures to the GPU, but we would just send the unpacked residuals there which is the only remaining way how to spare RAM in our case. The point is that all the recostruction steps would then have to be performed on the GPU, including creating an array and traversing it. The only way how this can be done on the GPU is with the help of CUDA, OpenCL or compute shaders. However, the implementation of the renderer uses neither CUDA, nor OpenCL and during the time of designing and writing the method, it was written in DirectX9 which does not support compute shaders.

The second alternative we considered was to directly utilize the information in vertices — their location. To determine the height of a certain vertex, we would have to average the suitably backward transformed heights of its neighbors (Sec. 4.2). This would have to be done in vertex shader. However, this is impossible, as in vertex shader, we cannot access neighboring vertices of the current vertex. The alternative to it could be to use a geometry or tessellation shader. However, this would be very complicated and would require serious modifications to the renderer's pipeline.

To conclude, the possibility to switch any part of the real-time decompression onto the GPU highly depends on the implementation of the renderer which uses the method. The limitations of our testing renderer made it very difficult to switch any part of it onto the GPU. However, just recently has the renderer switched to DirectX11 already supporting compute shaders which opens the door for all the alternatives impossible before. On the other hand, the GPU memory is currently the main bottleneck, so sacrifying it to spare RAM could actually decrease the overal performance.

Finally, after a consultation with people from the practice — the developers of the testing renderer — we decided to give up the practical implementation on the GPU. However, it still remains an interesting possibility of future work for interested researchers.

Bibliography

[1] C. Andújar, A. Chica, M. A. Vico, S. Moya, and P. Brunet. Inexpensive reconstruction and rendering of realistic roadside landscapes. *Computer Graphics Forum*, 33(6):101–117, 2014.

[2] C. A. Angulo, C. D. Hernández, G. RincĂłn, C. A. Boada, J. Castillo, and C. A. Fajardo. Accelerating huffman decoding of seismic data on gpus. In *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, pages 1–6, Sept 2015.

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

[4] 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.

[5] 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.

[6] Robert Louis Cloud, Matthew L. Curry, H. Lee Ward, Anthony Skjellum, and Purushotham Bangalore. Accelerating lossless data compression with gpus. *CoRR*, abs/1107.1525, 2011.

[7] Alessio Damato. 2-level wavelet transform-lichtenstein, 2007.

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

[9] Christian Dick, Jens Schneider, and Rüdiger Westermann. Efficient geometry compression for gpu-based decoding in realtime terrain rendering. *Comput. Graph. Forum*, 28(1):67–83, 2009.

[10] Dorde M. Durdevic and Igor I. Tartalja. Hfpac: GPU friendly height field parallel compression. *GeoInformatica*, 17(1):207–233, 2013.

[11] Tom G. Farr, Paul A. Rosen, Edward Caro, Robert Crippen, Riley Duren, Scott Hensley, Michael Kobrick, Mimi Paller, Ernesto Rodriguez, Ladislav Roth, David Seal, Scott Shaffer, Joanne Shimada, Jeffrey Umland, Marian Werner, Michael Oskin, Douglas Burbank, and Douglas Alsdorf. The shuttle radar topography mission. *Reviews of Geophysics*, 45(2):n/a–n/a, 2007. RG2004.

[12] Thomas Gerstner. Multiresolution compression and visualization of global topographic data. *GeoInformatica*, 7(1):7–32, 2003.

[13] 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.

[14] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[15] Petr HLAVINKA. Kompresní algoritmus lz77 na gpu [online], 2012 [cit. 2016-07-23].

[16] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.

[17] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer Graphics - Principles and Practice, 3rd Edition*. Addison-Wesley, 2014.

[18] Peter Lindstrom and Jonathan D. Cohen. On-the-fly decompression and rendering of multiresolution terrain. In *Proceedings of the 2010 Symposium on Interactive 3D Graphics, SI3D 2010, February 19-21, 2010, Washington, DC, USA*, pages 65–73, 2010.

