Multiresolution Compression and Reconstruction

Oliver G. Staadt, Markus H. Gross, Roger Weber

Department of Computer Science, ETH Zürich

Email: {staadt, grossm, weber}@inf.ethz.ch
WWW: http://www.inf.ethz.ch/department/WR/cg

Abstract

This paper presents a framework for multiresolution compression and geometric reconstruction of arbitrarily dimensioned data designed for distributed applications. Although being restricted to uniform sampled data, our versatile approach enables the handling of a large variety of real world elements. Examples include non-parametric, parametric and implicit lines, surfaces or volumes, all of which are common to large scale data sets. The framework is based on two fundamental steps: Compression is carried out by a remote server and generates a bitstream transmitted over the underlying network. Geometric reconstruction is performed by the local client and renders a piecewise linear approximation of the data. More precisely, our compression scheme consists of a newly developed pipeline starting from an initial B-spline wavelet precoding. The fundamental properties of wavelets allow progressive transmission and interactive control of the compression gain by means of global and local oracles. In particular we discuss the problem of oracles in semiorthogonal settings and propose sophisticated oracles to remove unimportant coefficients. In addition, geometric constraints such as boundary lines can be compressed in a lossless manner and are incorporated into the resulting bit-stream. The reconstruction pipeline performs a piecewise adaptive linear approximation of data using a fast and easy to use point removal strategy which works with any subsequent triangulation technique. As a result, the pipeline renders line segments, triangles or tetrahedra. Moreover, the underlying continuous approximation of the wavelet representation can be exploited to reconstruct implicit functions, such as isolines and isosurfaces more smoothly and precisely than commonplace methods. Although it scales straightforwardly to higher dimensions the performance of our framework is illustrated with results achieved on data very popular in practice: parametric curves and surfaces, digital terrain models, and volume data.

Keywords: wavelets, isosurfaces, volumes, triangulation, tetrahedralization, meshing, oracles.

1. Introduction

1.1. Motivation and Previous Work

Geometry compression is an attractive and emerging subfield in computer graphics research which has gained much importance in recent years. Especially when aiming at distributed, interactive rendering and visualization applications, many of which are closely related to the WWW, efficient data encoding is an essential prerequisite for both storage efficiency and real time performance. In this context, we often face client server setups where a remote server maintains complex data sets which have to be browsed, inspected, analyzed or rendered with low latency by a local client. Apart from rendering complex scenes, consider the case of visualizing large digital terrain or medical volume data sets located somewhere in a remote data base: For fast searching and browsing it is often sufficient to generate a low level of detail representation. Conversely, it is sometimes desirable to preserve interesting features such as boundaries, isolines, or spatially appealing regions in full detail while keeping the overall through-put of the communications channel as low as possible. Fig. 1 illustrates some examples where different criteria hold for a meaningful data representation.

Hence, the underlying data representation should be flexible and has to encompass both global and local level of detail while accounting for constraints imposed by special data features. Obviously, as opposed to standard image compression methods, information loss is a manifold problem and has to be controlled much more carefully in graphics applications. As a consequence, elaborate data encoding and compression methods are called for which successfully address the situations featured above. While, on the client side, visual data inspection and analysis is tightly related to

a) b) c)

Figure 1: *Illustration of situations arising from visual inspection of data sets: a) 2D nonparametric surface. b) 2D parametric surface. c) 3D implicit isosurface.*

the computation of geometric reconstructions from the data, mostly in terms of piecewise linear elements such as line segments, triangles, or tetrahedra. It is therefore desirable to perform reconstructions efficiently from the bitstream of incoming data. Moreover, geometry should be refined progressively as more and more data arrives at the client. All representations have to be adaptive, in the way that the number of triangles has to vary as a function of the client's performance and interest while still providing a meaningful representation.

It is clear that much successful research effort has been spent on developing appropriate methods for geometry compression. Early approaches go back to Douglas et. al. [6] who proposed a simplification method for line segments. We can also find a vast amount of literature on mesh representation strategies, a good survey of which is provided by Heckbert et. al. [11]. Later, Deering [5] for instance, proposed a scheme to compress triangular shapes and their attributes. Hoppe [12] and Popovic et. al. [16] suggested the concept of progressive meshes for triangulated shapes where edge split and collapse techniques lead to a continuous hierarchy of levels of detail of an object and constraints may be imposed easily. Others [15], [7] discussed representation and parametrization strategies for meshes of arbitrary topology using linear wavelets. However, high compression gain along with continuous approximation requires smooth, higher order polynomial wavelets, which are difficult to define over arbitrary meshes. The special case of digital terrain data was addressed, for instance, by Lindstrom et. al. [14] and Gross et. al. [10]. The latter one used an underlying wavelet representation to govern mesh refinement and featured both global and local levels of detail.

In summary, much effort has been spent on finding appropriate mesh simplification and representation methods which allow for fast and progressive transmission and rendering of complex scenes. However, little attention has been spent on the following issues:

- Many technical applications in practice, such as medical imaging systems, produce raw data sampled over uniform grids. Due to their complexity, these data sets have to be compressed and stored in a remote database server. Thus, visual inspection and browsing requires computation of piecewise linear geometric reconstructions from the compressed data set.
- Up to now compression was mostly considered a mesh representation problem. The manifold aspects of a full compression pipeline such as precoding, global and local oracles, quantization, and optimal bit allocation have rarely been discussed in full detail.
- Compression and reconstruction should be embedded in a framework which provides an interface for the client and offers a testbed for individual methods. In particular, both lossy and lossless compression must be combined to satisfy demands arising from geometric reconstructions.

1.2. Our Approach

The research presented in this paper was stimulated by the issues discussed above. The goal was to provide an efficient and versatile compression and reconstruction pipeline which accounts for client-server setups. The framework is hybrid in the sense that it combines both lossy and lossless compression. Being restricted to uniform sampled data, we can use bounded B–spline wavelets, such as in [20] and [18], for data precoding. Some of their relatives have successfully been used in image compression [22]. The underlying approximation features high compression gain, elimination of boundary problems, multiresolution progressive setups, and both global and local oracles within the error bounds of L^2. Furthermore, B–spline wavelets allow one to build linear time decomposition and reconstruction schemes forming a basis for fast compression and decompression. The geometric reconstruction of the data essentially combines a generalized point removal/insertion strategy with a subsequent triangulation. We restrict our attention to vertex removal and keep it independent of the meshing. That is, we consider a meaningful triangulation as a plug-in, such as provided by the *qhull* library [1]. Special emphasis, however, is given to implicit reconstruction tasks which occur in many applications. For this, we exploit the smoothness properties of the underlying approximation which allows more smooth and precise computation of implicit intersections than current methods. Again, triangulation algorithms, such as marching tetrahedra [2], are provided as plug–ins from other sources. Thus, our framework features modular and object oriented design, currently embedded in *AVS/Express*.

Fig. 2 depicts an overview of the framework. The individual components can be combined according to requirements of the application. The remote server performs data compression and is governed by parameter settings for global and local oracles, and a bitstream received by a client is produced. It is at this step where geometric reconstruction and interactive visualization are computed. The quality of the geometric reconstruction computed by the client can be controlled depending on network performance, computational and storage capabilities of the client, or on the data themselves.

