MASTER'S THESIS 2020

Compression of Surface Data on Dynamic Topologies

Oguz Taskin

ISSN 1650-2884
LU-CS-EX: 2020-26

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-26

Compression of Surface Data on Dynamic Topologies

Oguz Taskin

Compression of Surface Data on Dynamic Topologies

Oguz Taskin
oguz.taskin@icloud.com

June 12, 2019

Master's thesis work carried out at EA Digital Illusions CE AB.

Supervisor: Jan Schmid, jan.schmid@dice.se
Examiner: Michael Doggett, michael.doggett@cs.lth.se

Abstract

A common method for defining high-frequency detail on a 3D model is via the use of texture mapping. Problems with this technique such as discontinuities and seams appearing in the textured model has lead to the development of different solutions. Mesh Colors, proposed by C.Yuksel (2010), presents a solution where surface data is directly associated with the mesh in a topologically independent way, removing the need for a map and avoiding problems inherent to the use of them.

The purpose of this study is to develop an image codec for mesh colors data, data directly associated with the mesh. It focuses on compression of color data but supports compression of other textures as well. The result is a JPEG-like image codec with comparable compression ratios and image quality. Because of the similarities to the JPEG standard, arising compression artifacts resemble those that appear in JPEG images.

Keywords: Mesh Colors, compression, discrete cosine transform, JPEG, texture

Contents

Chapter 1
Introduction

1.1 Background

The traditional way of adding color detail to a 3D model is by using texture mapping, a technique that applies an image to a surface by mapping a 2D texture onto a 3D surface [2]. This can make a simple surface get the appearance of something more complex. Using a map for the 3D surface onto a 2D texture has, however, inherent problems such as mapping-discontinuities and limitations to editing of the 3D model after texturing.

Advantages of having a 2D texture as the source for detail are, to name a few, being able to use existing image codecs to store the image in a compressed format, and being supported by current graphics hardware.

Mesh colors [8] takes a different approach to providing detail to a geometric surface and associates surface data directly with the mesh geometry. Each texel (i.e. data point in the texture) has a hexagonal area of influence and is ordered in a hexagonal lattice. Because of differences in data representation, compression of mesh colors data using existing image codecs is not possible.

1.1.1 Related Work

The Mesh Colors technique is a different approach to providing detail to a geometric surface by associating surface data directly with the mesh geometry. It can be seen as an extension of vertex coloring, where color values are not only placed on vertices but also on faces and edges. Color values are placed on grid nodes of a triangular grid, of which the resolution can be set, resulting in each texel having a hexagonal area of influence (Figure 1.1).

Developed by the Joint Pictures Expert Group, JPEG is today one of the most widely used image compression standards. It is a lossy compression technique belonging to a family of techniques called "transform compression" where transformation of data into an-

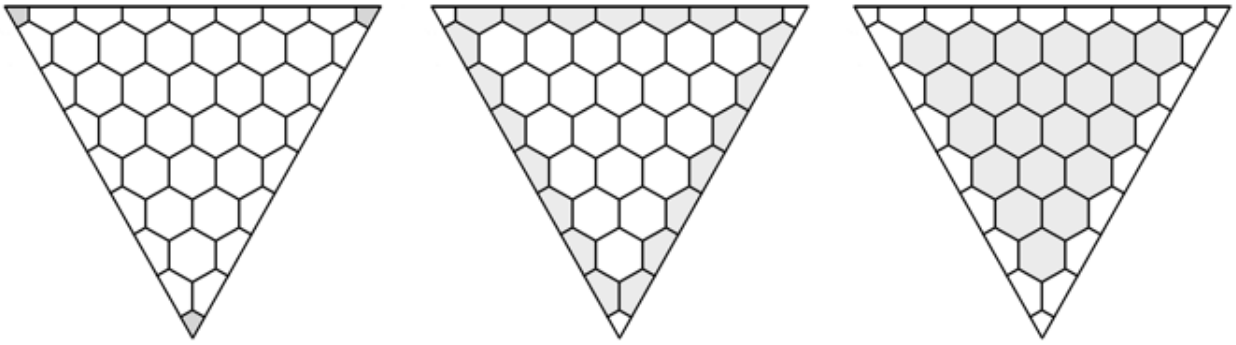

Figure 1.1: Triangle with resolution 8 with colored vertex elements, edge elements and face elements (left-to-right).

other domain is an essential part of the codec [1][6]. The baseline JPEG codec transforms data into the frequency domain using the discrete cosine transform (DCT), a lossless operation, that returns coefficients representing different frequency functions. Because the human visual system is less sensitive to high-frequency functions, corresponding transform coefficients can be represented with less accuracy without the image losing much of its perceptual quality.

The different steps of JPEG compression are illustrated by Figure 1.2. Decompression is similar and applies the steps in reverse order where the inverse operation is applied in each step. These are described in more detail in Chapter 2.

1.2 Problem Definition

In the mesh colors framework, surface data is stored in a linear array and ordered in a way that removes the possibility to directly apply existing image codecs to achieve compression. Furthermore, the different structure and layout of data requires a custom image codec that takes into account neighborhood conditions which differ between mesh colors texels and regular texels. While regular texels have the shape of a square and are ordered in a cartesian grid, mesh colors texels will have a hexagonal shape and be ordered in a hexagonal grid. In addition to this, the custom codec will also need to support compression of mesh colors data where data associated with different triangles can have different resolutions. Additionally, in video-game development, the use of MIP-levels (lower resolution versions of the original texture) opens up possibilities traditional codecs aren't developed to take advantage of.

1.3 Purpose

Texturing of 3D models is an important step for achieving high fidelity renders. For increasing resolutions of textures, compression becomes a necessity to store and transfer data efficiently. This becomes even more important when data is stored on physical mediums where storage space is limited. The aim of this thesis is to implement a Mesh Colors

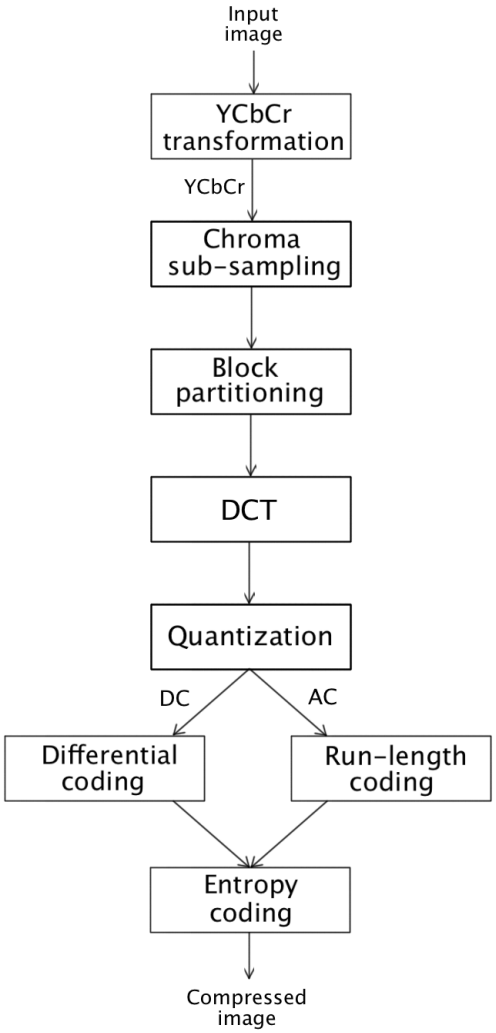

Figure 1.2: Baseline JPEG compression steps.

Codec (MCC), a JPEG-like image codec, for mesh colors data to enable effective storage and transmission.

1.4 Scope

The presented solution is mesh colors data compatible and builds off the baseline JPEG. It does not support options of, or improvements in later iterations of the technique. An entropy coder was not developed for the developed codec, rather, an in-house arithmetic coder was used. The focus of the codec lies on efficiency of compression, both in terms of quality and compression rate, rather than computational efficiency.

Chapter 2
Approach

In the mesh colors framework, surface data is associated directly with the mesh geometry. A main feature of this technique is to enable having varying levels of detail for individual faces and edges. This is accomplished by assigning each face a resolution which determines the number of data elements residing within it. The number of data elements P assigned to faces and edges for a resolution of r is given by equation (2.1) and (2.2) respectively. Vertices will always be represented with a single data element. Face data elements are considered to make up a two-dimensional image with a triangular shape made up of hexagonal texels. They are treated as hexagonal texels because each data element will have a hexagonal area of influence. Edge data elements are treated as a one-dimensional signal also with hexagonal texels. The developed MCC processes edge data elements and face data elements separately, but uses results from compression/decompression of edge data elements when processing face data elements to achieve a better compression ratio.

$$P_{face}(r) = \sum_{k=1}^{r-2} k = \frac{(r-2)+1}{2}(r-2) \tag{2.1}$$

$$P_{edge}(r) = r - 1 \tag{2.2}$$

2.1 YUV Transformation

The first step specified by the JPEG standard is the transformation of data elements in color space. Data in RGB format is transformed to Y'C_bC_r format with equation (2.3) where the Y' component represents brightness and components C_b and C_r represent the chrominance of the color element. The chroma components (C_b and C_r) have less effect on the perceptual quality of the image than the luma component (Y'). This transformation is applied to enable greater compression ratios by compressing less significant components more heavily. A second outcome of this transformation is statistical decorrelation, which

results in the entropy coder yielding a greater compression ratio. Since this procedure is applied to individual texels the shape or structure of texels do not matter. The color transformation is therefore applied to all data elements individually.

During decompression, the inverse transformation is applied to each data element in order to transform color data from Y'C_bC_r space to RGB space. The inverse equation is given by equation (2.4).

$$\begin{aligned} Y' &= 0.299R + 0.587G + 0.114B \\ C_\mathrm{b} &= (B - Y)0.5643 + 128 \\ C_\mathrm{r} &= (R - Y)0.7132 + 128 \end{aligned} \tag{2.3}$$

$$\begin{aligned} R &= Y' + 1.402(C_\mathrm{r} - 128) \\ G &= Y' - 0.714(C_\mathrm{r} - 128) - 0.334(C_\mathrm{b} - 128) \\ B &= Y' + 1.772(C_\mathrm{b} - 128) \end{aligned} \tag{2.4}$$

2.2 MIP-Maps

MIP-mapping is a texture minification technique used to reduce aliasing artifacts that occur when textured 3D objects move away from the camera. These artifacts can be avoided by using textures that are scaled down versions (in terms of resolution) when the object passes certain thresholds for distance. A series of down-sampled versions of the texture is therefore pre-calculated and stored. Each version of the texture is referred to as MIP-level x, where higher values for x correspond to lower resolution versions of the texture. This technique is supported by mesh colors for faces and edges that have a resolution equal to some power of two. MIP levels are compressed and decompressed independently except for when chroma sub-sampling is enabled.

