FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Compression of dynamic polygonal meshes with constant and variable connectivity

The State of the Art and Concept of PhD. Thesis

Jan Dvořák

Technical Report No. DCSE/TR-2022-01
April 2022

Technical Report No. DCSE/TR-2022-01
April 2022

Compression of dynamic polygonal meshes with constant and variable connectivity

The State of the Art and Concept of PhD. Thesis

Jan Dvořák

Abstract

Polygonal mesh sequences with variable connectivity are incredibly versatile dynamic surface representations as they allow a surface to change topology or details to suddenly appear or disappear. This, however, comes at the cost of large storage size. Current compression methods inefficiently exploit the temporal coherence of general data because the correspondences between two subsequent frames might not be bijective. We study the current state of the art including the special class of mesh sequences for which connectivity is static. We also focus on the state of the art of a related field of dynamic point cloud sequences. Further, we point out parts of the compression pipeline with the possibility of improvement. We present the progress we have already made in designing a temporal model capturing the temporal coherence of the sequence, and point out to directions for a future research.

This work was supported by the projects 20-02154S and 17-07690S of the Czech Science Foundation and SGS-2019-016 of the Czech Ministry of Education.

Acknowledgements

I am grateful for the great guidance, valuable feedback, and extreme patience from my supervisor, Doc. Ing. Libor Váša PhD. I would also like to acknowledge my colleagues from the scientific project 20-02154S, who contributed significantly to the research presented in this thesis. Last but not least, I would like to thank my family, who were always there to support me.

Contents

Chapter 1

Introduction

3D surface representations, such as meshes and point clouds, have incredible representative power. When studying a certain object or a collection of objects represented by a mesh or a point cloud, we can observe them from any desired viewpoint and thus be able to detect details that might not be observable in a static image. However, this is often still insufficient to infer dynamic behavior. For example, given a scene consisting of a sphere and a box, we cannot tell whether any of the objects are static or if any force is being applied to them, if any collision might occur between the two objects, and if so, whether any of the objects will deform. This can be addressed by multiple representations capturing the scene at consecutive points in time (see Figure 1.1).

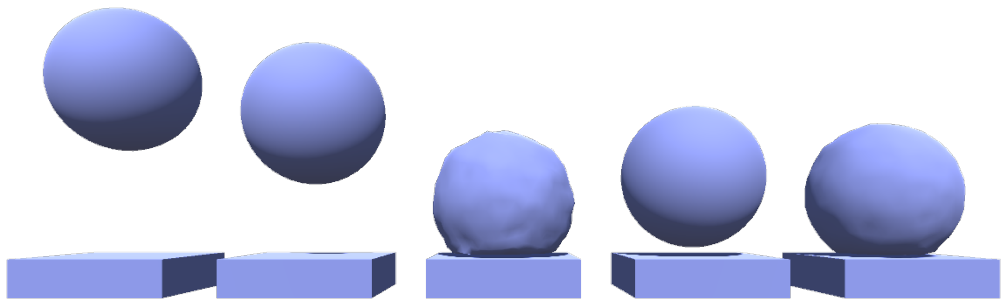


Figure 1.1: Selected frames of a dynamic scene in which a sphere is slightly deformed by collision with a box. Frames are sorted from left to right in order of appearance.

While modern surface scanning hardware can output such data at sufficient frame rates, there are, unfortunately, only a few publicly available datasets of mesh or point cloud sequences. This can be attributed to their large size. For example, the D-FAUST dataset [11] contains 129 sequences with over 40 000 meshes overall, which requires around 129 GB of storage in an uncompressed form. While the compression of a special class of mesh sequences with constant connectivity, called *dynamic meshes*, is already considered a solved problem and some recent advancement has been made in the compression of dynamic point clouds due to ongoing MPEG standardization [3], the compression of general triangle mesh sequences (often called *time-varying meshes*) remains an open problem. Even a few methods have already been proposed, currently the most efficient way to store such data is to compress each mesh separately even though the subsequent frames are temporally coherent, and this fact is not being exploited at all. We believe that providing an efficient time-varying compression format to the public would motivate others to publish new mesh sequence datasets.

Some progress in time-varying mesh compression has actually been made in a single specific scenario: tele-immersion. Tele-immersion is a way of communicating in augmented or virtual reality, where a person is being captured by a surface scanning device and the resulting 3D model is transmitted in real time to the receiver (see Figure 1.2). This situation is specific in the sense that the transmitted data represents a human body in varying poses, and thus it is a much simpler task to construct a mathematical model (e.g., a tracked skeleton) that drives the exploitation of the temporal coherence. Additionally, the objectives for compression are also different from those in the general scenario. Instead of preserving as much information as possible, it is much more important to achieve real-time performance. This allows the method, for example, to discard certain frame and replace it with an already-transmitted frame that is warped to resemble the discarded shape, or to remesh the surface. While some of the time-varying mesh compression methods for tele-immersion can also be adjusted to work in more general scenarios, they are quite inefficient in those cases. More interesting is the adjustment in the opposite direction – if there was an efficient general mesh sequence compression method, it might be altered to account for the real-time scenario of tele-immersion.

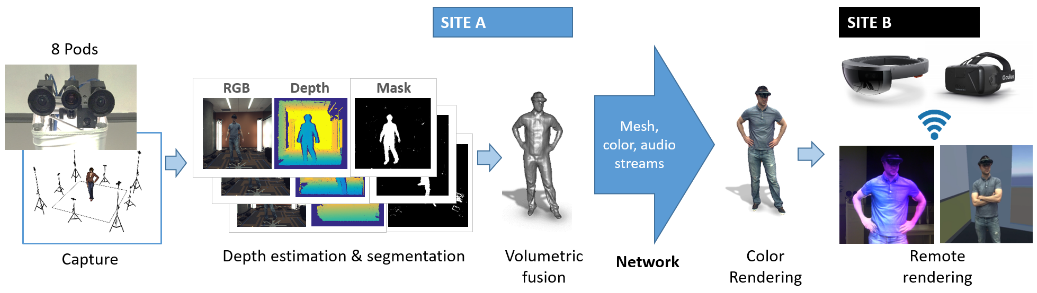


Figure 1.2: Pipeline of the *Holoportation* tele-immersion system [78].

The structure of this thesis is as follows. In the rest of this chapter, we will discuss mesh compression from a more general point of view and also point out how the methods usually measure compression performance. In Chapter 2, we will describe the mesh and point cloud sequences. The next three chapters will discuss the state of the art of such data compression. Chapter 6 will focus on methods addressing related problems that might be incorporated to improve compression performance. After that, we will describe the progress we have already made related to mesh sequence compression. In Chapter 8, we will outline directions for future research. Last, we will conclude the text in Chapter 9.

1.1 Problem Definition

Compression is the process of transforming input data X, which are represented by a certain sequence of m bits, into a reduced representation of n bits, where $m \gg n$, from which one can obtain a reconstruction $\hat{X}$ by a reverse process called decompression. If the compression is lossless, X and $\hat{X}$ must be identical. In lossy compression, $\hat{X}$ is distorted and is required to resemble X only in a defined sense.

In our case, the input data is a sequence of triangle meshes represented by geometry (positions of the elements of the mesh), connectivity (information about how the elements of the mesh are connected to each other) and other properties of each mesh. The scope of our research is compression of only the first two, with emphasis on the geometric information. The data will be described in more detail in Chapter 2.

In terms of mesh geometry compression, the majority of methods are lossy, meaning the reconstructed positions are different from the original ones. This is because positions are represented by vectors of floating point values, and such data is actually quite difficult to efficiently encode in a lossless manner. Loss of information in this case is also less likely to be detected by a viewer than if it occurred in connectivity.

For connectivity, the key criterion to classify a method as lossless is whether a map exists between the vertices of the original and reconstructed mesh, that is an isomorphism. Such definition permits lossless methods to reorder vertices. Lossy methods usually perform remeshing, simplification or filtering, which also implies loss of information in geometry. For a simpler comparison of original and distorted data, it is desirable to preserve the isomorphism.

1.1.1 Data Reduction Techniques

In this section, we will briefly describe several key concepts of general data compression frequently used in methods discussed in Chapters 3, 4, 5 and 6.

The most prominent tool of lossy geometry compression is *quantization*, a process of transforming a range or a large set of values into a smaller discrete set. The simplest and most commonly used type of quantization is performed by the trivial rounding:

$$\bar{x} = round(x/q),$$

where x is the input value and q is a *quantization constant* that controls the resolution. The values can be reconstructed up to a specified precision by simply multiplying by the quantization constant:

$$\hat{x} = q \cdot \bar{x}.$$

In our scenario, the quantization is usually applied to point coordinates or to data derived from them, which are usually real values represented in a single resp. double-precision floating-point number format requiring 32 (resp. 64) bits for storage in an uncompressed form. Assuming there are n possible values after quantization, we can already reduce the data rate by assigning each of the values a unique integer value represented by $\lceil \log_2 n \rceil$ bits. However, we may achieve better results by combining the quantization with other data-reduction techniques.

At the end of the compression pipeline, there is usually a lossless encoding method, which attempts to exploit an underlying model of encoded symbols. In terms of mesh compression, most methods use *entropy coding* (e.g., arithmetic or Huffman coding), which can adjust the number of bits representing each symbol according to its probability of occurrence in the data without considering the actual context. The entropy coding method attempts to obtain an average number of bits per encoded symbol that is close to its optimal lower bound – Shannon's entropy:

$$H = -\sum_{x \in S} p(x) \log_2 p(x), \tag{1.1}$$

where S is a set of all possible symbols and $p(x)$ is a probability of occurence of the symbol x. Less often, a *dictionary-based coding* method (e.g. LZW) is used. Such methods attempt to exploit the context of data by searching for recurring patterns of symbols and encoding a reference to a dictionary of patterns constructed during encoding instead.

Compressed data contains lot of redundant information that can be simply deduced from the coherence of values. A powerful tool for removing such redundancy is a *prediction*. Instead of encoding the original value, the encoder can predict it using the information available from already processed data (which is also available during the decompression) and encode only the difference. Prediction itself is a lossless process, since it does not reduce the number of encoded values, however, when combined with entropic coding, it may result in a decreased bit rate. In terms of mesh sequence compression, we distinguish two types of prediction: *intra-* and *inter*-based. Intra-based prediction exploits the coherence in a single frame, while inter-based prediction exploits the temporal coherence between frames. The majority of compression methods we will discuss use both prediction types to some extent.

We will omit the description of intra only methods, which are equivalent to applying a static mesh compression (a well-studied field) to each separate frame.

From Eq. 1.1, it follows that the higher the probability of the encoded symbol, the lower the number of bits required to represent its single instance in entropy coding. Similarly to prediction, one can use the information obtained from already-processed values to reduce the data rate using *context modeling.* Instead of the overall probability of symbol $p(x)$, a conditional probability $p(x|ctx)$, given a context ctx, is utilized by the coder. If ctx is chosen reasonably well, the conditional probability of the correct encoded symbol should be higher. A context can be deduced from various information, e.g., the previously encoded value.

Another powerful tool is *dimensionality reduction.* For a certain vector $\mathbf{v} \in \mathbb{R}^n$, the goal is to obtain a transformed vector $\hat{\mathbf{v}} \in \mathcal{X}$ in a certain subspace $\mathcal{X} \subset \mathbb{R}^n$ of dimension $k = dim(\mathcal{X}),\ k \ll n$, which we can represent more compactly. While there is no limitation on how this is achieved, due to its simplicity, most approaches are based on orthogonal projection – for a certain vector $\mathbf{u}$, the closest vector $\mathbf{u}_w$ in the direction of unit vector $\mathbf{w}$ is computed as follows:

$$\mathbf{u}_w = (\mathbf{u} \cdot \mathbf{w})\mathbf{w}.$$

Given a set of orthonormal vectors $B_{\mathcal{X}} = \{\mathbf{x}_1, \dots \mathbf{x}_k\}$ forming a basis of $\mathcal{X}$, the $\hat{\mathbf{v}}$ can be expressed as their linear combination:

$$\hat{\mathbf{v}} = \sum_{i=1}^{k} \alpha_i \mathbf{x}_i, \tag{1.2}$$

where $\alpha_i = \mathbf{v} \cdot \mathbf{x}_i$. It can be shown that $\hat{\mathbf{v}} = \arg\min_{\mathbf{w} \in \mathcal{X}} \|\mathbf{v} - \mathbf{w}\|_2$. While $\hat{\mathbf{v}}$ is still of size n, we can actually encode a vector $\mathbf{a} = (\alpha_1, \dots \alpha_k)^T,\ \mathbf{a} \in \mathbb{R}^k$. The $\hat{\mathbf{v}}$ can be fully recovered using Eq. 1.2, as long as both the encoder and the decoder have access to $B_{\mathcal{X}}$. Unless $\mathbf{v} \in \mathcal{X}$, the transformation is lossy. The process allows for progressive coding by incrementally extending the basis and encoding additional projection coefficients. The most crucial part is selecting the appropriate $B_{\mathcal{X}}$, with the objective being to achieve the lowest possible data rate while discarding the information that is less likely to be detected as missing.

One way to find $B_{\mathcal{X}}$ is using *Principal Component Analysis* (PCA), which, for a set of vectors in $\mathbb{R}^n$ with centroid at origin, finds an ordered orthonormal basis of the given space, where each basis vector represents a direction with the most variance in data if the information present in previous basis vectors was removed. This is achieved by singular value decomposition (SVD) of a matrix with encoded vectors as rows. The PCA requires all the encoded vectors to be known, and thus the basis cannot be constructed by the decoder. However, for a fairly coherent set of encoded vectors, the majority of the information is present in the first few principal directions, and a significant reduction of the number of encoded values can be achieved even when the basis vectors are encoded alongside the projection coefficients.

If the encoded vector $\mathbf{v} \in \mathbb{R}^n$ represents a signal over a certain discrete domain (e.g., n vertices of a mesh or a graph), $B_{\mathcal{X}}$ can be chosen as a subset of frequency basis. For a relatively smooth signal, most of the information is contained in lower frequencies, and the high frequency information can be discarded without being noticeable. The dimensionality reduction in this case is equivalent to the *Discrete Fourier Transform* (the projection) followed by a low-pass filter (selection of certain elements of frequency basis to form the $B_{\mathcal{X}}$). One can exploit the fact that in the continuous case, the sine and cosine functions forming the frequency basis are eigenfunctions of the Laplace operator and find the $B_{\mathcal{X}}$ as a set of eigenvectors of the Laplacian matrix $\mathbf{L}$ corresponding to a certain discretization of the Laplace operator over the specified domain, as long as the $\mathbf{L}$ is real, symmetric, and positive semi-definite. For this purpose, on graphs, the compression methods use a graph Laplacian matrix. On meshes, this matrix corresponds to a discretization mostly referred to as Kirchhoff's (the connection between meshes and graphs will be explained in Section 2.1). Additionally, the Cotan discretization [81] can be used. In such a case, the $B_{\mathcal{X}}$ is referred to as a *Manifold Harmonic Basis*. However, since Cotan discretization requires mesh geometry to construct L, some reference geometry must be known, or it can be only used to compress other mesh properties. Full eigendecomposition of $\mathbf{L}$ is not required since we are interested in only a selected subset of eigenvectors.

1.1.2 Performance Evaluation

Efficiency of lossless compression is quantified by the resulting *data size*. Lossy compression, however, requires relating the size to the amount of introduced *distortion*. In the rest of this section, we will describe how such measures are evaluated in terms of meshes.

While the *overall size* of the compressed data is the simplest measure, it is much more difficult to relate the results between different input datasets. Much more prominent is *data rate*, which measures the number of bits required to represent a certain element of data, e.g., a vertex or a face. Another popular measure is the *compression ratio*, the relationship between the sizes of the original and compressed data.

Measuring the distortion that is present in compressed meshes is a much more difficult problem. Depending on the purpose of the data, one must select a correct metric. Additionally, the performance of various compression methods differs depending on the metrics used in evaluation.

Technical applications require an upper bound on error in absolute coordinates, which is usually determined from the precision of the technology processing the data (e.g., in manufacturing). These objectives require *mechanistic* error metrics such as *Mean Squared Error* (MSE) and *Peak Signal to*

Noise Ratio (PSNR), which relate corresponding vertex positions in original and distorted meshes. When the isomorphism between meshes is lost, one can use the *Haussdorf distance* or *Chamfer distance*. These are, however, much more expensive to evaluate, because they require frequent closest point query evaluation. An example of a static mesh compression method that performs well under mechanistic criteria is the *Parallelogram prediction* [95].

In other areas (e.g., in entertainment or marketing), visual similarity is much more important. To this end, it is better to use *perceptual* metrics. These metrics are shown to better correlate with human perception of distortion in data by comparing their results to various user studies. Examples of such metrics are *MSDM2* [57], *DAME* [102], *FMPD* [107], *TPDM* [94] and *TPDMSP* [28], to name a few. For more information on this topic, we refer the reader to the survey by Corsini et al. [18]. An example of a static mesh compression method that performs well under perceptual criteria is *High-pass coding* [86].