[19] David J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[20] John Miano. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[21] University of Southampton. Median filtering, 2005.

[22] 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.

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

[24] Dietmar Saupe and Raouf Hamzaoui. A review of the fractal image compression literature. *SIGGRAPH Comput. Graph.*, 28(4):268–276, November 1994.

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

[26] Thomas Wiegand and Heiko Schwarz. Source coding: Part I of fundamentals of source and video coding. *Foundations and Trends in Signal Processing*, 4(1-2):1–222, 2010.

[27] 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.

[28] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theor.*, 23(3):337–343, September 2006.

List of Tables

List of Abbreviations

Attachments

Attachments

A. Results on a reference image

This attachment demonstrates on a reference image how the output of compression varies depending on the chosen maximum deviation. On the top, there will be always the original image, in the center, the compressed one and on the bottom, their difference in blue-to-yellow scale, with blue being the minimum value and yellow being the maximum value.

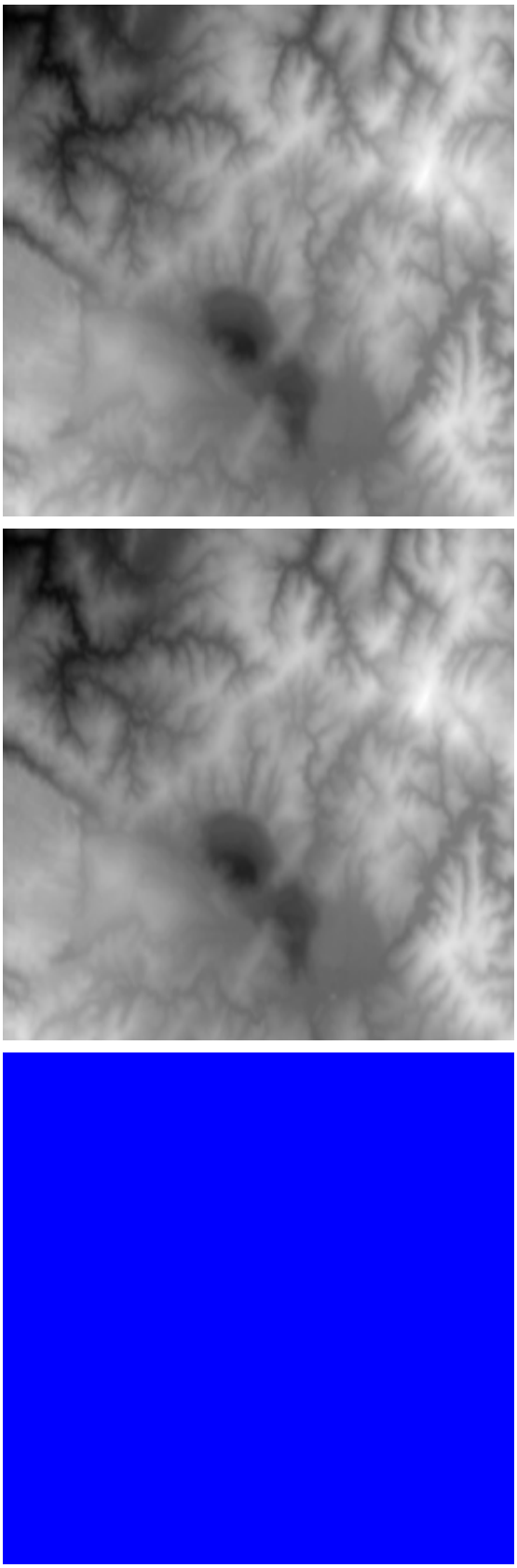

Figure A.1: Maximum deviation 0.