2.3 Chroma Sub-Sampling

The JPEG standard supports chroma sub-sampling, which is the process of storing down-sampled versions of the chroma components C_b and C_r. This has minimal effect on the perceptual quality of the image on which accurate representation of the luma component has a greater impact. The compression ratio achieved by chroma sub-sampling alone depends on the coefficient of decimation. For a decimation coefficient of c, c-by-c neighboring data values are replaced by their mean value and the resulting compression ratio by this oprration alone can be calculated as

$$compression\_ratio = \frac{3}{1 + \frac{1}{c^2} + \frac{1}{c^2}} = \frac{3c^2}{c^2 + 2}.$$

For example, a decimation coefficient of 2 results in a compression ratio of 2, achieved by the chroma sub-sampling alone. During decompression, each c-by-c block of neighboring elements (in the original size) will be assigned their mean value.

The developed MCC supports chroma sub-sampling for decimation coefficients equal to powers of two. Because higher MIP-levels are by definition down-sampled versions of the base texture, MIP-level 0, chroma components do not have to be decimated nor stored as they are provided by the neighboring MIP-level. This is one of the reasons why only decimation coefficients that are a power of two are supported.

During decompression, the chroma components of MIP-level 0 are recreated using the chroma components of MIP-level 1. Each data element in MIP-1 has a corresponding data element in MIP-0 with the same relative position in the triangle as illustrated by Figure 2.1 (this relationship holds true for any two neighboring MIP-levels). The rest of the data elements will have exactly two of these elements as neighbors. Up-sampling of chroma components is implemented by applying a linear interpolation of the two closest data elements from the higher MIP-level. Edges and faces are processed separately, where edges are up-sampled before faces as up-sampling of face data elements are dependent on edge data elements.

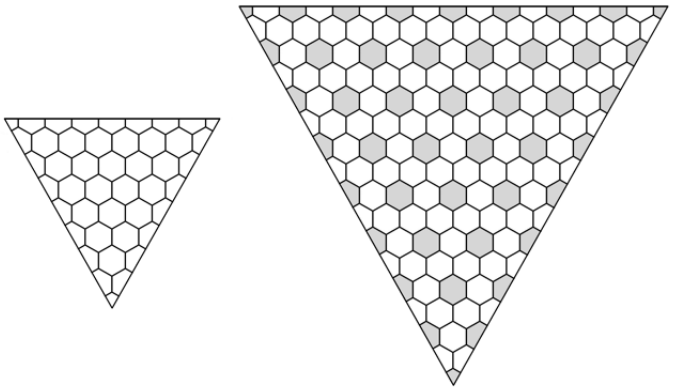

Figure 2.1: Elements in the large triangle (MIP-0) with positions reappearing in the smaller triangle (MIP-1) are marked with grey.

2.3.1 Edges

As illustrated by Figure 2.2, edge data elements are divided into three categories where elements of the first category, marked 1 in the figure, are assigned the value of the data element in the higher MIP-level with the same relative position. Category 2 elements are assigned the mean value of the two neighboring category 1 elements. Category 3 elements are assigned the mean value of the closest category 1 element and the closest vertex element. The last category of data elements will, at most, have two members for any resolution.

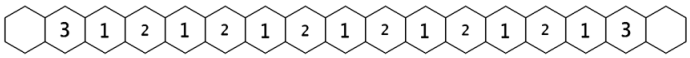

Figure 2.2: Edge with resolution 16 where elements are numbered after their category. The two outermost elements are vertex data elements.

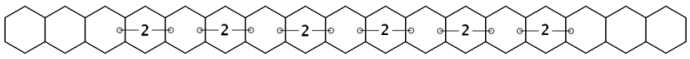

Figure 2.3: Each category two element is assigned the mean value of its surrounding data values.

2.3.2 Faces

Face data elements are divided into four categories. Category 1 consists of elements assigned the value of the data element in the higher MIP-level with the same position. The number of data elements in this group is equal to the number of face data elements of a face with half the resolution. Category 2 data elements are assigned the mean value of two neighboring face data elements belonging to category 1. They only appear if the down-sampled face in MIP-1 has more than one element. The third category of elements are those that are assigned the mean value of one face data value and one edge data value. These elements appear only if the down-sampled MIP-level has face elements. Data elements of the first three categories appear only in faces with a resolution greater than 4. Each face for which chroma sub-subsampling is supported will also have three category 4 elements that are assigned the interpolated value of two closest edge data elements. These data elements appear on the corners of the face and are not present for low resolutions.

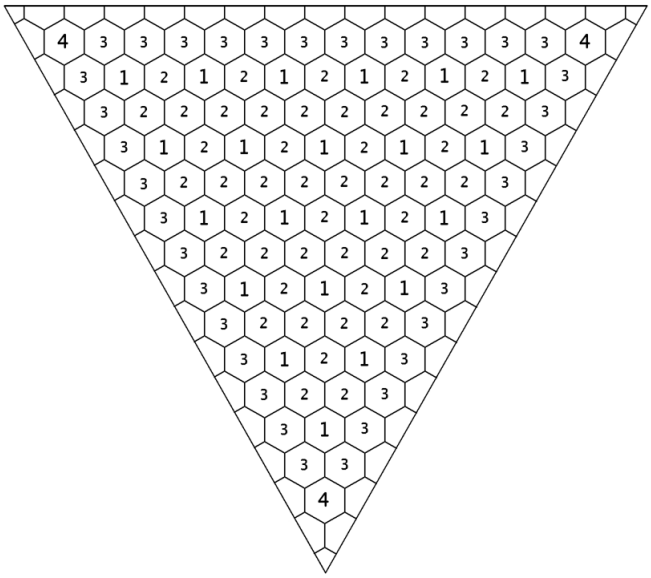

Figure 2.4: Triangle with resolution 16 where face data elements are numbered after their category.

Data elements of category 3 and 4 are assigned the mean value of two data elements where at least one of them is an edge data element. When an edge is shared by two faces with different resolutions, one of the faces will have a different resolution than that of the shared edge. If the resolution of the shared edge is greater than the resolution of the face then the appropriate MIP-level of the edge is used for up-sampling. If the resolution of the edge is lower than the resolution of the face, the edge is first up-sampled prior to using elements for reconstruction of face elements.

2.4 Block Partitioning

In the JPEG standard, data elements are split into blocks before transformation to the frequency domain. Triangles, in the MCC, are processed individually to not partition data elements into blocks that transcend triangle boundaries. Having such blocks would introduce a number of issues such as neighboring triangles having different resolutions, having different spacial properties due to varying sizes and shapes, and having vastly different texture data. Edge and face data elements are processed separately and have their elements partitioned into one-dimensional and two-dimensional blocks respectively.

2.4.1 Edges

Edge data elements are split into sequences of 8 elements in a one-dimensional fashion as illustrated by Figure 2.5. This pattern of partitioning edge data elements will at most result in a single block with less than 8 elements, which appears at the end of the sequence of blocks.

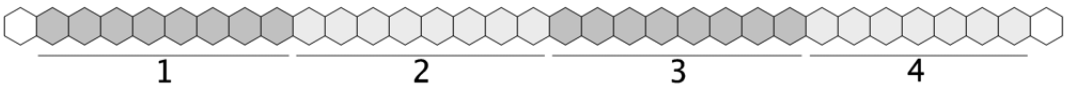

Figure 2.5: Splitting of edge data elements into blocks. The unmarked cells are vertex data elements.

2.4.2 Faces

Face data elements are split into *8*-by-*8* blocks as illustrated by Figure 2.6. Resulting blocks have a shape resembling a quadrilateral with 64 hexagonally shaped data elements. Because of the triangular shape of the face, blocks closest to the right edge will have missing diagonals of data elements. In the example below, blocks 1, 2, and 5 are "full" blocks consisting of 64 elements, blocks 3, 6, and 8 are blocks with a single missing diagonal (63 elements each), and blocks 4, 7, 9, and 10 are blocks with 9 missing diagonals each (21 elements).

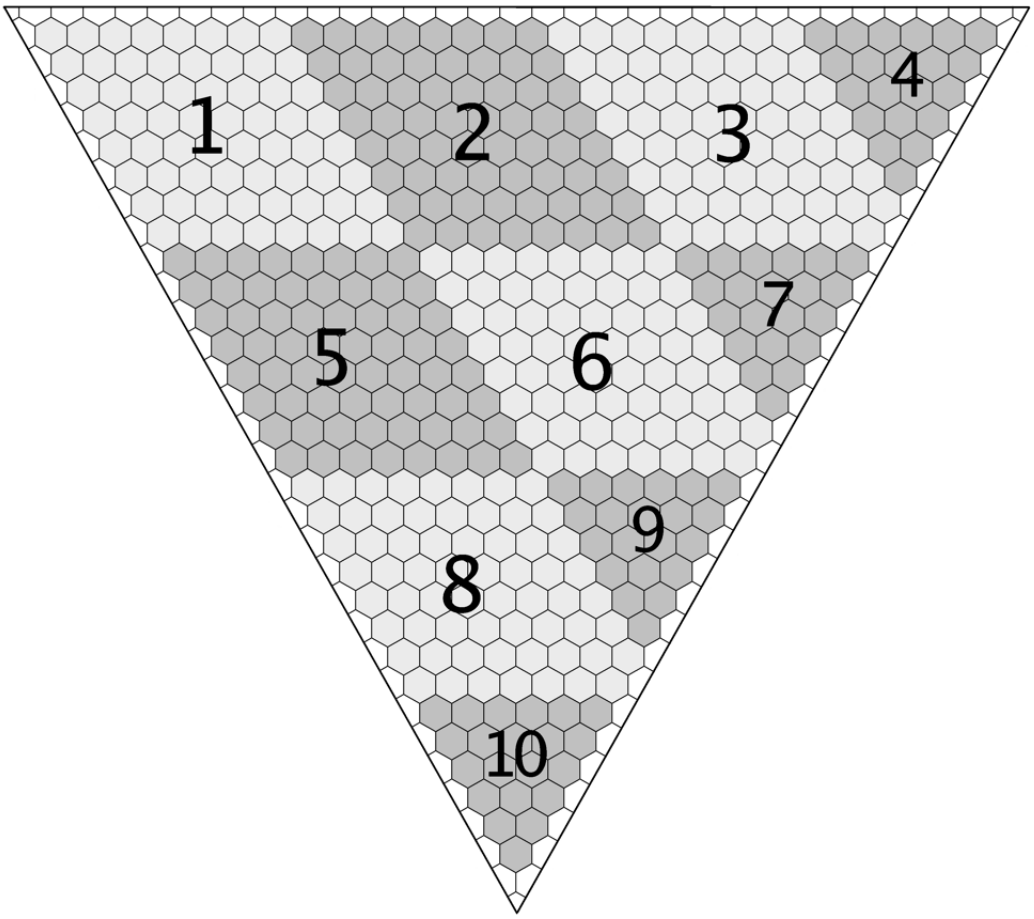

Figure 2.6: Splitting of face (resolution = 32) data elements into blocks.