An additional concern arises in the dynamic setting, where a sequence of meshes is considered. Even for a small distortion, a large discrepancy between the frames can occur. As an example of such behavior, Corsini et al. [18] discuss a case in which one frame of the animation is distorted by applying a sine function, while the next frame is distorted by a cosine function. Although this distortion might not be visible when examining the frames separately, it results in a flickering effect, which is easily detectable when the frames are examined in fast succession. Quite a few methods for compressing mesh sequences use static metrics applied on separate frames even though such an approach cannot detect this behavior. This is because no temporal metric has been proposed for general mesh sequences. Only a few temporal metrics are designed for dynamic meshes, a special class of mesh sequences with a common connectivity (e.g. *KG* error [50] and *STED* [104]).

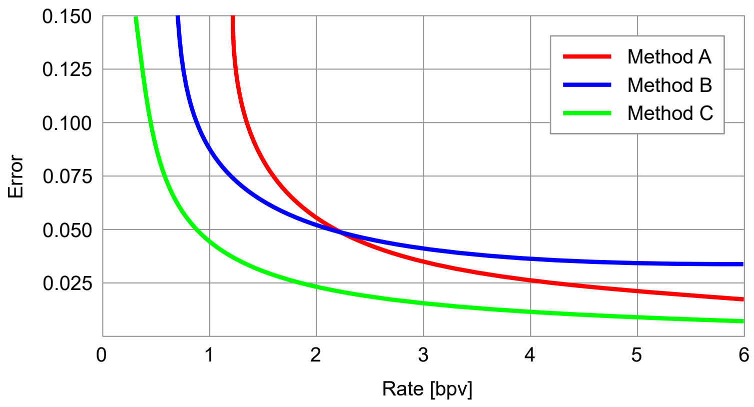


Figure 1.3: An example of an RD curve. While Method A performs better than Method B at higher data rates, it is unclear which method has a better overall performance. On the other hand, Method C clearly outperforms other methods at all data rates.

Not only do various methods perform differently under various metrics, but their performance also differs for different data rates. For a more thorough evaluation of performance, *Rate-Distortion* (RD) curves are used. They show how the change in data rate influences the distortion in a selected metric. Figure 1.3 shows an example comparison of multiple compression methods using RD curves. It is trivial to obtain an RD curve for a method that is controlled by a single parameter directly influencing the data rate (e.g., the quantization constant). A more complex configuration, however, requires an optimization process (e.g., [100]) to find a curve that actually represents the best performance achievable by the method.

Chapter 2

Mesh and Point Cloud Representations of Dynamic Surfaces

Data considered in this thesis is assumed to represent a continuously moving two-dimensional, smooth manifold surface $\mathcal{S}$ embedded in $\mathbb{R}^3$ and sampled at discrete points in time. A representation $\mathcal{F}_i$ (resp. $\mathcal{M}_i$ for mesh, $\mathcal{P}_i$ for point cloud) of each sampled time point t_i will be referred to as a *frame*. The continuous movement implies temporal coherence in frames in the sense that two consecutive frames are often visually nearly indistinguishable.

The temporal coherence between the frames can be described by a *correspondence* function $f_{ij} : \mathcal{F}_i \mapsto \mathcal{F}_j$, which for any point $\mathbf{x} \in \mathcal{F}_i$ assigns a corresponding point $f_{ij}(\mathbf{x}) \in \mathcal{F}_j$ if it exists. Correspondences not only can be used for inter-frame prediction, but also allow temporally coherent mapping of values on the surfaces, e.g., texture [12].

There are multiple criteria to consider when choosing an appropriate surface representation. The most important is representation versatility. We are interested not only in what classes of surfaces can be represented, but also at what cost. Other things to consider are obtaining, rendering, and processing complexity. Since this thesis studies the representations from the compression perspective, we will also list some of the properties specific for this problem.

2.1 Triangle Mesh Sequences

Triangle mesh is currently considered the most popular representation of 3D surfaces due to its simplicity, approximation quality, and native support on graphics cards. It is a piecewise planar surface usually defined as $\mathcal{M} = (V, T)$, where V is a set of vertices, and T is a set of triangles that connect them. A vertex is usually represented by its geometry (position) and properties (normal, color, texture coordinate, etc.), while a triangle is represented as an ordered triplet of indices, and the order of vertices induces its orientation (the direction of a normal). Considering a set of edges E that form all the triangles in T, we can also interpret the mesh as an undirected graph $G = (V, E)$. This allows application of many techniques from graph theory. When talking about mesh geometry, we will refer to the positions of vertices, while connectivity will refer to the combinatorial information (vertex indices, triangles, or edges). For the sake of simplicity, we will focus only on orientable manifold meshes, i.e., meshes in which the orientation of triangles can be unified, vertices coincide only with a single triangle fan, and no edge is connected to more than two triangles. Nevertheless, some of the listed related work in Section 4 did not make such assumptions.

2.1.1 Dynamic Mesh

Dynamic meshes (DMs) are a special class of triangle mesh sequences. The distinguishing property of a dynamic mesh is a *common connectivity* T shared by all frames:

$$T_0 = T_1 = \ldots = T_{n-1} = T,$$

where n is the number of frames. This also indicates that the number of vertices and their order are constant through time. Only the geometry and properties change between frames, and thus the connectivity needs to be encoded only once.

One of the main advantages of a dynamic mesh is that the vertex correspondences are explicitly coded in the connectivity:

$$\forall v_k \in V : f_{ij}(\mathbf{x}_k^i) = \mathbf{x}_k^j$$

for any pair of frames (i, j), where $\mathbf{x}_k^i$ is the position of k-th vertex in the i-th frame. The correspondence function for any point $\mathbf{x}^i \in \mathcal{M}_i$ can be directly generalized from vertex correspondences using barycentric coordinates $(\lambda_a, \lambda_b, \lambda_c) : \mathbf{x}^i = \lambda_a \mathbf{x}_a^i + \lambda_b \mathbf{x}_b^i + \lambda_c \mathbf{x}_c^i$ in triangle $t = (v_a, v_b, v_c)$, which contains x:

$$f_{ij}(\mathbf{x}^i) = \lambda_a f_{ij}(\mathbf{x}_a^i) + \lambda_b f_{ij}(\mathbf{x}_b^i) + \lambda_c f_{ij}(\mathbf{x}_c^i) = \lambda_a \mathbf{x}_a^j + \lambda_b \mathbf{x}_b^j + \lambda_c \mathbf{x}_c^j.$$

For dynamic meshes, the function f_{ij} is always an isomorphism. Instead of treating the geometry of each frame separately by assigning each vertex v_k a position $\mathbf{x}_k^i \in \mathbb{R}^3$, a static mesh $\mathcal{M} = (V, T)$ can be considered, where for each vertex v_k the geometry is represented as a trajectory $\mathbf{t}_k \in \mathbb{R}^{3n}$. This allows for global coding approaches (in terms of time), e.g., PCA coding [103].

The simplest dynamic meshes are usually synthetic data obtained by continuously deforming a certain shape using, for example, skinning. It is, however, often more efficient to encode the deformation parameters than to encode the resulting mesh sequence. More complex sequences are obtained by surface tracking or by 4D reconstruction (e.g., [91, 105]); however, most methods require a prior mathematical model of the data as an input (e.g., template shape), and the whole process is quite complex and prone to errors.

Visualizing a dynamic mesh usually consists of rendering the first frame and then updating only the geometry, which allows for a higher rendering frequency. While simple and easy to work with, the dynamic mesh lacks the representation versatility. Not only is the time-evolving topology not allowed, but since the connectivity complexity directly influences the ability to represent fine details, any fine detail must also be accounted for even if it appears only in a single frame.

2.1.2 Time-Varying Mesh

Time-varying mesh (TVM) is any mesh sequence in which the number of vertices and/or connectivity changes over time. While the temporal coherence of the connectivity might be present (e.g., in synthetic data), it cannot be generally assumed. Thus, most of the time, it must be encoded for each frame separately.

Not only are the correspondences of time-varying mesh frames unable to be directly derived from the connectivity, but they are also difficult to estimate. This is caused by the fact that the bijective property of the correspondence function is lost with the merging and separation of parts (see Figure 2.1). This renders exploiting the temporal coherence a much more difficult problem.

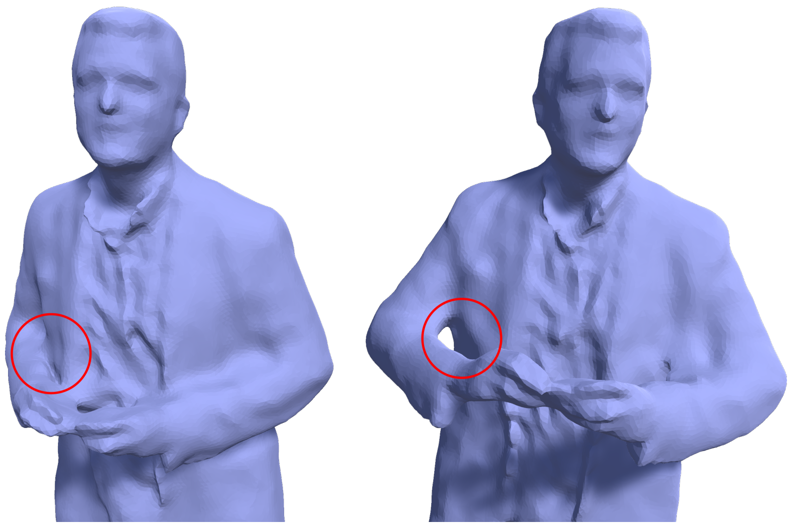


Figure 2.1: Example of bijectivity loss of correspondences. There are no correspondences for vertices at the back of the arm in the highlighted area.

The earliest time-varying meshes were similar to dynamic meshes in the sense that the connectivity changed only by simple updates (e.g., by subdivision, contraction, or vertex removal). More importantly, TVMs have been used to represent dynamic volumetric environments (e.g., fluid simulations). Such data is usually obtained by extracting the iso-surface of each frame from a certain implicit function. In recent years, improvements in the performance of capturing systems allowed real-time surface capture of dynamic scenes. Such systems can output a large number of mesh frames; however, these contain a considerable amount of noise and self-contact. This leads to frequent spurious topology changes, even for surfaces that originally showed constant topology. While in some scenarios such noisy TVMs could be converted to a dynamic mesh with great difficulty, often it is much more preferable to work directly with the time-varying mesh data.

Visualizing a TVM requires rendering a different mesh in each separate frame, which made it almost impossible to render at satisfying frame rates on past consumer-grade devices. As the computer technology advances, this is becoming less of an issue and the advantages of the TVMs prevail. The connectivity can adapt to accommodate any detail in the data at any time, allowing an increase in the complexity of the mesh frame only when it is necessary.

2.2 Dynamic Point Cloud

A *point cloud* is a set of points $\mathcal{P} = \{\mathbf{x}_0, \dots \mathbf{x}_{m-1}\}, \mathbf{x}_k \in \mathbb{R}^3$, sampled from the represented surface, where m is the number of points. Similarly to a triangle mesh vertex, a point can be represented by its geometry and attributes. In the case of point clouds, the literature does not consider any special cases of sequences in which only the positions of points change over time, and any point cloud sequence is usually referred to as a dynamic point cloud. From the compression standpoint, a point cloud frame $\mathcal{P}_i$ is equivalent to a mesh $\mathcal{M}_i = (V_i, \emptyset)$ with empty set of triangles. For this reason, a dynamic point cloud compression combined with connectivity coding can be considered a method for compression of time-varying meshes and, conversely, a TVM compression method that ignores the connectivity data can be considered a method for compression of dynamic point clouds.

A correspondence function of point clouds can be defined only on the points since the point cloud is discrete and there is no surface representation between the points. Similarly to TVMs, correspondences on point clouds are generally not bijective and are difficult to estimate.

Dynamic point clouds have also benefited from recent improvements in capturing systems. Compared to time-varying meshes, they can be obtained at a much lower cost. A special type of dynamic point cloud is obtained using LiDAR (Light Detection and Ranging) sensors. Such sensors are usually mounted on a moving vehicle (e.g., a car or drone) and emit light rays in rotating motion while measuring the distance from the point where the ray hits the scanned environment. LiDAR data is commonly used for navigating autonomous vehicles. Unlike the general point clouds, the frames are usually also accompanied by additional measurements (e.g., accelerometer data). While it is possible to treat the LiDAR point cloud as an image, in which the position of each pixel represents cylindrical coordinates and the value represents the distance, several compression methods treat it as a point cloud, while still using its specific properties.

Rendering dynamic point clouds is easier than rendering TVMs, since no connectivity has to be passed to the GPU. The points are usually rendered as a certain primitive, e.g., a cube, a sphere, or a disc. Selecting an appropriate scale for the rendered primitives given the current viewpoint should result in the illusion of a smooth surface.

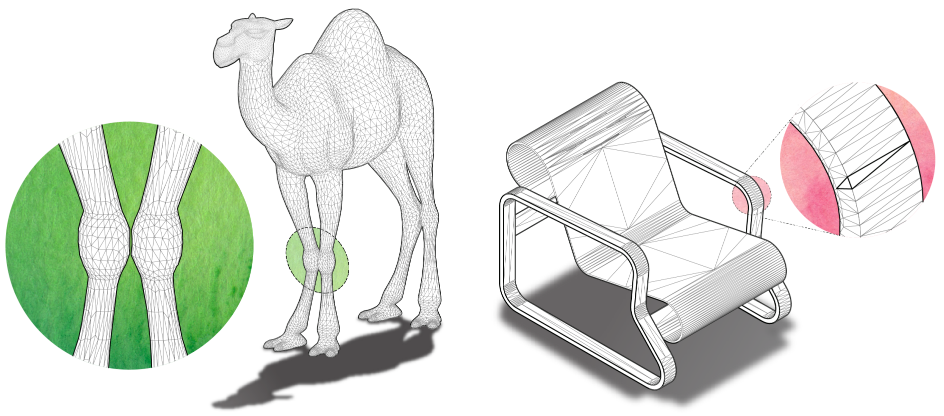


Figure 2.2: Advantages of the mesh representation over point clouds. *Left*: Separation of geodetically distant vertices in near proximity in space. *Right*: Highly non-uniform representation. Illustrations are courtesy of Hanocka et al. [41].

The main advantage of the point cloud representation over a triangle mesh is its simplicity, which allows storage and processing of even large models with points numbered in the millions. On the other hand, since it lacks information about connectivity, obtaining the direct neighborhood of a point must be done by querying positions in space, which is much more difficult than just examining the edges in the case of a triangle mesh, and the result might contain points that are in near proximity in space, but quite far apart in terms of geodetic distance. Additionally, the mesh allows for a highly non-uniform representation in which planar regions with a lack of detail can be represented by a small number of large triangles and, conversely, regions with lots of detail can be represented by a large number of small triangles. In the case of point clouds, highly non-uniform sampling might result in visible holes in a surface during rendering. Both of these advantages of the triangle mesh are shown in Figure 2.2.

2.2.1 Voxelization

A majority of dynamic point cloud compression methods consider so-called voxelized point clouds. Voxelization is a process in which the point cloud is stored in a cubic grid of size $2^d \times 2^d \times 2^d$, where d is a parameter controlling the level of detail. Each cell is marked occupied or unoccupied, based on whether it contains any of the input points. All the points inside an occupied grid cell are discarded and replaced by the centroid of the cell. An octree is usually built from the binary occupancy data (see Figure 2.3), since it allows more efficient storage and progressive coding.

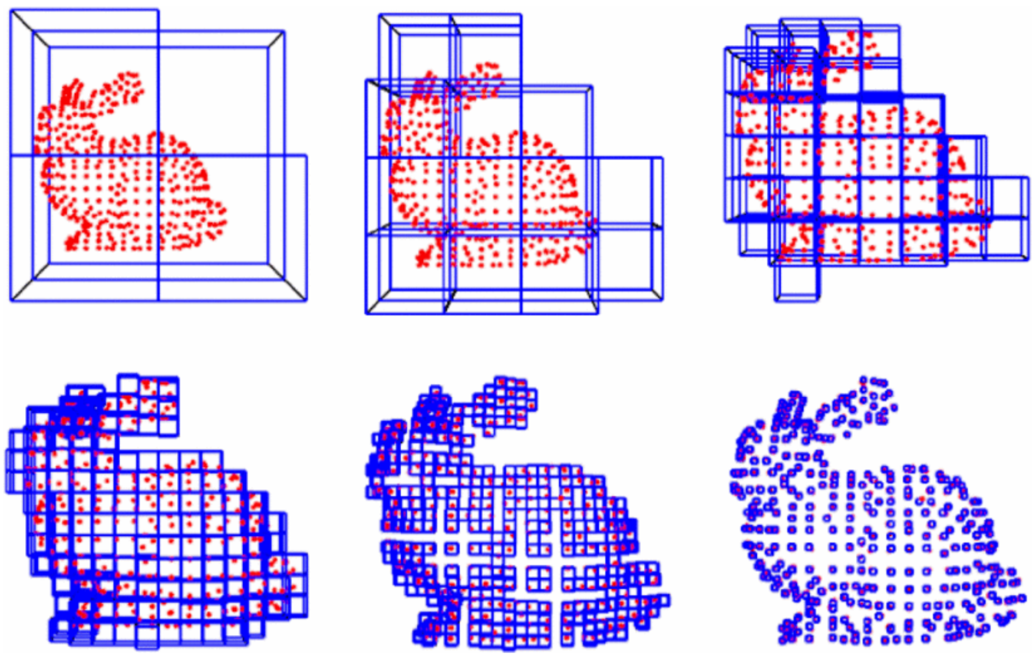


Figure 2.3: Different levels of a single octree representing a voxelized point cloud. Source: [48]

Voxelization is a process similar to quantization in the sense that the real valued coordinates of points are transformed to grid indices. However, it also alters the number of points, compromising the one-to-one correspondence between the original and voxelized data. This is why certain dynamic voxelized point cloud compression methods claim to be lossless – the voxelized data already has reduced precision.