We are aware that the restriction to uniform sampled data might be considered a major drawback. We believe, however, that the rich variety of applications covered by our approach justifies the presented research.

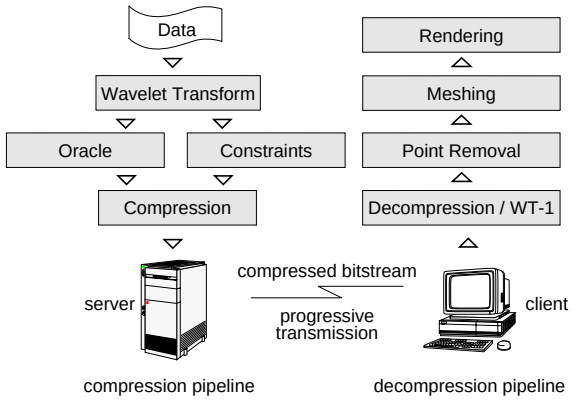

Figure 2: *Illustration of the conceptual components of our compression and reconstruction framework embedded into a client-server setup*

The remainder of our contribution is organized as follows: In Section 2 we describe the fundamentals of the multiresolution data precoding emphasizing new methods for the construction of global oracles for semiorthogonal B–spline wavelets. Section 3 addresses all relevant issues related to our compression strategies for quantization and bit allocation. A fast and easy to use geometric reconstruction technique based on progressive point removal/insertion is explained in Section 4. The special problem of implicit interpolation for isolines and -surfaces is elucidated in Section 5. Finally, we illustrate the versatility of our framework with various examples ranging from different surface types to volume data.

2. MR Approximations

In this section we discuss the mathematical fundamentals of the preprocessing we employ for data preconditioning. As stated earlier, B–spline wavelets are used as a precoding transform since they combine various advantageous features, such as vanishing moments, continuous approximation, bounded interval definition, linear time algorithms, and localization. For reasons of readability, we first review some basics of cardinal B–spline wavelets. However, our attention is mostly directed to the definition of global oracles, that is, schemes to reject unimportant coefficients. Our global oracle consists of a greedy algorithm resulting from an elaborate analysis of L^2 errors in semiorthogonal settings [9], an excerpt of which is given in Section 2.2. Additionally, we will demonstrate how local oracles reject coefficients in unimportant spatial regions and thus enable the construction of *electronic magnifying glasses* for interactive data inspection. For reasons of simplicity, we perform all computations for 1D functions, but the results extend straightforwardly to higher dimensions.

2.1. B-Spline Wavelets

B–spline wavelets were initially introduced by Chui [4], and were extended to bounded intervals by [18] and [20], while nonuniform knot sequences were addressed for instance by [3]. Due to a rich variety of literature in this area, we restrict our introduction to those topics essential for an understanding of our framework.

B–spline wavelets can be constructed from a multiresolution hierarchy of cardinal B–spline scaling functions. Semiorthogonality invokes an additional degree of freedom, however. Thus, approaches as in [18] or [20] end up in slightly different construction schemes. We adapted the methods of Quak et. al. [18] to construct B–spline wavelets of arbitrary order bounded to the interval.

Assuming the reader is familiar with some fundamentals of discrete wavelet transforms (DWT), the implementation of the forward transform is carried out by sequences of projection operators $\mathbf{A}^m, \mathbf{B}^m$, where $m = 1 \ldots M$ stands for the decomposition level. An initial function $f(x)$ is mapped from the higher resolution approximation space V^m onto a lower resolution space V^{m+1} and onto its orthogonal complement space W^{m+1}. Given the coefficient vectors $\mathbf{c}^m$ and $\mathbf{d}^m$ for the scaling functions $\phi_i^m(x)$, and wavelets $\psi_i^m(x)$ in the 1D setting, with

$$c_i^m = \langle f, \phi_i^m \rangle \qquad d_i^m = \langle f, \psi_i^m \rangle \tag{1}$$

i: $1 \ldots N/2^m + \text{order} - 1$, order: B-Spline order ,

the decomposition is performed by matrix operations

$$\boldsymbol{c}^{m+1} = \mathbf{A}^m \mathbf{c}^m \qquad \mathbf{d}^{m+1} = \mathbf{B}^m \mathbf{c}^m \quad . \tag{2}$$

Due to the semiorthogonality, we require the inverse operators $\mathbf{P}^m$ and $\mathbf{Q}^m$ to compute the reconstruction with

$$\mathbf{c}^m = \mathbf{P}^m \mathbf{c}^{m+1} + \mathbf{Q}^m \mathbf{d}^{m+1} . \tag{3}$$

It can be easily proven [18] that the operators relate to each other by

$$\left[\frac{\mathbf{A}^m}{\mathbf{B}^m}\right] = [\mathbf{P}^m | \mathbf{Q}^m]^{-1} . \tag{4}$$

In the case of cardinal B–spline wavelets, sparse operators $\mathbf{P}^m$ and $\mathbf{Q}^m$ come along with dense matrices $\mathbf{A}^m$ and $\mathbf{B}^m$. In order to construct linear time algorithms for both decomposition and reconstruction, it is sufficient to know the sequences $\mathbf{P}^m$ and $\mathbf{Q}^m$ to perform an additional base transform of the coefficients into their duals $\tilde{\mathbf{c}}^m$ and $\tilde{\mathbf{d}}^m$ using the inner product matrices Φ^m and Ψ^m. This results in a decomposition and reconstruction scheme as depicted in Fig. 3.

Note that the decomposition involves solutions of the sparse linear systems of type $\Psi^m \cdot \mathbf{d}^m = \tilde{\mathbf{d}}^m$ for each iteration and $\Phi^M \cdot \mathbf{c}^M = \tilde{\mathbf{c}}^M$ for the last iteration step. Fortunately, this can be accomplished in linear time as well. For brevity we abandon all mathematical details associated with the construction of these transforms and refer the reader to [18]. Instead, we direct our attention in the following section to the problem of global oracles.

2.2. Global Oracles

A global oracle rejects unimportant wavelet coefficients from the transform while minimizing a given error norm. It is clear that the global oracle controls the compression and

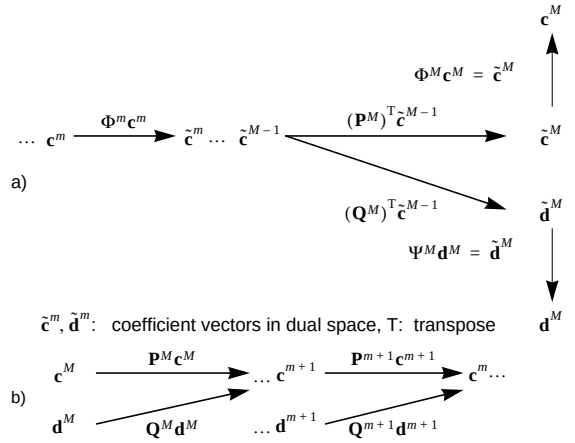

Figure 3: *Linear time decomposition and reconstruction pyramids for cardinal B–spline wavelet transforms a) decomposition. b) reconstruction.*

is essential for information loss. In orthogonal settings, L^2 optimal oracles can be constructed easily by sorting coefficients according to their magnitude, and by rejecting the smallest ones from the list [20]. This strategy is commonplace in many applications and offers good results [8]. Unfortunately, in semiorthogonal spaces, construction becomes more difficult and has hardly been addressed in depth. Maximum distance norm oracles have been proposed by [21] for biorthogonal wavelets. Mathematical analysis of the behavior of approximation errors for semiorthogonal wavelets is closely related to signal energy computations.