2.5 Discrete Cosine Transform

The type-II Discrete Cosine Transform, DCT-II (referred to as DCT), is an essential part of the JPEG standard. Applied to blocks of YUV-transformed texels, it transforms each block from the spatial domain to the frequency domain. Resulting data elements are amplitudes of the basis functions of the transform matrix, which is dependent on the dimensions of the DCT matrix (which in turn is dependent on the block dimensions). The transform is separable and therefore applied to rows and columns of the block separately, where the order in which they are applied does not matter.

For transformation of data elements, the MCC uses a modified version of the DCT, where transform coefficients are given by

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1. \tag{2.5}$$

Equation (2.5) multiplies X_0 by *1/N*, where N is the number of elements in the dataset (either a row or column of data elements). For inverse transformation, a modified version

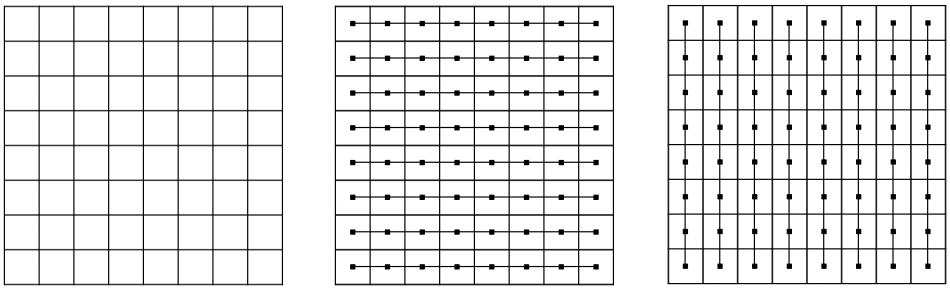

Figure 2.7: Transform directions defined by the JPEG standard in an *8*-by-*8* block.

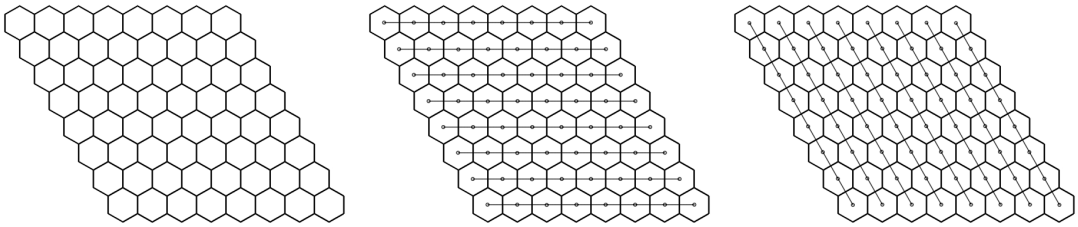

Figure 2.8: Transform directions for a quadrilateral block.

of DCT-III is used (equation 2.6) (from now on referred to as IDCT or inverse DCT), where x_0 is multiplied by N to reflect the modification in equation (2.5).

$$X_k = \frac{1}{2}x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N} n \left(k + \frac{1}{2}\right)\right] \qquad k = 0, \ldots, N-1 \tag{2.6}$$

2.5.1 Faces

Transformation of blocks is done in the directions illustrated by Figure 2.8, where the first transform direction is horizontal and the second one is parallel to the left edge of the triangle. These transform directions are similar to the transformation directions defined by the JPEG standard for *8*-by-*8* blocks of data elements in a Cartesian grid (Figure 2.7).

Blocks with missing diagonals are transformed using the same transformation directions used for full *8*-by-*8* blocks, but will include transformations of datasets with varying lengths. Resulting coefficients of the first transformation are combined with coefficients of the same index in the second transformation (2.9). This approach of transforming blocks has similarities to certain aspects of the Directional Discrete Cosine Transform (DDCT) [9]. Like the DDCT, the MCC combines, in the second transformation, coefficients belonging to data sequences of different lengths, and takes care to pair those with the same index. That is, DC coefficients of the first transformation are transformed separately from first-order AC coefficients and so on. Transformation directions for the diagonal down-right mode of the DDCT are illustrated by Figure 2.11.

The modification made to the type-II DCT (equation 2.5) results in X_0 being equal to the mean value rather than the sum. Color data close in proximity generally have similar

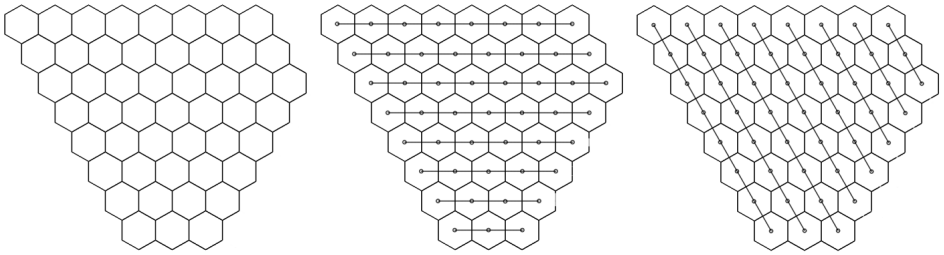

Figure 2.9: *8*-by-*8* block with 5 missing diagonals of data elements (left), transform direction one (middle) and transform direction two (right).

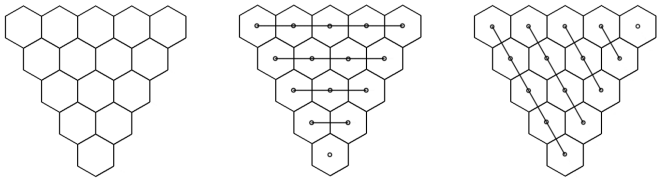

Figure 2.10: *8*-by-*8* block with 10 missing diagonals of data elements (left), transform direction one (middle) and transform direction two (right).

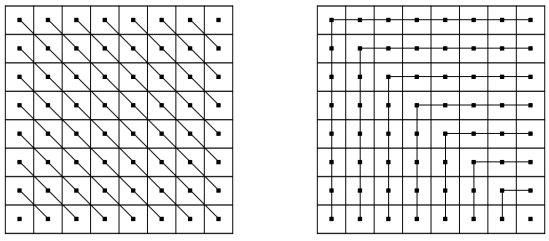

Figure 2.11: Transformation directions of the diagonal down-right mode as defined by the DDCT. The first transformation direction (left) is diagonal and the second one (right) is constructed to make sure coefficients of the same index are combined.

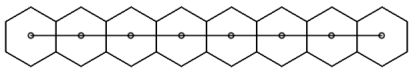

Figure 2.12: Transform direction for a one-dimensional block of edge data elements.

content, thus, by normalizing with the number of data points, X_0 coefficients will have similar magnitudes as well, making them good estimates of close-by X_0 coefficients even when they belong to blocks of different sizes. This fact will be taken advantage of during coding of DC coefficients (i.e. X_0 coefficients) in Section 2.7.

Inverse transformation of blocks is analogous to forward transformation where the modified IDCT (equation 2.6) is first applied along transform direction two (i.e. vertically), and then along transform direction one.

2.5.2 Edges

Blocks of edge data elements are transformed in the only dimension available (Figure 2.12) using equation (2.5) and (2.6) for forward and inverse transformation respectively. As was the case with face elements, using the modified type-II DCT will yield DC coefficients of comparable size even when block sizes differ, as they will equal the mean value of the block rather than the sum.

2.6 Quantization

Quantization is a procedure used by codecs to compress a range of values into a single one, with the aim of reducing the alphabet size (the number of unique values in a data collection) and make the data stream more compressible when deploying an entropy coder later in the process. Forward quantization is achieved by rounding off pixel values divided by a quantization step:

$$q_{i,j} = round\left(\frac{p_{i,j}}{s}\right) = \left\lfloor \frac{p_{i,j} + \frac{s}{2}}{s} \right\rfloor.$$

Inverse quantization is analogous to forward quantization and is done by multiplying the quantized value with the quantization step:

$$\hat{p}_{i,j} = q_{i,j} \cdot s.$$

JPEG codecs implement a uniform scalar quantizer which is applied on elements in the frequency domain, i.e. the resulting coefficients of the modified DCT transformation. The standard defines a quantization matrix for the luma component as

$$\begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{2.7}$$

, and defines a single quantization matrix for both of the chroma components as

$$\begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}. \tag{2.8}$$

Each coefficient in a transformed block is quantized with the corresponding quantization step in the appropriate matrix. Coefficients representing higher frequencies are quantized with a greater quantization step. This is done to take advantage of the fact that humans are not as good at distinguishing the exact strength of a high-frequency brightness variation as it is at detecting small differences in brightness over a relatively large area (i.e. low-frequency variations).

The MCC uses the quantization tables defined by the JPEG standard and achieves a greater/lesser compression ratio by multiplying each of the quantization steps by an adjustable quantization factor.

2.6.1 Faces

DCT coefficients in blocks of face data elements are quantized with the quantization steps presented in equation (2.7) and (2.8) with corresponding positions in the appropriate quantization matrix. Blocks with missing diagonals will only quantize existing elements and will not make use of the entire matrix.

2.6.2 Edges

As edge data elements are partitioned into one-dimensional "blocks", the JPEG quantization matrix is not directly applicable. For quantization of these elements, the first 8 steps of the quantization matrices along the zig-zag pattern (Figure 2.16) are used. The quantization steps presented in equation (2.9) are used for the luma component, and the ones in equation (2.10) are used for the chroma components. Because edge data elements are only transformed in a single direction, which makes them less prone to quantization, steps are further modified by a factor *0.1*. As with blocks of face data elements, blocks of edge elements with *N* missing elements will only use the first *8-N* quantization steps.

$$0.1\begin{pmatrix}16 & 11 & 12 & 14 & 12 & 10 & 16 & 14\end{pmatrix} \tag{2.9}$$

$$0.1\begin{pmatrix}17 & 18 & 18 & 24 & 21 & 24 & 47 & 26\end{pmatrix} \tag{2.10}$$

2.7 Differential Coding of DC Coefficients

The JPEG standard encodes DC coefficients with a differential encoder. By using the DC coefficient of a nearby block as an estimation, DC coefficients are encoded as the difference between the estimate and the absolute value. This feature is supported by the MCC and is the reason for the modification of the X_0 coefficient in equation (2.5). With the modification, the number of data elements in blocks, which can vary, becomes irrelevant as the DC coefficients will be equal to the mean value of elements rather than the sum.