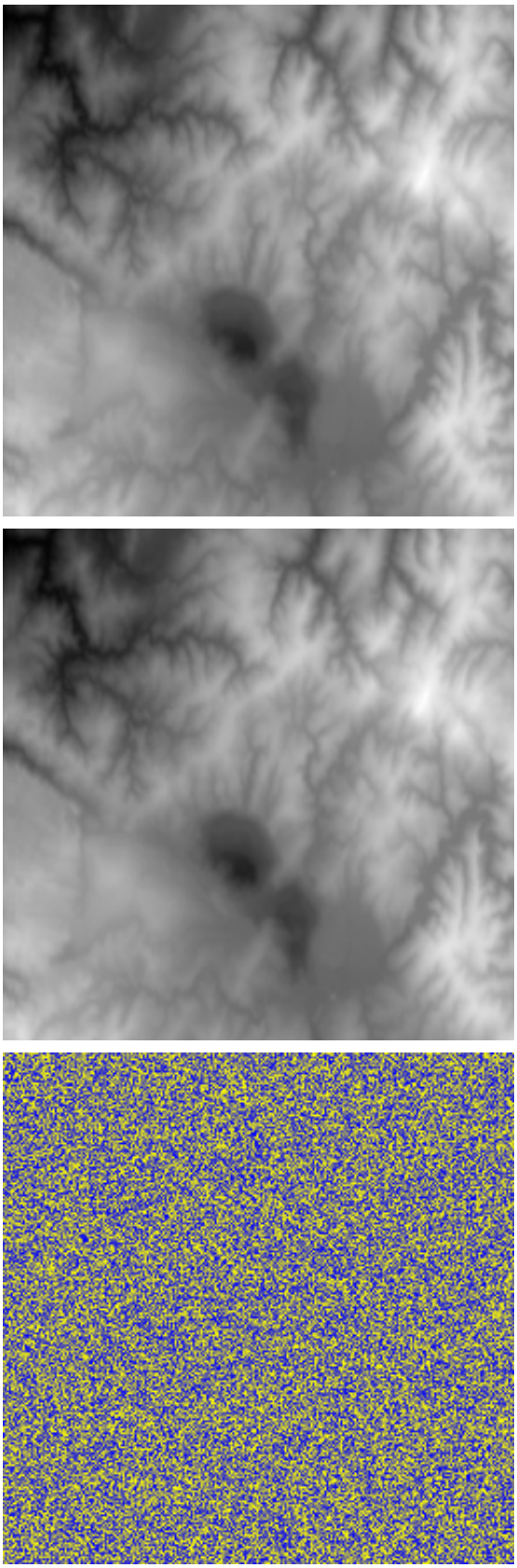

Figure A.2: Maximum deviation 1.