The computational scheme for the global oracle is based on the observation that the energy of a function $f(x)$ expanded by semiorthogonal wavelets is obtained by the following sum of scalar products:

$$\|f(x)\|_{L^2}^2 = \mathbf{c}^M \bullet \tilde{\mathbf{c}}^M + \sum_{m=1}^{M} \mathbf{d}^m \bullet \tilde{\mathbf{d}}^m \tag{5}$$

• : scalar product operator.

Due to the orthogonality of different complement spaces it is sufficient to analyze the error norm of a single space W^m. In order to derive an incremental method we assume K out of $N/2^m + \text{order} - 1$ coefficients in this space to vanish. The approximation error is determined by the following relation:

$$\begin{aligned} &\left\|\Delta f^m(x) - \Delta f'^m(x)\right\|_{L^2}^2\Big|_{\mathrm{Rej}(K)} \\ &= \left\langle \sum_{i \in \mathrm{Rej}(K)} d_i^m \psi_i^m(x), \sum_{i \in \mathrm{Rej}(K)} d_i^m \psi_i^m(x) \right\rangle \\ &= \sum_{i \in \mathrm{Rej}(K)} \sum_{j \in \mathrm{Rej}(K)} d_i^m d_j^m \cdot \langle \psi_i^m(x), \psi_j^m(x) \rangle \end{aligned} \tag{6}$$

where $\Delta f'^m(x)$ represents the residual approximation and $\mathrm{Rej}(K)$ denotes the set of all coefficients being rejected from the initial transform. In a next step we compute how the upper error behaves when rejecting an arbitrary $d_k^m \neq 0$, assuming K coefficients to have already been rejected in previous procedures. That is, we compute an expression for the error increment generated by a single coefficient.

$$\begin{aligned} &\left\|\Delta f^m(x) - \Delta f'^m(x)\right\|_{L^2}^2\Big|_{\mathrm{Rej}(K,\,k)} \\ &= \left\|\Delta f^m(x) - \Delta f'^m(x)\right\|_{L^2}^2\Big|_{\mathrm{Rej}(K)} + (d_k^m)^2 \\ &+ \; 2 \cdot \sum_{i \in \mathrm{Rej}(K)} d_k^m d_i^m \cdot \langle \psi_k^m(x), \psi_i^m(x) \rangle \end{aligned} \tag{7}$$

Equation (7) expresses the dependence of the error on an increment of the rejection set. We will refer to it as the *conditional approximation error* in all subsequent discussions. The factor of 2 follows immediately from the symmetry of the inner product matrix. Apparently, the conditional increment is computed by adding one row and one column to the matrix type structure representing the double summation of (6), such as depicted in Fig. 4. In summary, the error can be updated by adding the products of the coefficient d_k^m and the elements of the rejection set times the associated entry of the inner product matrix. Note that this error can be considered a score which reflects the conditional importance of an individual coefficient.

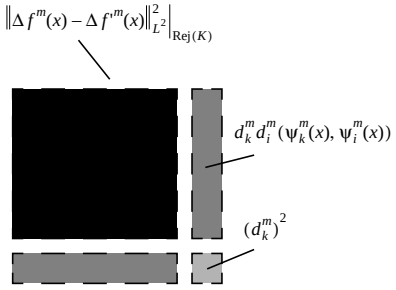

Figure 4: *Illustration of the conditional approximation error increment.*

The relations derived represent an essential step towards the development of an oracle. They allow one to predict how the approximation error changes when rejecting an individual wavelet coefficient under the precondition that K other coefficients have been rejected earlier. Based on this fundamental relationship, it is possible to develop a greedy rejection algorithm which optimizes locally and computes a minimum error rejection set of coefficients. In essence, the greedy oracle operates as follows: It first assigns an initial score to all wavelet coefficients of all iterations m. The score is defined by the overall conditional approximation error, which governs the oracle. In a second step, the oracle iteratively selects the coefficient with the minimum score, rejects it, and recomputes the scores of all other coefficients. The iteration loop runs up to a predefined number of cycles K or up to a predefined error bound E_b. As with equation (7), the score can be recomputed by an appropriate increment after each iteration. Thus we end up with a simple reject–and–update scheme for our oracle. The pseudocode is provided below:

```
Initialize: score[i,m] ← d[i,m] d[i,m];
for i ← 1 to K
  for m ← 1 to M do
    Search: coefficient[i_rej,m_rej]|score[irej,mrej] = min ≠ 0;
      Reject: d[i_rej,m_rej] ← 0;
      if m = m_rej && score[i,m] ≠ 0 then
        increment[i,m] ← 2 d[i,m] d[i_rej,m_rej] y[i,i_rej,m]
              + score[i_rej,m_rej] - old_score
      else if score[i,m] ≠ 0
        increment[i,m] ← score[i_rej,m_rej] - old_score;
      Update: score[i,m] ← score[i,m] + increment[i,m];
  end;
```

After each step, the `score[i,m]` of a coefficient `d[i,m]` represents the overall conditional approximation error when rejecting `d[i,m]`. Note that the time–complexity of the oracle equals $O(N^2)$ and applies only on forward compression.

2.3. Local Oracles and Selective Refinement

A local oracle allows one to control the approximation locally in interesting regions. Here, the spatial localization of the basis functions enables us to accentuate particular wavelets while suppressing the influence of others. In this understanding, a straightforward local oracle consists of a weighting function which operates on the coefficients of the transform. A first approach to this is given in Gross et. al. [10] who employed a Gaussian weighting. The basic idea is to assume some ellipsoidal weighting area as a local region of interest in the spatial domain. Localization of the wavelet transform enables the projection of scaled and translated versions of it into wavelet space, where individual coefficients are influenced. The initial version presented in [10], however, did not consider the support regions of individual basis functions, and can lead to some artifacts by rejecting wavelets ranging into the region of interest. Therefore, we extended the method with the computation of support and bounded wavelets.

An illustration for geometry based image representation is given in Fig. 5. In Fig. 5a, we computed a mesh of the IEEE Computer Society logo using the approach explained subsequently. Selectively refined meshes using Gaussian weighting are presented in Fig. 5b and Fig. 5c respectively, where the areas of interest are located around different areas of the logo. For illustration, the mesh was kept artificially dense.

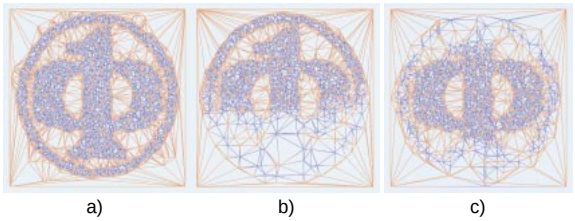

Figure 5: *Illustration of the effect of a local oracle on a triangulated image. a) Initial triangulation. b), c) Local oracle is centered at the upper and central area of the triangulation.*

3. Compression Strategies

This section explains in detail all essential processing steps associated with the definition of appropriate compression strategies. We first give an overview of the compression and decompression pipelines, which are hybrid, in the sense that they combine both lossy and lossless methods depending on the type of feature to encode.