2.7.1 Faces

Depending on the position of the block within the triangle, either DC coefficients from nearby edges or a DC coefficient from a nearby face data block will be used. The top-left block will use as estimation the mean value of the two closest DC coefficients from the two neighboring edges. Remaining blocks will either refer to blocks to the left or to the top for DC coefficient estimates (see Figure 2.13).

2.7.2 Edges

The DC coefficient of the first block of edge elements is estimated with the closest vertex data value. Data values positioned at vertices are part part of the 3D mesh and are not compressed by the MCC and therefore never processed. They are therefore YUV-transformed and quantized before being used as estimates. Transformation of vertex data values with equation (2.5) is not necessary, as application of the formula for a single data point leaves it unaltered. Remaining DC coefficients are estimated with the DC coefficient of the preceding block, as illustrated by Figure 2.14.

2.8 Run-Length Coding

Due to the frequency functions that make up AC coefficients (equation 2.5) and the greater quantization steps used, the probability of them being equal to zero is greater. This is especially true for two-dimensional blocks of face data elements, as quantization steps in the quantization matrices (equation 2.7 and 2.8) increase in amplitude towards the lower right corner.

Like the JPEG standard, the MCC supports zero run-length encoding of AC coefficients. It is a lossless data compression technique applied on linear arrays and is suitable for streams of data elements with sub-sequent zero values. Because zero values become increasingly probable towards the lower right corner of a block, the data elements are iterated over by the run-length encoder in a zig-zag pattern as specified by the JPEG standard

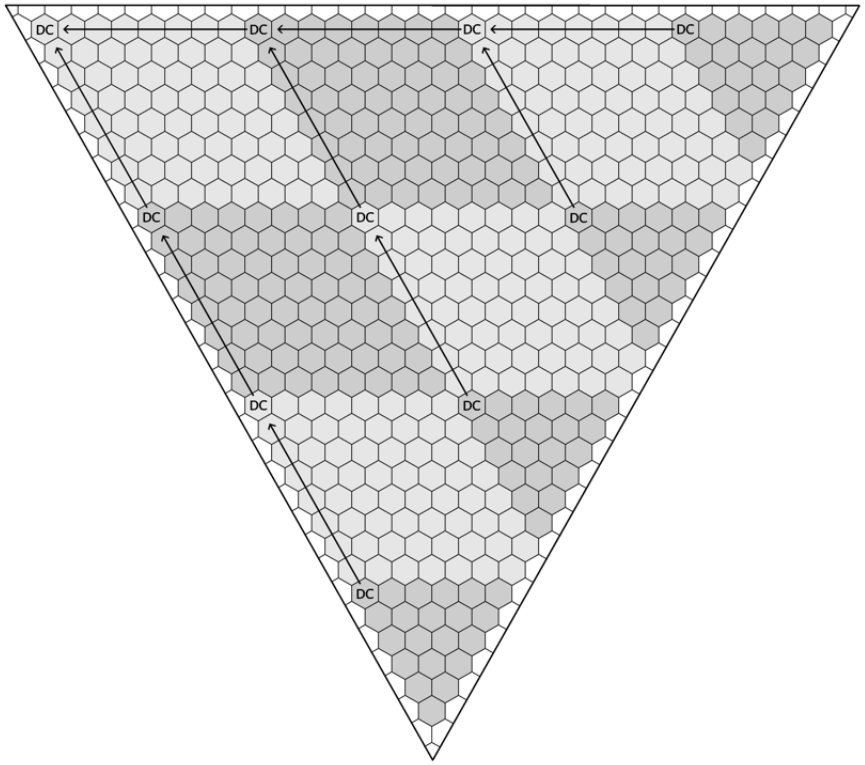

Figure 2.13: The arrows point to the DC coefficient used as estimation by the DC coefficient the arrow originates from.

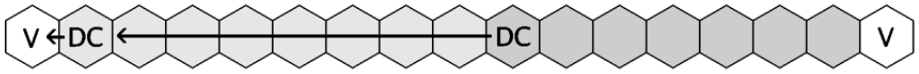

Figure 2.14: Estimation directions for DC coefficients in blocks of edge data elements.

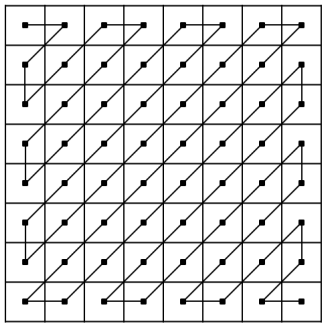

Figure 2.15: Zig-zag pattern defined by the JPEG standard for *8*-by-*8* blocks. The first data element in the pattern located at the top-left corner of the block is not processed by the zero run-length encoder.

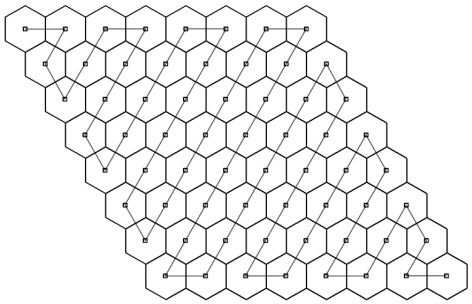

Figure 2.16: Zig-zag pattern defined for *8*-by-*8* blocks of face data elements. The first data element in the pattern, located at the top-right corner of the block, is not processed by the zero run-length encoder.

(2.15). Similar to zero run-length coding in JPEG codecs, the MCC encodes all non-zero values as a pair of values consisting of a run-length and an amplitude. The run-length is the number of zeroes since the last non-zero data value (limited to 255). The amplitude is the non-zero value that is at the end of the sequence of zeroes. This differs slightly from how the JPEG standard specifies encoding of the run-length and amplitude.

2.8.1 Faces

The zig-zag pattern used for run-length encoding of face data elements is illustrated in Figure 2.16. This pattern is used for blocks with missing diagonals as well, in which case the zig-zag pattern will end prematurely. It is possible to do so, as blocks with less than the maximum number of data elements are always missing full diagonals closest to the bottom right corner of the block. Because the maximum *run-length* allowed (*255*) exceeds the number of AC coefficients in a block, a single pair of [*run-length*, *amplitude*] can be made to encapsulate coefficients from up to five blocks. Depending on the resolution of the triangle, a single pair can also encapsulate AC coefficients from several blocks.

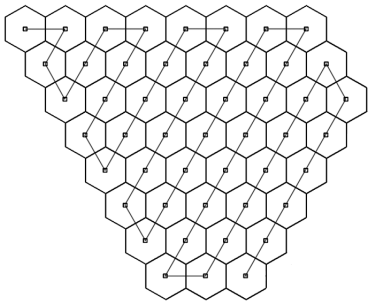

Figure 2.17: Zig-zag pattern defined for a block of face data elements with 5 missing diagonals. The first data element in the pattern, located at the top-right corner of the block, is not processed by the zero run-length encoder.

2.8.2 Edges

The zero run-length encoder iterates through coefficients in one-dimensional blocks of edge elements from left to right. As in the case of face elements, AC coefficients from several blocks can be combined into a single data pair of [*run-length*, *amplitude*].

2.9 Entropy Coding

The next step after differential coding of DC coefficients and run-length coding of AC coefficients is, as defined by the JPEG standard, entropy coding of the obtained data. Though the standard supports both Huffman coding and Arithmetic coding, the latter one is rarely implemented by codecs due to historical reasons concerning patents. Entropy coding offers lossless compression and may either be responsible for most of the compression or complement the compression obtained in the previous steps [7]. The MCC uses an existing arithmetic coder to encode DC and AC coefficients. Since independent and identically distributed sources provide better compression ratios when using arithmetic coders [7], the data is split into a total of 18 contexts as described by Table 2.1.

2.9.1 DC Coefficients

DC coefficients encoded with a differential encoder are split into three data contexts depending on which component (Y', C_b, C_r) they belong to. Furthermore, coefficients are split into an additional two contexts for face and edge data elements. The reason for this separation is due to the use of different quantization steps, and the difference in transformation. While DC coefficients of edge elements express the mean value in a one-dimensional direction, those of face elements do so for a two-dimensional area.

Table 2.1: Data split into contexts.

data	context
$DC_{Y',face}$ coefficients	1
$DC_{C_b,face}$ coefficients	2
$DC_{C_r,face}$ coefficients	3
Run-lengths$_{Y',face}$	4
Run-lengths$_{C_b,face}$	5
Run-lengths$_{C_r,face}$	6
Amplitudes$_{Y',face}$	7
Amplitudes$_{C_b,face}$	8
Amplitudes$_{C_r,face}$	9
$DC_{Y',edge}$ coefficients	10
$DC_{C_b,edge}$ coefficients	11
$DC_{C_r,edge}$ coefficients	12
Run-lengths$_{Y',edge}$	13
Run-lengths$_{C_b,edge}$	14
Run-lengths$_{C_r,edge}$	15
Amplitudes$_{Y',edge}$	16
Amplitudes$_{C_b,edge}$	17
Amplitudes$_{C_r,edge}$	18

2.9.2 AC Coefficients

The same arguments are valid for AC coefficients, which is why *run-lengths* and *amplitudes* of edge data elements and face data elements are encoded using different contexts. This results in a total of 12 contexts used for AC coefficients.

Chapter 3
Results

The purpose of image codecs is to compress data while maintaining an acceptable visual fidelity. Compression ratio in this report refers to compression of color data only and is defined as the ratio between size of uncompressed color data and that of compressed color data. The quality of the reconstructed data is characterized by the *Peak signal-to-noise ratio (PSNR)* defined as

$$PSNR = 10 \log_{10} \left(255^2 / E_{na} \right),$$

where 255 is the maximal pixel value of 8-bit words. It is calculated using the average quantization noise defined as

$$E_{na} = \frac{1}{N} \sum_{i=0}^{N} (p_i - \hat{p}_i)^2 ,$$

where p_i denotes a texel value of the image component (Y', C_b or C_r), $\hat{p}_i$ the reconstructed texel value of the component, and N the number of color data elements. Higher *PSNR* values correspond to greater image quality of decompressed images. The degree of compression ratio is affected by the chroma sub-sampling option and the quantization factor, both of which allow for an adjustable trade-off between compression ratio and visual quality. Two 3D models with their accompanying mesh colors data were used for gathering results.

