

# UC Davis

## IDAV Publications

### Title

Compression Methods for Visualization

### Permalink

<https://escholarship.org/uc/item/6rq8w34g>

### Journal

Future Generation Computer Systems, 15

### Authors

Gross, Markus

Lippert, L.

Stadt, Oliver G.

### Publication Date

1999

Peer reviewed

# Compression Methods for Visualization

Markus H. Gross<sup>†</sup>, Lars Lippert<sup>‡</sup>, Oliver G. Staadt<sup>†</sup>

<sup>†</sup>Department of Computer Science  
ETH Zürich, Switzerland  
E-Mail: {grossm, staadt}@inf.ethz.ch  
<http://www.cg.inf.ethz.ch/>

<sup>‡</sup>Alcatel Telecommunications, Zürich, Switzerland  
E-Mail: lars.lippert@alcatel.ch

## ABSTRACT

Compression methods have become of fundamental importance in almost every subfield of scientific visualization. However, unlike image compression, advanced visualization applications impose manifold constraints on the design of appropriate algorithms, where progressiveness, multiresolution or topology preservation are some of the key issues. This paper demonstrates the importance of multiresolution compression methods for visualization using two examples: The first, compression domain volume rendering, enables one to visualize volume data progressively and instantaneously from its compressed data format and has been designed for WWW and networked applications. The second one is a multiresolution compression and reconstruction method that allows for progressive coding, transmission and geometric reconstruction of surfaces and volumes. Both of the presented methods are so-called transform coding schemes and use wavelets for data representation.

## KEYWORDS

wavelets, multiresolution, volume rendering, mesh simplification, compression.

## 1 INTRODUCTION

### 1.1 Motivation

From the early days of computer science, data compression has been of fundamental importance for efficient representation, transmission, storage, and archival of large data volumes [7]. Thus, various strategies and different classes of compression schemes have been devised over the past decades. The most successful ones are almost ubiquitous and can be found in many high tech appliances.

Modern data compression [35], essentially distinguishes between *lossless* compression and *lossy* compression. Among the lossless techniques, very popular algorithms can be found, such as the Huffman or arithmetic coding, which exploit the statistical structure of the data. Others, like Ziv-Lempel-based methods, make use of repetitive data patterns and can be found in many operating system libraries.

One of the essential steps in lossy compression methods is *quantization*, by which the information loss is ultimately controlled. Sophisticated lossy compression methods, such as the JPEG [41] for image coding, perform compression in a transform domain, that is, they project onto Fourier,

cosine, PCA, or wavelet spaces before quantization and compression. By providing a more efficient data representation, they amplify the compression gain at a given loss.

The design of a compression method is highly application dependent and has to thoroughly balance competitive requirements, such as information loss and speed of decompression.

Due to its broad range of applications, image compression [6] has become paramount and countless algorithms have been devised to efficiently represent and compress still and moving images or video. Recent developments, as in MPEG-4 comprise so-called model-based image compression [2, 12] where image content is encoded individually. Many modern image compression methods are *progressive* and *incremental*, that is, they allow reconstructing the image successively as data comes in from the network.

Unlike image compression, the graphics and visualization community have neglected the importance of compression methods for quite a long time. Specifically, in [29] the importance of compression in visualization was pointed out, however, only recently, with emerging WWW and distributed applications, graphics researchers have faced the challenge of devising compression algorithms. [9], for instance, proposed a lossy compression scheme for meshes. An elegant data structure to represent and compress meshes was proposed by [21] and [32]. In [20], Rossignac presented a compression algorithm, which allows to efficiently represent both mesh geometry and topology. Conversely, [4] and [26] invented methods for compression domain volume rendering.

### 1.2 The Power of Hierarchy

Wavelets, as devised by approximation theory a decade ago [8, 5, 28], provide an extremely powerful method for various types of compression strategies. Specifically, they can be used in the context of transform coding algorithms, where the initial data sets are transformed into a wavelet representation prior to quantization and compression. The power of the wavelets lies in the combination of various useful mathematical properties, such as local support, vanishing moments, (bi-) orthogonality, progressive approximation, hierarchical setup, fast decomposition and reconstruction, error control and many others. Therefore, many lossy compression methods are based on wavelets. One of the most popular ones is the 'zero-tree' for image coding [37].

Consequently, wavelets had soon been discovered by the graphics and visualization communities for efficient data approximation [40] and have been used for many different applications, as for instance, global illumination [15], hierarchical meshing [27, 19], geometric modeling [13] or volume rendering [42, 30, 18, 24].

The major purpose of a wavelet transform embedded in a compression scheme is to approximate the data with as few nonvanishing coefficients as possible given a predefined information loss. The following pictures illustrate the approximation power of wavelets [24]. Here, semiorthogonal B-spline wavelets of polynomial degree 3 [5] had been used for volume data approximation. Fig. 1 shows a sequence of images computed with a decreasing amount of coefficients. For high compression rates, high frequency components are washed out and artifacts become visible. Specifically, the smoothness of the hierarchical basis functions is striking.

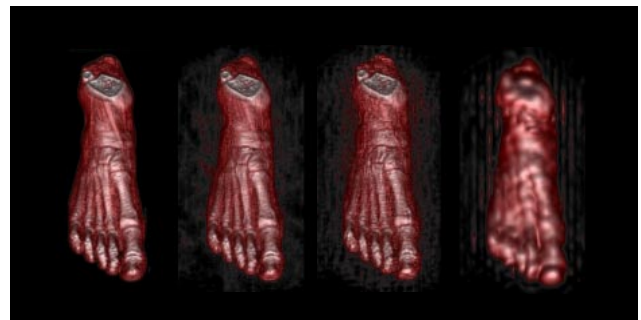
The localization properties of the wavelets are demonstrated in Fig. 2. Here, a filter operating in wavelet space allows controlling the approximation quality locally and has the effect of an electronic magnifier enhancing spatially interesting subregions. Obviously, compression gain and information loss are functions of the spatial position.

Similar methods can be developed for mesh generation and mesh control [19, 38]. The series of images depicted in Fig. 3 shows meshes, whose quality is controlled by an underlying wavelet representation of the data. Similar filter operations were applied on the data sets in the middle and right hand side image to influence the mesh quality.

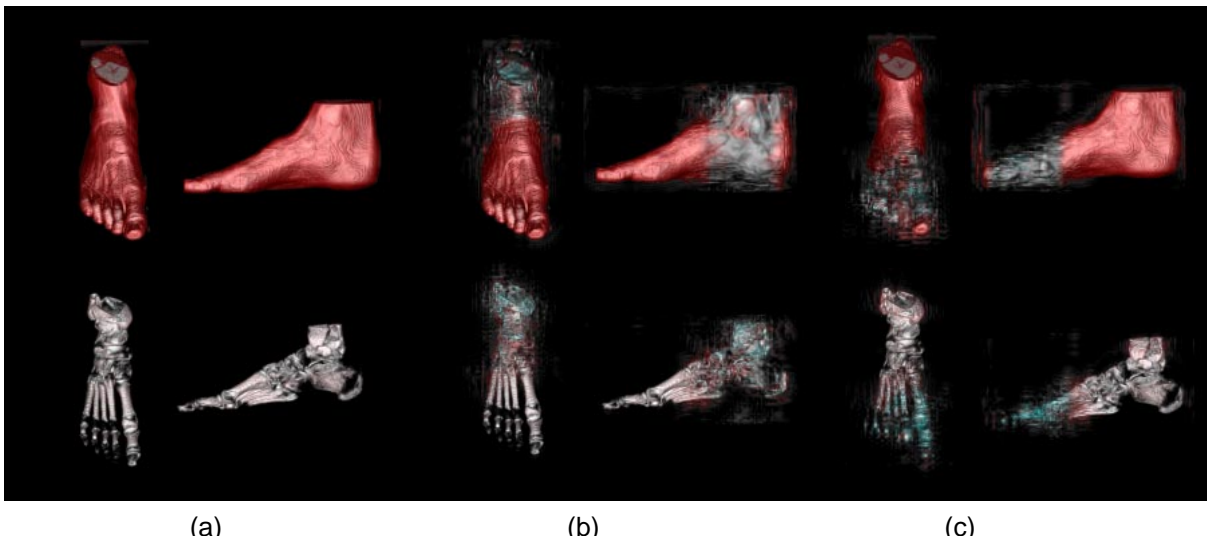
The purpose of the following paper is twofold: First, to point out the fundamental importance of compression methods for visualization and image generation and second, to illustrate how efficient compression methods can be designed using wavelets. Therefore, we present two differ-

ent examples of compression schemes devised by the authors: The first example describes a compression domain volume rendering method, such as presented in [26]. Here, volume data sets can be rendered instantaneously from a highly compressed file format that has been computed from a wavelet representation of the data. The method is especially designed for distributed applications and allows for progressive rendering as data comes in from the network. The second example elaborates on wavelet-based compression and visualization of surface and volume meshes and was presented by the authors in [38]. Again, the wavelet transform provides full approximation error control and progressive transmission of surface and volume data over networks. At the client side, piecewise linear representations with simplices can be computed from the decompressed data.