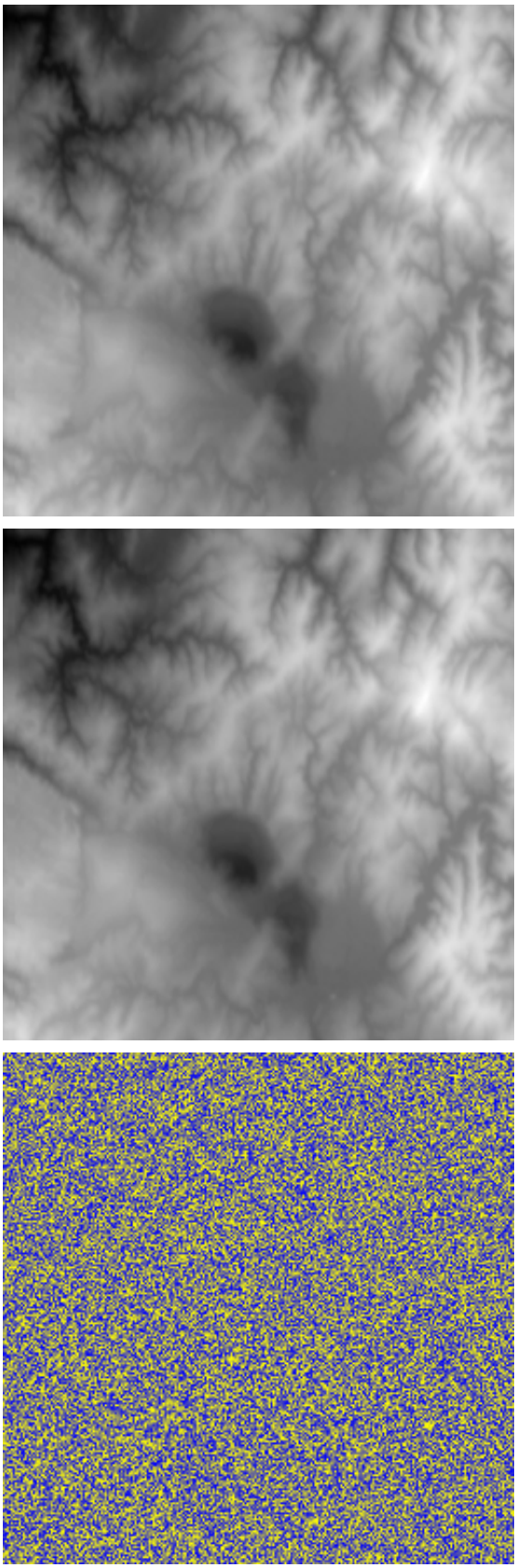

Figure A.3: Maximum deviation 2.

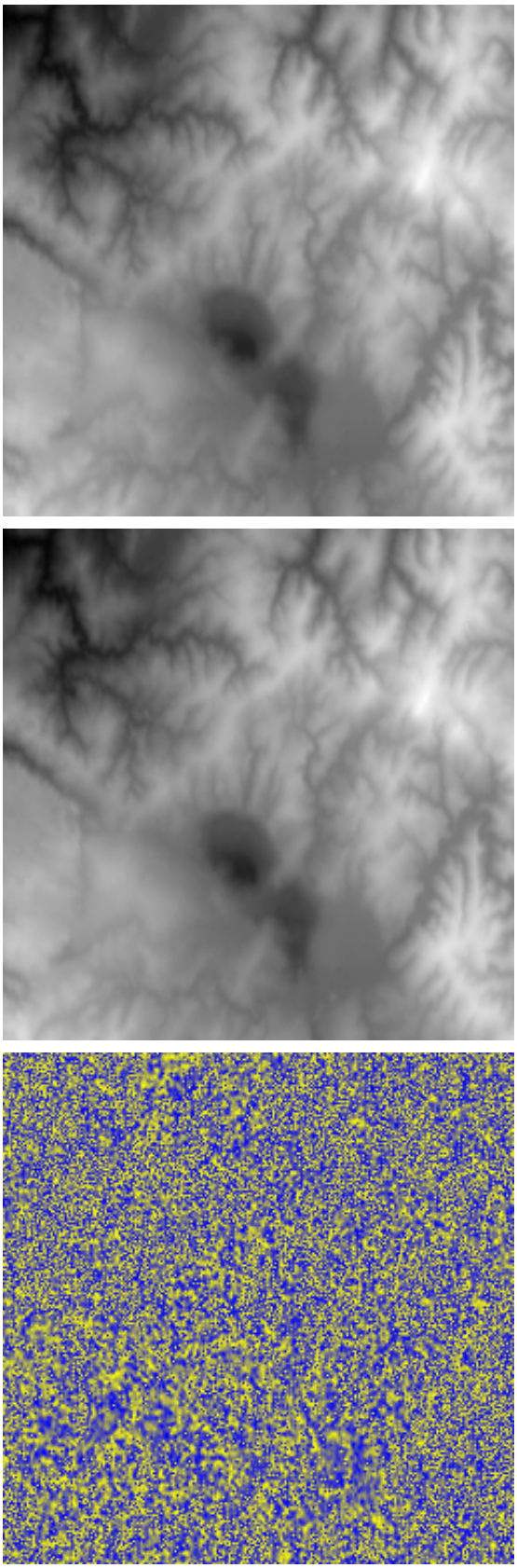

Figure A.4: Maximum deviation 5.