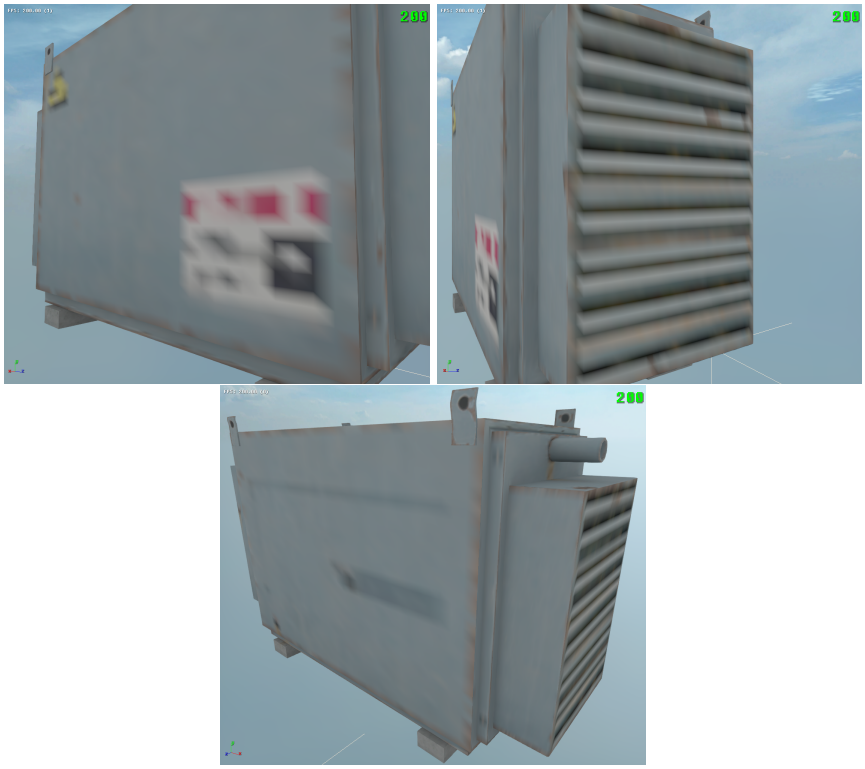

Figure 3.1: Model 1 textured with uncompressed mesh colors data. Each face has a resolution of 32.

3.1 Model 1

Figure 3.1 presents the first 3D model used for measurements. The same model with faces of higher resolution is presented in Figure 3.2. Disk size of uncompressed mesh colors data of different resolutions is presented in Table 3.1. They were generated from the source file in Figure 3.3.

Table 3.1: Sizes of color data in kB for uncompressed mesh colors data with different resolutions.

resolution	size
32	627 kB
64	2,413 kB
128	9,464 kB

3.1.1 Full Chroma Resolution

A comparison of the model rendered with uncompressed data and compressed data can be seen in Figure 3.4. The mesh colors were compressed without chroma sub-sampling and with a low quantization factor, resulting in a compression ratio of 6.47. The resulting *PSNR* values were 51.56, 54.18, and 53.46 for components Y', C_b and C_r respectively.

Figure 3.2: Model 1 textured with uncompressed mesh colors data. Each face has a resolution of 128.

Figure 3.3: Source texture file that the mesh colors data was generated from.

Figure 3.4: Rendering with MCC-compressed texture (left) and uncompressed texture (right). Faces in both models have a resolution of 64. Compression ratio: 6.4517.

For a rough comparison, one can consider compression of the source texture file with JPEG. Without chroma sub-sampling and adjusted to result in a compression ratio of 6.72, the resulting *PSNR* values were 49.94, 51.13, and 50.49 for components Y', C_b, and C_r. These values are, when compared to the MCC for a similar compression ratio, 1.62, 3.05, and 2.97 lower for components Y', C_b, and C_r respectively. Renders of the model with decompressed mesh colors data with greater compression rates can be be seen in Figure 3.5.

While high compression ratios are desirable, the visual quality of the reconstructed image must be taken into consideration. Figure 3.6 illustrates how the *PSNR* values of components Y', C_b and C_r are affected for increasing compressing ratios. The dashed line represents the *PSNR* for the source texture file compressed with a JPEG codec, and the solid line represents the *PSNR* of the compressed mesh colors data.

It can be observed that, in terms of *PSNR*, the MCC compares differently to the JPEG codec depending on the component (Y', C_b, or C_r). For compression of the luma component with ratios between 5 and 33, the MCC exhibits greater *PSNR* values. The same comparison for the chroma components, C_b and C_r, puts the MCC ahead for compression ratios up to 13 and 14 respectively. Figure 3.6 shows that *PSNR* values of the mesh colors data that was compressed with the MCC deteriorate quicker for increasing compression ratios.

Figure 3.5: Rendering of model textured with decompressed mesh colors data with compression ratios of 9.54 (top), 14.62 (middle) and 23.43 (bottom).

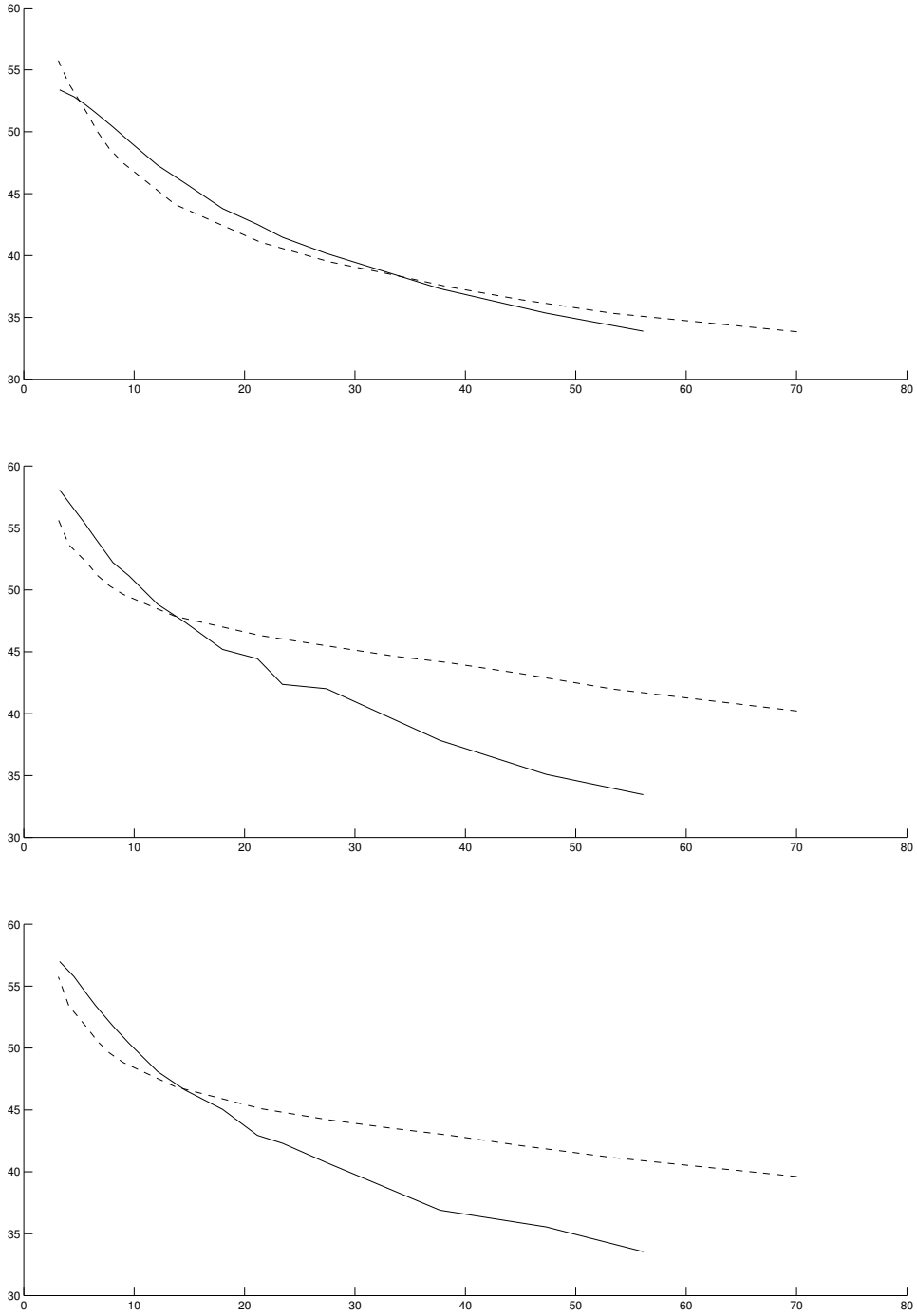

Figure 3.6: *PSNR* of model 1 texture components Y' (top), C_b (middle), and C_r (bottom), when chroma sub-sampling is disabled, plotted as a function of compression ratio. The solid lines represent the quality of the compressed mesh colors data and the dashed lines represent the quality of the JPEG-compressed source data.

Figure 3.7: Renders with compressed mesh colors data (left-column) and uncompressed mesh colors data (right-column).

3.1.2 Chroma Sub-Sampling

Results in this section were gathered with the chroma sub-sampling option activated in the MCC. Renders of the model textured with compressed and uncompressed mesh colors data can be seen in Figure 3.7. The decompressed data used for texturing in both the top and bottom images in the figure was compressed with a compression ratio of 8.41, resulting in *PSNR* ratings of 51.42, 48.3, and 46.38 for components Y', C_b, and C_r respectively. Compression of the source image with a JPEG codec with a compression ratio of 7.92 resulted in *PSNR* values of 51.27, 46.42, and 44.77 for components Y', C_b and C_r respectively.

Figure 3.8 presents results of compression with greater quantization factors. The mesh colors data was compressed with compression ratios of 12.00, 17.87, and 27.74 respectively.

Plots of the *PSNR* values of components Y', C_b, and C_r of both the mesh colors data and the source file are presented in Figure 3.9. The dashed line plots the *PSNR* of the source data (compressed with a JPEG codec) for different compression ratios.

Enabling chroma sub-sampling has minimal effect on the shape of the *PSNR* plot of

Figure 3.8: Renders of the model textured using mesh colors data with compression ratios of 12.00 (top), 17.87 (middle) and 27.74 (bottom).

component Y', as is made clear by comparing the top graph in Figure 3.9 with the corresponding graph in Figure 3.6. *PSNR* values of chroma components, in contrast, exhibit an initial drop when chroma sub-sampling is enabled though this trend changes as compression ratio increases. For high enough compression ratios, chroma sub-sampling has a positive effect on *PSNR* values. Comparing chroma components of the source file (compressed with a JPEG codec) and the mesh colors data (compressed with the developed codec), it can be observed that JPEG starts performing better than the MCC for compression ratios of 25 and higher for both chroma components (3.9). In contrast, the corresponding values when chroma sub-sampling is disabled is 13 and 14 for components C_b and C_r respectively (3.6). Thus, with chroma sub-sampling enabled, the MCC maintains its lead over JPEG for even greater compression ratios. The faster deterioration of quality in reconstructed images for increasing compression ratios is, however, still present.