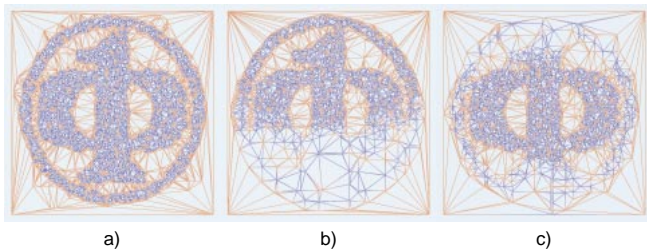
For reasons of brevity, we omit all mathematical details of wavelets.



**Figure 1:** Illustration of the approximation power of cubic B-spline wavelets: (a) original CT data set (100%), (b) 5.04%, (c) 1.92%, (d) 0.15% of nonvanishing coefficients (courtesy of Mallinckrodt Institute of Radiology, Washington University).



**Figure 2:** Localization properties of cubic B-spline wavelets: (a) original data set, (b) Magnifier centered at toe-region, (c) Magnifier centered at heel-region.



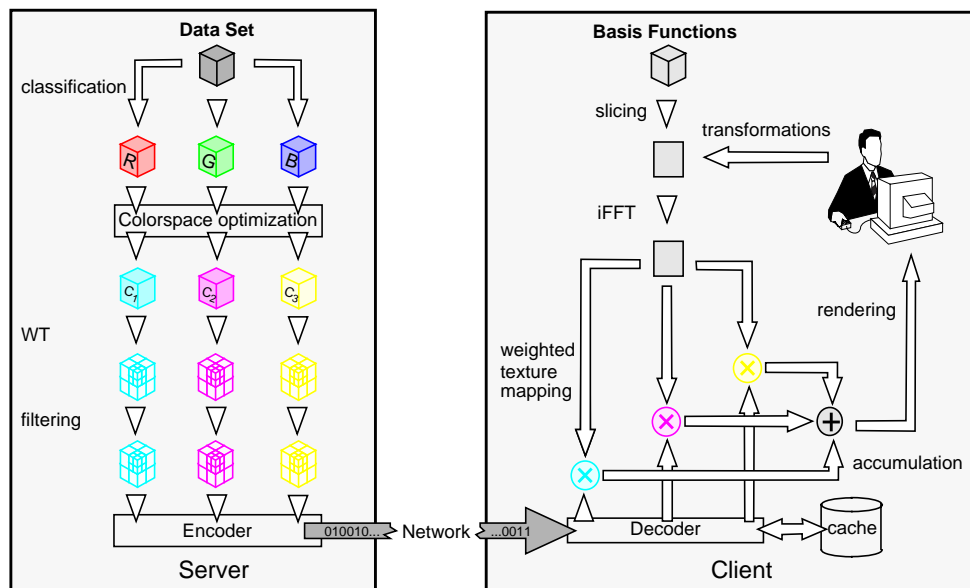
**Figure 3:** Localization in triangulations: a) Initial triangulation. b), c) Magnifier is centered at the upper and central area of the triangulation.

## 2 COMPRESSION DOMAIN VOLUME RENDERING

### 2.1 Overview

Our first example is targeted at networked applications where, for instance, a local client with low computational power browses through a remote volume database. Thus, in

order to transmit our volume data efficiently we have to find appropriate compression strategies. The underlying framework of the wavelet representation being used here proposes to develop an optimized compression technique that allows progressive transmission, decompression and direct rendering at interactive frame rates. Moreover, wavelet domain rendering avoids full decompression of the data prior to so-called splatting which itself as an image based method does not require to store the full volume data at the client's side. Although much research has been done on wavelet compression methods [40], [43] the specific needs of compression domain rendering encouraged us to develop a new compression pipeline which will be explained below.



**Figure 4:** Setup for distributed compression domain volume rendering.

Fig. 4 illustrates the data flow in our compression and rendering setup. The data preprocessing comprises five stages. It enables both intensity and RGB volumes to be handled which might be the result of an optional data classification step [3]. In case of RGB volumes the second step consists of a colorspace optimization which essentially decorrelates the data and allows color sensitive quantization. Next, a wavelet transform is performed independently on the three channels. Lossy compression is carried out by an oracle [16] operating locally or globally in the wavelet domain. The final step includes a data compression and encoding scheme to achieve a binary output stream that can be stored locally or transmitted directly through a network. Note that forward compression does not have any real-time constraints as opposed to the decompression.

### 2.2 Wavelet Splats

In order to understand our renderer, it is necessary to recall that in classic volume rendering, the amount of light  $I(t_L, \mathbf{x}, \mathbf{s})$  received at point  $\mathbf{x}$  from direction  $\mathbf{s}$  up to a ray length  $t_L$  is computed as:

$$I(t_L, \mathbf{x}, \mathbf{s}) = \int_0^{t_L} q(\mathbf{x} + t \cdot \mathbf{s}) e^{-\int_0^t \alpha(\mathbf{x} + s\mathbf{u}) du} dt \quad (1)$$

where  $q$  denotes the volume source term and  $\alpha$  the opacity function. For an isotropic medium with constant opacity  $\alpha$  equation (1) reduces to

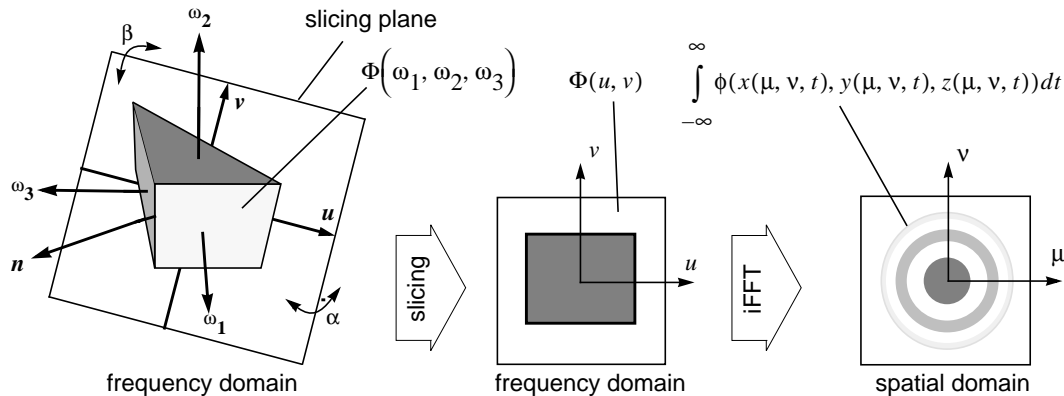
$$I(t_L, \mathbf{x}, \mathbf{s}) = \int_0^{t_L} q(\mathbf{x} + t \cdot \mathbf{s}) e^{-\alpha t} dt \quad (2)$$

Note that for  $\alpha \equiv 0$  we end up in a X-ray-like image.

Especially in those cases, splatting has proved its usability for fast volume rendering. In contrast to ray-casting, it allows one to reduce the computational complexity for interpolation and integration to a minimum, since the pre-projected footprints of high-order interpolation functions can be stored as lookup tables. The projections themselves can be computed with an accurate quadrature technique. Besides hierarchical splats [23], wavelet splatting [25] is a sophisticated extension.

In wavelet splatting, the renderer computes the projection such as defined in (2). Taking into account the wavelet decomposition level  $m$  up to  $M$  and moving the summation outside the integral, the formulation collapses to:

$$I(\infty, \mathbf{x}, \mathbf{s}) = \sum_{m=1}^M \sum_{\substack{\text{type} = 1 \\ p, q, r \in Z}}^7 d_{mpqr}^{\text{type}} \int_{-\infty}^{\infty} \psi_{mpqr}^{3, \text{type}}(\mathbf{x} + \mathbf{s}t) dt \\ + \sum_{p, q, r \in Z} c_{Mpqqr} \int_{-\infty}^{\infty} \phi_{Mpqqr}^3(\mathbf{x} + \mathbf{s}t) dt \quad (3)$$



**Figure 5:** Illustration of the Fourier projection slicing theorem in 3D for an idealized wavelet.

Once the renderer builds the viewpoint dependent integral tables, the screen position of the table is calculated, mapped, weighted by the wavelet coefficient and accumulated into the framebuffer. Since the basis functions of different iteration levels  $m$  differ by dilation, only eight different splats of depth  $M$  have to be calculated. All other footprints are derived by subsampling in the spirit of a mipmap. Correct sampling and optimized data-structures of the calculated splats are discussed in [18].

### 2.3 Multiview Rendering

One way to overcome the drawback of missing occlusion in X-ray images is the introduction of a multiview arrangement. Here, the volume is rendered simultaneously from different directions and presented to the user in a single win-

where  $d_{mpqr}^{\text{type}}$  denote the wavelet coefficients of decomposition level  $m$  at the spatial position  $p, q, r$  and wavelet-type  $\text{type}$  and  $c_{Mpqqr}$  the coefficient of the scaling function  $\phi$  of level  $M$ . The computation of the line integrals for a particular view can be accomplished by Fourier projection slicing (FPS). It allows to compute accurate projections of any basis function. This theorem states that the 2D Fourier transform of a projection of a function  $f(x, y, z)$  onto a given plane  $\mathbf{P}$  equals a plane that slices the Fourier transform  $F(\omega_1, \omega_2, \omega_3)$  parallel to  $\mathbf{P}$  and intersects the origin.