Chapter 3

Dynamic Mesh Compression

Due to all the advantages of the dynamic mesh representation listed in Section 2.1.1, it is possible to design highly efficient compression methods using both spatial and temporal prediction. This is why the development of dynamic mesh compression was so divergent from the development of time-varying mesh compression. As of 2021, this field was already considered well-studied. Since the main scope of this thesis is the TVM compression, we will describe only the most recent dynamic mesh compression methods. For more details about this field, we refer the reader to the 3D mesh compression survey by Maglo et al. [69].

3.1 High-Fidelity Compression of Dynamic Meshes with Fine Details Using Piecewise Manifold Harmonic Bases

The method proposed by Chen et al. [17] is based on the assumption that since the dynamic mesh is usually created by deforming a static mesh over time, the frames with similar poses should have similar fine details.

The method represents the positions of vertices in each frame by a single vector $\mathbf{f}_i = (x_{i,1}, \ldots, x_{i,n}, y_{i,1}, \ldots, y_{i,n}, z_{i,1}, \ldots, z_{i,n})^T$, where i is the index of the frame and n is the number of vertices. The frames are clustered using a k-medoids algorithm based on the Manhattan norm. Unlike in k-means clustering, the k-medoids algorithm selects a center frame (medoid) that is closest to all frames in the cluster instead of to the mean cluster frame. The medoids are then selected as key frames and encoded using a state-of-the-art static mesh compression algorithm. For each frame in the cluster, the low-frequency information is encoded using dimensionality reduction with manifold harmonic basis [96] constructed from the key frame geometry. The projection coefficients are further compressed using linear predictive coding. Finally, the fine detail (high-frequency information) is introduced by detail transfer from the key frame (see Figure 3.1).

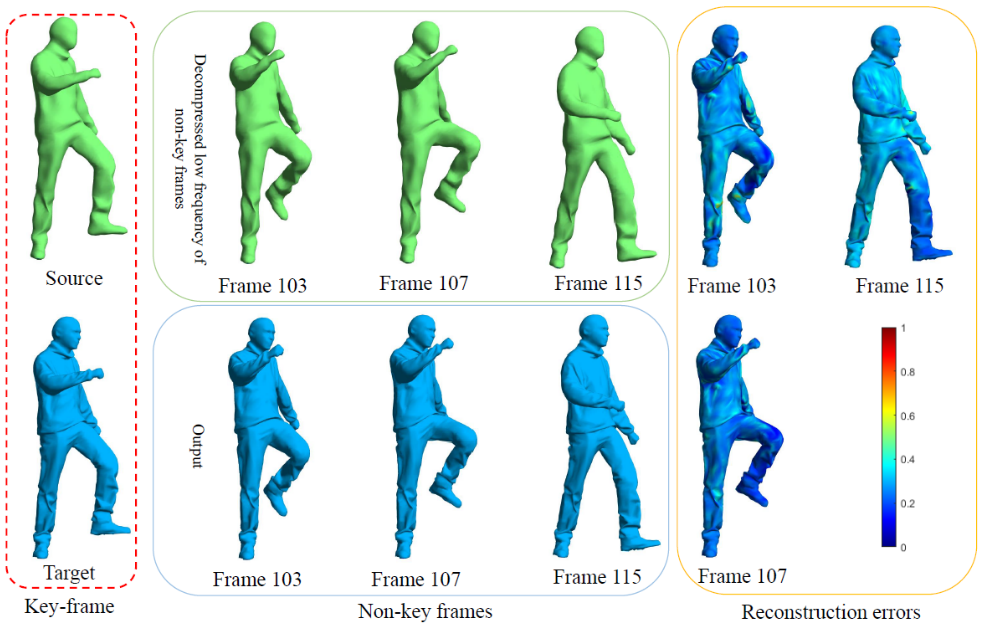


Figure 3.1: Detail transfer from the key-frame. Source: [17]

3.2 Motion-Aware Compression and Transmission of Mesh Animation Sequences

The method introduced by Yang et al. [110] is also based on clustering of frames; however, it does not consider similarity of shape, but of motion.

An overview of the method is shown in Figure 3.2. The method first represents the movement of each vertex as a spatial curve it traces through time. Curvature κ and torsion τ of the curve are estimated using finite differences for points on the curve corresponding to all the frames. The method then proceeds with k-means clustering representing the frame as a vector $\mathbf{f}_i = (\kappa_{i,1}, \tau_{i,1}, \ldots, \kappa_{i,n}, \tau_{i,n})^T$, where i is the index of the frame and n is the number of vertices, using Euclidean distance. Each cluster is then spatially segmented by applying the k-means method to vertices, considering the average position of the vertex through all the frames in the cluster, and curvature and torsion estimations in such frames. Connectivity, clustering, and spatial segmentation data is encoded. To compress the geometry of each segment in each frame cluster, Graph Fourier Transform (GFT) is used. After reducing the number of projection coefficients, which removes high-frequency information, the remaining coefficients are encoded using SPIHT coding [84].

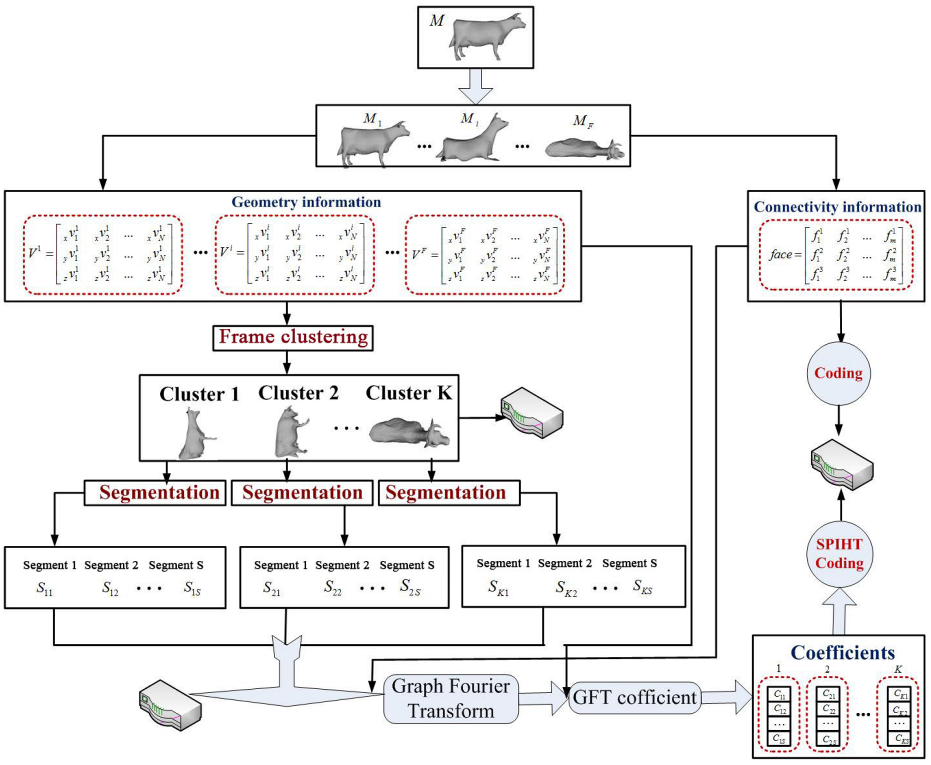


Figure 3.2: Overview of the compression process of [110]. Source: [110]

3.3 3D Mesh Animation Compression Based on Adaptive Spatio-temporal Segmentation

Contrary to [17] and [110], Luo et al. [65–67] proposed a more local approach. The frames are processed sequentially. Initially, the method attempts to find a boundary frame index b_1 in a sequence S of the first γ_{init} unprocessed frames, which separates the S into two subsequences with distinctive dynamic behavior. If $b_1 = \gamma_{init}$, no distinctive behavior was detected and S is passed directly into the encoding stage (Fig. 3.3 (I)). Otherwise a spatial segmentation follows. The method measures the maximal edge length change and a certain percentage of the least deforming edges when modeling the measured values as exponential probability distribution is selected and all the vertices connected to such edges are labeled as *rigid*. Clusters are created by merging topologically connected rigid vertices. Connected clusters with similar trajectories are merged together until a specified number of vertex groups is achieved. Then, the initial temporal boundary is adjusted by examining the next γ_{max} frames. For each segment separately, the PCA is applied to its trajectories and the number of the most significant principal components n is obtained by thresholding the corresponding eigenvalues of the covariance matrix used to obtain the components. The method gradually increments the temporal boundary and examines the recomputed n until it changes. The largest frame index b_2, where n does not change, is the final temporal boundary of the segment. Segmentation data and geometry up to frame b_2 are encoded. If $b_2 = \gamma_{max}$, no temporal segmentation was detected in the examined frames (Fig. 3.3 (III)). Thus, the spatial segmentation is reused for temporal segmentation on the next γ_{max} frames (Fig. 3.3 (IV)). Otherwise (Fig. 3.3 (II)), the spatial segmentation is discarded and the method continues to the initial temporal cut of the next γ_{init} frames.

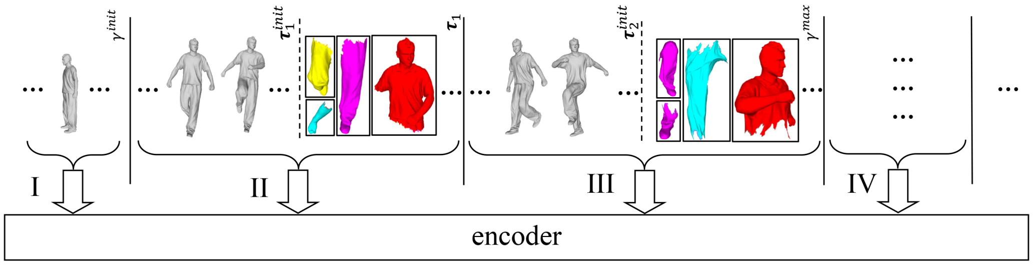


Figure 3.3: Four different classes of encoded subsequences. (I) directly encoded, (II) temporal boundary detected, (III) temporal boundary not detected, (IV) spatial segmentation reused from previous subsequence. Source: [66]

The geometry of each spatio-temporal segment is compressed using projection onto a reduced PCA basis. In [65], the PCA data required for the reconstruction (basis, projection coefficients, and average trajectory) was encoded directly. In [66], such data was further compressed using lossless zLib compression. Most recently [67], the authors proposed a more efficient strategy based on reshaping. Once all the spatio-temporal segments are obtained, they are clustered using k-means clustering. To obtain the vector representations required for clustering, for each segment, a histogram of a fixed number of bins is constructed over its coded values. The earth mover's distance [83] $d_{i,j}$ is computed for each pair of histograms (H_i, H_j) and the vector representation of the k-th segment is $\mathbf{s}_k = (d_{k,1}, \ldots, d_{k,N})^T$, where N is the number of all segments. Each PCA cluster is processed separately by combining PCA data for all of its segments in a 2D matrix, which is then compressed. Although the authors used zLib compression which does not fully use the structure of 2D data, image compression can also be used.

3.4 Summary of Dynamic Mesh Compression

Most modern dynamic mesh compression methods are based on global approaches, mainly the dimensionality reduction. The methods presented in this chapter usually introduced only an incremental improvement in data rates by grouping similar data to reduce the number of dimensions required to store the information to be preserved. Nevertheless, it is actually difficult to decrease the average data rate significantly since the reconstructed data already looks indistinguishable from the original at ranges from 2 to 5 bits per vertex in a single frame, depending on the complexity of the input data. For this reason, we do not believe there will be any further breakthrough in this area in the near future.

Chapter 4

Time-Varying Mesh Compression

An overview of all the time-varying mesh compression methods presented in this chapter and their classifications is shown in Table 4.1. The methods can be classified by the type of inter-prediction they utilize. The simplest approaches, based on motion estimation (ME), usually align two consecutive frames and predict the encoded geometry using their proximity. Another simple approach is to construct a spatial data structure from the vertices, e.g., a grid or octree, and inter-predict the structure (Structural). These two approaches usually make few to no assumptions about the input data. To benefit from the already mature fields of image and video compression, the projection-based methods transform the encoded data into the 2D domain (Projection). The last class of methods uses a mathematical model representing the sequence, e.g., a skeleton or tracked template (Model).

Table 4.1: Overview of all methods presented in this chapter in the order of corresponding sections. *Conn.* denotes whether the method considers connectivity coding; *Iso.* denotes whether the method preserves the one-to-one mapping between encoded and decoded vertices. *[40] does not preserve the one-to-one mapping in special cases.

Section	Method	Type	Input data	Conn.	Iso.
4.1	Gupta et al. [37]	ME	synthetic	yes	yes
4.2	Yang et al. [111]	Model	general	yes	no
4.3	Han et al. [39]	ME/Structural	general	no	yes
4.4	Han et al. [40]	Structural	general	no	no*
4.5	Yamasaki – Aizawa [109]	ME	general	yes	yes
4.6	Nakagawa et al. [76]	Model	human	yes	no
4.7	Doumanoglou et al. [23]	Model	human	yes	opt.
4.8	Hou et al. [42]	Projection	const. top.	yes	no
4.9	Faramarzi et al. [27]	Projection	general	yes	yes

The methods also differ in the types of input data they are able to handle.

While the majority of methods handle general TVMs, some focus on more specific input. For example, [37] assumes synthetic data created by modeling software (synthetic), in which the connectivity changes with simple updates. Other methods expect the input data to represent a human person (human), and therefore data can be effectively represented by various mathematical models. Lastly, the method presented in [42] is limited only to TVMs with constant topology (const. top.).

4.1 Registration and Partitioning-Based Compression of 3-D Dynamic Data

To the best of our knowledge, the earliest work in the field of time-varying mesh compression was done by Gupta et al. [37]. They assumed that the movement of the compressed model was piece-wise rigid. Their algorithm divides the model into segments, which are then matched between two consecutive frames using the iterative closest point (ICP) registration algorithm [9]. The registration process yields a rigid transformation for each segment, mapping its positions from the previous to the current frame with some error introduced. Depending on the amount of error, the segments are then divided into three groups. The first group can be sufficiently described by the rigid transform. The second group requires residuals to be encoded to correct the encoded positions. The third group cannot be described by the rigid transform and thus must be encoded independently.

To encode the connectivity of a frame, correspondences found by alignment with segments of the previous frame are used to find vertices that have been added or deleted. For deleted vertices, only their index is encoded. For added vertices, the algorithm encodes their closest neighbor. Then, a Delaunay triangulation is performed to obtain a connectivity prediction in changed parts of the mesh. To correct the prediction, the following data needs to be encoded: indices of triangles that are to be deleted and indices of all vertices of added triangles.

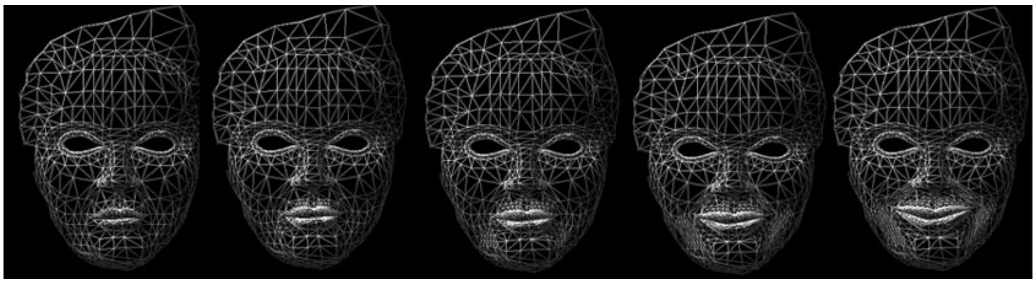


Figure 4.1: Example of synthetic time-varying mesh data. An adaptive refinement approach is used to represent more complex facial expressions. Source: [37]

The algorithm assumes that the compressed data is synthetic (i.e., created by a software; see Figure 4.1), limiting the change in connectivity only to the operation of insertion or deletion of vertices. Nowadays, this is a very special case of a time-varying triangle mesh. For real data (3D scans), this connectivity coding is very inefficient and highly impractical.