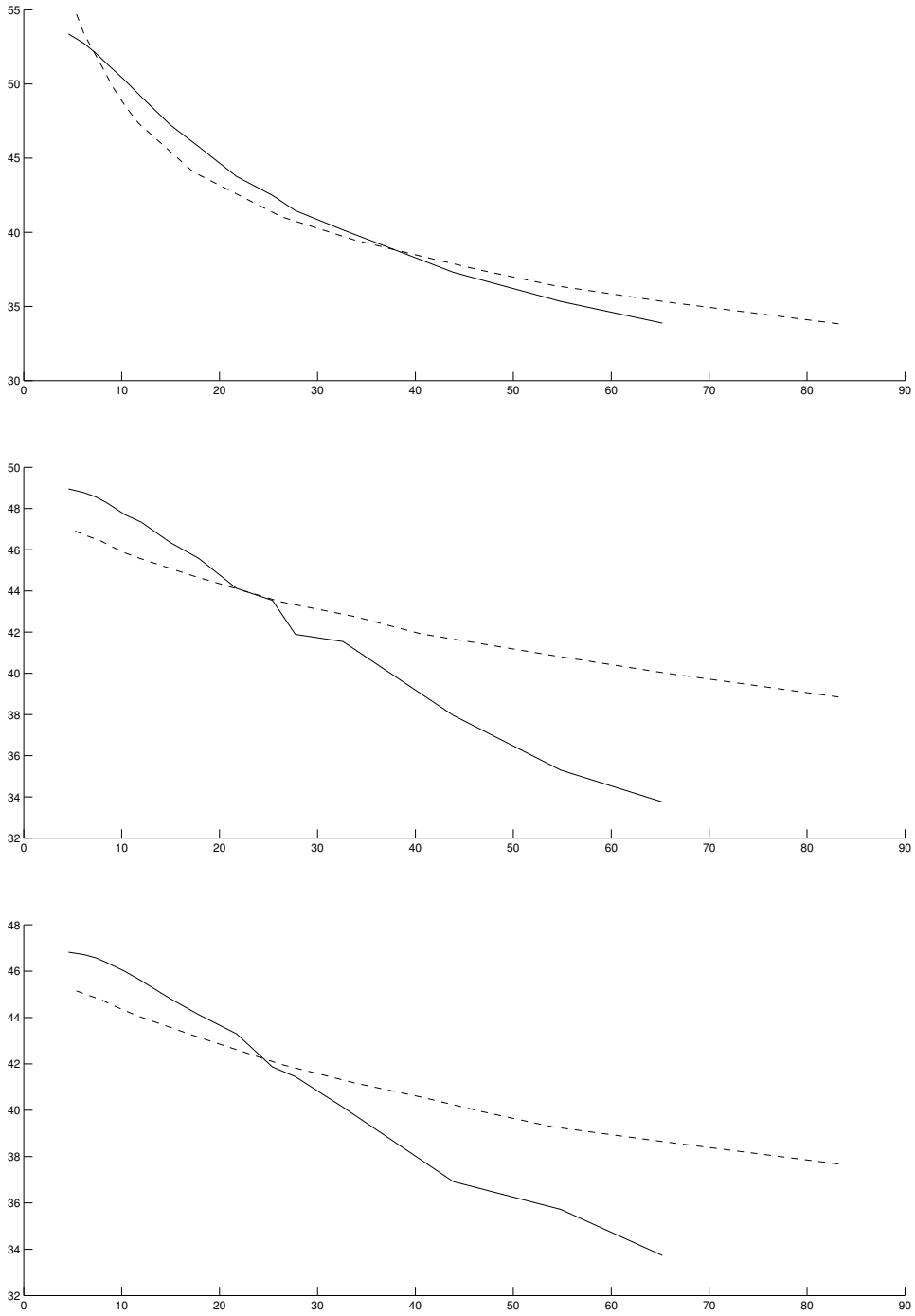

Figure 3.9: *PSNR* of model 1 texture components Y' (top), C_b (middle), and C_r (bottom), when chroma sub-sampling is enabled, plotted as a function of compression ratio. The solid lines represent the quality of the compressed mesh colors data and the dashed lines represent the quality of the JPEG-compressed source data.

Figure 3.10: Model of a building textured with mesh color data. Each face has a resolution of 128.

3.2 Model 2

The second model used for measurements is presented in Figure 3.10, textured with mesh colors data generated from the image in Figure 3.11.

3.2.1 Full Chroma Resolution

Figure 3.12 presents the changes in *PSNR* of the Y', C_b, and C_r components for increasing compression ratios. The solid line in each graph represents the quality of mesh colors data compressed with the MCC, and the dashed lines represent the quality of the source data compressed with a JPEG codec. Similar to the results in Figure 3.6, the MCC performs better than JPEG in compression of component Y' for certain compression ratios, but is otherwise close to the performance of JPEG in terms of quality. Compression of chroma components also behave as expected, where the measured quality of decompressed mesh colors data diminishes quicker than that of the source data for increasing compression ratios. As in Figure 3.6, the MCC outperforms JPEG up to a certain compression ratio but starts to perform worse after that.

3.2.2 Chroma Sub-Sampling

Figure 3.13 plots *PSNR* values of the YUV-transformed image components as a function of compression ratio. The biggest change when enabling chroma sub-sampling is the drop in quality of chroma components. Similar to the results in 3.9, the MCC performs better than JPEG in the compression of chroma components up to certain compression ratios. As in the case for model 1, enabling chroma sub-sampling results in the MCC performing better than JPEG for even greater compression ratios.

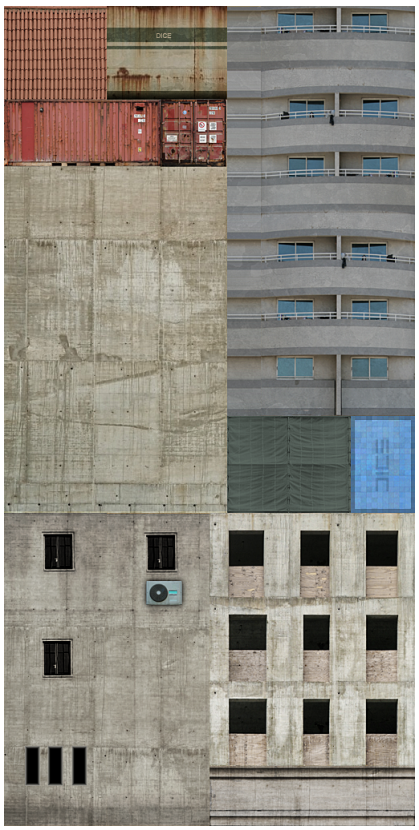

Figure 3.11: Source texture file that the mesh colors data for model 2 was generated from.

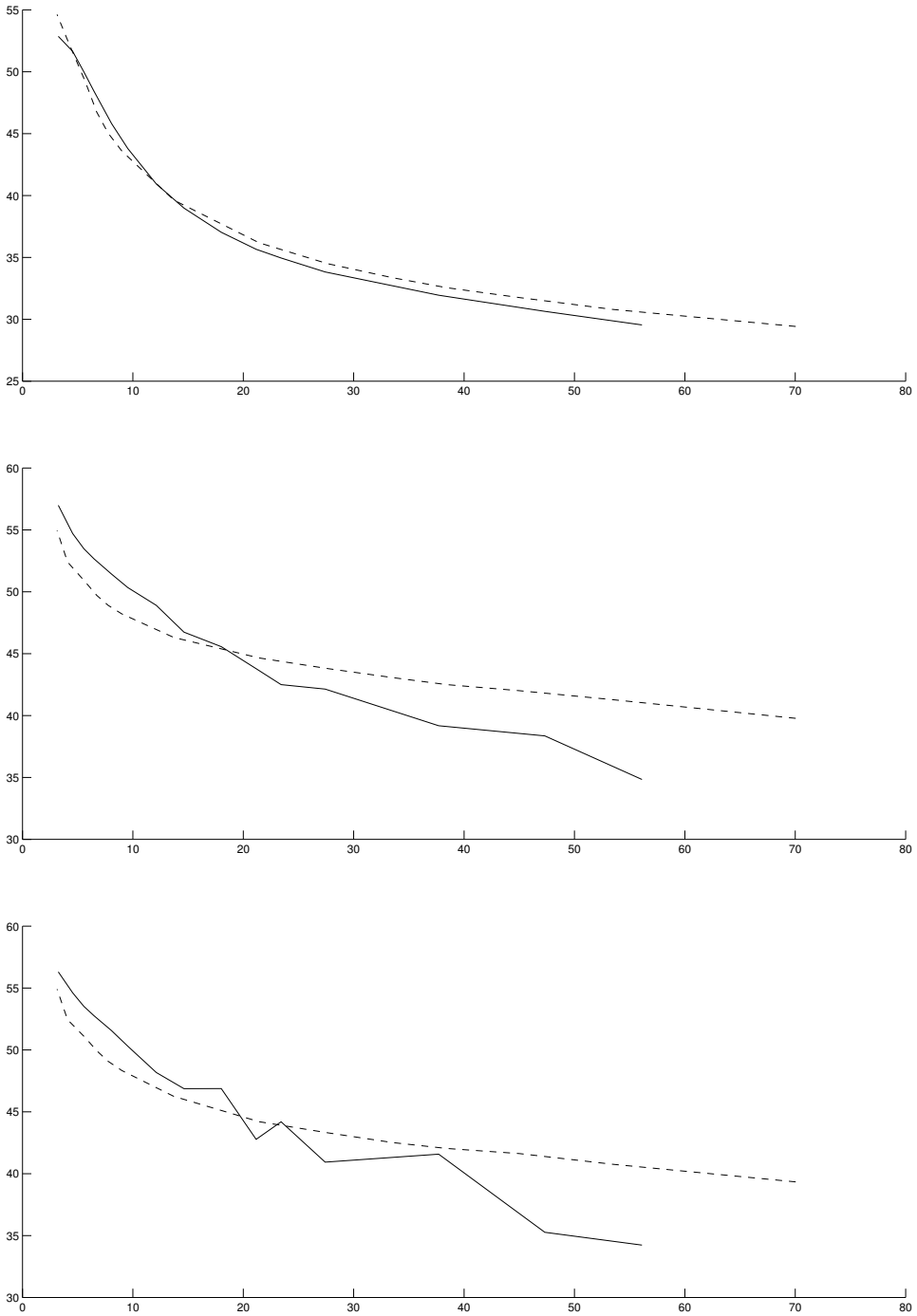

Figure 3.12: *PSNR* of model 2 texture components Y' (top), C_b (middle), and C_r (bottom), when chroma sub-sampling is disabled, plotted as a function of compression ratio. The solid lines represent the quality of the compressed mesh colors data and the dashed lines represent the quality of the JPEG-compressed source data.