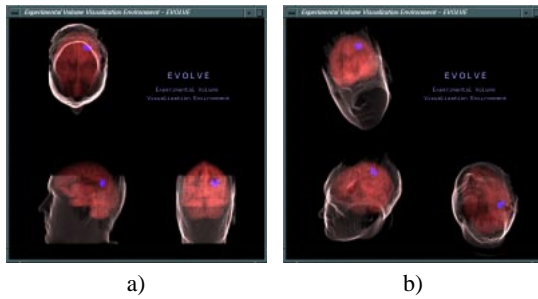
Since many wavelet types such as B-splines come along with closed form representations in the frequency domain, it is straightforward to apply this theorem to get the required splats. Fig. 5 depicts the setting, where an inverse FFT processes the slices to obtain the wavelet splat.

The intersection plane spanned by  $\mathbf{u}, \mathbf{v}$  defines the 2D Fourier transform of the texture splats  $I(u, v) = F(\omega_1(u, v), \omega_2(u, v), \omega_3(u, v))$ , whereas the normal vector  $\mathbf{n}$  of the plane equals the direction of the projection. The definition of the viewing parameters is figured out in spherical coordinates  $(\alpha, \beta)$ .

Exploiting the coherence of different viewing angles given by the symmetry of tensor-product constructions, the basic single-view splatting approach can be extended to a multiview renderer without computational overhead for splat calculation [26].

The triple view rendering is illustrated in Fig. 6, where a classified data set is displayed from three directions. The image was grabbed directly from the screen as it is displayed to the user. Skin is colored white, brain tissue red and a tumor is colored in blue.

In our implementation the software decompressor works at a rate of  $\sim 30$  k wavelet-coefficients/sec. on an Indy R4400 workstation and allows on-line decompression. The renderer splats the computed footprints weighted with the decoded wavelet-coefficients directly into a software or



**Figure 6:** Triple views seen from different viewing angles. (volume size: 256 x 128 x 256) a)  $\alpha = \beta = 0.0$ . b)  $\alpha = 0.66, \beta = 0.91$ .

hardware accumulation buffer. In addition, our framework incorporates a local cache to store coefficients at the client’s side. The required splats are computed locally. Since the wavelet coefficients are transmitted in significance order the rendering quality is fully controlled by a user-defined framerate, the client’s hardware and, if no cache mechanism is enabled, also by the bandwidth of the network. Hence, this concept balances CPU, network and graphics performance and allows scalability. In a minimum configuration we have to provide client storage only for the eight mother-wavelet splats and three Huffman tables. Together they take less than 5KB of memory even for huge data sets. This allows the scheme to run on clients such as the upcoming network computers.

## 2.4 Colorspaces for Compression

If the initial volume is given in RGB, it is critical to transform the volume into an optimized colorspace prior to compression. Here, we assume the optimized space to be spanned by the three vectors  $C_1, C_2$  and  $C_3$ . This allows to assign an additional significance to each vector. As a result we get two independent significance weights per coefficient which affect encoding and quantization. The first weight is defined by the energy of the associated function [16]. The second one is a global significance determined by the colorspace-coordinates. For instance, a coefficient with the coordinates (1,0,0) is regarded as more relevant than a coefficient (0,1,0), if the vector  $C_1$  is considered to be more significant than  $C_2$ .

In addition to RGB we employ two colorspace: The first one is data-independent and equals the YIQ-colorspace obtained by a simple matrix transform [34]. The  $Y$  component encodes the luminance information, whereas the chromaticity is encoded in  $I$  and  $Q$ . We followed the NTSC bandwidth conventions and assigned a quantization factor 4 to  $Y$ , 1.5 to  $I$  and 0.6 to  $Q$ . In practical use this colorspace allows to compute a black and white image ( $Y$ ) as a rough sketch and to refine color progressively. Thus, progression is figured out both in the spatial *and* in color domain. Alternatively, our second colorspace is calculated by a statistically optimal principle component analysis (PCA) or Karhunen-Loève expansion [14] whose matrix has to be computed individually for each data set. Although the solutions of the eigenproblems are computationally more chal-

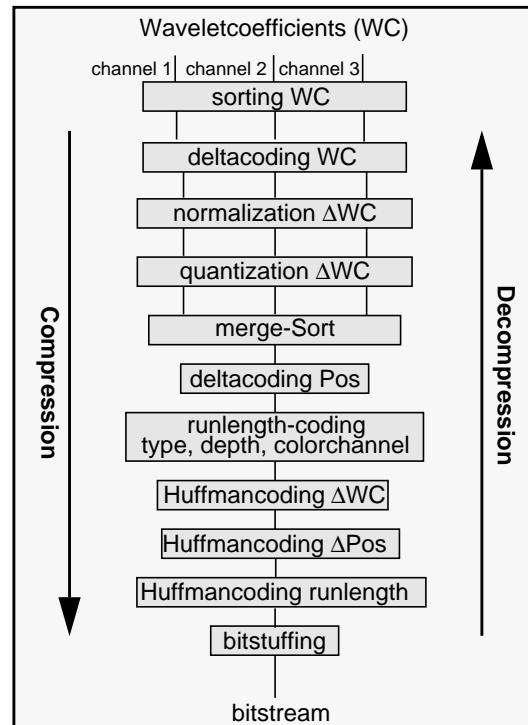
lenging on offline forward compression, yet they provide better results. In this case the absolute values of the eigenvalues are taken to describe the significance of the corresponding eigenvector. Note that this step can be skipped for intensity volumes, such as raw CT or MRI data sets.

**Table 1:** Colorspace significance assignments for different color coding schemes

COLORSPACE	IMPORTANCE $I$		
	$I(C_1)$	$I(C_2)$	$I(C_3)$
RGB	1	1	1
YIQ	4	1.5	0.6
EIGEN-VECTORS (PCA)	EIGEN-VALUE 1	EIGEN-VALUE 2	EIGEN-VALUE 3

## 2.5 Data Compression Pipeline

The algorithmic steps for data compression are executed sequentially in the pipeline summarized in Fig. 7 and convert the wavelet transformed data sets into a sequential bitstream.



**Figure 7:** Pipeline representing the individual steps of the compression scheme.

Therefore, we start with a sorting operation that generates a sequence of coefficients and positional data in significance order. The significance score  $S$  is determined for each color channel  $C \in \{C_1, C_2, C_3\}$  and coefficient  $w(C) \in \{d_{mpqr}^{type}(C), c_{Mpq}(C)\}$  individually according to its associated wavelet energy  $E$  and color channel importance  $I$ .

$$S(w, C) = E(w(C)) \cdot I(C) \quad (4)$$

Table 1 summarizes the importance factors for the three different colorspace. For each color channel ( $C$ ), wavelet type (*type*) and decomposition level ( $m$ ) a deltacoding, normalization and quantization operation is performed sepa-

rately depending on the individual ranges of the coefficients  $w$ . More precisely, if quantization is restricted to  $P$  bits for a given sequence of  $N+1$  significance sorted non-zero wavelet and scaling function coefficients  $(w_n(C, m, type))_{n=0, \dots, N}$ , we compute the coefficient's delta factor  $\Delta(C, m, type)$  as:

$$\Delta(C, m, type) = \frac{\max_{n=0, \dots, N-1} (|w_n(C, m, type)| - |w_{n+1}(C, m, type)|)}{2^P} \quad (5)$$

This factor is used to store the coefficient's range with respect to the assigned bytes. Thus, it has to be transmitted once for each wavelet type, decomposition level and color coordinate. In contrast, the normalized and quantized difference of two coefficients  $\delta$  has to be transmitted for each coefficient individually. It is computed as:

$$\delta(C, m, type, n) = \text{round}\left(\frac{(|w_n(C, m, type)| - |w_{n+1}(C, m, type)|)}{\Delta(C, m, type)}\right) \quad (6)$$

Upon reconstruction we end up with approximated wavelet and scaling function coefficients  $\tilde{w}_n(C, m, type)$ ,  $n = 1, \dots, N$  which can be computed by:

$$\tilde{w}_{n+1}(C, m, type) = \text{sign}(n+1)(|\tilde{w}_n(C, m, type)| - \delta(C, m, type, n) \cdot \Delta(C, m, type)) \quad (7)$$

Since the coefficients are sorted according to their individual scores we optimize the residual approximation error as a function of the parameters introduced above. Note that the sign of each coefficient is encoded by an additional flag, referred to as  $\text{sign}(n)$ .

We choose to limit the quantization  $P$  to eight bits, since standard framebuffer use eight bits for  $\text{RGB}\alpha$  each. However, observations in practice encourage us to reduce quantization to even three bits without significant loss of visual quality (see Section 2.6). The three data sequences are merged and sorted according to the scores of their wavelet coefficients. In particular, the sorted sequence requires for each coefficient to encode additionally its spatial position, wavelet type and color-channel in a lossless scheme. In order to overcome the drawbacks in compression performance arising from this requirement we introduce an additional spatial clustering mechanism [26]. That is, we balance spatial coherence and the energy-based sorting order of the coefficients. Note that clustering also speeds up the rendering process, since splats outside the field of view can be detected easily and skipped without further computation.