Data compression has a long tradition and has been studied intensively [19]. However, the individual requirements of a geometry based approach encouraged us to design the pipeline explained subsequently. For instance, in the context of lossy compression, issues of floating point data handling and quantization must be adapted to our needs where the structure of the wavelet representation plays an important role. Furthermore, additional effort has to be spent on progressive settings. Since the preservation of constraints, such as iso- or boundary values or lines, is desirable in many applications we propose a lossless compression strategy for these features.

3.1. Overview

Based on the wavelet precoding steps explained previously, we designed a compression/decompression pipeline as depicted in Fig. 6. 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 compression scheme. Conversely, the decompression pipeline inverts the procedure and prepares the data for subsequent geometric reconstruction.

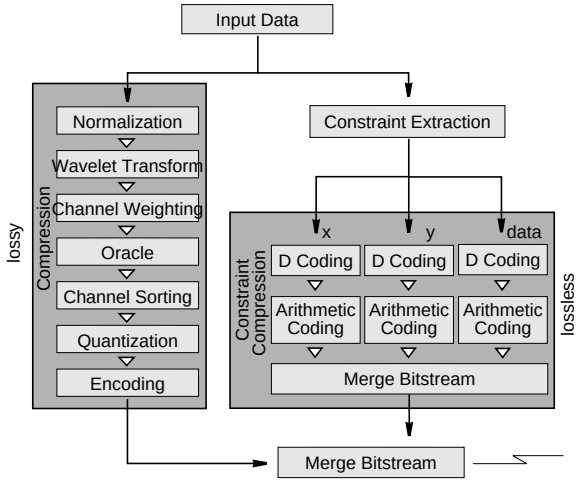

Figure 6: *Compression pipeline including both lossless and lossy data compression. For decompression, all of the above steps have to be reversed.*

3.2. Progressive Lossy Compression

Handling of Floating Point Values First the data is normalized, i. e. the values are scaled to $[0,\ldots,1]$. We decided to carry this out *before* transformation, because post–nor-

malization maps an offset onto small wavelet coefficients and is more difficult to handle upon compression.

In order to prepare the data for bandwise progressive transmission, we sort the multidimensional coefficient array into a 1D vector as displayed in Fig. 7. Here, the array is traversed from the most significant scaling function coefficients to the high frequency bands representing fine grained detail.

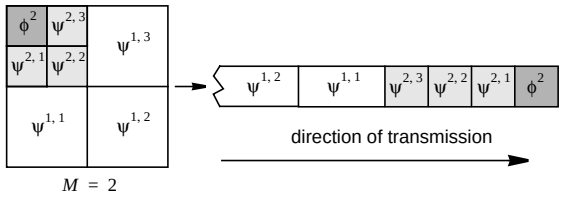

Figure 7: *Conversion of the multidimensional array into a 1D coefficient vector depicted for a 2D WT.*

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

Quantization 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_{float} be a coefficient, we obtain it's quantized version c_{quant} by

$$c_{\text{quant}} = \text{round}(2^{n-1} \cdot c_{\text{float}}) . \qquad (8)$$

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 IEEE–754 floating point format.

Coding and Bit Allocation 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 [23]. Assuming that many of the coefficients will equal zero, encoding is carried out as follows: All nonzero 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.

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 pseudocode illustrates the procedural flow of the scheme:

```
// N:total number of integer coefficients
// d_i: coefficient i
// hufflen_i: length of Huffman-code for symbol i
// huffcode_i: Huffman-code for symbol i
// WriteBits(l,i):
//     appends the last l bits of i to bitsream
// Make2Tupel(i,first,second):
//     converts integer into 2-tuple
i ← 0;
while i < N do
  if d_i = 0 then
    j ← 0;
    while j<32 && d_i ← 0 do inc(i); inc(j); end;
    WriteBits(hufflen_{j+31}, huffcode_{j+31});
  else
    Make2Tupel(d_i,first,second);
    WriteBits(hufflen_{first-1},huffcode_{first-1});
    WriteBits(first,second);
    inc(i);
  end;
end;
WriteBits(hufflen_31,huffcode_31);
```

For brevity we do not explain the construction of the Huffman–table and refer to standard literature, such as [19]. However, 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, which is presented in Table 2. 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. 8.

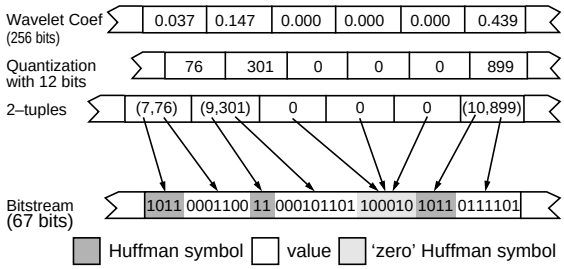

Figure 8: *Example of encoding a sequence of coefficients and the resulting bitstream.*

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. 9. In order to decompose the control points of this B–spline surface we used the pipeline explained in detail in Section 4.3.

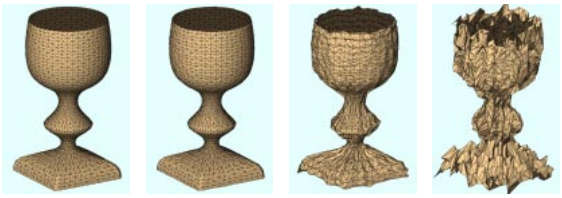

Figure 9: *Compression of a B–spline surface with different quantizations. No additional point removal is performed ($\varepsilon = 0$). Some triangles degenerate due to quantization. a) 50% coefficients, 23 bit quantization, compression gain 1:1.33. b) 10 bit, 1:4. c) 7 bit, 1:5. d) 5 bit, 1:10.*
(Data source: Courtesy Advanced Visual System Inc.)

Finally, Table 1 compares the proposed encoding scheme (*encode*) with some of the most popular lossless compression methods, like *zip*, *arc*, *urbon* 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.

Table 1: Comparison of the proposed method (encode) with some popular compression algorithms (3D volume data set of Fig. 19 and Fig. 20: 128x64x64 voxels).

	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. 10 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, however. The constraint is imposed in terms of a terrain following polyline of a given extent.

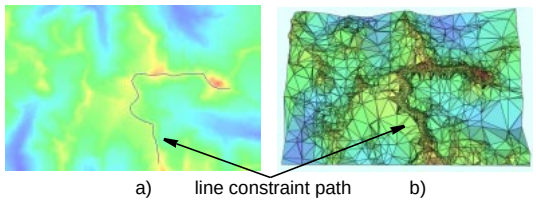

Figure 10: *Illustration of constraints in a digital terrain data set. a) Interactive specification of the constraint path. b) Mesh after constraint insertion.*
(Data source: Courtesy Bundesamt für Landestopographie, Bern, Switzerland)

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. 6.