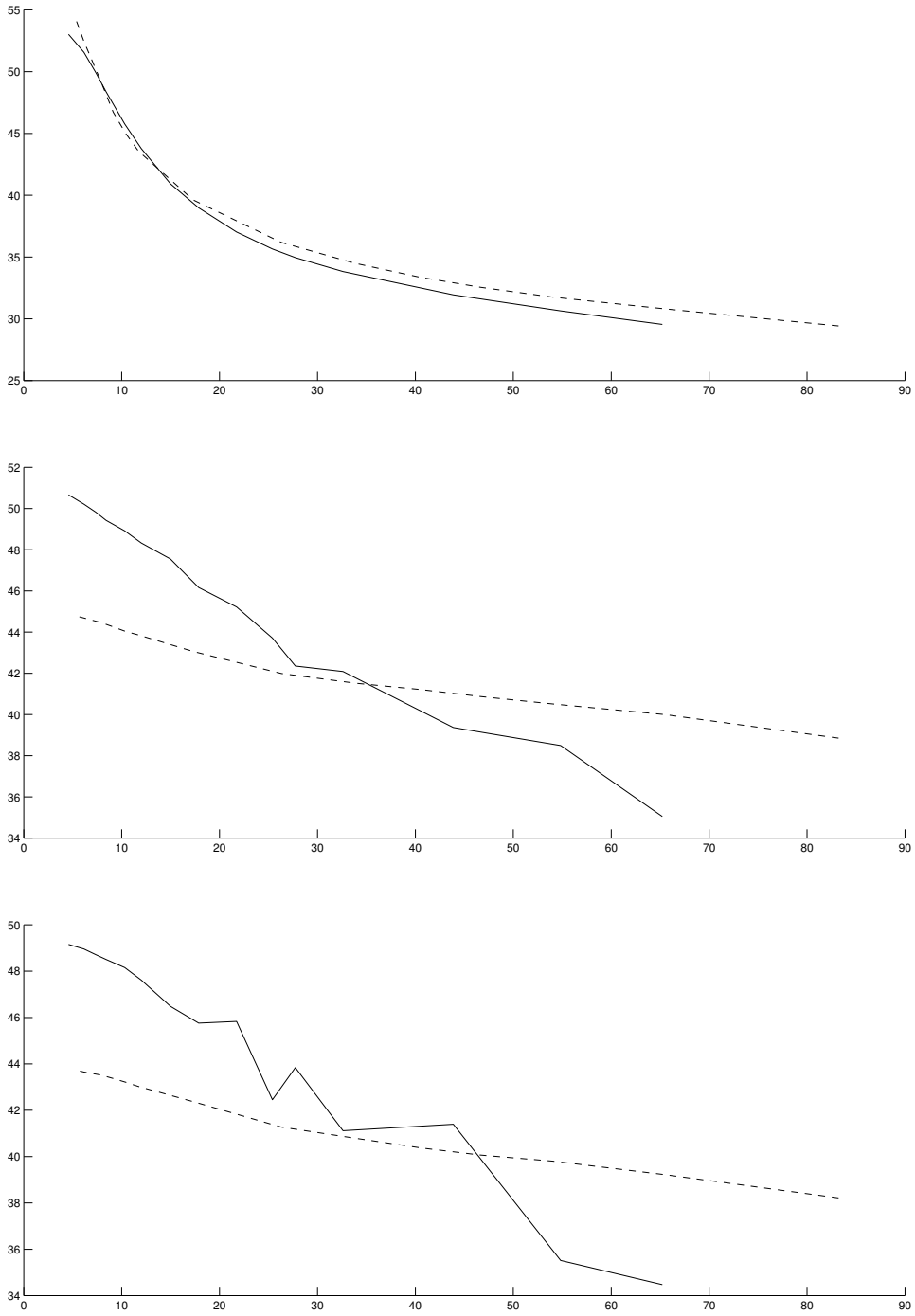

Figure 3.13: *PSNR* of model 2 texture components Y' (top), C_b (middle), and C_r (bottom), when chroma sub-sampling is enabled, plotted as a function of compression ratio. The solid lines represent the quality of the compressed mesh colors data and the dashed lines represent the quality of the JPEG-compressed source data.

Figure 3.14: Close-up of model 1, rendered with highly compressed mesh colors data.

3.3 Compression Artifacts

Below are the most common compression artifacts that can be found in mesh colors textures compressed with the MCC, and 2D textures compressed with JPEG.

3.3.1 Blocking

A common compression artifact found in compressed images and videos is blocking (illustrated by Figure 3.14). Blocking in mesh colors data appear as quadrilaterals, with shapes depending on the stretching of the triangle, and not squares as they usually do in JPEG files. The model in 3.14 is textured with decompressed mesh colors data with a compression ratio of 42.8069. This artifact is also visible in images compressed with a JPEG codec (3.15). As the compression ratio increases, the blocks become more visible.

3.3.2 Ringing

Another compression artifact that can appear in JPEG images (and compressed mesh colors data) is ringing. These artifacts appear near edges in the image and resemble them in shape. Figure 3.16 illustrates ringing artifacts around a brightly colored area appearing in compressed mesh colors data. Figure 3.17 illustrates ringing artifacts found in the JPEG-compressed 2D source file.

Figure 3.15: JPEG compression of the source file for model 1 with compression ratios of 1.7687 (left) and 39.0244 (right).

Figure 3.16: Close-up of model 1 textured with mesh colors data compressed with a ratio of 42.8069.

Figure 3.17: Ringing artifacts near an edge in the JPEG compressed source image. The images have a compression ratio of 21.7687 (top), and 39.0244 (bottom).

3.3.3 Noise

Noise is a typical compression artifact that is found in JPEG images, even for lower compression ratios. These artifacts are not as easy to spot on a 3D render due to the filtering involved. Figure 3.18 illustrates noise introduced by compression for different compression ratios. When rendered with more heavily compressed mesh colors data, noise becomes more apparent near detail in the texture.

An alternative way of presenting a triangle with its associated mesh colors data is presented in 3.19. The triangle is transformed into a right-angled triangle to assume the form of a matrix with missing diagonals. In this form, the noise that appears in compressed data is more visible.

3.3.4 Loss of Color Detail

As can be seen in Figure 3.7, compression of mesh colors data with sub-sampled chroma components result in loss of color quality, even when compression ratios are minimal. Loss of color detail near edges where two areas with different color tones meet can be observed in the top row of images in the figure (e.g. the top-left edge of the white square). Loss of color information can also be observed in the bottom row of images where orange detail that can be found in the uncompressed data is not present in the decompressed data.

Similar loss of color details can also be observed in JPEG files with chroma subsampling enabled. Color details concentrated to a few pixels either diminishes or is absent (Figure 3.20).

Figure 3.18: Models textured with compressed mesh colors data with compression ratios of 16.2321 (top) and 25.8704 (bottom). Noise can be observed close to the colored detail.

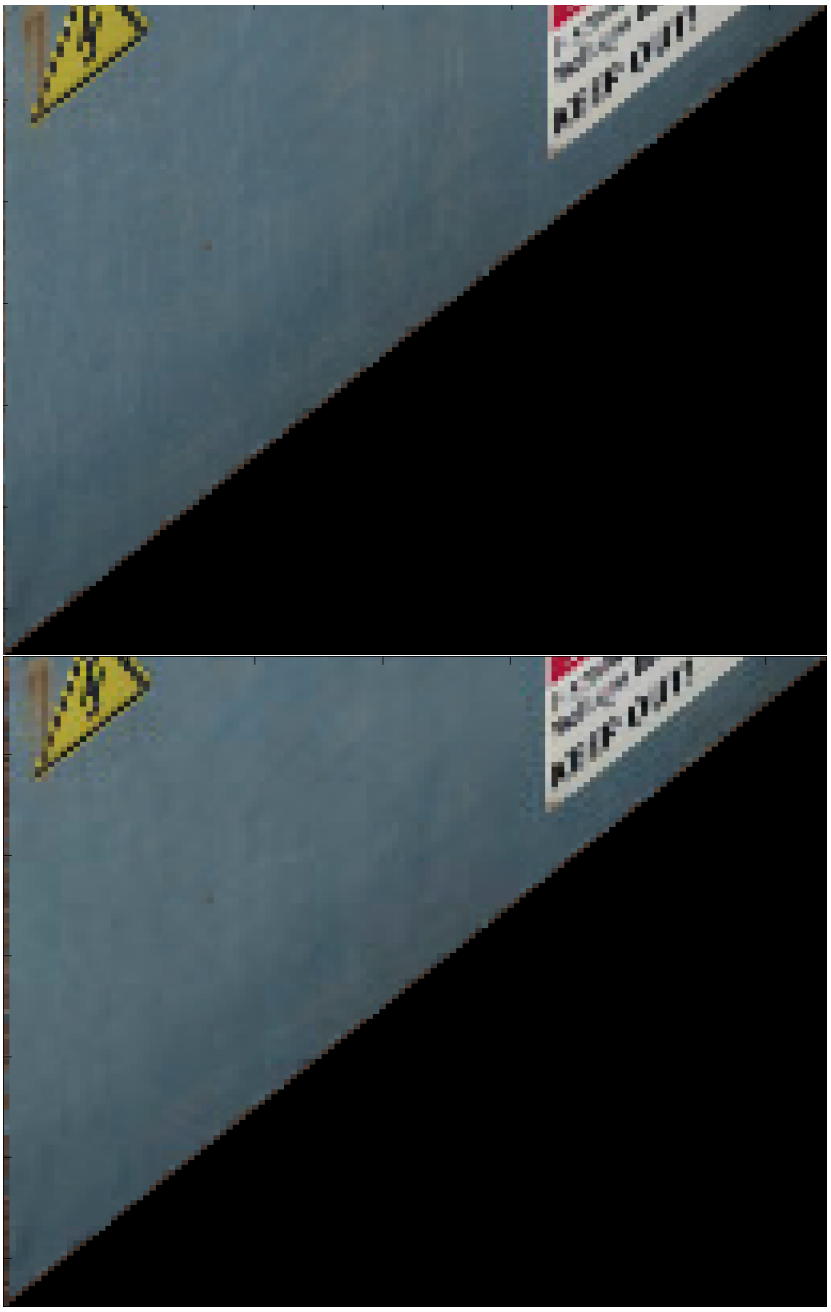

Figure 3.19: Transformed surface data for a single triangle with compression ratios of 16.2321 (top) and 25.8704 (bottom). Noise appears near detail, most notably around the details near the top-left and top-right corners of the triangle.

Figure 3.20: Close-up of the uncompressed source data (left) and compressed source data with a compression ratio of 5.67 (right).