Additional runlength-codings of wavelet type, decomposition depth  $m$  and colorchannel are performed and transmitted as variable length Huffman tag codes. The third Huffman-table encodes the deltas  $\delta$  of wavelet-coefficients. These three tables together take about 2kBytes and have to be transmitted separately prior to the data. In addition the transmitted meta-data includes information about the basis vectors of the colorspace, maximum depth of the wavelet transform ( $M$ ), exact initial wavelet coefficients  $(w_0(C, m, type))$  and the coefficient delta factors  $(\Delta(C, m, type))$ .

The reconstruction scheme has to decode all required information, such as the spatial position, wavelet type, depth  $m$ , colorchannel and the data value. For computational efficiency, we propose to precompute 10-bit Huffman look-up tables.

Fig. 8 illustrates a fraction of the bitstream as generated by our method. The number of bits varies as a function of the individual Huffman codes.

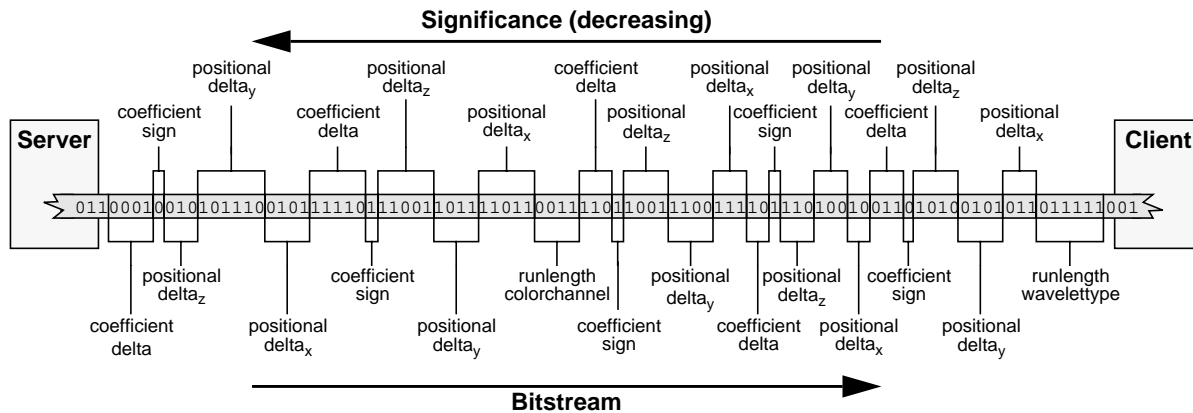


Figure 8: Fraction of the bitstream generated by the compression scheme

## 2.6 Examples

To investigate the performance of the proposed method, we applied the approach to the  $\text{RGB-Visible Human}$  data set of size  $128 \times 128 \times 128$  voxels ( $3 \times 8$  bits/voxel). The wavelet decomposition was performed with Haar wavelets up to level  $M=3$ . In order to quantize image quality we define an

$L^2$  image measure  $Q_I$  conforming to the signal-noise ratio ( $\text{SNR}$ ) in [dB] well known from signal processing applications as:

$$Q_I = 20 \cdot \log_{10} \left( \frac{\sum_{pixel} \sum_{color} [i_{ref}(pix, col)]^2}{\sum_{pixel} \sum_{color} [i_{im}(pix, col) - i_{ref}(pix, col)]^2} \right) \quad (8)$$

where,  $i_{im}(pix, col)$  denotes the intensity of a given *pixel* of the computed image for the *color*-component in RGB-colorspace, e.g.  $col \in \{R, G, B\}$  or  $col \equiv Y$  respectively. Note specifically that in image compression ratios  $> 40$  dB refer to reasonable visual qualities and at ratios  $> 60$  dB images are perceived as „noise-free“. The reference image was generated by the proposed splatting method for  $M=0$  and 100% of the coefficients. We observe that ratios  $> 60$  dB are achieved at compression gains of almost 95%.

The colorplates in Fig. 10 display the image quality achieved from the RGB data set for different colorspace and compression rates with respect to the original data size of 6291456 bytes. The qualitative differences of the three colorspace reveal mostly for small datasizes. Note that YIQ favors the *Y*-component and renders greyscale images at high compression rates. This is contrasted by the RGB color space where the method reconstructs the volume both in the spatial and colorspace domain and ends up in a poorer image quality. Finally the best results are obtained by the

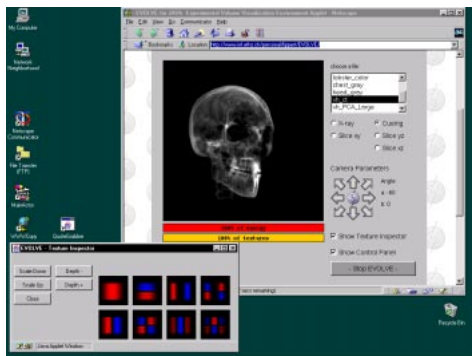
PCA-based color representations. However, as progression proceeds the representations converge to each other. It is clear that the entropy of the color information is lower than in the *Y* channel. Therefore, we observe in general higher compression gains (SNRs) on color volume compression.

Table 2 summarizes the performance of our algorithm. Timings are given for a SGI-Indy workstation (MIPS R 4400/150 MHz) and a SGI Maximum Impact workstation (MIPS R10000/195 MHz). Both workstations use our software-accumulation scheme as introduced in [18]. The resolution of the rendered image was 160x180 pixels. For the delta-coding of the wavelet coefficients we assigned three bits. The timings reveal, that we still achieve interactive framerates for fast previewing. Note in particular, that competitive high quality renderers, such as shear warp factorization [22] are significantly slower at these data sizes and require careful setting of the transfer function for speed-up. Our proposed splatting technique is well-suited for hardware support [25]. The hardware assisted accumulation of the calculated splats is done within the accumulation buffer or uses alpha-blending operations, depending on the available hardware platform. Hardware support allows to further increase the rendering speed significantly, especially for the generation of high resolution images.

**Table 2:** Performance of the method.

	RGB			YIQ			COMPUTED BY PCA		
	# COEFF	TIME (INDY) IN [SEC.]	TIME (IMPACT) IN [SEC.]	# COEFF	TIME (INDY) IN [SEC.]	TIME (IMPACT) IN [SEC.]	# COEFF	TIME (INDY) IN [SEC.]	TIME (IMPACT) IN [SEC.]
4.6 KBYTE	2155	0.44	0.15	2175	0.46	0.15	2184	0.45	0.15
9.4 KBYTE	4480	0.9	0.31	4221	0.93	0.3	3820	0.86	0.26
79 KBYTE	31714	3.21	1.14	35655	3.06	1.09	30554	2.61	0.9
368 KBYTE	170761	11.52	4.01	178065	12.55	3.75	172240	10.77	3.58

Fig. 9 shows a Java applet as running on a standard WWW browser. Note again that the method renders instantaneously as the bits arrive and does not require full volume expansion at the client side.



**Figure 9:** JAVA-Applet for compression domain volume rendering (URL: <http://www.inf.ethz.ch/department/IS/cg/html/research/evolve/>)

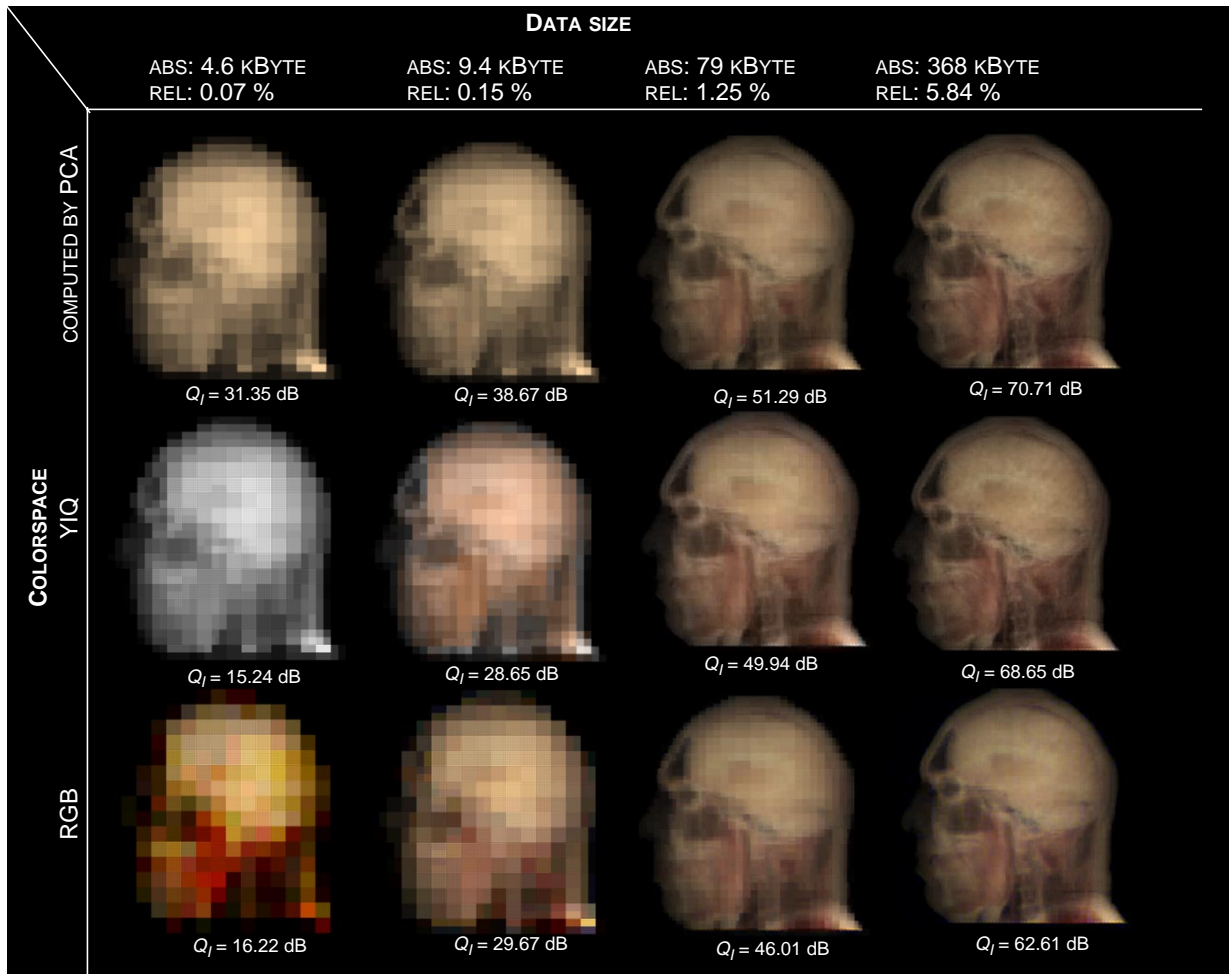
### 3 COMPRESSION AND GEOMETRIC RECONSTRUCTION