The position (x, y) and the data value are encoded separately using both delta and higher order arithmetic compression algorithms. For details see [19].

The resulting bitstream format is presented below in Fig. 11, where two headers are followed by the individual *x*-, *y*- and data-streams.

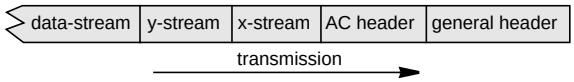

Figure 11: *Data format of the bitstream for constraint compression. The individual header formats are given in Table 2.*

Table 2: Header formats of bitstream.

	Name	Type	Description
General Header	magic_number	byte	ASCII '67'
	stream_size	integer	total size
	xValues_size	integer	size of x-stream
	yValues_size	integer	size of y-stream
	info	byte	misc info
	width	float	constraint width
	field_dims	integer[2]	mesh dimension
	npoints	integer	# points of constraint
	ndata	integer	# extracted data values
	xFirstValue	float	first x-coordinate
	yFirstValue	float	first y-coordinate
Header for Arithmettic Coding	arithFirstValue	float	first extracted data value
	maxValue	float	maximal value
	minValue	float	minimal value
	nIntBits	integer	multiplication factor
	huffFirstValue	integer	1st integer value
Header for Encode	iterationDepth	short	iteration depth
	weightArr	float[]	array of weights
	huffTable	integer[64]	Huffman–table
	quantBits	short	quantization (# of bits)
	degree	short	degree of B–spline bases
	minValue	float	minimum coefficient value
	maxValue	float	maximum coefficient value
	nDim	short	# of dimensions
	dimArr	short[]	dimension array

4. Vertex Removal Strategies

The following section is dedicated to vertex removal methods, which 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. [6] 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 and parametric 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 [17] to fast look–up tables [10].

4.1. 1D Settings

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 [6] we extended these ideas and modified the method to a recursive and progressive algorithm, illustrated in Fig. 12. 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 ε_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.

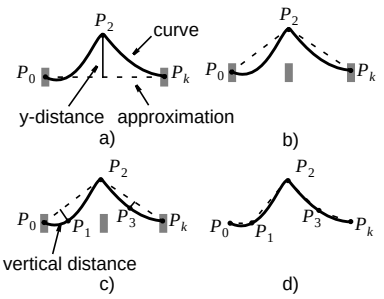

Figure 12: *a) Recursive algorithm assuming a smooth representation of the underlying curve: a) P_2 has largest vertical distance. b) new approximation after insertion of P_2. c) example for vertical distance measure. d) final result.*

The distance can be computed in different ways, where, however, the computation of the vertical distance, such as depicted in Fig. 12c, is computationally much more expensive for general multidimensional settings. Therefore, we recommend computation of the *y*-distance (see Fig. 12a) approximating nonparametric data. The problem of parametric data sets will be discussed in upcoming sections.

4.2. Generalizations to Multiple Dimensions

Generalizations of the method towards multidimensional nonparametric data is straightforward. Starting from an initial grid, as in Fig. 13, 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.

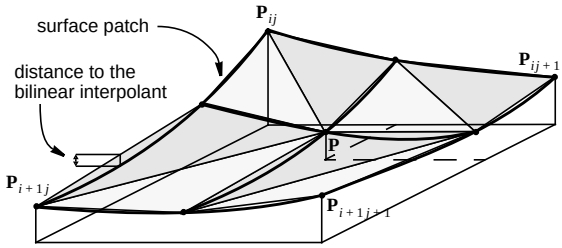

Figure 13: *Extension towards multiple dimensions exemplified for nonparametric data: 2D version. The underlying B–spline patch is outlined in bold. A new vertex is inserted at position* $\mathbf{P}$ *and the distance is computed with respect to the bilinear-interpolant of* $\mathbf{P}_{ij}, \mathbf{P}_{ij+1}, \mathbf{P}_{i+1j}, \mathbf{P}_{i+1j+1}$.

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 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 ε_0 to the iteration. That is if m stands for the current iteration step, the associated threshold $\varepsilon(m)$ is computed by

$$\varepsilon(m) = \varepsilon_0 \cdot e^{M-m} \tag{9}$$

ε_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. 14 further elucidates the data representation.

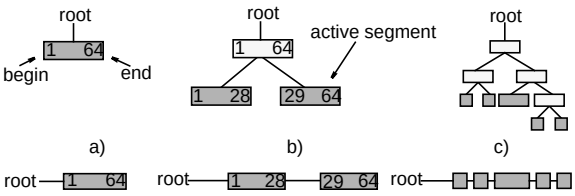

Figure 14: *Construction of a 1D tree data structure with 64 vertices and its growth during progression. The equivalent list structure is given below. a) First segment at the beginning. b) Insertion of P_{29} causes split into two segments. c) Final tree after inserting all points.*

For subsequent triangulations we employed the *qhull* library from [1] in 2D and 3D. Note, that the *N*–tree type cell structure enables computation of very fast meshings using look–up tables, such as the ones presented in [10]. An example of progressive point removal is depicted in Fig. 18, where the mesh is refined gradually with each wavelet channel arriving at the client side.

4.3. Parametric Data Sets

A parametric version of the introduced algorithm can be constructed as elucidated below. For reasons of simplicity, we restrict our description to 2D parameter spaces, however higher dimensional spaces can be easily generalized from that. The conceptual components of our pipeline are illustrated in the diagram of Fig. 15. We assume an initial parametric B–spline surface $\mathbf{s}(u, v)$ to be defined by its vector valued control points $\mathbf{c}_{ij}^0 = (c_{ij,x}^0, c_{ij,y}^0, c_{ij,z}^0)^T$ at iteration $m = 0$.

$$\boldsymbol{s}(u, v) = \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{c}_{ij}^0 \phi_j^0(v) \phi_i^0(u) \tag{10}$$

$J \cdot I$: number of tensor product B–spline scaling functions.

Thus compression has to proceed separately on the *x*, *y* and *z* components of the control vertices. Specifically, the WT and the oracles operate for now independently on the individual coordinates.

However things become more complicated upon reconstruction, which operates again in parameter space (u, v) independently for the spatial coordinates $s_x(u, v)$, $s_y(u, v)$, and $s_z(u, v)$. As a result three binary label fields are generated indicating the importance of individual vertices in