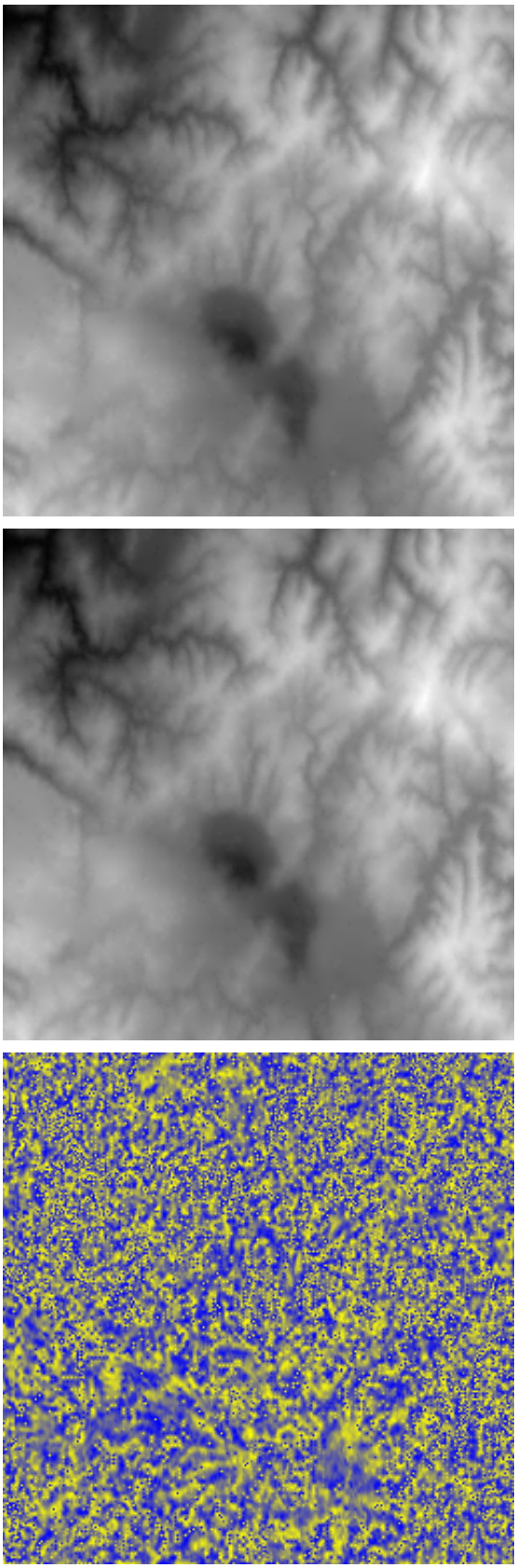

Figure A.5: Maximum deviation 10.