#### 3.1 Overview

In our second example, we present a framework for multi-resolution compression and geometric reconstruction of arbitrarily dimensioned data designed for distributed applications. Although being restricted to uniformly sampled data, our versatile approach enables the handling of a large variety of real world elements, such as nonparametric, parametric and implicit lines, surfaces or volumes.

Here, we designed a compression/decompression pipeline as depicted in Fig. 11. The forward compression proceeds as follows: After extraction of constraints, the data set is normalized, wavelet-transformed and both local and global approximation errors are controlled by the oracles introduced above. Sorting of the individual channels of the WT transforms the multidimensional array into a 1D data vector which is quantized and encoded subsequently. Line-constraints, as extracted earlier, are fed into a lossless compress-





**Figure 10:** Progressive compression in three different colorspace. (data source: [31]) volume size:  $128^3$ , max. decomposition level  $M=3$ .

sion scheme. Conversely, the decompression pipeline inverts the procedure and prepares the data for subsequent geometric reconstruction.

### 3.2 Progressive Lossy Compression

First the data is normalized, i. e. the values are scaled to  $[0, \dots, 1]$ . In order to prepare the data for bandwise progressive transmission, we sort the multidimensional coefficient array into a 1D vector as displayed in Fig. 12. Here, the array is traversed from the most significant scaling function coefficients to the high frequency bands representing fine grained detail.

Note that the vector contains floating point values and has to be converted into an array of integers.

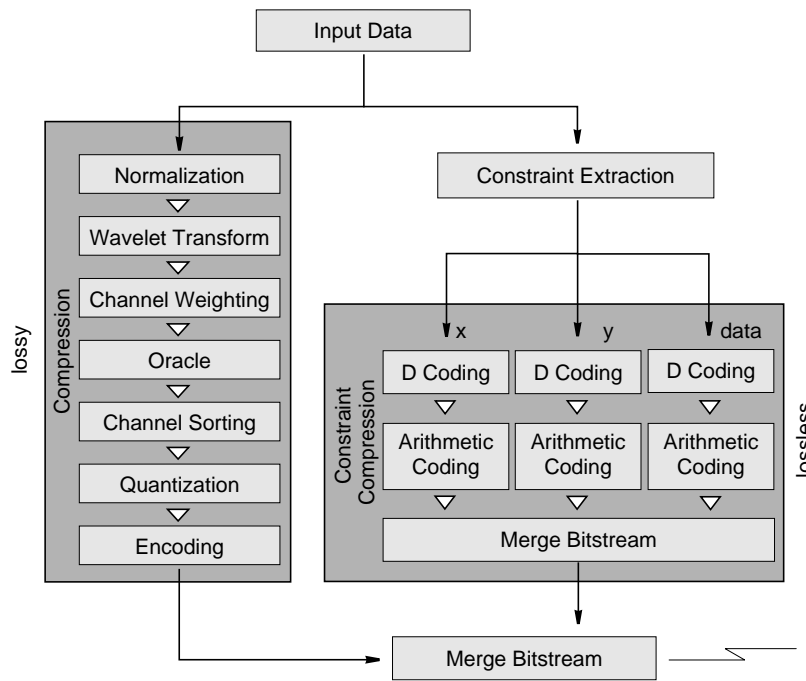
The quantization step comprises a multiplication of the initial floating point coefficients with a factor of  $2^{n-1}$ , where  $n$  represents the number of bits to be assigned for each coefficient. Subsequent rounding operations transform the floating point value into signed integer formats of size  $n$ .

Let  $c_{\text{float}}$  be a coefficient, we obtain it's quantized version  $c_{\text{quant}}$  by

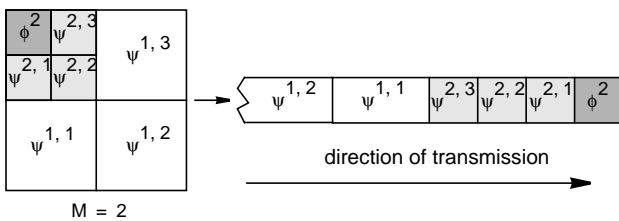
$$c_{\text{quant}} = \text{round}(2^{n-1} \cdot c_{\text{float}}). \quad (9)$$

Note that  $n$  strongly affects the quantization error and appears as noise after reconstruction. Lossless quantization would typically require 23 bits on a 32 bit machine for single precision due to the normalized IEEE-754 floating point format.

The major task in the proposed compression is to convert the quantized integer vector into a bitstream of data. Therefore, we employ an entropy coding scheme in the spirit of JPEG [41]. Assuming that many of the coefficients will equal zero, encoding is carried out as follows: All non-zero coefficients are represented by 2-tuples, where the first element represents the number of bits of the second one. The second element contains the data value itself. All negative numbers are thus replaced by their absolute values, where in the case of a positive number the first bit is cleared. This enables the encoding of the sign. Let's say to encode a value of 17 we get (5, 00001), whereas to encode -17 we obtain (5, 10001). Similarly, 5 is represented by (3, 001), whereas -5 is converted to (3, 101). Note specifically that since the number of bits is known in advance, the representation is unique and the additional encoding of the sign bit in the most significant bit is possible.



**Figure 11:** Compression pipeline including both lossless and lossy data compression. For decompression, all of the above steps have to be reversed.



**Figure 12:** Conversion of the multidimensional array into a 1D coefficient vector depicted for a 2D WT.

Zero valued coefficients are encoded differently. Here we recommend a runlength coding up to a length of  $2^5 = 32$  which generates a set of 32 new symbols. These symbols, together with the first part of our 2-tuples, are stored in a Huffman-table which has essentially 64 entries. The Huffman symbols are as follows:

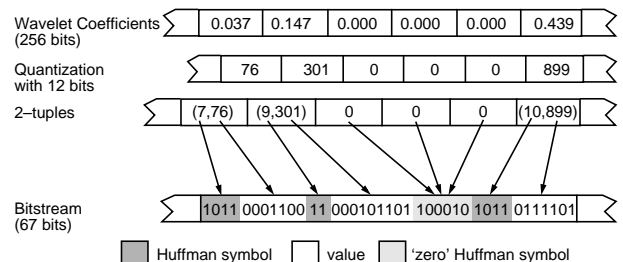
- Symbols 0 – 30: First element of a 2-tuple minus 1
- Symbol 31: ‘EOB’ (End Of Bitstream)
- Symbols 32 – 63: Runlength of ‘zero’-coefficients

The scheme proposed here compromises the complexity of the Huffman-table with the maximum number of zero coefficients (32) to be encoded in one symbol. The ‘EOB’ Symbol usually allows the encoding of long sequences of ‘zero’-coefficients in the least significant positions of our data vector. However, it is only used where the Huffman table has not been built individually. The following pseudo-code illustrates the procedural flow of the scheme:

```
// N: total number of integer coefficients
// di: coefficient i
// huffleni: length of Huffman-code for symbol i
// huffcodei: Huffman-code for symbol i
// WriteBits(l,i):
//     appends the last l bits of i to bitstream
// Make2Tupel(i,first,second):
```

```
// converts integer into 2-tuple
i ← 0;
while i < N do
  if di = 0 then
    j ← 0;
    while j < 32 && di ← 0 do inc(i); inc(j); end;
    WriteBits(hufflenj+31, huffcodej+31);
  else
    Make2Tupel(di, first, second);
    WriteBits(hufflenfirst-1, huffcodefirst-1);
    WriteBits(first, second);
    inc(i);
  end;
end;
WriteBits(hufflen31, huffcode31);
```

In our framework the Huffman-table is generated individually for each data set upon compression and is transmitted along with the data and header information. Since the size of the table is fixed to 64 entries, this does not lead to a notable overhead. Another solution would be the employment of a generic table, such as in image compression which, however, drops the compression gain and, due to the variety of geometric data, is much more difficult to construct. An example of encoding a sequence of coefficients is given in Fig. 13.