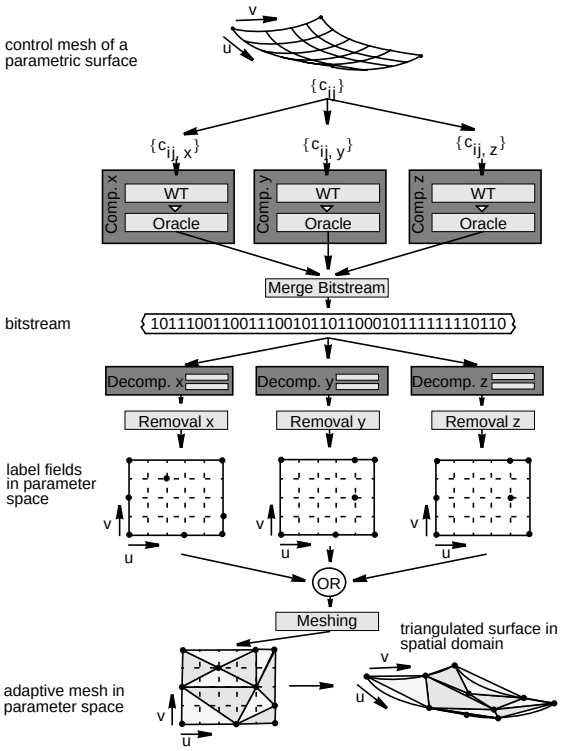

Figure 15: *Illustration of the conceptual components for parametric compression and reconstruction.*

parameter space for subsequent triangulation. Unfortunately, different results are obtained for $s_x(u, v)$, $s_y(u, v)$, and $s_z(u, v)$ and we have to decide on the final removal. This decision is accomplished by applying a Boolean **OR** operator over the individual vertex fields a motivation of which is given as follows: As explained earlier the non-parametric version of our removal strategy holds for linear approximations in terms of triangulations and thus refines the mesh in spatial regions, where the underlying function features nonlinear behavior. In the parametric setting similar criteria are valid for a linear approximation of a surface. The mesh has to be refined in those regions where the surface shows nonlinear behavior, that is where the local curvature does not equal zero. This however happens if either $s_x(u, v)$, $s_y(u, v)$ or $s_z(u, v)$ indicate nonlinearity. Obviously, the Boolean **OR** of the label fields considers a vertex important if one or more of the three coordinates behave locally nonlinear. The usefulness of this approach can be seen in Fig. 17, where a parametric surface is compressed and reconstructed with different parameter settings. Here, we end up with a dense mesh in spatial regions of high curvature and simplification occurs in regions of local planarity.

5. Implicit Interpolation

In the following section the problem of implicit reconstruction is addressed. In practice, implicit structures are mostly isolines or isosurfaces. An advantageous feature coming along with the multiresolution B–spline representation is

the higher order continuous approximation of the underlying data. Although any computation bases on adaptive triangulations obtained from previous procedures, this property can be exploited to reconstruct implicit structures more precisely. For instance, in 2D data sets, piecewise linear representations of isolines can be recovered by immediate computation of the intersections along the triangle edges from the B–spline approximation. Similar procedures hold for isosurface reconstructions from tetrahedralizations. The cubic polynomials perform data smoothing and cancel out most of the jags and discontinuities commonplace in standard methods.

5.1. Isolines

In order to handle isolines we start from the initial B–spline description of the underlying 2D height function $f(x, y)$ and obtain an implicit formulation by

$$f(x, y) = \sum_{i=1}^{I} \sum_{j=1}^{J} c_{ij}^{0} \phi_j^0(y) \phi_i^0(x) = \tau \qquad (11)$$

τ : isovalue.

Recalling the approximation properties of B–splines we recommend to precompute an interpolation problem to get the appropriate coefficients c_{ij}^{0} related to the data samples to be interpolated. These types of interpolations are extensively investigated and relate tightly to inverse B–spline filtering problems which perform in linear time [22].

For (11) we provide a polyline approximation using a marching triangle-like look-up table which operates on a triangle mesh representing $f(x, y)$. A slight extension of the look-up table enables the extraction of those parts of the surface which are interior to the isoline, that is whose function values $f(x, y) > \tau$. The vertices of the describing polygonal hull are given by the intersections of the isoline with triangle edges, such as shown in Fig. 16a.

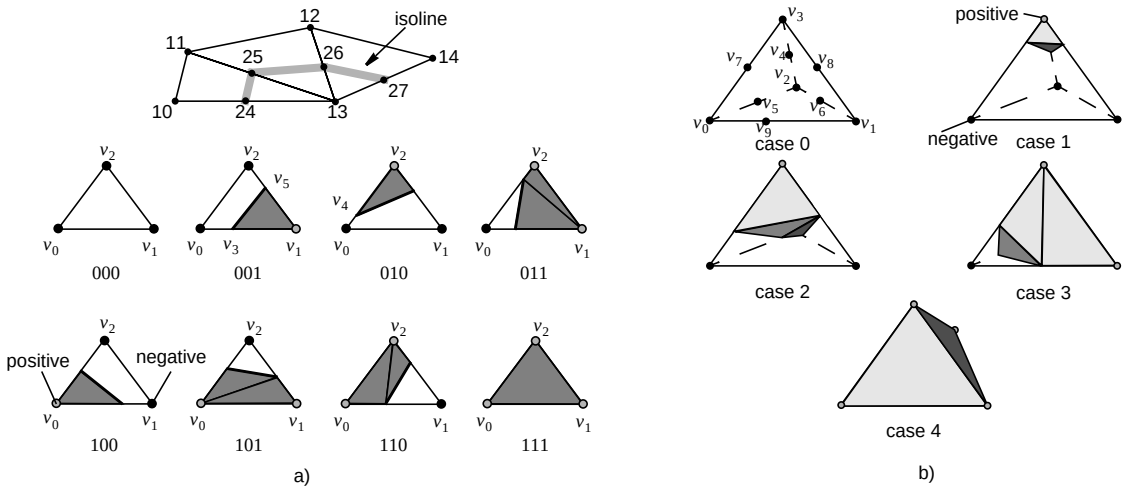

Figure 16: *Polyline approximation of isolines: a) Isoline as computed by intersections with the triangle edges and look-up table to extract interior or exterior parts of the surface. b) Generation of isosurfaces and interior volumes using a marching tetrahedron algorithm: 5 basis cases arising upon triangulation. The connectivity table for generation of interior volumes is presented in Table 3.*

Note that in cases 011, 101, and 110 the initial triangle representing the surface is split into 2 primitives. The intersection of the isoline, implicitly defined by (11), with the triangle edge is calculated using a binary search along the edge. Here we exploit the regional separation with respect to τ provided by the isoline.

Fig. 18 illustrates the performance of the method on digital terrain data. We employed progressive triangular approximations of different quality to compute both isolines and to extract interior regions. Comparing the isolines computed by our method with those of a linear interpolation reveals the superiority of the approach.

5.2. Isosurfaces

Similar relations hold for the generation of isosurfaces and interior or exterior volumes. Here we start from a B-spline volume approximation of $f(x, y, z)$:

$$f(x, y, z) = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{h=1}^{H} c_{ijh}^{0} \phi_h^0(z) \phi_j^0(y) \phi_i^0(x) = \tau \qquad (12)$$

$H \cdot J \cdot I$: number of tensor product volume B–splines.

After solving the initial B–spline interpolation problem the isosurface is obtained by a marching tetrahedron [2] algorithm, where the intersections of the surface with the tetrahedral edges are computed using the binary search on the B–spline volume. Again, a little work on the look-up table enables one to extract interior or exterior volume seg-

ments which are important for many applications, such as finite element simulations [13]. Fig. 16b exploits symmetry and illustrates the 5 out of 16 cases arising upon triangulation giving the connectivity for the extraction of interior volumes.

Table 3: Connectivity table for generation of interior volumes (see Fig. 16b).