4.2 Semi-Regular Representation and Progressive Compression of 3-D Dynamic Mesh Sequences

The method proposed by Yang et al. [111] is based on remeshing the sequence to obtain a semi-regular structure. First, the first frame of the sequence is simplified using the quadric error metric [35], resulting in a so-called base mesh. Vertices of the original mesh are then mapped on the base mesh, and by applying the butterfly subdivision scheme, multiple levels of detail are created.

The base mesh is then gradually transformed to match each frame. This is done by first simplifying the current frame to obtain its base mesh. The base meshes are then aligned using rigid ICP [9]. Then, alignment-driven segmentation is done so that if an error for a single segment exceeds a threshold, the segment is split and an alignment is done on the split regions. If there is a sufficiently small segment next to a large one, these are merged together. Subsequently, a vertex-wise motion refinement for each subdivision level of the previous frame is done by minimizing energy consisting of three parts: change of spring energy, change of curvature and normals, and matching itself.

At each subdivision level, to obtain a prediction of a currently encoded vertex, a butterfly subdivision is done on the lower level of the current frame and the motion-extrapolated previous frame. The residuals are then encoded using wavelet compression.

The method is progressive because first, only the base mesh data is transmitted, and the decompressed result grows more precise with each subdivision step. However, the resulting sequence is no longer a time-varying mesh. Also, issues might occur when the frames are not complete and thus not every vertex has a correspondence with a vertex in the previous frame.

4.3 Time-Varying Mesh Compression Using an Extended Block Matching Algorithm

Han et al. [39] proposed a method based on the block matching algorithm (BMA) widely used in video compression [47]. The method divides the bounding box of a frame into cubic blocks of a specified size. The surface in each block is then translated for its centroid to lie at the center of the block. For each block, a corresponding block in the previous frame is found by matching weighted average normal vectors of the surfaces. Residuals (so-called motion vectors) are then calculated as displacement vectors between the closest points between the two blocks (see Figure 4.2). Such residuals are then quantized and transformed in the frequency domain.

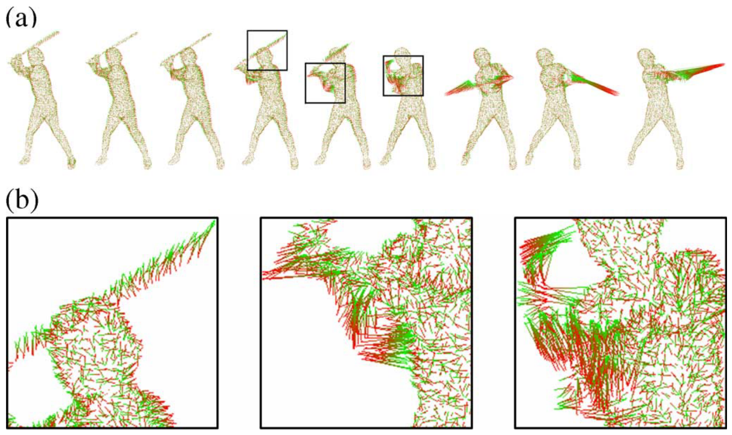


Figure 4.2: (a) Motion vectors estimated by the EBMA. (b) Detail for the rectangular regions. Source: [39]

The method is still quite limited, as it requires the encoded TVM to consist of only a single connected component (regardless of the topology of the component). The method also assumes the connectivity of each frame is known to both the coder and decoder and does not consider such data when measuring bit rates. It turns out, however, that such data should be considered as well since the connectivity might actually take a large portion of the complete encoded data size. Additionally, to restore the position of a vertex, the ID of a corresponding vertex needs to be encoded as well. This is unfortunately not apparent from the paper since authors compared their method only to a slightly degraded version of the method itself in spite of the existence of many static compression algorithms at that time.

4.4 Geometry Compression for Time-Varying Meshes Using Coarse and Fine Levels of Quantization and Run-Length Encoding

As a follow-up to their work described in the previous section, Han et al. [40] proposed another method based on domain subdivision into blocks. The subdivision occurs on two levels. The coarse level is used to indicate which parts of the domain contain surface and which do not (see Figure 4.3). This is represented by a binary function. To reduce the temporal redundancy, an XOR operation is performed on functions of consecutive frames. The frames must be aligned beforehand to further improve the results. Since the result of the XOR operation indicates which blocks changed their values between frames, it is expected that the number of these blocks is small relative to the number of all blocks, and thus run-length coding of a raster scan is performed to reduce bit rate. Each block containing a surface is then further divided into sub-blocks. The edge length of each sub-block should be smaller than the length of the shortest edge in the TVM to prevent singularities. A binary function is used to represent which sub-blocks contain vertices. Again, raster scanning is performed to obtain a sequence of blocks for which the values of the given binary function are encoded using run-length coding.

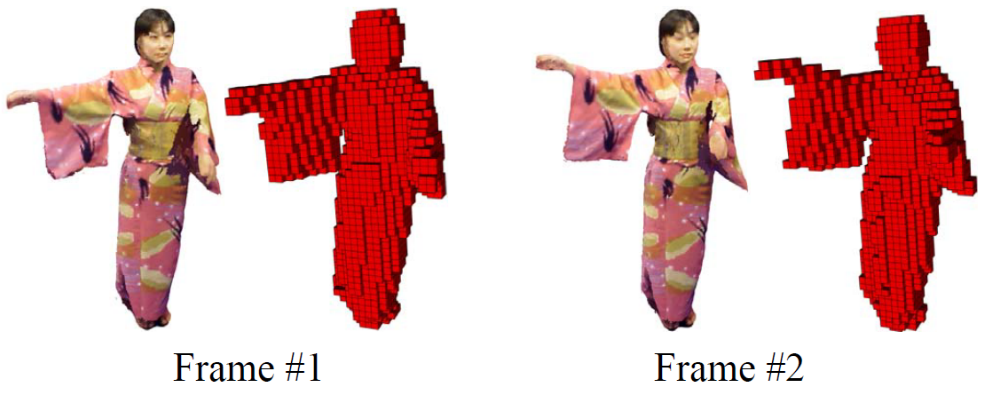


Figure 4.3: Block splitting of the TVM. There is visible temporal coherence of occupied blocks between consecutive frames. Source: [40]

This method does not consider connectivity data either. Although it is not explicitly stated, the approach changes vertex ordering, and some vertices might also be deleted in the fine-level quantization process. This makes the connectivity coding even more difficult. However, if the vertex order does not matter, the method can also be used for compression of dynamic point clouds. Although a comparison with [39] was done showing great improvements, the results still cannot be related to any static mesh compression approach.

4.5 Patch-Based Compression for Time-Varying Meshes

The first method that accounted for connectivity data was proposed by Yamasaki – Aizawa [109]. They proposed two approaches, inter- and intra-frame. Both approaches divide each frame into patches of almost constant surface area. PCA is then applied to each patch and it is transformed so that its centroid lies at the origin and its principal components are aligned with the coordinate system.

In the intra-frame approach, the patch is then compressed using spectral compression [49]. In the inter-frame approach, the corresponding aligned patch from previous frame which minimizes vertex-wise Chamfer distances is found. Instead of encoding IDs of corresponding vertices, for each vertex only a certain number of correspondences is encoded, which results in reordering. Then, the corresponding vertex is used as a prediction and the residuals are encoded using vector quantization.

Although they did not propose any specific method for connectivity encoding, they were the first to admit that it significantly influences the resulting bitrate. The method was compared to both [39] and [40], with connectivity data omitted. The authors attempted to make a comparison with static mesh compression methods, which, however, were not suited for the data used in the experiments (non-manifold meshes with irregularities and noise that were produced by 3D scanning techniques).

4.6 Deformation-Based Data Reduction of Time-Varying Meshes for Displaying on Mobile Terminals

Nakagawa et al. [68, 76] proposed a skeleton-based compression method motivated by the limited processing and rendering capabilities of smart mobile devices at the time of publishing their work. To be able to decode and render time-varying triangle meshes at high enough rates using such devices required a drastic data reduction and low computational complexity on the decoder side. The method is a direct continuation of their previous work described in [68]. Since the main idea behind both approaches is identical, we will describe only the latter approach.

The overview of the method is shown in Figure 4.4. First, the skeletal information is extracted for each frame. After that, all the frames are simplified using a modified QSLIM algorithm [35], which preserves the important parts of the surface, such as the hands, face, and surroundings of skeletal joints. A representing frame (e.g., first frame) is then propagated in space using skeletal information to match the subsequent frames, creating a mesh sequence with constant connectivity described only by the representing frame and its deformation, while the original sequence is discarded. A separate reference triangle mesh with precomputed skeletal information is available to both the coder and the decoder and is used to further reduce the data rate by transforming it to match the currently coded frame. For each point on the frame, a corresponding point on the reference mesh is found by considering the relative position of the point to the skeleton and is used as a prediction. Only the skeletal transformations and correction vectors are encoded.

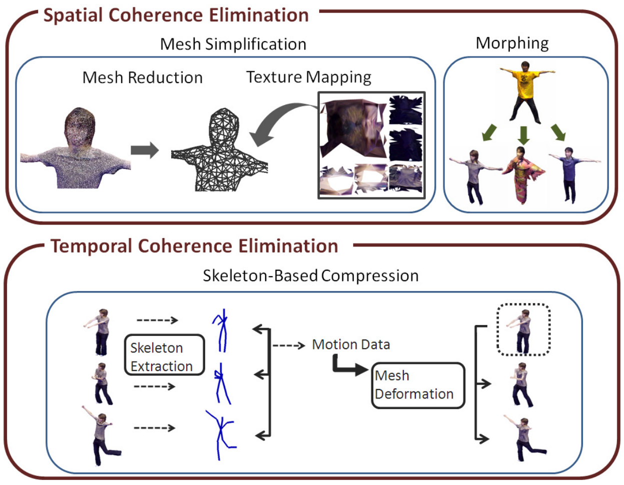


Figure 4.4: Overview of the compression method proposed by Nakagawa et al. Source: [76]

The method can be used only for human models, and discarding the original data causes loss of the one-to-one correspondence between encoded and decoded vertices. Also, the method depends on the quality of the skeletal information found. To address this, the authors created an interactive application in which the user could modify this information to obtain better results. The method achieved high compression rates and low decoding complexity, which allowed it to be performed on mobile devices at the time. Since then, however, the computational and rendering capabilities of mobile devices have increased significantly and these obstacles are of no concern. The increased resolution of the mobile displaying devices also makes the high distortion of the decompressed data more apparent.

4.7 Toward Real-Time and Efficient Compression of Human Time-Varying Meshes

The method by Doumanoglou et al. [23] was motivated by the needs of tele-immersion systems. Its main idea was to exploit the skeletal information extracted in real time by the 3D scanning tools used to produce human time-varying meshes. They introduced three types of frames that should be treated differently: I (intra-coded), P (predicted), and EP (enhanced P).

The I-frame is treated as a static triangle mesh and compressed using the method proposed in [71]. P-frames are reconstructed by animating the last encoded I-frame using rigid skinning. Although the P-frame is described in the paper, the final version of the method uses only I and EP-frames. To encode an EP-frame, the last encoded I-frame is used for prediction. Rigging is performed on both frames and a per-bone ICP algorithm is used to describe skeletal motion as a rigid transformation of bones. Those transformations need to be encoded to be known to both the encoder and the decoder. Next, the I-frame is transformed and a nearest neighbor for each vertex of the EP-frame is found and used as a prediction. Similarly to [109], instead of encoding the correspondences explicitly, the number of correspondences for each vertex of the I-frame is encoded, leading to a reordering of vertices of the EP-frame.

To force the reordering in the encoded connectivity, the authors proposed a modification of the TFAN algorithm [71]. They assumed there is a high probability that if two vertices share an edge, they are both in the set of k nearest neighbors of each other ($k \ll |V|$). During the TFAN traversal, for each processed vertex v_i, an ordered list $N_m(v_i)$ of m nearest neighbors is constructed. If a topological neighbor v_j is part of the list, its number excluding already encoded vertices is encoded. Otherwise, the correct index value must be encoded, indicated by a negative sign.

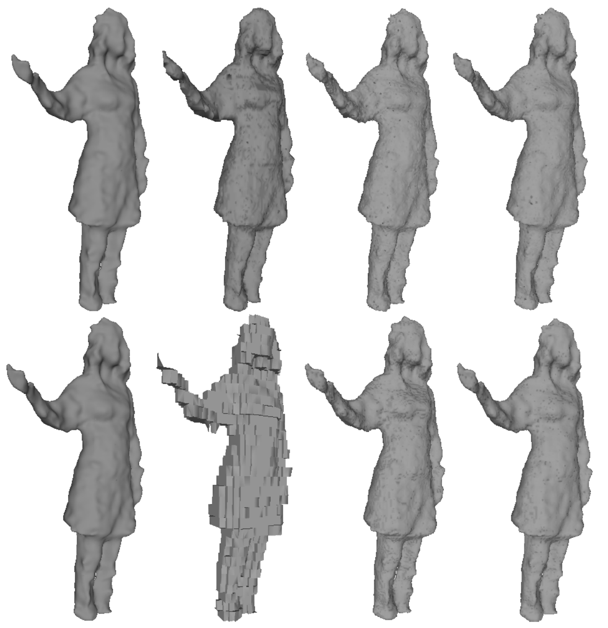


Figure 4.5: Comparison of compression result on a single frame of the Xenia sequence [7]. The first picture of every row is the original uncompressed mesh. Top row from left to right: method proposed by Doumanoglou et al. at 5.7 | 10.81 | 13.4 bpvf. Bottom row from left to right: TFAN [71] at 7.5 | 12.58 | 14.75 bpvf. Source: [23]

The authors also propose a slight modification of the method by segmenting the human model into parts according to the skinning information and encoding only the parts with larger movement. The rest is reconstructed by animating the corresponding part of the I-frame. After that, the parts are merged together. Although this results in a lower data rate, original connectivity information is lost.

Doumanoglou et al. [23] were the first to admit that the previous time-varying mesh compression methods did not compare well to the static mesh compression approaches. Their approach, on the other hand, performed better than [71] at lower data rates, and comparably at higher data rates (see Figure 4.5). Its main limitation is that the method is suited only for humanoid models due to the fixed skeletal structure proposed by the authors. It can be altered to work with other data by adjusting the structure; however, it is not suited for data in which the ground truth structure is changing over time. Nevertheless, the erroneous topological changes in human data due to self-contact should result only in an increased data rate due to the imperfection of the predictions. This method is considered the state of the art of human time-varying mesh compression methods preserving the original connectivity.

4.8 A Novel Compression Framework for 3D Time-Varying Meshes

A method proposed by Hou et al. [42] is based on so-called geometry videos (GVs) [13]. It transforms time-varying 3D data into a sequence of 2D images by encoding the x, y, and z coordinates as R, G, and B channels. To obtain a GV, a polycube parameterization is performed, mapping the salient points of each frame onto the same position in the parametric domain. Next, the polycube is flattened and fitted into an image (see Figure 4.6).

The GV is further reduced before encoding. For each dimension (resp. a color channel of the geometry video), a matrix $\mathbf{M}_d \in \mathbb{R}^{V \times F}$ is formed, where V is the number of vertices of GV (these are not the vertices of the original models) and F is the number of frames. A matrix $\mathbf{K}_d \in \mathbb{R}^{V \times k}$, where $k \leq F \ll V$ is a compression parameter controlling the number of preserved values, is computed as a low-rank approximation of $\mathbf{M}_d$ using randomized SVD, which is a form of dimensionality reduction. The reduced matrices $\mathbf{K}_x$, $\mathbf{K}_y$, and $\mathbf{K}_z$ are then reshaped to form so-called EigenGVs, which have a reduced number of frames. The EigenGVs are then compressed using a standard video compression algorithm.

To reconstruct $\mathbf{M}_d$, a reconstruction matrix $\mathbf{W}_d \in \mathbb{R}^{k \times F}$ is found so that

$$\mathbf{M}_d \approx \mathbf{K}_d \mathbf{W}_d.$$

This is done by minimizing energy $E(\mathbf{W}_d)$:

$$E(\mathbf{W}_d) = \left\| \mathbf{M}_d - \mathbf{K}_d \mathbf{W}_d \right\|_F^2 ,$$

where $\| \cdot \|_F^2$ is the Frobenius norm of the matrix. The reconstruction matrix is encoded directly with no compression applied.