**Figure 13:** Encoding a sequence of coefficients.

It should be stated again that progression is achieved channel by channel. That is, we transmit the low frequency scaling function coefficients first, followed by the wavelet coefficient channels in order of ascending frequency.

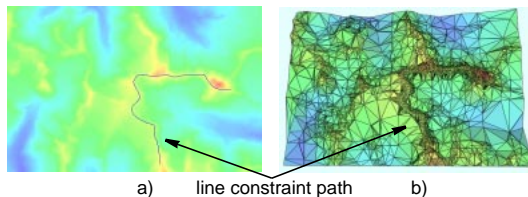
Some results of the lossy compression of a B-spline surface with different parameter settings are depicted in Fig. 21. In order to decompose the control points of this B-spline surface we used the pipeline explained in detail in [38]. We observe that quantization noise is seriously disturbing the B-spline surface. Sophisticated least-square estimators, such as the Wiener-Filter [17] allows one to remove noise and to reconstruct the surface.

**Table 3:** Comparison of the proposed method (encode) with some popular compression algorithms (3D volume data set of Fig. 20: 128x64x64 voxels).

ENCODE	8 BIT QUANT.		16 BIT QUANT.		CPU (IN S)
	50% COEFF. (IN KB)	10% COEFF. (IN KB)	50% COEFF. (IN KB)	10% COEFF. (IN KB)	
ENCODE	568	245	1,835	466	2
ZIP	618	290	2,399	660	5
ARC	711	300	2,727	764	13
URBAN	501	233	1,888	496	69
COMPRESS	533	253	2,407	607	3
UNCOMPRESSED	2,248	2,248	4,496	4,496	0

### 3.3 Compression of Constraints

In many cases it is desirable to compress spatially interesting features, such as boundary- or isolines and individual vertices in a lossless manner. We call these data *constraints*, since they usually constrain subsequent geometric reconstruction. In our pipeline we represent constraints as polylines or polygons. Fig. 14 illustrates the use of constraints in a digital terrain data set of the Swiss Alps. Here the geometric reconstruction, i. e. triangulation of the surface, was simplified up to a given bound. The constraints invoked by the polygon force the reconstruction to keep the triangulation dense. The constraint is imposed in terms of a terrain following polyline of a given extent.



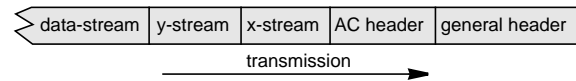
**Figure 14:** Illustration of constraints in a digital terrain data set. a) Interactive specification of the constraint path. b) Mesh after constraint insertion. (Data source: Courtesy *Bundesamt für Landestopographie*, Bern, Switzerland)

Assuming the polyline constraint is represented as a stream of vertices of type  $(x, y, data)$ , we employ a lossless compression strategy, as shown in Fig. 11.

The position  $(x, y)$  and the data value are encoded separately using both delta and higher order arithmetic compression algorithms. For details see [35].

The resulting bitstream format is presented below in Fig. 15, where two headers are followed by the individual  $x$ -,  $y$ - and data-streams.

Table 3 compares the proposed encoding scheme (*encode*) with some of the most popular lossless compression methods, like *zip*, *arc*, *urban* and *compress*. Note that information loss occurs only upon coefficient removal and quantization. Thus, all subsequent steps in our pipeline are lossless and can be compared with some standard algorithms. Results are given for a 3D volume data set, where the data was prequantized with 8 bits and 16 bits respectively. Interestingly, even in lossless mode our method competes with popular algorithms in overall performance.



**Figure 15:** Data format of the bitstream for constraint compression.

Any further details, such as the header formats of the bitstream, can be found in [38].

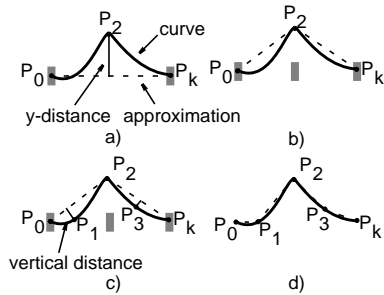
### 3.4 Vertex Removal Strategies in 1D

Vertex removal methods enable the client to compute geometric reconstructions adaptively and progressively from the incoming bitstream of data. When seeking an appropriate algorithm, computational performance and invariance to the dimensionality are important considerations. Due to the rich literature on vertex removal in graphics and computational geometry we found that the well-known algorithm of Douglas et. al. [10] is a good starting point. First, we briefly explain its initial form in a nonparametric 1D setting and illustrate its application in multiresolution representations. Here, special emphasis is given to extension of the method for progressive reconstruction. Next, we generalize the method to multidimensional cases and give some examples of how it works. The versatility of the introduced method imposes no restriction on subsequent triangulation methods, which can range from constraint Delaunay [33] to fast look-up tables [19].

In order to construct a point removal strategy, let's first consider the 1D setting. Here, the problem reduces to finding a strategy for the reduction of line segments in piecewise linear approximations. Inspired by the algorithm of [10] we extended these ideas and modified the method to a recursive and progressive algorithm, illustrated in Fig. 16. It starts by connecting the first point of a curve,  $P_0$ , with the last point  $P_k$ . All intermediate points representing the curve are compared against the line segment  $\overline{P_0P_k}$  and the point

with the largest distance, for instance  $P_2$ , is identified. If its distance exceeds a predefined threshold  $\epsilon_0$ , the vertex is considered *important* and labeled. We split the initial line segment in two halves, on each of which the algorithm can be applied recursively. Obviously, the quality of the removal can be controlled by the distance threshold. The advantage of this extension to the original method lies in the tree type refinement of the vertex analysis coming along with the recurrence relations.

The distance can be computed in different ways, where, however, the computation of the vertical distance, such as depicted in Fig. 16c, is computationally much more expensive for general multidimensional settings. Therefore, we recommend computation of the y-distance (see Fig. 16a) approximating nonparametric data.

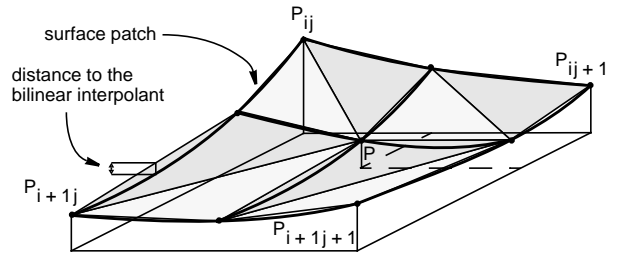


**Figure 16:** a) Recursive algorithm assuming a smooth representation of the underlying curve: a)  $P_2$  has largest vertical distance. b) new approximation after insertion of  $P_2$ . c) example for vertical distance measure. d) final result.

### 3.5 Generalizations to Multiple Dimensions

Generalizations of the method towards multidimensional nonparametric data is straightforward. Starting from an initial grid, as in Fig. 17, the algorithm seeks the vertex  $P$  with the maximum distance and subdivides the field into 4 (in 2D) or 8 (in 3D) subcells on which the method is applied recursively. In these cases the distances to the bilinear and trilinear interpolants of the cell vertices are computed, respectively.

Recalling the multiresolution B-spline approximation of the data motivates the extension of the algorithm towards a channelwise progressive point insertion. Therefore, the algorithm analyzes mesh vertices progressively and labels unimportant points as new data comes in. In 2D, for instance, the basic idea is to start from an initial vertex field



**Figure 17:** Extension towards multiple dimensions exemplified for nonparametric data: 2D version. A new vertex is inserted at position  $P$  and the distance is computed with respect to the bilinear-interpolant of  $P_{ij}, P_{ij+1}, P_{i+1j}, P_{i+1j+1}$ .

of resolution  $2^{m-M}$  in each direction, where  $M$  represents the maximum iteration. The vertices are provided by the scaling function approximation  $f^M(x, y)$  and are processed further by our algorithm. To define a distance metric, we assume a bilinear interpolant between the vertices which approximates the B-spline scaling function representation. If the difference signal  $\Delta f^m(x, y)$  is received, the resolution is refined by 2 and all newly inserted vertices are checked conforming to our distance metric. If required, they will be inserted.

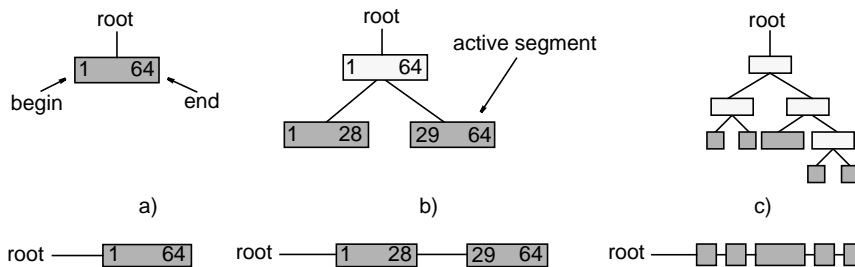
In order to compute the intermediate vertices for each iteration, an inverse wavelet transform has to be applied on all coefficients of a given iteration  $m$  as soon as they are received and decompressed.