No.	v_0	v_1	v_2	v_3	Tetrahedra (vertex list)
0	-	-	-	-	{}
1	-	-	-	+	{ {v_4, v_7, v_8, v_3} }
2	-	-	+	+	{ {v_8, v_5, v_6, v_2}, {v_5, v_8, v_7, v_3}, {v_8, v_5, v_2, v_3} }
3	-	+	+	+	{ {v_7, v_5, v_9, v_2}, {v_3, v_1, v_2, v_7}, {v_7, v_2, v_9, v_1} }
4	+	+	+	+	{ {v_0, v_1, v_2, v_3} }

Note especially that individual tetrahedra may split up into three primitives for representing the bounding surface of the interior volume.

The results given in Fig. 19 and Fig. 20 illustrate the approximation behavior of the method, where standard marching cubes and marching tetrahedron with linear interpolation are contrasted to the B-spline binary search algorithm. Although intersection computation is more expensive we observe that the resulting isosurface is figured out more precisely and smoothly using the new approach.

6. Results

In this section we demonstrate the versatility of the introduced compression and reconstruction framework by investigating its performance on different data sets. First, Fig. 17 shows a series of triangulations of a parametric interpolating B-spline surface generated in accordance to the diagram in Fig. 15. The initial control mesh of 100×100 points was decomposed and reconstructed using 60% of the coefficients. By fixing the quantization to 10 bits we achieved a compression gain of 1:5. We observe that the quality of the reconstruction is governed by the parameter ε_0. The triangulation was computed using the *qhull* library from [1]. Interestingly, our method generates numerous *slivers*, long thin triangles, which are mostly located along the shaft of the object. This effect can be explained easily by analyzing the curvature in those regions. We find that it differs significantly in u and v direction in parameter space and therefore long, thin structures provide an efficient planar approximation.

Further progressive mesh refinements and isoline reconstructions are depicted in Fig. 18 for a digital terrain data set of the Swiss Alps, Matterhorn region. The initial grid consists of 701×481 vertices and 168,000 triangles. We applied a forward compression by keeping only 5% of the coefficients at a decomposition level of $M = 7$, which corresponds to a compression gain of 1:40 at 10 bits quantization. The series reveals how the mesh is refined progressively and adaptively upon reconstruction with each incoming wavelet channel. The extracted isolines were computed using the method of Section 5.1 and are contrasted against the bilinear interpolations of Fig. 18e. By comparing them to Fig. 18a we note that artifacts coming along with linear interpolation are avoided in our approach using the binary search technique. Further pictures from this series illustrate the extraction of interior and exterior regions and variations of the compression gain. Especially Fig. 18h illustrates the quality of the approximation at a compression gain of 1:100.

A set of 3D isosurface reconstructions is presented in Fig. 19. Here a $128 \times 64 \times 64$ voxels subset of the CT–VHD (Fig. 20a) was decomposed, compressed and tetrahedralized adaptively to obtain a fraction of the skull surface. We fixed the parameters to $M = 4$ and achieved a compression gain of 1:15 at 20 bits. In this picture our method is compared to a standard marching cubes technique and to a trilinear interpolating marching tetrahedron, as included in the libraries of *AVS/Express 3.0*. Again, the higher order interpolation provided by the cubics smooths out most artifacts striking in the reconstruction of Fig. 19c, where continuity is lost and the surface "breaks up". Moreover, we avoid even some of the "voxel-like" artifacts of the marching cubes reconstruction shown in Fig. 19b.

Reconstructions of interior and exterior volumes from the same data set are depicted in Fig. 20b and Fig. 20c. We observe that our point removal strategy keeps the tetrahedra dense in those regions, where the underlying volume features high spatial frequencies. The adaptive tetrahedralizations were computed using the look–up table extensions proposed in Section 5.2. This allows one to extract anatomic substructures for further processing, such as FEM. Finally, the computing times of some of the examples are listed in Table 4.

Table 4: Computing times for various steps of our compression and reconstruction framework.

Step	B-spline surface (Fig. 17)	DTM (Fig. 18)	VHD (Fig. 19/20)
Compression	0.1 sec.	1.5 sec.	2 sec.
Decompression	0.1 sec.	1.5 sec.	2 sec.
Point Removal	1 sec.	2 sec.	2 sec.

7. Conclusion and Future Work

We presented a versatile framework for multiresolution compression and reconstruction of non-parametric, parametric and implicit data which bases on wavelet approximations. Although being restricted to uniform grids the scheme handles many real world data types and features numerous advantageous properties, such as both lossless and lossy compression or progressive and selective mesh refinement. However the current implementation only supports channelwise progressive mesh refinement. Hence, future research activities have to comprise the development of a true "coefficient-wise" progressive mesh refinement procedure, which improves the approximation as data comes in. The extension of the framework toward nonuniform sample grids is still interesting, in spite of the fact that appropriate WTs have already been introduced to the community [3]. In addition the inclusion of 2D and 3D texture compression and reconstruction is an important issue for ongoing investigations.

8. Acknowledgment

This research was supported in parts by the ETH research council under grant No. 41–2642.5. We are grateful to the

Bundesamt für Landestopographie, Bern, Switzerland, for the digital terrain data and to the NLM for providing the VHD set. Our special thanks to R. Kisseleff, U. Hengartner, J.–P. Hofstetter, and P. Ruser for implementing parts of the framework.

9. References

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. "Qhull," 1996. http://www.geom.umn.edu/locate/qhull.

[2] J. Bloomenthal. "An implicit surface polygonizer." In P. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[3] M. D. Buhmann and C. A. Micchelli. "Spline prewavelets for non-uniform knots." *Numerische Mathematik*, 61(4):455–474, May 1992.

[4] C. Chui. *An Introduction to Wavelets*. Academic Press, 1992.

[5] 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.

[6] 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.

[7] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. "Multiresolution analysis of arbitrary meshes." In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[8] S. J. Gortler, P. Schroder, M. F. Cohen, and P. Hanrahan. "Wavelet radiosity." In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 221–230, 1993.

[9] M. H. Gross. "L^2 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.

[10] M. H. Gross, O. G. Staadt, and R. Gatti. "Efficient triangular surface approximations using wavelts and quadtree data structures." *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.

[11] P. S. Heckbert and M. Garland. "Survey of polygonal surface simplification algorithms." Technical report, CS Dept., Carnegie Mellon U., to appear. http://www.cs.cmu.edu/~ph.

[12] H. Hoppe. "Progressive meshes." In H. Rushmeier, editor, *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 99–108, Aug. 1996.

[13] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, and Y. I. H. Parish. "Simulationg facial surgery using finite element models." In H. Rushmeier, editor, *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 421–428, Aug. 1996.

[14] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. "Real-time, continuous level of detail rendering of height fields." In H. Rushmeier, editor, *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 109–118, Aug. 1996.

[15] J. M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, Seattle, 1994.

[16] J. Popovic and H. Hoppe. "Progressive simplicial complexes." In *Computer Graphics (SIGGRAPH '97 Proceedings)*, to appear, Aug. 1997.

[17] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer, New York, 1985.

[18] E. Quak and N. Weyrich. "Decomposition and reconstruction algorithms for spline wavelets on a bounded inverval." *Applied and Computational Harmonic Analysis*, 1(3):217–231, June 1994.