The main issue with this approach is that converting the sequence into a GV causes loss of the original connectivity of all frames and there is no one-to-one correspondence between the original and decoded vertices. The GVs have a regular structure-induced connectivity that is the same across all the frames, thus the reconstructed sequence is no longer a TVM, but a DM. Additionally, the polycube used for parameterization must be manually set prior to the compression, must remain static through all the frames, and must reflect the topology of the data. Thus, the method is actually limited only to TVMs with constant topology, which makes it impractical in real-world compression scenarios.

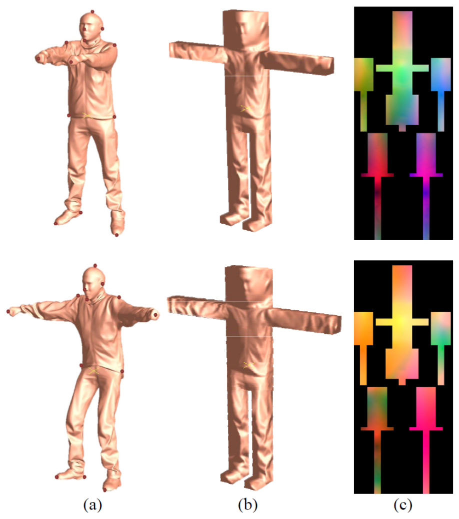


Figure 4.6: Process of obtaining a geometry video. (a) Salient points. (b) Polycube parameterization. (c) Geometry video. Source: [42]

4.9 Mesh Coding Extensions to MPEG-I V-PCC

A time-varying triangle mesh can be also compressed by combining a temporal coherence-based point cloud compression applied on its vertices and a connectivity codec. One such approach was presented in [27].

The method was one of the results of point cloud compression standardization efforts by the MPEG, which will be described in more detail in Section 5.9. The authors propose combining the video-based point cloud compression method (MPEG V-PCC [4]) with Edgebreaker [82] or TFAN [71] connectivity coding algorithms. The video-based codec divides the surface into patches with similar orientation, which are then projected into the image domain and encoded using video compression. However, the unmodified codec, reorders the vertices, which is in conflict with connectivity coding, which also results in reordering. It is thus impossible to relate the decoded positions and connectivity without encoding any additional data, e.g., an index map. To address this, the authors proposed to drive the patch generation process of the V-PCC by the order of vertices in the connectivity coding traversal.

Compared with [31], the method performed slightly worse, possibly because the patch generation process was not as effective as in the original V-PCC. However, since the standardization is still in process, there is a high potential for follow-up work that will improve upon such results.

4.10 Summary of Time-Varying Mesh Compression

Although the earliest work on TVM compression [37] dates back only four years after the earliest work on dynamic meshes [58], the field remains mostly unexplored. The simplest methods based on motion estimation or prediction of spatial structures can handle general time-varying meshes. They are, however, inefficient in comparison with other approaches. The model-based methods are more promising since they achieve better compression performance. Unfortunately, none of the model based methods presented to date are suited for more general input. Both projection-based methods are also quite effective; however, they have their own limitations.

When considering the methods that preserve the isomorphism between encoded and decoded points, only the method shown in [23] outperforms state-of-the-art static mesh (intra-frame) compression, though it is applicable only to human time-varying triangle meshes. For more general data, the static mesh methods are still preferable. Although it was already addressed, connectivity coding remains one of the main challenges. The data rates of current methods at satisfyingly low distortion range from 15 to 20 bits per vertex which is significantly more than of dynamic mesh compression methods.

Chapter 5

Dynamic Point Cloud Compression

The classification of inter-prediction methods for TVMs can also be applied to dynamic point clouds. Table 5.1 shows an overview of all the methods described in this chapter.

Table 5.1: Overview of all methods presented in this chapter in the order of corresponding sections. *Iso.* denotes whether the method preserves one-to-one mapping between encoded and decoded vertices.

Section	Method	Type	Input data	Iso.
5.1	Lien et al. [62]	Model/Projection	Articulated	no
5.2	Daribo et al. [19]	ME/Special	Struct. light scans	yes
5.3	Kammerl et al. [48]	Structural	General	yes
5.4	Thanou et al. [93]	ME/Structural	General	yes
5.5	Mekuria et al. [73]	ME/Structural	General	no
5.6	Kathariya et al. [52]	ME	General	yes
5.7	de Queiroz – Chou [21]	ME/Structural	General	yes
5.8	Garcia – de Queiroz [33]	Structural	General	yes
5.9.1	MPEG V-PCC [4]	Projection	General	opt.
5.9.2	MPEG G-PCC [2]	Intra	General	opt.
5.9.3	Schwarz et al. [85]	Projection	General	opt.
5.9.3	Li et al. [61]	Projection/ME	General	opt.
5.9.3	Kim et al. [54]	Projection/ME	General	opt.
5.9.3	Cao et al. [14]	Model/Projection	General	no
5.10	Garcia et al. [34]	Structural	General	yes
5.11	Biswas et al. [10]	Structural	General	yes
5.12	Milani et al. [74]	Structural	General	yes
5.13	Feng et al. [29]	Model/Projection	LiDAR	no
5.14	Peixoto et al. [80]	Structural	General	yes

5.1 Multi-camera Tele-Immersion System with Real-Time Model Driven Data Compression

The method proposed by Lien et al. [62] was motivated to significantly reduce the transmission rate of geometry data in tele-immersion systems (e.g., in online tai-chi classes) with near-real-time performance.

The method is based on skinning and assumes that most points will move under rigid-body transform with the associated skeleton and that the appearances of the initial and remaining frames are usually similar. In order to fit the data model, a skeleton is estimated for the first frame point cloud P of the sequence offline during preprocessing. To encode a frame Q, a skeletal transform $T(P)$ must be found. To achieve this, the authors propose an articulated version of the ICP algorithm [9]. Q is first clustered according to normals represented in spherical coordinates and according to colors represented by hue and saturation. For each cluster, a k-d tree is constructed using the positions of its points. The skeleton is treated as a tree graph with bones as nodes and the torso as a root node in case of human data. The transforms of each bone are found recursively, with a parent node transform influencing the transforms of its children, using the ICP algorithm with correspondences found in the surrounding clusters by using a nearest neighbor search in the given k-d trees.

After the articulated ICP is finished, the $T(P)$ is roughly aligned with Q. However, the global error of the alignment can still be large enough. To this end, the Q and P are segmented so that each segment Q_l and P_l are assigned to a body part l. To calculate the global error for l, the second moment μ is used:

$$|\mu(P_l) - \mu(Q_l)|.$$

If this error occurs over a given threshold for any of the parts, a global fitting is applied. During this process, the articulated ICP algorithm is repeated, but only on those parts that have larger value than the given threshold, ignoring the rest. This process is iterated until the global error is low enough for all the body parts or until it does not result in further improvement. After that, the joint positions and angles of the skeleton are encoded.

To further improve the reconstruction, so-called prediction residuals are encoded. These are acquired by projecting both P_l and Q_l onto a grid embedded in cylindrical coordinates defined by l, which results in two sets of images: color and depth. Since P_l and Q_l are aligned, it is expected that the images are similar, and thus only their difference can be encoded to further reduce the data rate (see Figure 5.1). By changing the resolution of the grid, one can control the compression ratio and distortion.

The authors themselves have pointed out that the main limitation is the assumption of a rigid movement of body parts, which does not hold, for example, if the person has long hair or wears a skirt. However, such behavior in data results only in an increase of the data rate. Other issues might arise if the encoded sequence represents non-articulated moving objects, where the skeleton structure is dynamic. Additionally, the points of decoded point clouds do not correspond to the original, and thus distortion of the data is more difficult to measure. The authors address this by measuring the difference in the rendered images.

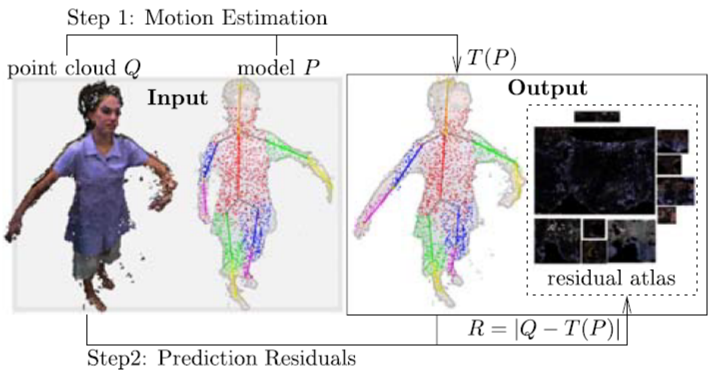


Figure 5.1: Skeleton-based compression of point clouds. Source: [62]

5.2 Efficient Rate-Distortion Compression of Dynamic Point Cloud for Grid-Pattern-Based 3D Scanning Systems

The work of Daribo et al. [19] is inspired by properties of grid-pattern-based scanning systems (e.g., [30, 53]). Such systems consist of one or more cameras and projectors. Each projector projects a different pattern onto the scanned object, which is then captured by the cameras (see Figure 5.2). The resulting points lie on the spatial curves formed by the distorted lines of the grid pattern (see Figure 5.3). The authors have observed that such points are also ordered so that if two points lie next to each other on a captured spatial curve, they are also next to each other in the output stream of the scanning system.

Generally, finding neighbors of a point in a point cloud is not a trivial task. However, the structure of the grid-pattern-based 3D scan data allows one to at least alleviate the problem by detecting the corresponding spatial curves. This is done incrementally – initially, there is an empty set of curves S and a curve C. Each incoming point $\mathbf{x}$ is tested to see if it is an outlier for C. If it is not, it is inserted into C. Otherwise, C is inserted into S and a new curve is created containing $\mathbf{x}$. After the partitioning is done, all the curves are coded separately.

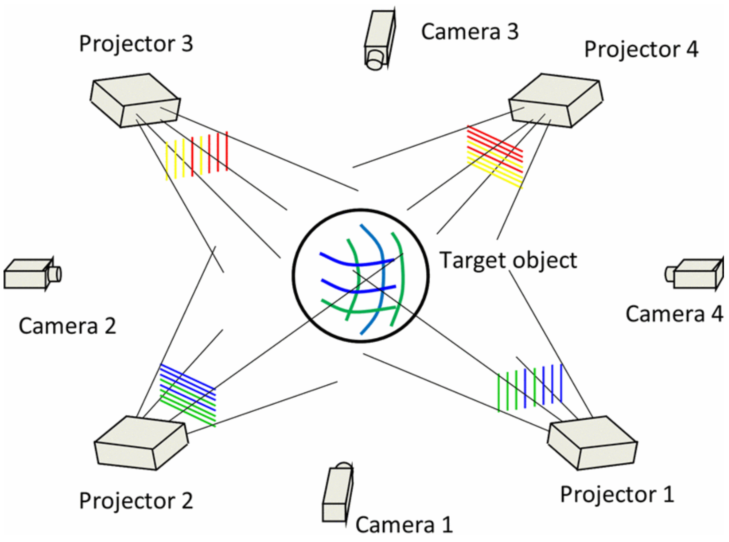


Figure 5.2: Example configuration of a grid-pattern-based scanning system. Source: [30]

To encode the points of the spatial curve, the method uses a competition-based predictive approach. Multiple spatio-temporal predictors are considered for each point, and the one that is the most efficient is selected, quantized, and encoded using entropy coding (e.g., a Huffman coder). The authors suggest considering the following predictions:

- No prediction at all,
- Previous point on the curve,
- Continuation of a line through the previous two points on the curve,
- Continuation of a line fitted to all the previous points on the curve,
- Continuation of a curve with a turning angle determined from the previous two points,
- Corresponding point on the closest curve in the previous frame to the currently coded curve, and
- Continuation of a curve with a turning angle determined from the most similar curve in the previous frame, considering curvature and torsion.

The prediction mode P that minimizes the following Lagrangian functional is selected:

$$J(\mathbf{x}, P, bp) = D_{pred}(\mathbf{x}, P, bp) + \lambda(bp) \cdot R_{rec}(\mathbf{x}, P, bp),$$

where $\mathbf{x}$ is the encoded point, bp is a quantization constant, $D_{pred}(\mathbf{x}, P, bp)$ is the distance between $\mathbf{x}$ and the prediction, $R_{rec}(\mathbf{x}, P, bp)$ is the number of bits needed to encode all the data needed for P, and $\lambda(bp) = 8 \cdot \exp(-1.53 \cdot bp)$ is a Lagrange parameter estimated experimentally by the authors.

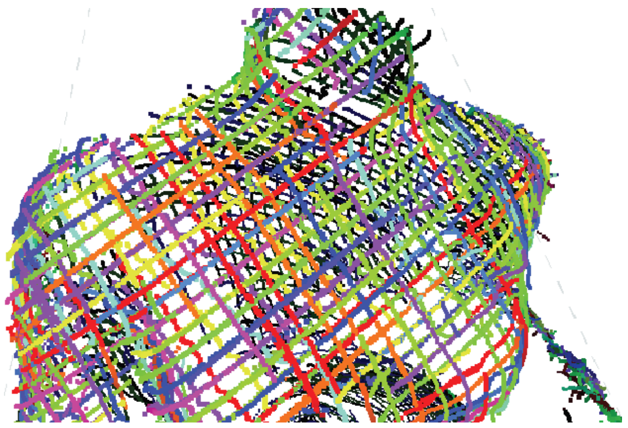


Figure 5.3: Points are fitted to spatial curves during grid-pattern-based scanning. Source: [19]

The method proposed by Daribo et al. [19] allows control of error propagation, random access, and parallel encoding since all the curves are encoded separately. It is also one of the few methods that do not disrupt the isomorphism between the original and decoded points. Unfortunately, it is limited by the expected structure of the data, which is generally not present.

5.3 Real-Time Compression of Point Cloud Streams

To the best of our knowledge, the method proposed by Kammerl et al. [48] was the first method suited for general dynamic point clouds to consider temporal coherence while preserving one-to-one mapping between encoded and decoded vertices. The method stores the frames of the sequence in an octree data structure. This structure can be serialized into a sequence of bytes during a breadth-first traversal from the root of the tree to the occupied nodes. Each non-leaf node yields a byte, whereas each bit represents occupancy of the child node (see Figure 5.4).

To address the temporal coherence, the authors proposed so-called double-buffering octree representing two consecutive frames of the sequence. Nodes of this structure contain 16 instead of 8 children, corresponding to the nodes of octrees of both frames. This allows one to serialize only the occurring changes by using an XOR operation between the occupancies of the two halves of the child nodes. The serialized sequence is then entropy coded.

The authors also compared two approaches to representing points in leaf nodes. The first approach was to use a centroid of the voxel of the node. This, however, results in the loss of the one-to-one mapping if two or more points are in the same voxel. It can be resolved by increasing the resolution of the octree. However, the computational and memory requirements increase exponentially. Another approach is to use the origin of a voxel as a prediction value and encode the corrections. Unfortunately, this means additional values must be encoded.

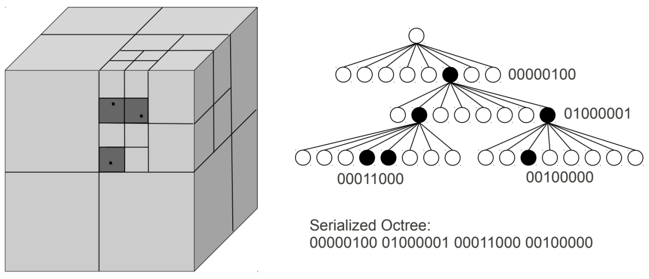


Figure 5.4: Serialization of the octree. Source: [48]

5.4 Graph-Based Compression of Dynamic 3D Point Cloud Sequences

The method proposed by Thanou et al. [93] can be considered an improved version of the previous method. It also represents point clouds using an octree. However, motion compensation as presented in [92] is used to reduce the bit rate (see Figure 5.5).