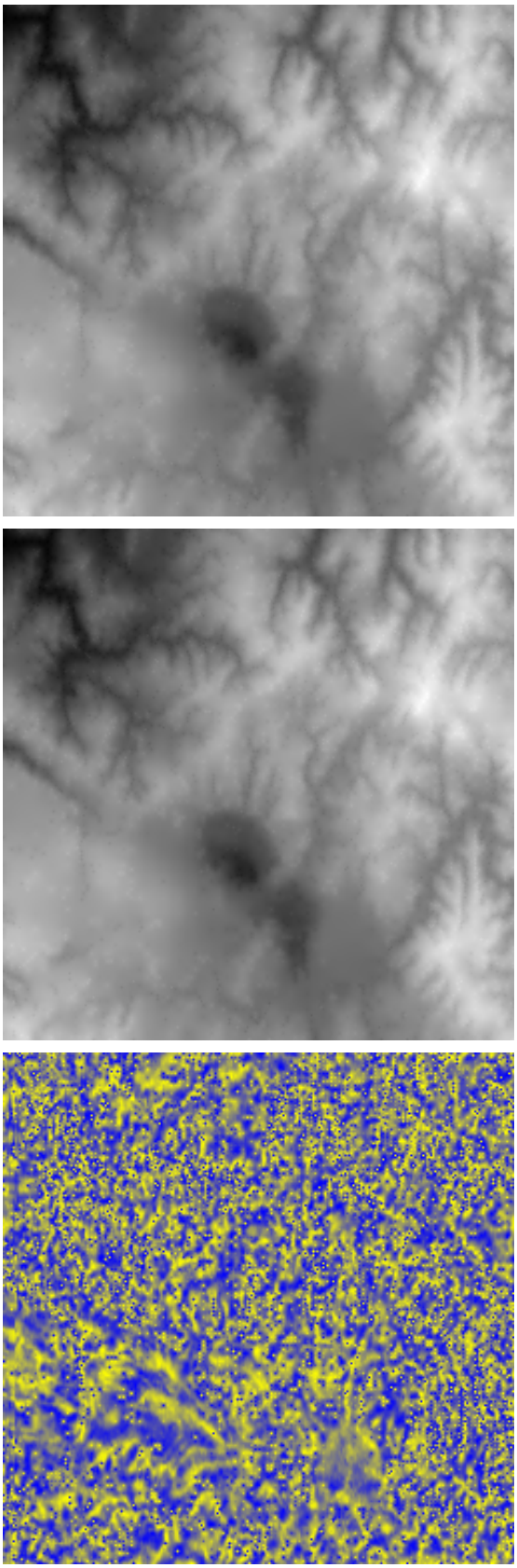

Figure A.6: Maximum deviation 15.

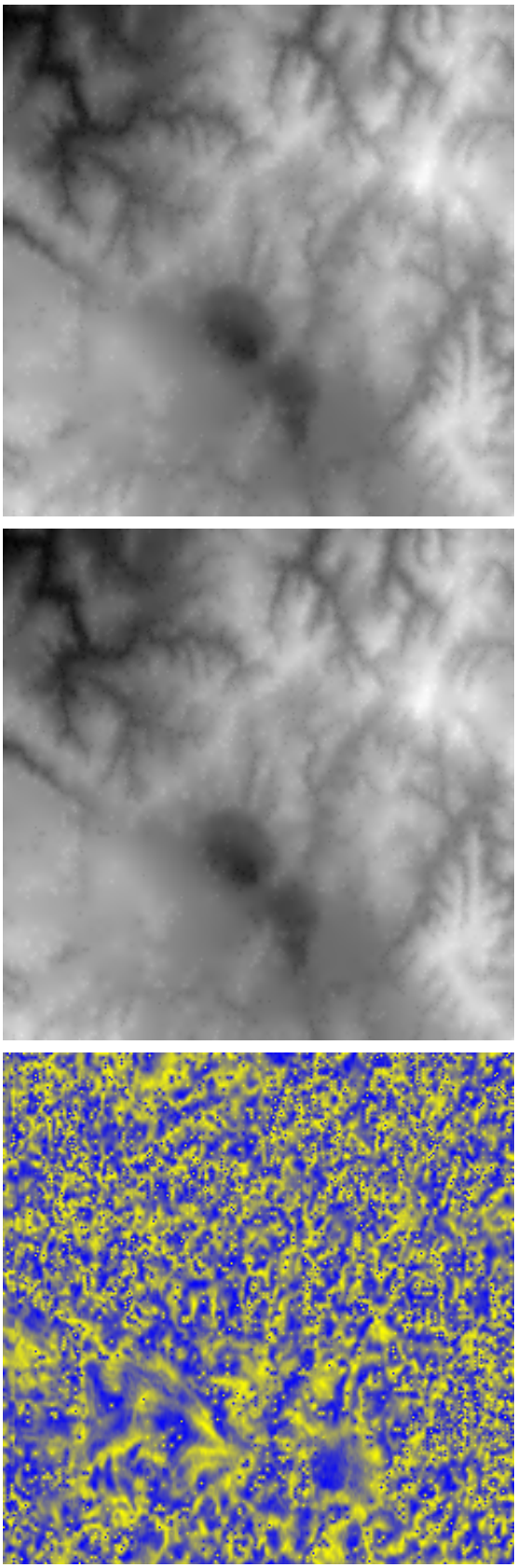

Figure A.7: Maximum deviation 20.