Chapter 4
Discussion

When comparing the quality of MCC-compressed mesh colors textures with the quality of JPEG-compressed 2D textures, results are comparable, and for some configurations, favorable for the MCC. Generally, the MCC outperforms JPEG for lower compression ratios as illustrated by Figure 3.6, 3.9, 3.12, and 3.13. Because of a faster decay in quality for increasing compression ratios, however, JPEG outperforms the MCC after a certain threshold in compression ratio has been passed. Such high compression ratios, however, are unlikely to be used due to the poor quality of reconstructed images (3.8, 3.5).

The only option available, other than chroma sub-sampling, to alter compression ratio is the quantization factor. A faster decay thus suggests that the MCC handles increasing quantization factors less efficiently than regular JPEG codecs. The root cause of this is suspected to be quantization of edges. A one-dimensional transformation will not decouple high-frequency information from low-frequency information as well as a two-dimensional transformation. Thus, smaller quantization steps will need to be applied to coefficients of edge elements to achieve equal PSNR values (when compared to quantization of face elements), or conversely, for the same quantization factor, edge elements will yield lower PSNR values.

Illustrations of compression artifacts in compressed mesh colors textures in Section 3.3 are presented with their JPEG counterparts. Because of the similarities between JPEG and the MCC, similarities in compression artifacts are inevitable.

Because triangles are transformed into different triangular shapes depending on the 3D model, blocking artifacts can assume different shapes. For instance, blocks can appear skewed and elongated if the triangle is shaped as such. Partitioning of face data elements into blocks is another factor affecting the shape of compression artifacts. As can be seen in Figure 2.6, blocks with full sets of data elements originate from the top-left vertex of the triangle. Had a different vertex been chosen as a reference point at creation of the mesh colors data, partitioning would have resulted in blocks that looked different. Because of this, two triangles with identical shape and orientation (even color data) can display block-

ing artifacts aligned differently. This is in contrast to JPEG, where all blocking artifacts are aligned with the *x*- and *y*-axis.

As can be seen in Figure 3.16 and 3.17, ringing artifacts appear in the same area in the compressed mesh colors texture, as it does in the compressed source images. As in the case for blocking artifacts, ringing artifacts may appear differently depending on the partitioning of face data elements into blocks. This explains why the ringing artifacts appear to be oriented differently in Figure 3.16.

Sub-sampling of chroma components is, as stated earlier, a procedure with often minimal effects on the perceptual quality of the image. It does, however, become apparent in areas of the image where color detail is confined to very small areas. Because of the down-sampling during generation of MIP-levels, and up-sampling during decompression, fine color detail will either be absent or blurred in the reconstructed image. Color detail spanning over larger areas however appear to be unaffected by this procedure.

There are two notable effects of chroma sub-sampling in terms of measured quality. When comparing Figure 3.6 with 3.9, and Figure 3.12 with 3.13, it can be noted that the quality of chroma components experience a drop in PSNR for low compression ratios when sub-sampling is enabled. As the compression ratio increases, however, sub-sampled chroma components yield higher PSNR ratings than their counterparts that are not sub-sampled. This behavior indicates that low compression ratios are best accomplished with small quantization factors alone, rather than with chroma sub-sampling, and that increasing compression ratios are best achieved with the addition of chroma sub-sampling, as high quantization factors alone prove to be too destructive.

Plots of *PSNR* as a function of compression ratio also shows that the threshold at which JPEG starts outperforming the MCC is affected by the chroma sub-sampling option. For instance, Figure 3.6 shows that compression with JPEG doesn't start yielding better quality than the MCC until compression ratios of 13.8 and 14.4 are reached for components C_b and C_r respectively. Enabling chroma sub-sampling, however, results in these values increasing to 22.2 and 24.4, as can be seen in 3.9. For model 2, these values increase from 18.0 and 19.7, to 35.2 and 46.3 respectively. This behavior is most likely due to the fact that all chroma data elements are sub-sampled in JPEG, while only those in the highest resolution version of the texture, MIP-level 0, are sub-sampled by the MCC. Because the chroma components of higher MIP-levels are kept in their original resolution, they retain the same quality as when chroma sub-sampling is disabled, thus influencing the overall quality of the compressed texture.

Chapter 5
Conclusion

Obtained results show that measured quality of textures compressed with the developed MCC is similar to the quality of 2D textures compressed with JPEG. The MCC tends to yield better quality for lower compression ratios than JPEG but has a more rapid decay in quality for increasing compression ratios. Such high compression ratios are suspected to be irrelevant for most cases, however, as the reconstructed images display great compression artifacts.

The MCC also achieves a greater compression ratio with chroma sub-sampling compared to JPEG. This is partly due to sub-sampled versions already being available in the MIP-hierarchy. This leads to the MCC outperforming JPEG for increasing compression ratios when compared to the luma component.

While compression artifacts in textures compressed with the MCC are similar to those observed in images compressed with JPEG, they tend to differ in shape and orientation. This is because triangles in meshes are transformed in various ways.

Though computational performance has not been the focus of this thesis, it is of importance when mesh colors are used in time-critical applications such as video games. If importance of compression is secondary to that of load times, the MCC must be able to work within a limited time-window. If it were to take too long to decompress data, it might prove to be more beneficial to loading times to skip compression, and store uncompressed data instead. An advantage the MCC has over JPEG in this regard is the likelihood of graphics hardware being present. Because compression and decompression of mesh colors data is highly parallelizable, if such hardware is present, it could be adapted to run in parallel computing platforms such as CUDA [5] to increase efficiency.

A possible improvement to the MCC could be implemented by taking further advantage of the MIP-hierarchy. Each MIP-level of the texture, except the highest one (MIP-0), has a corresponding version of lower resolution. Instead of compressing absolute color values, differences between MIP-levels and up-sampled versions of their lower resolution coun-

terparts could be compressed. This is similar to DC coefficient encoding using estimates. As these differences are more likely to be similar than absolute values between faces and edges, this could result in greater compression ratios achieved with entropy coding.

The DDCT improves compression of blocks that contain directional edges not aligned with the vertical or horizontal axes by using transformation directions that align more closely with them [9]. Because of the hexagonal shapes of texels in the mesh colors framework, each texel will have 6 neighbors whereas a square texel has 4. This means transform directions can assume three different directions, as opposed to 2 in the case of square texels, that do not require turns or shifts as those seen in various modes of the DDCT (e.g. diagonal down-right mode as shown in Figure 2.11). Using this fact, some of the gains provided by the DDCT should be possible to reproduce for mesh colors data by choosing different reference vertices.

The developed MCC compresses edge and face data elements but is not concerned with compression of vertex data as single data elements are not affected by the DCT. To ensure compression of all color data, and further increase the compression ratio, it is possible to pair the MCC with various compression techniques developed for vertex color data. Such techniques include, among others, prediction-based methods or methods achieving compression via use of mapping tables [4].

Because of its similarity to JPEG, the MCC could be improved with various techniques aimed at improving the former codec. For instance, the DC separation and ΔDC method could be incorporated to the MCC to improve the quality of the reconstructed data [3].

Bibliography

[1] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, Jan 1974.

[2] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, 1974.

[3] Peter Kauff and Klaas Schuur. Shape-adaptive DCT with block-based DC separation and ΔDC correction. *Circuits and Systems for Video Technology, IEEE Transactions on*, 8:237 – 242, 07 1998.

[4] Ho Lee, Guillame Lavoué, and Florent Dupont. New methods for progressive compression of colored 3d mesh. 01 2010.

[5] NVIDIA. Cuda faq. https://developer.nvidia.com/cuda-faq.

[6] K. Rao and P. Yip. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Boston: Academic Press, 1990.

[7] A. Said. Introduction to arithmetic coding - theory and practice. http://www.hpl.hp.com/techreports/2004/HPL-2004-76.pdf [Accessed 2014-06-10].

[8] Cem Yuksel, John Keyser, and Donald H. House. Mesh colors. Technical report, Department of Computer Science, Texas AM University, 2008.

[9] Bing Zeng and Jingjing Fu. Directional discrete cosine transforms for image coding. In *IEEE International Conference on Multimedia and Expo*, 2006.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2019-06-11

EXAMENSARBETE Kompression av ytdata på dynamiska topologier
STUDENT Oguz Taskin
HANDLEDARE Jan Schmid (DICE)
EXAMINATOR Michael Doggett (LTH)

Kompression av ytdata på dynamiska topologier

POPULÄRVETENSKAPLIG SAMMANFATTNING **Oguz Taskin**

Mesh Colors är en alternativ textureringmetod för texturering av 3D modeller som undviker problem som sömmar och diskontinuiteter genom att associera data direkt med enskilda trianglar. I detta arbete har en kodek för komprimering av Mesh Colors texturer utvecklats.

Mesh colors tilldelar varje hörn, kant och yta av trianglar ett antal element som bestäms av den upplösning som anges triangeln. Till skillnad från traditionella texturer lagras dock mesh colors data endimensionellt.

JPEG är ett välkänt och ett ofta använt format för kompression av bilder. Formatet är en standard för destruktiv komprimering, dvs. där den dekomprimerade bilden har en försämrad bildkvalitet. Detta görs dock i enighet med det mänskliga synsystemets egenskaper vilket resulterar i, beroende på kompressionsgrad, återskapade bilder som uppfattas ha minimal förlorad kvalitet. Bilddata partitioneras till block av data som transformeras till frekvensdomänen med diskret cosinustransform. Transformerad data kvantiseras, dvs. lagras med mindre precision, där koefficienter som representerar lägre frekvenser lagras med högre precision. Det faktum att det mänskliga synsystemet är mindre känsligt för förvrängningar av högfrekventa egenskaper i bilder än lågfrekventa gör att detta steg inte får den återskapade bilden att se särskilt förvrängd ut.

Den utvecklade mesh colors-kodeken är lik JPEG-standarden, dock med vissa avvikelser. Element i mesh colors textureringsalgoritmen som associeras med hörn är en del av 3D-modellen och behöver inte lagras med resterande data. De element som positioneras på kanter och ytor komprimeras i två separata pass där kantelement bearbetas först. Dessa partitioneras in till längder av element och transformeras endimensionellt innan kvantisering. Element som positioneras på ytor partitioneras in till tvådimensionella block och transformeras tvådimensionellt för att sedan kvantiseras. På grund av den triangulära formen av ytor resulterar partitionering av associerade element i block där diagonaler av element eventuellt kan saknas.

Den implementerade mesh colors-kodeken uppnår en grad av kompression jämförbar med JPEG. Då man jämför den uppmätta kvaliteten för dekomprimerad mesh colors-data och texturfil i JPEG där båda är genererade från samma källa kan det observeras att mesh colors kodeken presterar marginellt bättre upp till en viss kompressionsgrad men även att kvaliteten för mesh colors-data minskar snabbare än JPEG-filer för ökande kompressionsgrader. Dessa resultat visar att det går att uppnå JPEG-grad av kompression för mesh colors-data och att dess otraditionella struktur inte är ett hinder för att nå relativt goda kompressionsresultat.