An apparent drawback of this approach, however, deserves some attention: Once a vertex is labeled as important there is no way to reject it in subsequent steps. Obviously, the detail signals added during progression influence the importance of each vertex. Therefore, we recommend an exponential alignment of the threshold  $\epsilon_0$  to the iteration. That is if  $m$  stands for the current iteration step, the associated threshold  $\epsilon(m)$  is computed by

$$\epsilon(m) = \epsilon_0 \cdot e^{M-m} \quad (10)$$

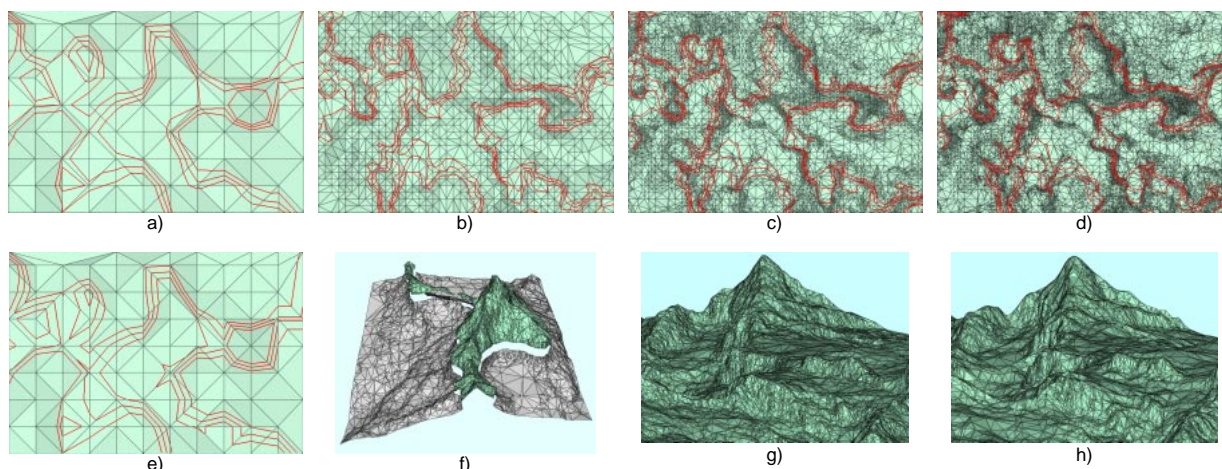
$\epsilon_0$ : global threshold governing the point removal.

In our implementation we employ a tree type data structure to maintain the individual cells representing the mesh. The tree grows iteratively as progression proceeds. After iteration, the leaves of the tree represent the remaining cells and can be triangulated with appropriate methods. Fig. 18 further elucidates the data representation.

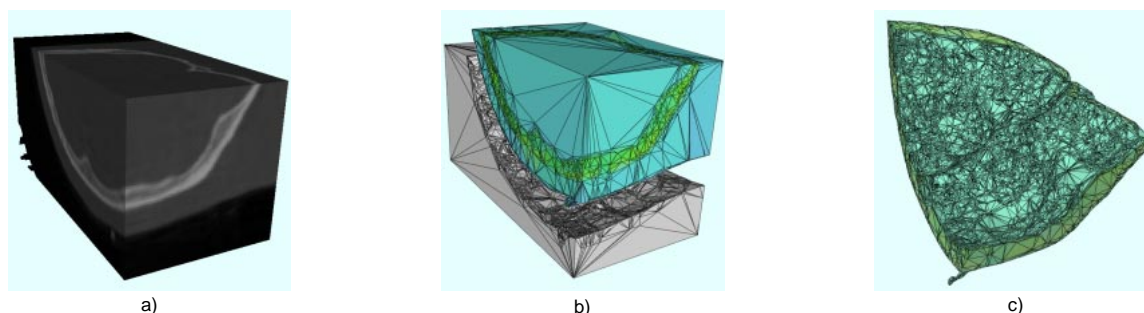


**Figure 18:** Construction of a 1D tree data structure with 64 vertices and its growth during progression. The equivalent list structure is given below. a) First segment at the beginning. b) Insertion of  $P_{29}$  causes split into two segments. c) Final tree after inserting all points.

For subsequent triangulations we employed the *qhull* library from [1] in 2D and 3D. An example of progressive point removal is depicted in Fig. 19, where the mesh is refined gradually with each wavelet channel arriving at the client side.



**Figure 19:** Extraction of isolines and interior surfaces from a digital terrain model of the Swiss Alps and progressive mesh refinement: 3 isolines are extracted for  $\tau = 120$ ,  $\tau = 125$  and  $\tau = 130$ , respectively. a)  $\epsilon_0 = 0.01$ , Wavelet channel 1, 0.1% triangles. b) Channel 3, 1.15% triangles. c) Channel 5, 5.80% triangles. d) Channel 7, 15.83% triangles. e) Standard isoline algorithm for channel 1. f) DTM split into interior and exterior regions at  $\tau = 130$ . g) 5% coeff., compression gain 1:33,  $\epsilon_0 = 0.0035$ , 62% triangles. h) 1% coeff., compression gain 1:100,  $\epsilon_0 = 0.0035$ , 62% triangles (data set courtesy of Bundesamt für Landestopographie, Bern, Switzerland).



**Figure 20:** Extraction of interior and exterior volumes. a) Initial CT volume data set with 2,704,000 tetrahedrons. b) Interior and exterior volumes,  $\tau = 42$  (skin surface), 133,091+ 34,290 tetrahedrons. c) Interior volume,  $\tau = 75$  (skull), 124,491 tetrahedrons.

## CONCLUSIONS

We presented two examples for compression schemes designed for applications in visualization. Both methods are lossy transform coding schemes and used wavelets for the underlying data representation. The advantages are obvious: High compression gain, precise approximation error control in  $L^2$ , localization, progressive refinement of the bitstream and very fast decompression – to name a few. The major drawback of wavelet based methods is their restriction to trivial data topologies. The current examples employed tensor product constructions, as they appear mostly in practice, however, more sophisticated multidimensional constructions are highly desirable. Although various extensions to spherical [36] or triangular domains [11] are emerging, much research has to be pursued to develop more powerful wavelets are necessary to cope, for instance, with complex boundary conditions.

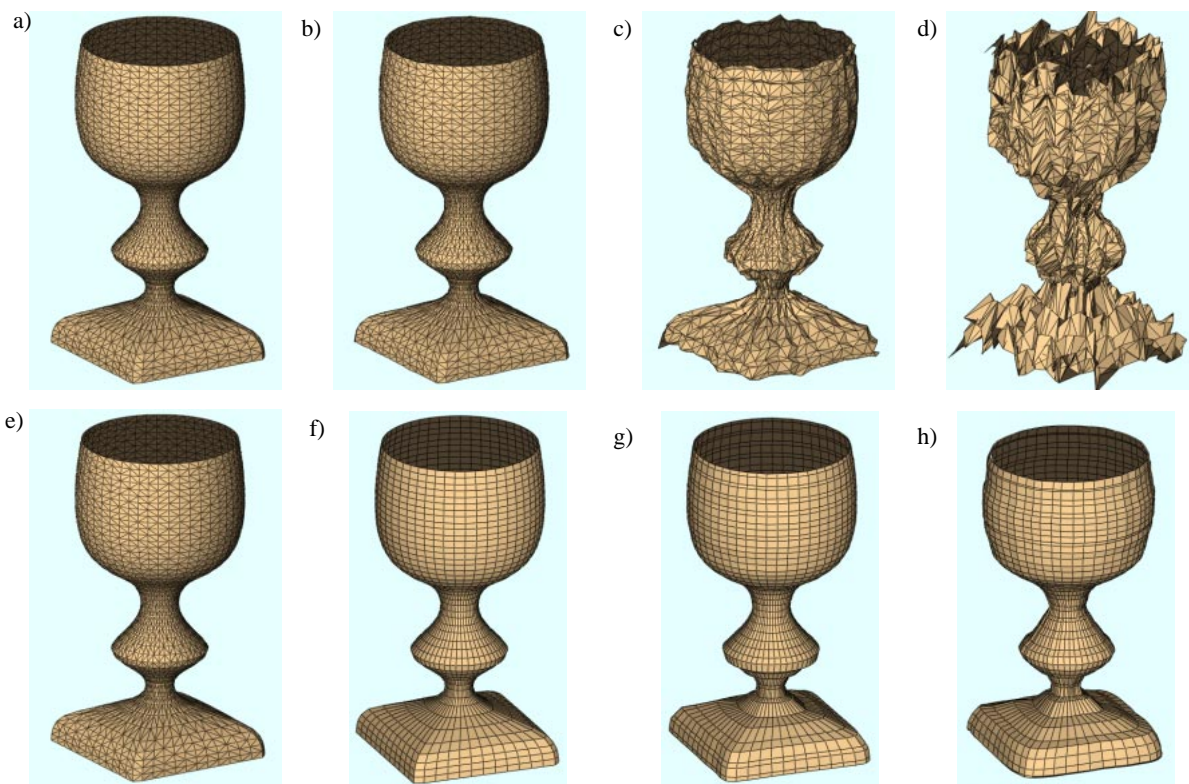
Apart from the hierarchical bases provided by the wavelets, mere multiresolution compression schemes, such as the progressive mesh [21] might bear much potential for compression, since they are less restrictive and give a fine grain control over the progression. A sequence of progressive tetrahedralizations is presented in Fig. 22, where an irregular turbine data set has been approximated with an increasing amount of simplices [39].