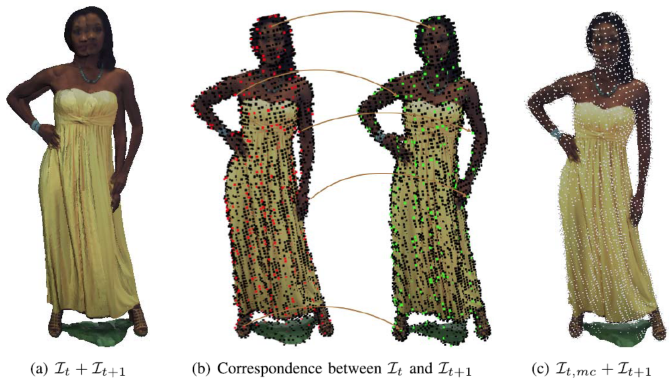

(a) $\mathcal{I}_t + \mathcal{I}_{t+1}$
(b) Correspondence between $\mathcal{I}_t$ and $\mathcal{I}_{t+1}$
(c) $\mathcal{I}_{t,mc} + \mathcal{I}_{t+1}$

Figure 5.5: Motion estimation of two frames of the sequence. (a) Superposition of both frames. (b) Correspondence estimation. (c) Superposition of the encoded frame and motion-compensated first frame. The cubes represent the leaf voxels. Source: [93]

To compensate for motion between two frames of the sequence, where the first of the two frames is already known to both the encoder and the decoder, both frames are inserted into octrees, and k-NN graphs are constructed between leaf voxels. The positions and colors are then interpreted as signals on vertices of the graph. Then, feature vectors ϕ_i, where i is the index of a vertex of a graph, are computed for each vertex of both graphs using spectral graph wavelets. Instead of computing the full spectrum of the graph Laplacian matrix, the wavelets are approximated using Chebyshev polynomials [38]. The vertices between graphs are then matched using the Mahalanobis distance:

$$\sigma(m, n) = (\phi_m - \phi_n)^T \mathbf{P}(\phi_m - \phi_n),\ m \in V_t, n \in V_{t+1}$$

where V_t and V_{t+1} are sets of vertices of the two frames and $\mathbf{P}^{-1}$ is a covariance matrix estimated from training features that are known to be in correspondence and describes the relation between different feature components. After that, the vertices are clustered and a representing vertex with the best score in each cluster is selected. The motion vectors of the selected vertices are initially set as displacements to their correspondences. These values are then propagated to the rest of the graph during an optimization process that enforces smoothness of motion vectors with respect to the neighboring vertices. The vectors are transformed into the frequency domain using GFT, quantized, and encoded using an RLGR coder [70].

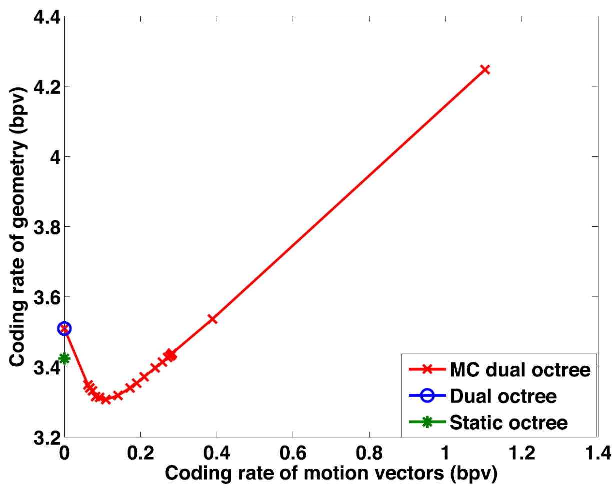


Figure 5.6: Relation between the coding rate of motion vectors and the coding rate of geometry (including the motion vectors). The comparison was also made to the static octree compression method proposed in [51]. Source: [93]

Then, the previous frame is motion compensated using the reconstructed motion vectors, and the result is used to compute the serialized difference in the octree structure as described in [48]. The authors show in their experiments, that when the motion vectors are encoded at a low bit rate (≈ 0.1 bpv), the method achieves slightly better results (see Figure 5.6). The precision of the motion vectors affects only the bit rate.

5.5 Design, Implementation, and Evaluation of a Point Cloud Codec for Tele-Immersive Video

To meet the requirements of their tele-immersion system [106], Mekuria et al. [73] proposed a progressive compression algorithm with near-real-time performance. Like many other compression methods, it works with I- and P-frames.

The I-frame is encoded as follows: In the first step, outliers are filtered and a bounding box is calculated. The bounding box is then slightly expanded by a certain margin. The box is used for the octree representation of the point cloud. The geometry is encoded using a modified version of the approach in [48], which allows coding of two levels of detail. For color compression, the values are serialized to a grid in a zig-zag pattern in order of depth-first tree traversal and encoded using a JPEG codec.

To determine whether the frame is intra- or inter-coded (I or P), the method checks whether it fits into the extended bounding box of the previous I-frame. If it does not, it is treated as a new I-frame and coded as such. Otherwise, the extended bounding box of the previous I-frame is used for the octree representation of this frame as well. For octrees of both I- and P-frames, macroblocks are generated at level K above the final level of detail (K is a parameter, and the authors chose $K = 4$). For each macroblock, a decision is made whether it should be inter- or intra-coded. To be able to code a macroblock using prediction, the same macroblock should be occupied in the I-frame as well, their number of points and color values should not differ much, and ICP registration should result in rigid transformation with a fitness value over a certain threshold. If a macroblock fits all the criteria, it can be replaced by the predicting macroblock. Only a rigid transform, color difference, and indices representing the position of the predicting macroblock are thus encoded. The indices of macroblocks are encoded to allow the macroblock coding to be parallelized. If the macroblock does not match all of the criteria, it is more conveniently encoded in the intra-predictive manner. For this purpose, a separate point cloud frame consisting of all the rejected macroblocks is constructed and encoded separately.

The algorithm is simple enough to meet the near-real-time requirements of tele-immersion systems. It was also used as a first reference platform for the development of point cloud compression in MPEG-4 [3]. However, since some of the blocks of the point cloud in a P-frame are replaced by blocks of another point cloud, it is no longer possible to trivially calculate the distortion of the data. For this reason, the authors have proposed using the sum of the distances to the nearest points of the distorted point cloud for each vertex of the original point cloud. Additionally, they have carried out a user study to evaluate the subjective quality of the compression. However, the study was more driven to compare the algorithm with the avatar user representation that was formerly used in their system (see Figure 5.7). No comparison with any other compression method was done.

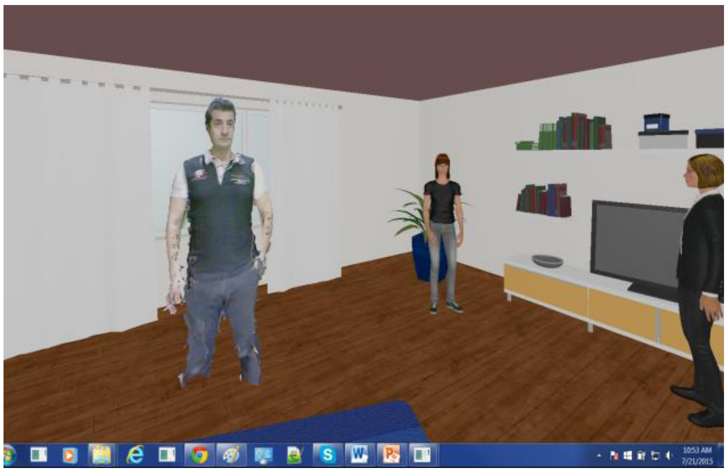


Figure 5.7: Reverie tele-immersion system [106]. Comparison with real-time encoded point clouds and synthetic avatar-based representations. Source: [73]

5.6 Embedded Binary Tree for Dynamic Point Cloud Geometry Compression with Graph Signal Resampling and Prediction

Kathariya et al. [52] proposed a lossy compression based on motion compensation between blocks represented by cells of a k-d tree.

The method first divides the frames into subgroups. For each subgroup, a k-d tree is constructed from the points of the first frame. The tree represents a partitioning of the domain into blocks, and this partitioning is reused for each following frame in the subgroup. Each frame is encoded block-wise using the corresponding blocks in the previous (reference) frame. First, the block in the reference frame is resampled so that the number of points in both blocks is identical. To this end, the authors proposed to construct a k-NN graph ($k = 10$) and to use eigenvectors of the graph Laplacian matrix to cluster the points. Each eigenvector partitions the graph into a set of connected groups of vertices with the same sign of values it assigns to them. The number of groups grows with the corresponding frequency that the eigenvector represents. The first partitioning that yields at least $|m - n|$ groups, where m is the number of points of the currently coded block and n is the number of points in the reference block, is selected. If there are fewer points in the reference block, a centroid for each group of points is inserted. Otherwise, the median point of each group is deleted. After that, the correspondences are found by graph pruning of a k-NN graph between the two blocks until the graph is bipartite and the vertices of the coded block are reordered to match the order of the corresponding reference points. The residues are then computed between the corresponding reference and coded points. The residues are treated as points centered around the origin and encoded using octree serialization. It is unclear from the paper how to relate the residuals back to the points, since such coding alters the number of points and results in additional reordering.

According to the authors, to be able to reconstruct the values, the decoder needs to know which eigenvector was selected for partitioning, the serialized octree and its bounding box coordinates. For the first frame of each subgroup, there is no reference for prediction. Thus, instead of encoding residues, the positions themselves are encoded in the octree.

The authors did not provide any rate-distortion or performance comparison with other compression methods, which is surprising since the dataset used in their experiments is bundled with the implementation of [73]. Additionally, one could be concerned with the computational complexity of the method since it requires finding an unknown number of eigenvectors of a graph Laplacian matrix for each block of the frame.

5.7 Motion-Compensated Compression of Dynamic Voxelized Point Clouds

The method proposed by de Queiroz – Chou [21] also works with I- and P-frames. It represents the voxels of the point cloud with x, y, and z coordinates of the centroids and their YUV color values. For an I-frame, a static octree compression is applied, combined with Region-Adaptive Hierarchical Transform (RAHT) [20] coding for color compression.

A P-frame is divided into $N \times N \times N$ blocks, where N is a parameter, and motion vectors are computed for each block. The authors do not propose any method for motion estimation; however, they suggest that motion vectors are sometimes a byproduct of point cloud reconstruction methods (e.g., [22]). If the motion vector of each voxel in the block is known, the representing motion vector of the block is selected as one that is closest to the average value. The method then decides whether the block is inter- or intra-coded based on the estimated encoding rates and distortions of both approaches.

The calculated distortion consists of correspondence and projection-based metrics. The correspondence metric is calculated using spatial and color distances between corresponding voxels. For the projection-based metric, the point cloud is projected onto six sides of a cube at the limits of the voxel space and differences in Y color values are used. The intra-coding of the P-frame block also uses a static octree method and RAHT for color compression. If the block is inter coded, only a motion vector is encoded using the RAHT.

The authors describe blocking artifacts that might occur when using inter-prediction. To address this issue, they propose using a smoothing filter followed by morphological closing to close any gaps.

5.8 Context-Based Octree Coding for Point-Cloud Video

Garcia – de Queiroz [33] proposed a simple lossless compression method based on the work of Kammerl et al. [48]. It assumes that an encoded octree has a similar distribution of symbols representing the nodes to that of the prediction.

During the encoding, the predicting tree is constructed similarly to the method in [48], with a single difference in the fact that Garcia – de Queiroz propose using multiple frames merged into a single octree instead of a single previous frame. However, instead of encoding the XOR of the two octrees, the nodes of the prediction octree are sorted in ascending order of the bytes representing their occupancy, since they can be interpreted as numbers between 0 and 255. The sorting order of the prediction is applied to the encoded octree, which is then coded. Since the sorted prediction contains long sequences of repeated symbols, the reordered data should in theory contain such sequences as well. Additionally, the worst case results only in a scrambled version with similar properties to the original octree.

Context-based octree coding is very simple and outperforms [48]. Although the authors did not make that claim, in theory, one could be able to combine it with [93] as well.

5.9 MPEG Standards for Point Cloud Compression

The growing need for an efficient compression method for point cloud data was also recognized by the Moving Picture Experts Group (MPEG). This led to the call for proposals in early 2017 [3]. The work of Mekuria et al. [73] was selected as a baseline for comparison with all the proposed methods. As a result, 13 proposed solutions were collected from various industry and research contributors and three different test model cases were identified: TMC1 for static data (e.g, cultural heritage), TMC2 for dynamic data, and TMC3 for dynamically acquired data (e.g., LiDAR). Eventually, due to the similarities in the approaches, TMC1 and TMC3 were merged to form TMC13, which led to the development of geometry-based point cloud compression (MPEG G-PCC), the ISO/IEC 23090-9 standard [2]. The method evolved from TMC2 is called video-based point cloud compression (MPEG V-PCC), the ISO/IEC 23090-5 standard [4]. As of early 2021, the V-PCC was in the approval stage of the final draft, while the draft of the G-PCC was still being finalized. Since both methods are quite complex and still under continuous development, we will give only an overview of the approaches in both codecs.

5.9.1 MPEG V-PCC

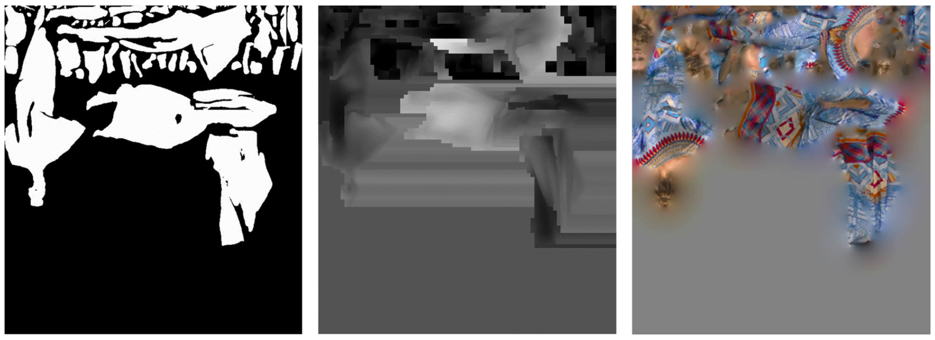


Figure 5.8: Example of the images encoded in V-PCC (from left to right): occupancy map, geometry image, attribute image. Source: [36]

The motivation for the video-based codec was to quickly deliver an efficient method benefiting from video compression, which is already a mature field of research. Similarly to [42], it is based on compressing projected geometry into the image domain. First, the point cloud frame is divided into patches. For each point, a normal vector is estimated. The point is then assigned to one of six principal orthographic projection directions by selecting the direction that yields the highest dot product with the normal vector. The assignment is refined with the local neighborhood of the point taken into account. The points are then clustered using the connected component algorithm and projected to the faces of the bounding box in the assigned direction. This results in a set of 2D patches $H(u, v)$, where u and v are the coordinates not altered by the projection and the value is the distance of the point along the projection. It is important to note that multiple points can share the same u and v in a single patch, which must be accounted for if the compression should be lossless. After that, all the patches must be placed into the image domain. The patches are sorted by size and incrementally inserted into the image in the first empty place found in the raster scan order, considering eight different rotations of the patch. To exploit the temporal coherence, the method attempts to place similar patches in the same place as in the previous frame. This is done by estimating correspondences using the intersection-over-union (IOU) metric. A patch pair is considered corresponding when its IOU is over a specified threshold. Not only does this allow more efficient video coding, but it can also be used later. After the position of each patch is established, three sets of images are constructed: geometry and attribute images, and occupancy maps (see Figure 5.8). The geometry images contain the values $H(u, v)$ in the luminance channel, and all the other channels are zero. Attribute images contain the colors assigned to points and have the same resolution as the geometry images. To address the multiple points projected onto the same pixel, more than one geometry and attribute image might be constructed for a single frame. The occupancy maps are binary images that indicate the parts of the image domain that contain the desired information. They can be encoded in a lossy manner by reducing the resolution in comparison with the other two images. To utilize the properties of the video codec used, the empty pixels in the geometry and attribute images are filled to obtain a piecewise smooth transition of values. After that, all the images are encoded using a state-of-the-art video compression algorithm (e.g., MPEG-HEVC [89]). To reconstruct the values, patch metadata (e.g., position in the 2D and 3D domain and bounding box size) is encoded as well. The temporal patch correspondence, if available, can be used to predict such values. After the reconstruction, the positions and attributes of the patch boundary points can be smoothed to attenuate any visual artifacts that might occur.

5.9.2 MPEG G-PCC

Contrary to the previously described standard, the G-PCC encodes the point cloud directly in 3D. The attribute data is encoded separately and the description of the coding process will be omitted. The standard consists of three geometry compression modes: octree entropy coding, trisoup, and geometry prediction.

The octree entropy coding mode represents the point cloud by an octree with variable depth, which means that not all leaf nodes are on the same level. In fact, the boxes of the octree are subdivided until they contain only a single point or until a maximum depth is reached. The tree is then encoded during a breadth-first search traversal by an entropic coding strategy with context modeling called OBUF (Optimal Binarization with Update On-the-Fly), in which an occupancy bit is encoded by a coder chosen from a set of N continuously updated entropic coders by examining the dependency state of the bit. The state is determined from information available to both sides, e.g., the occupancy of already coded neighbors of the current node, parent node, and its neighbors and value predictions. The predictions are ternary functions of values: predicted occupied, predicted unoccupied, and unpredicted. Currently, only a single intra-prediction method is utilized, which takes the weighted sum of adjacent node values and deduces the value by thresholding. However, the authors plan to propose other types of predictors, including ones utilizing inter-frame coherence, for future successors of the standard [56]. The OBUF strategy is, however, inefficient for isolated points since such points have no neighborhood for deducing the dependency state. For such points, their relative position to the node volume is directly encoded and the selection of the direct coding is indicated in the stream.