[19] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, San Francisco, 1996.

[20] E. J. Stollnitz, T. D. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, Inc., 1996.

[21] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. "Wavelets for computer graphics: A primer." *IEEE Computer Graphics and Applications*, 15(3):76–84, May 1995 (part 1) and 15(4):75–85, July 1995 (part 2).

[22] M. Unser, A. Aldroubi, and M. Eden. "Fast b–spline transforms for continous image representation and interpolation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):277–285, Mar. 1991.

[23] G. K. Wallace. "The jpeg still picture compression standard." *Communications of the ACM*, 34(4):30–44, Apr. 1991.

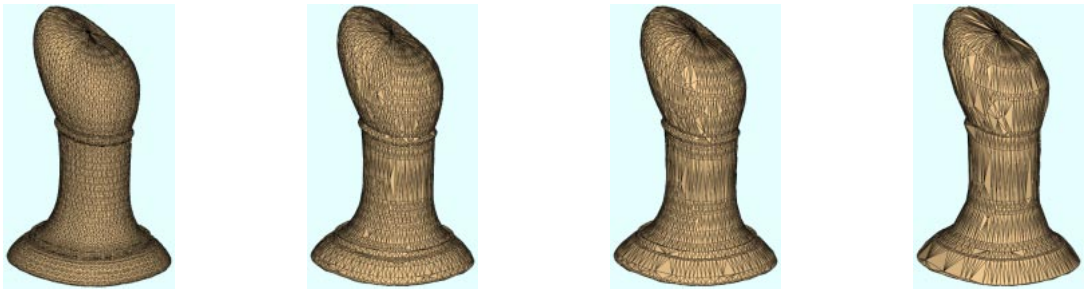

Figure 17: *Compression and reconstruction of a parametric B–spline surface for different levels of linear approximation. a)* $\varepsilon_0 = 0$ *, 19602 triangles (100%). b)* $\varepsilon_0 = 0.005$ *, 43% triangles. c)* $\varepsilon_0 = 0.01$ *, 34% triangles. d)* $\varepsilon_0 = 0.02$ *, 22% triangles.*

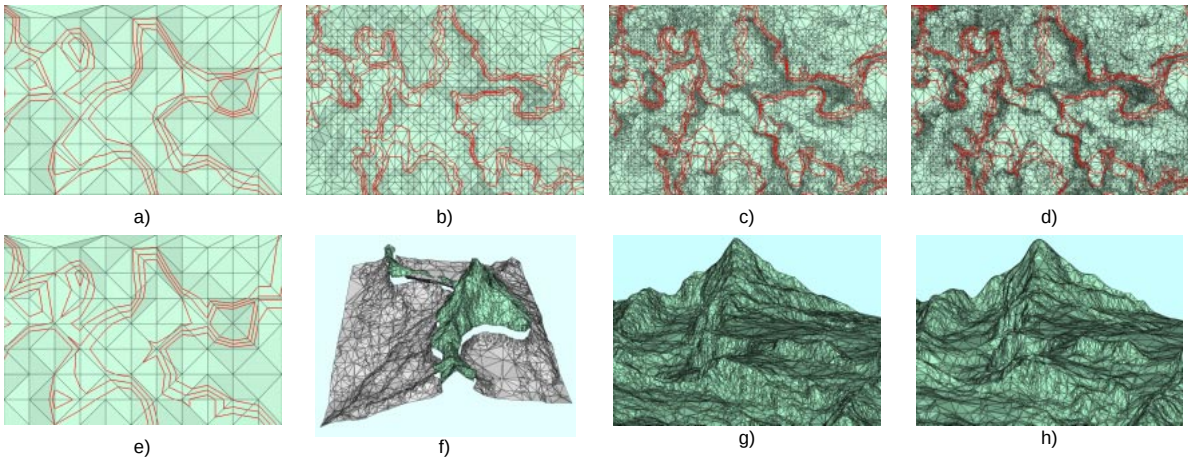

Figure 18: *Extraction of isolines and interior surfaces from a digital terrain model of the Swiss Alps and progressive mesh refinement: 3 isolines are extracted for* $\tau = 120$, $\tau = 125$ *and* $\tau = 130$, *respectively. a)* $\varepsilon_0 = 0.01$, *Wavelet channel 1, 0.1% triangles. b) Channel 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,* $\varepsilon_0 = 0.0035$, *62% triangles. h) 1% coeff., compression gain 1:100,* $\varepsilon_0 = 0.0035$, *62% triangles.*

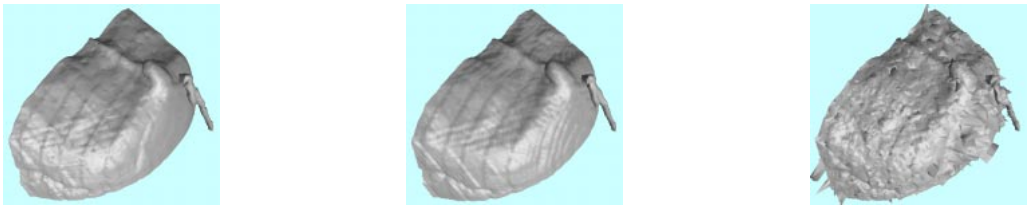

Figure 19: *Extraction of isosurfaces volume data. Isovalue* $\tau = 75$ *(skull). a) Our method, 20% coefficients,* $\varepsilon_0 = 0.1$, *124,343 tetrahedrons, compression gain 1:15. 51,970 triangles b) Marching Cubes, same compression settings, 540,800 cells, 40,522 triangles. c) Marching Tetrahedron (as provided by AVS/Express 3.0), same settings as a). Data source: Visible Human Project. Courtesy National Library of Medicine.*

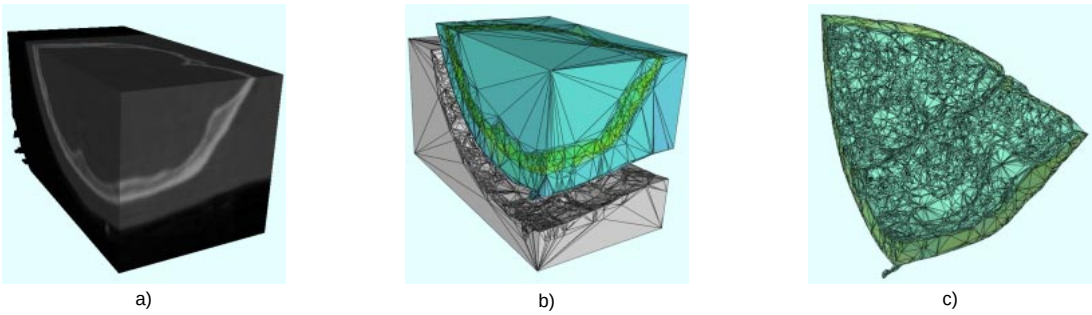

Figure 20: *Extraction of interior and exterior volumes. a) Initial CT volume data set with 2,704,000 tetrahedrons. b) Interior and exterior volumes,* $\tau = 42$ *(skin surface), 133,091+ 34,290 tetrahedrons. c) Interior volume,* $\tau = 75$ *(skull), 124,491 tetrahedrons.*