## ACKNOWLEDGEMENT

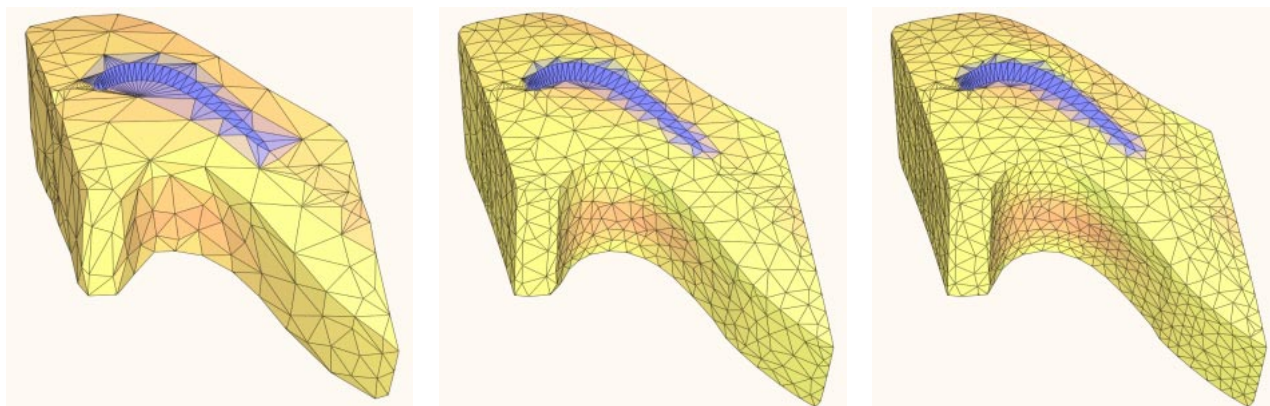
This research was supported in parts by the ETH research council under grant No. 41-2642.5.

## REFERENCES

- [1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. “Qhull,” 1996. <http://www.geom.umn.edu/locate/qhull>.
- [2] F. Bossen and T. Ebrahimi. “Region shape coding.” Technical Report ISO/IEC JTC1/SC29/WG11/M0318, MPEG Document, Nov. 1995.



**Figure 21:** Compression of a B-spline surface with different quantizations. Some triangles degenerate due to quantization. a) 50% coefficients, 23 bit quantization, compression gain 1:1.33. b) 10 bit, 1:4. c) 7 bit, 1:5. d) 5 bit, 1:10. (Data set courtesy of Advanced Visual System Inc.) Wiener-filtering on the parametric ‘Goblet’ data set corrupted by quantization noise: a) - d) different bit rates used for compression, e) - h) Resulting surfaces computed by the Wiener filter.



**Figure 22:** PT representations of an irregular turbine blade mesh. a)-b) Part of the data set at different reconstruction levels with increasing number of tetrahedra (data set courtesy of Advanced Visual Systems Inc.).

- [3] D. Busch and M. D. Gross. “Interactive neural network texture analysis and visualization for surface reconstruction in medical imaging.” In R. J. Hubbard and R. Juan, editors, *Eurographics '93*, pages 49–60, Oxford, UK, 1993. Eurographics, Blackwell Publishers.
- [4] T. Chiueh, C. Yang, T. He, H. Pfister, and A. Kaufman. “Integrated volume compression and visualization.” In *Proceedings of the IEEE Visualization 97*, pages 329–336, 1997.
- [5] C. Chui. *An Introduction to Wavelets*. Academic Press, 1992.
- [6] R. J. Clarke. *Digital Compression of Still Images and Video*. Academic Press, 1995.
- [7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1994.
- [8] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF regional conference series in applied mathematics, no.61, SIAM, 1992.

- [9] M. F. Deering. "Geometry compression." In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 13–20. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.
- [10] D. Douglas and T. Peucker. "Algorithms for the reduction of the number of points required to present a digitized line or its caricature." *The Canadian Cartographer*, 10(2):112–122, December 1973.
- [11] A. Dreger, M. H. Gross, and J. Schlegel. "Multiresolution triangular b-spline surfaces." In *Proceedings of CGI 98*, 1998.
- [12] T. Ebrahimi, F. Bossen, R. Castagno, C. DeSola, C. LeBuhan, L. Piron, E. Reusens, and V. Vaerman. "Dynamic coding of visual information." Technical Report ISO/IEC JTC/SC29/WG11/M0320, MPEG Document, Nov. 1995.
- [13] A. Finkelstein and D. H. Salesin. "Multiresolution curves." In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 261–268. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
- [14] K. Fukunaga. *Introduction to Statistical Pattern Recognition. 2nd Ed.*. Academic Press, New York, 1990.
- [15] S. J. Gortler, P. Schroder, M. F. Cohen, and P. Hanrahan. "Wavelet Radiosity." In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pages 221–230, 1993.
- [16] M. H. Gross. "L<sup>2</sup> optimal oracles and compression strategies for semiorthogonal wavelets." Technical Report 254, Computer Science Department, ETH Zürich, 1996. <http://www.inf.ethz.ch/publications/tr200.html>.
- [17] M. H. Gross and D. Kleiner. "Wiener splines." Technical Report 274, Department of Computer Science, ETH Zürich, Oct. 1997.
- [18] M. H. Gross, L. Lippert, R. Dittrich, and S. Häring. "Two methods for wavelet-based volume rendering." *Computers & Graphics*, 21(2):237–252, 1997.
- [19] M. H. Gross, O. G. Staadt, and R. Gatti. "Efficient triangular surface approximations using wavelets and quadtree data structures." *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.
- [20] P. Heckbert, J. Rossignac, H. Hoppe, W. Schroeder, M. Soucy, and A. Varshney. "Course no. 25: Multiresolution surface modeling." In *Course Notes for SIGGRAPH '97*. ACM SIGGRAPH, 1997.
- [21] H. Hoppe. "Progressive meshes." In H. Rushmeier, editor, *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 99–108, Aug. 1996.
- [22] P. Lacroute and M. Levoy. "Fast volume rendering using a shear-warp factorization of the viewing transformation." In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 451–458. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
- [23] D. Laur and P. Hanrahan. "Hierarchical splatting: A progressive refinement algorithm for volume rendering." In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 285–288, July 1991.
- [24] L. Lippert. *Wavelet Based Volume Rendering*. PhD thesis, Federal Institute of Technology (ETH), Zürich, to appear 1998.
- [25] L. Lippert and M. H. Gross. "Fast wavelet based volume rendering by accumulation of transparent texture maps." In *Proceedings of Eurographics '95*, pages 431–443, 1995.
- [26] L. Lippert, M. H. Gross, and C. Kurmann. "Compression domain volume rendering for distributed environments." In *Proceedings of Eurographics '97*, pages 95–107, 1997.
- [27] J. M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, Seattle, 1994.
- [28] S. Mallat. "A theory for multiresolution signal decomposition: The wavelet representation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.
- [29] B. McCormick. "Visualization in scientific computing." *Computer Graphics*, 21(6), 1987.
- [30] S. Muraki. "Approximation and rendering of volume data using wavelet transform." *IEEE Computer Graphics & Applications*, (9):21–28, 1992.
- [31] National Library of Medicine. *The Visible Human Project*. [http://www.nlm.nih.gov/research/visible/visible\\_human.html](http://www.nlm.nih.gov/research/visible/visible_human.html), 1995.
- [32] J. Popovic and H. Hoppe. "Progressive simplicial complexes." In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 217–224, Aug. 1997.
- [33] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer, New York, 1985.
- [34] D. H. Pritchard. "U.S. color television fundamentals – A review." *IEEE Transactions on Consumer Electronics*, 23(4):467–478, Nov. 1977.
- [35] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, San Francisco, 1996.
- [36] P. Schroder and W. Sweldens. "Spherical Wavelets: Efficiently Representing Functions on the Sphere." In *Computer Graphics Proceedings, Annual Conference Series, 1995 (ACM SIGGRAPH '95 Proceedings)*, pages 161–172, 1995.
- [37] J. M. Shapiro. "Embedded image coding using zerotrees of wavelet coefficients." *IEEE Trans. Signal Processing*, 41(12):3445–3462, Dec. 1993.
- [38] O. G. Staadt, M. Gross, and R. Weber. "Multiresolution compression and reconstruction." In *Proceedings of IEEE Visualization 1997*, pages 337–346. IEEE, 1997.
- [39] O. G. Staadt and M. H. Gross. "Avoiding errors in progressive tetrahedralizations." Technical Report 287, Department of Computer Science, ETH Zürich, Jan. 1998.
- [40] E. J. Stollnitz, T. D. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, Inc., 1996.
- [41] G. K. Wallace. "The jpeg still picture compression standard." *Communications of the ACM*, 34(4):30–44, Apr. 1991.
- [42] R. Westermann. "A multiresolution framework for volume rendering." In *Proceedings of ACM Volume Visualization 94*, pages 51–57, 1994.
- [43] M. V. Wickerhauser. *Adapted Wavelet Analysis from Theory to Software*. A. K. Peters, Ltd., 1994.