The trisoup approach represents the coded point cloud by octree with reduced depth. After encoding the octree parent nodes, the points within each leaf cell are represented by a surface that intersects with each edge of the block at most once. Instead of coding the points, only the vertices, which are all the intersections between the surfaces and edges, are encoded. The intersections are shared between neighboring blocks. For each edge, a flag representing whether it contains an intersection or not is encoded. Then, for each occupied edge, a scalar is coded representing the position of the intersection. The scalar value is quantized to a specified number of levels of precision. The decoder reconstructs the surfaces by triangulated non-planar polygon. The reconstructed point cloud is then generated by regular grid sampling of the triangles.

The geometry prediction is a mode introduced later in the development of the standard [32]. It was designed for LiDAR data, however, its core concepts could be used to compress general point clouds. It is based on reordering the points in space (e.g., using Morton codes or azimuthal sorting). A prediction tree is then constructed over the points, where up to three hierarchical predecessors are used to predict the positions of the points. There are multiple prediction modes and multiple strategies for constructing the prediction tree. For the sake of simplicity, these are omitted here.

Despite the original use cases, both standards can be used to represent general point cloud data. The main difference is that for now, V-PCC is more suited for dense clouds, while G-PCC is better for sparse data. One limitation of the V-PCC codec is that it uses only the temporal coherence in the image domain since the position of the patch in 2D does not correspond to its position in 3D. In the future, V-PCC will improve with the development of video coding, while G-PCC will be improved itself [15]. It is expected that G-PCC will utilize inter-prediction and should at some point outperform the video-based codec even for dense data.

5.9.3 Inter-based Methods Directly Inspired by MPEG Point Cloud Compression Standardization Efforts

The development of both MPEG standards motivated a considerable amount of effort that resulted in quite a few scientific papers studying various aspects of point cloud compression. For the sake of simplicity, in this section we will describe only the work that studied the inter-frame coherence of the data and was not eventually included in the standard itself.

The method proposed by Schwarz et al. [85] was one of the contributions to the initial call for proposals. It was similar to the one selected for TMC2, which evolved into the V-PCC standard. The approach was also video-based. However, instead of segmenting the point cloud into patches, it is orthogonally projected as is onto a series of planes rotating around the bounding box. Since some of the points might be occluded by the projection, the point cloud is projected multiple times onto the same plane while removing the previously projected points (see Figure 5.9). Instead of using the occupancy map, this information is signaled through the geometry image. This, however, means that only the attribute image can be padded to improve the compression performance of the video codec. Although the method is not as effective as V-PCC, it is much simpler since no patch segmentation is required. Also, the projection method allows one to utilize more temporal coherence information since points of two subsequent frames projected on the same pixels are usually close to each other spatially. This, however, does not hold for V-PCC.

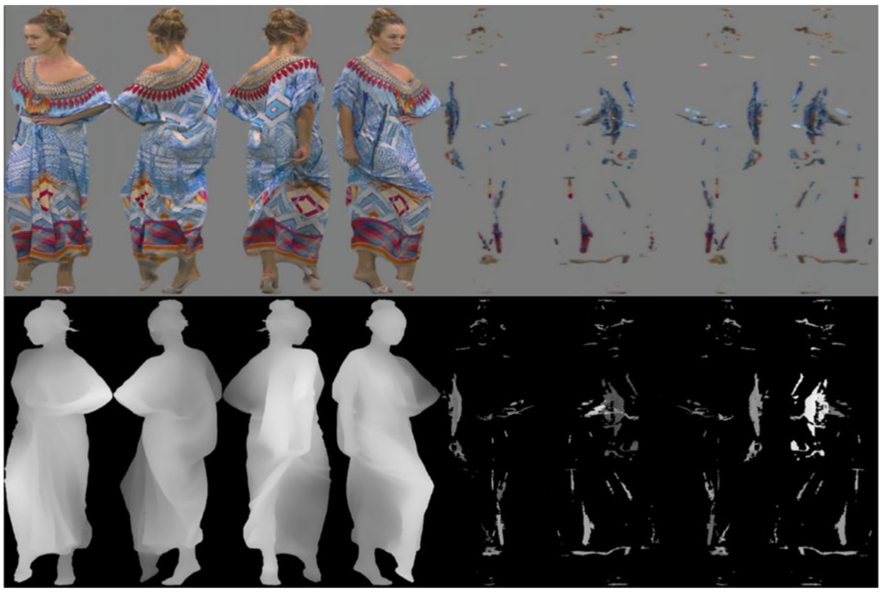


Figure 5.9: Projected attribute and geometry information. After four projections, the successfully projected points were removed. Source: [85]

Li et al. [61] pointed out that projecting the patches onto the image domain causes the loss of a decent amount of temporal coherence. For video coding, V-PCC usually uses the HEVC algorithm [89], which uses motion prediction in the image domain, where the predicting vector is selected from a candidate list. The authors propose estimating the correspondences in 3D by searching for the nearest points in the previous frame. Then, a 2D motion vector is extracted from positions of both points in the image domain and is inserted into the candidate list. While this approach results in compression improvements, it relies heavily on the fact that the HEVC algorithm is used, which is not enforced by the V-PCC standard.

To address the issues of [61], Kim et al. [54] proposed a 3D motion-estimation-based method independent of the video codec used. The method works with intra (I) and predicted (P) frames. A frame is considered an I-frame if it cannot be efficiently predicted by the previous I-frame. Such frames are encoded using the original V-PCC method. When a P-frame is encoded, each point of the I-frame is paired with a corresponding point in a search range of the P-frame based on similarity of color. Instead of the geometry and attribute images, the motion vector and attribute difference are encoded. Since the x, y, and z coordinates of the motion vectors are independent, those are encoded as separate images. The P-frame is reconstructed by translating the points of the I-frame and adjusting the colors using the encoded data. Compared to the original V-PCC, the method performs well at lower bit rates. However, the MPEG standard is still more efficient for higher bit rates.

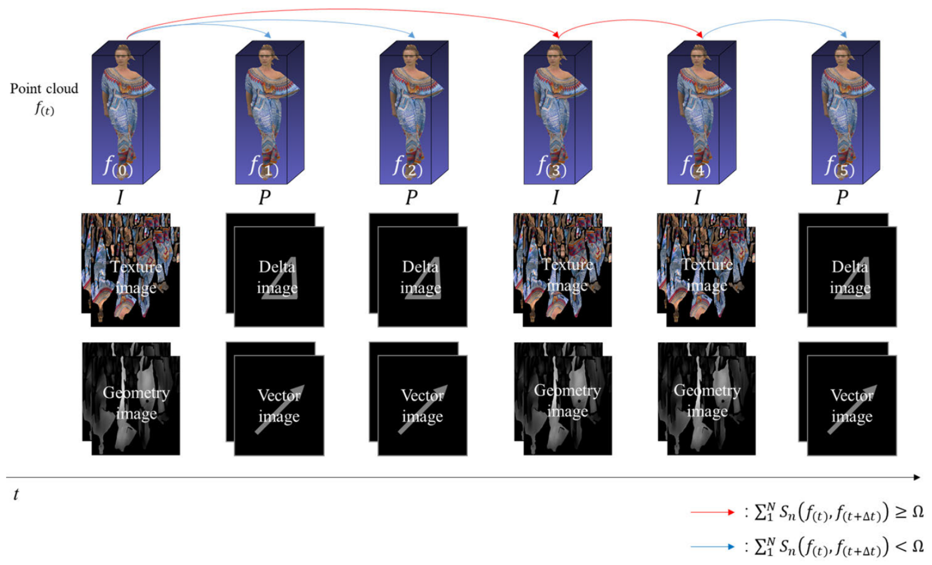
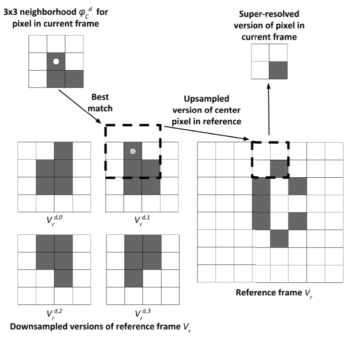


Figure 5.10: Motion prediction of [54]. The arrows on top denote which frame was used for prediction, and the red arrows indicate a failed prediction. Source: [54]

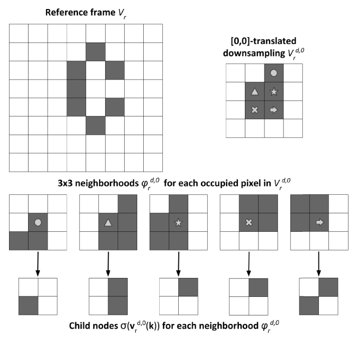
Cao et al. [14] proposed a V-PCC-based method for compression of dynamic human point clouds. The method segments points of each frame into detected body parts by projecting the point cloud orthogonally onto four planes around the bounding box and applying 2D pose detection on each of the projections. Each point is then classified considering the assignments found by the pose detection algorithm. Then, for a pair of subsequent frames, an affine transform for each body part is found. The sequence is subdivided into groups of subsequent frames. The first frame of each group is encoded using the original V-PCC algorithm. For the rest of the frames in the group, only the metadata representing the affine transformations is encoded. To reconstruct the frames, the first frame of the group is deformed using the reconstructed transformations. Finally, an interpolation is applied on boundaries of each body part to prevent gaps. Although the method achieves a decent bit rate reduction, it has quite a few limitations. For example, it relies heavily on the efficiency of the pose estimation algorithm. If the captured data contains deformable clothes, long hair, and objects, the pose estimation labels such surfaces incorrectly, which might lead to issues with estimating the affine transformations.

5.10 Geometry Coding for Dynamic Voxelized Point Clouds Using Octrees and Multiple Contexts

The method presented in [34] is based on an octree serialization (see Figure 5.4), similar to [33, 48, 93]. It is based on the assumption that once the predicting octree represents the encoded frame well, there are only a few possible outcome reconstructions for a single predicting byte representing the node of the tree.


3x3 neighborhood φ_c^d for pixel in current frame
Best match
Upsampled version of center pixel in reference
Super-resolved version of pixel in current frame
$V_r^{d,0}$
$V_r^{d,1}$
$V_r^{d,2}$
$V_r^{d,3}$
Downsampled versions of reference frame V_r
Reference frame V_r

(a) Super-Resolution by Example


Reference frame V_r
[0,0]-translated downsampling $V_r^{d,0}$
3x3 neighborhoods $\varphi_r^{d,0}$ for each occupied pixel in $V_r^{d,0}$
Child nodes $\sigma(v_r^{d,0}(k))$ for each neighborhood $\varphi_r^{d,0}$

(b) Super-Resolution by Neighborhood Inheritance

Figure 5.11: Inter-prediction modes for constructing the reference octree. Source: [34]

Each frame of the sequence is encoded as follows. First, the predicting octree O_p is constructed. Then, both octrees are serialized and the sequences are compared to find a map of frequencies ϕ_{ij}, which stores the number of predicting nodes with symbol i that predict a value j. The map is treated as a 3D surface and encoded as another octree with entries $\phi_{ij} = 0$ ignored. The first level of the coded octree is transmitted as is, and on each subsequent level, only the nonzero symbols are coded. For each symbol of the currently coded level, both the encoder and the decoder examine the value b in the prediction and select a context for the entropy encoding of the symbol determined from the probabilities of symbols relative to the symbol of b. Since only few possible symbols are expected when b is present, the context switching should, in theory, result in a much lower bit rate than using a single context for every symbol.

In order for the context switching to be effective, the prediction should be as close to the predicted octree as possible. To address this, the authors proposed three different prediction modes based on the fact that when encoding a level of the octree of the current frame, the previous frame and the previously encoded levels of the current frame are known to both the encoder and the decoder. The first mode is intra-predictive and can be used for the first frame. When encoding a symbol of a node using this mode, the prediction is simply the value of the parent node. Both other modes shown in Figure 5.11 are inter-predictive and use the previous frame octree O_r as a reference. The first inter-prediction mode, called *Super-Resolution by Example*, matches the neighborhoods of the previous levels of the current and previous frames. Once the voxel with the best-matching neighborhood within a spatial window is found, its value is used for prediction. To find better matches, the currently coded level in the reference octree is incrementally displaced in all directions, and down-sampled versions of each displacement are considered. In the second inter-prediction mode, called *Super-Resolution by Neighborhood Inheritance*, the reference octree is once again incrementally displaced in all directions and the symbols are analyzed to construct a dictionary, which for each voxel with a given neighborhood stores the most likely child configuration. Such values are then used for prediction. The prediction mode that yields the smallest Hamming distance to the correct value is selected for a set of nodes with a shared parent. A symbol indicating which mode was selected is encoded alongside the octree data using an arithmetic coder.

The proposed method is simple and easily extendable in the sense that one can introduce a different prediction mode from the three above mentioned. It compresses the data in a lossless manner, which can be considered an advantage but also a limitation since the lossy methods allow for additional bitrate reduction. Both inter-prediction modes are also quite complex and do not allow for near-real-time performance.

5.11 MuSCLE: Multi Sweep Compression of LiDAR Using Deep Entropy Models

Biswas et al. [10] focused on size reduction of LiDAR data. Their method is also based on octree coding. Like [34], it is based on modeling the probabilities of the symbols during entropy coding based on the context. The data they consider consists of the occupancy symbols, voxel intensities, and offset of points in each voxel. however, they claim that in the case of offsets, no patterns in the data could be found, so such data was coded directly. Since we focus on geometry encoding, the compression of intensities will also be omitted.

To model the probabilities of the occupancy symbols, the authors proposed trainning a neural network. The overview of the whole probability model can be seen in Figure 5.12. For each symbol, feature vectors are constructed using the information already known to both coding sides: the occupancy of the ancestors of the node and the occupancy of the previous frame. Such vectors are then processed by multilayer perceptrons, one for hierarchical and one for temporal data. The data from the previous frame is aggregated through continuous convolution [108] to create spatio-temporal feature vectors so that only the nodes spatially close to the currently coded node influence the probability. The spatio-temporal and hierarchical feature vectors are finally aggregated and given as an input to the neural network. The output layer represents each symbol $(0, \ldots, 255)$ and is converted to probabilities using the *softmax* function. The loss used to train the network is computed as a cross entropy with the probabilities derived from the already encoded data.

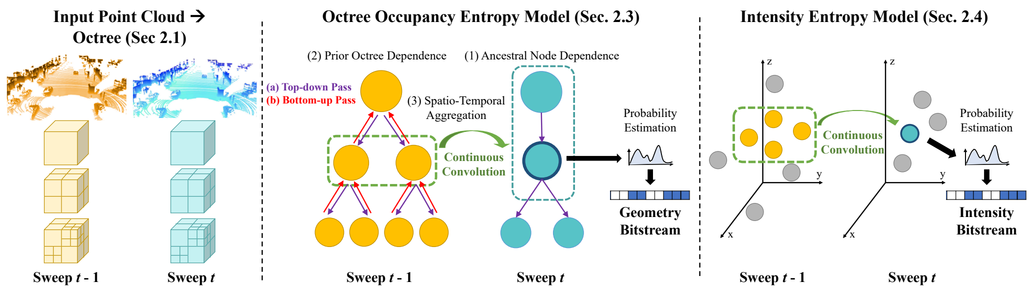


Figure 5.12: Overview of the deep entropy model. Source: [10]

The authors compared their results with multiple compression methods, e.g., [2] and [73]. In addition to compression performance, they studied how the compression impacts downstream tasks, for example, segmentation or detection. In every case, the method outperformed its competitors.

Although it was proposed to work with LiDAR data, it should, in our opinion, be applicable to a general dynamic point cloud. Additionally, only a single GPU pass is required per level of the octree in a frame, and thus the method should perform quite fast.

5.12 A Transform Coding Strategy for Dynamic Point Clouds

Recently, Milani et al. [74] proposed yet another octree coding strategy based on a cellular automata transform of symbols representing blocks of the octree (see Figure 5.13).

They assumed that if binary symbols representing the occupancy of nodes that are most likely to occur contain the greatest number of bits equal to 1, the encoded sequence, when treated in separate streams per each bit position in symbols, should contain large runs of 1, which could be exploited by the entropy coder. To be able to reconstruct the values, the transform must be reversible, which limits it to a permutation of the symbols. The authors propose sorting the symbols by probability, assigning the highest value (255) to the most likely symbol and the lowest (1) to the least likely one. The symbol 0 never occurs since such symbols can be detected from the structure of the previous level of the octree. The authors have proven that such a transform results in minimal bit rate when using an arithmetic coder. To decrease the entropy of the data, conditional probabilities given the context of the node, i.e., the already coded neighboring nodes, are considered. Instead of encoding the probabilities, which would lead to an increase of the data size, the authors propose two strategies. The first is intra-predictive and involves constructing the probability model on an example dataset known to both coder and decoder. Better results can be obtained by tailoring the probabilities to the actual data as in the second strategy, which is inter-predictive and calculates the probabilities using the previous frame.

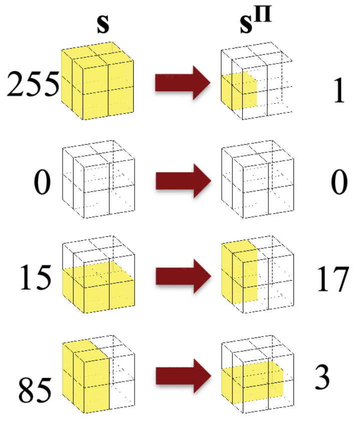


Figure 5.13: Example of cellular automata transform of symbols. Source: [74]

5.13 Real-Time Spatio-temporal LiDAR Point Cloud Compression

Feng et al. [29] proposed a compression method utilizing the properties of the LiDAR point cloud sequences, where each frame contains a full 360 degree view around the scanning device and is accompanied by internal sensory data that allows one to estimate the difference in position and orientation between frames. Figure 5.14 shows an overview of the method.

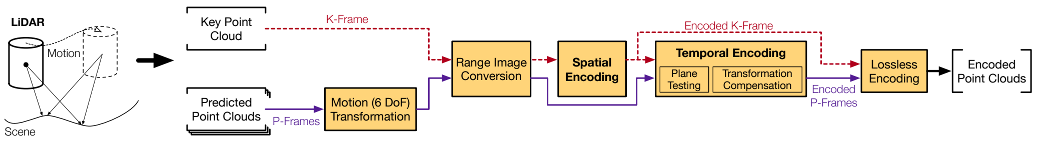


Figure 5.14: Overview of the plane-fitting-based LiDAR compression method. Source: [29]

The method considers two types of frames, K (key) and P (predicted). For the K-frame, it first converts the point cloud into a range image by transforming the Cartesian coordinates (x, y, z) of points into spherical coordinates (θ, ϕ, r). The θ and ϕ are quantized to obtain the coordinates of the corresponding pixel, while r represents its value. The pixels in each horizontal line are then processed by a divide-and-conquer plane-fitting algorithm, in which two neighboring pixel regions are merged if there is a plane representing both regions with error below a specified threshold. For each region, only the following values are encoded: the starting index of the region, the length, and coefficients a, b, and c representing the fitting plane $x + ay + bz + c = 0$. The positions are reconstructed by calculating the intersection between a ray of given θ and ϕ and the decoded plane. Since no additional offset is encoded, the error threshold for the plane fitting controls the distortion. Not all the pixels can be represented efficiently by plane fitting. Thus, a residual map is constructed for such points and encoded directly.

The P-frame is first transformed into the coordinate system of the previous K-frame using the internal sensory data. After that, it is converted to a range image as well. However, the transform causes some points to be quantized into the same pixel. If this occurs, the method considers only the point closer to the sensor, while the rest are discarded. To account for temporal coherence, the planes parallel to the ones found in the previous K-frame are used first. This accounts for some translation error introduced by the motion transformation and allows omitting the a and b coefficients of the plane. In the second phase, an attempt is made to fit different planes to the rest of the pixel groups. Such planes are encoded identically to how they were encoded in the K-frame. The rest of the pixels are again encoded in a residual map.

The method is fast enough to accommodate the capturing process in real time. It is one of the few methods that truly exploit the temporal coherence of the geometry. The authors also pointed out that the planes fitted during the compression could be utilized directly by downstream tasks (e.g., segmentation or object detection), essentially skipping the decompression process. Unfortunately, the method is tailored closely to LiDAR data and does not work with general dynamic point clouds since the frames of such sequences cannot generally be converted to range images.

5.14 Silhouette 4D: An Inter-Frame Lossless Geometry Coder if Dynamic Voxelized Point Clouds

Peixoto et al. [80] proposed an inter-frame extension of their lossless intra-frame compression method based on dyadic decomposition of the voxelized dynamic point clouds [79]. Although it utilizes structural coherence, it is not based on an octree coding scheme.

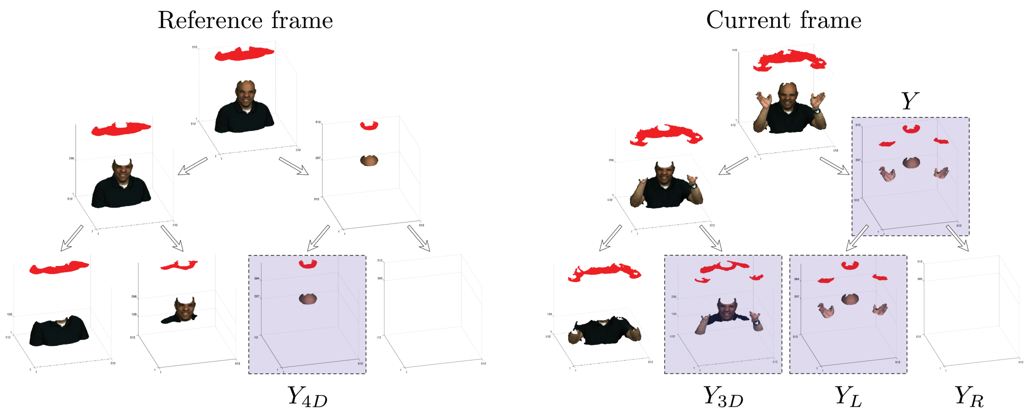


Figure 5.15: All the slices involved in encoding Y_L. Source: [80]

The point cloud is voxelized into a regular grid of size $N \times N \times N$. For each frame, a binary tree structure is constructed in which each node represents a slice of the volume along a selected principal axis, and its children split the slice in half. The tree has N leaf nodes, each representing a binary image of the size $N \times N$. During the compression, the tree is traversed in pre-order fashion, from left to right. The slice of the processed node is projected along the axis to create a so-called silhouette image. Silhouettes are encoded using a context-adaptive arithmetic coder suited for binary images. However, the number of encoded pixels can be reduced significantly when considering that a pixel in non-leaf node is occupied if and only if a corresponding pixel of at least one of the child nodes is occupied. If a pixel in a parent node is empty, all of the corresponding pixels in child nodes must be empty as well. Additionally, if a parent pixel is occupied, but the corresponding pixel in the left child is not, the same pixel must be occupied in the right child. In both cases, this information is apparent to both sides and thus it can be omitted from the coding stream. The bit rate is also reduced by building a context for the encoded pixel used in the entropy coder. It consists of three components: a 2D context that consists of the values of the three already encoded neighbor pixels in the same slice, a 3D context that consists of the corresponding pixel and its eight neighbor pixels in a slice above the axis if such a slice exists, and a 4D context that consists only of the value of the corresponding pixel in the previous frame (see Figure 5.15). In their experiments, the authors noted that in some cases, the previous intra method that did not utilize the 4D context and had more pixels in the 2D context still obtained better compression results. To address this, they proposed mode selection, which switches between the two prediction modes based on which mode provides better conditional probabilities.

The authors showed, that although for some datasets the method performed comparable to their previous work, it provides up to 15% gain for others. They also compared it with [34], which it outperformed on all datasets.

5.15 Summary of Dynamic Point Cloud Compression

Although dynamic point cloud representation became popular only recently, dynamic point cloud compression is already a well-studied area. This can be mainly credited to the attention this topic received thanks to the MPEG standardization process. Since an octree is a natural representation of voxelized point cloud data, it is not a surprise that the majority of the inter-prediction methods are octree-based. Most recent methods are versatile and can outperform intra-frame prediction methods with data rates already around 1 bit per occupied voxel. MPEG standardization is still ongoing and multiple research groups are still interested in improving the current state of the art. Additional large improvements are still expected in the near future.

Chapter 6

Potentially Related Methods in Different Research Areas

Some of the limitations of current time-varying mesh compression methods have already been addressed, albeit in different contexts. We will briefly describe some of this work in this chapter. We believe the field of TVM compression would benefit significantly from incorporating similar strategies.

6.1 Geometry-Based Connectivity Coding

Many static mesh and some dynamic mesh compression methods are connectivity driven – the geometry and connectivity are encoded simultaneously during a mesh connectivity traversal. This allows for efficient exploitation of spatial coherence in the data. In the case of the time-varying mesh, however, the connectivity-driven compression significantly limits how much of the temporal coherence is utilized. To the best of our knowledge, all of the time-varying mesh compression methods that consider connectivity encode such data separately from geometry. The issue with this approach is that both geometry and connectivity compression often lead to reordering of vertices, causing inconsistency in data. As was shown in [27], enforcing the order of vertices in geometry compression might hurt performance. A more feasible approach might be to encode the geometry first and then use this information to reconstruct the connectivity. Note that separate encoding allows one to treat the geometry as a dynamic point cloud.

Doumanoglou et al. [23] have already considered the geometry-based connectivity coding in terms of time-varying mesh compression (see Section 4.7). However, their approach is quite inefficient even though at the time which the method was proposed, several geometry-based connectivity compression methods had also already beem proposed.

The geometry-based approach was popularized by Lewiner et al., who proposed a compression method for general convex complexes in arbitrary dimensions [59]. An overview of the method is shown in Figure 6.1. It traverses the mesh from the initial cell, where the adjacent cells are reconstructed by encoding the tip vertices. This is done by first encoding the number of tip vertices (in the case of triangle meshes, the number is always equal to 1, and thus this step can be omitted). Then, values are assigned to all candidate vertices by a so-called geometric function. A range that contains all the correct tip vertices is encoded. All the vertices in the encoded range are then sorted by a geometric criterion. After that, an index in the sorted list for each tip vertex is encoded. For geometric functions and criteria, the authors propose, for example, the distance to the barycenter of the front cell or the circumradius.

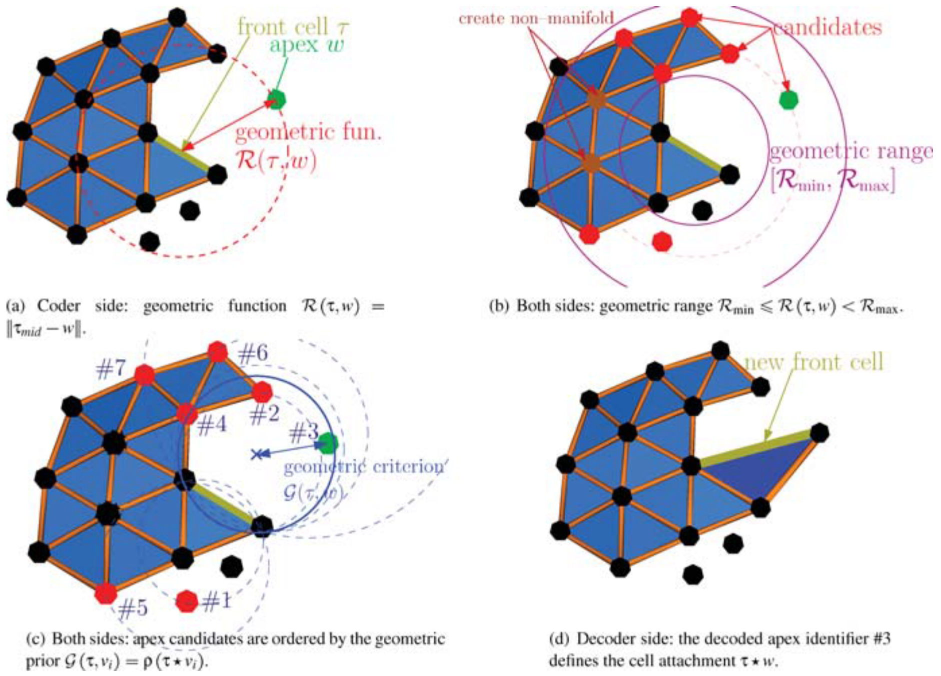

front cell τ
apex w
geometric fun.
$\mathcal{R}(\tau, w)$
(a) Coder side: geometric function $\mathcal{R}(\tau,w) = \|\tau_{mid} - w\|$.
create non-manifold
candidates
geometric range
$[\mathcal{R}_{\min}, \mathcal{R}_{\max}]$
(b) Both sides: geometric range $\mathcal{R}_{\min} \leqslant \mathcal{R}(\tau,w) < \mathcal{R}_{\max}$.
#7
#6
#4
#2
#3
geometric criterion
$\mathcal{G}(\tau, w)$
#5
#1
(c) Both sides: apex candidates are ordered by the geometric prior $\mathcal{G}(\tau, v_i) = \rho(\tau \star v_i)$.
new front cell
(d) Decoder side: the decoded apex identifier #3 defines the cell attachment $\tau \star w$.

Figure 6.1: Overview of the connectivity compression scheme proposed by Lewiner et al. Source: [59]

The method proposed by Marais et al. [72] traverses the mesh similarly to the Edgebreaker algorithm [82]. To encode the tip of an unprocessed triangle, the method predicts its position and encodes the rank of the correct vertex in the list of nearest neighbors of the prediction. The method discusses two prediction strategies: parallelogram [95] and midpoint prediction. To better suit the geometry, the prediction is further improved by considering curvature and scaling (see Figure 6.2). For the first of the two, the prediction is rotated around the gate edge so that it lies on the plane formed by the edge and the average position of the vertices in front of it. The scaling is adjusted so that the prediction lies as close as possible to the nearest vertex in a frustum in front of the gate edge. To choose between the two strategies, the method first tests both on all the triangles and selects the one with a higher number of successful attempts. An attempt is successful if the correct vertex has rank 1. If none of the strategies achieves this, the successful attempt is selected as the one with the lower rank.

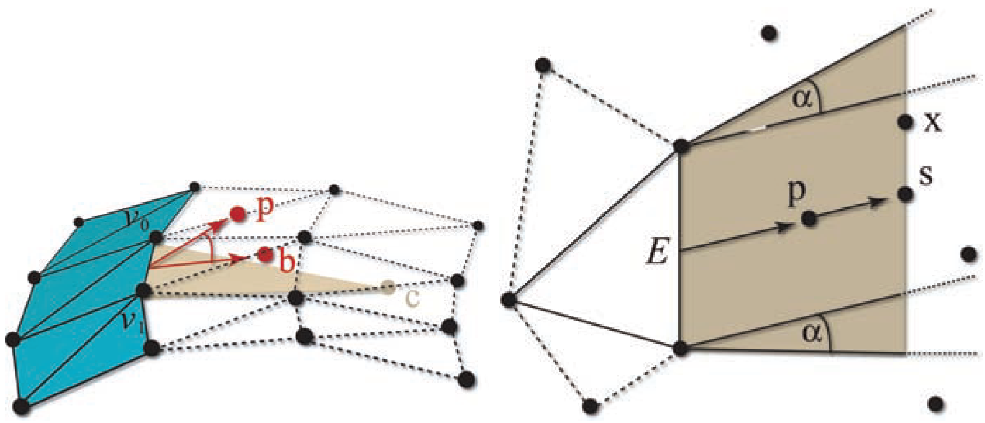


Figure 6.2: Prediction adjustments performed by Marais et al. [72]. *Left*: Rotation to accommodate for curvature. *Right*: Scaling to accommodate sampling irregularity. Source: [72].

The most recent geometry-based approach, presented in [16], is very well suited for highly regular data. It is based on a surface reconstruction algorithm called *convection* (see Figure 6.3). In convection, a Delaunay tetrahedronization of the points is constructed. A surface that is initialized as a convex hull of the points is incrementally adjusted. All its faces are inserted into the queue. Once a face has been removed from the queue, its Gabriel half-sphere is checked. If it contains any other vertex, the face is removed from the surface and replaced by the three other faces of its tetrahedron, which are also inserted into the queue. The process ends when the queue is empty. Since this reconstruction can be performed on both sides, only the times between events are encoded when the process attempts to remove a face that is present in the data or when it preserves a face that is not present. This process is applicable only to faces that are part of the Delaunay tetrahedronization of the points. The other faces are encoded separately in a less efficient manner.

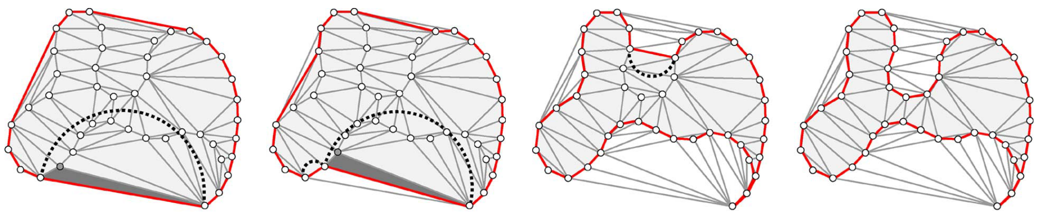


Figure 6.3: Example process of convection. Source: [16]

6.2 Versatile Temporal Coherence Estimation

As pointed out in Section 4.10, the main issue with the current model-based TVM compression methods is their lack of versatility. This is, however, not an issue of the approach itself, but of the applied models. In order to achieve better results for more general data, the model should make little to no assumptions other than that of the presence of the temporal coherence itself. In this section, we will briefly describe the most prominent approaches for revealing temporal coherence.

The most popular approach is tracking, which for any point on a surface at a time t_a attempts to estimate its position at a time t_b. Most tracking methods propagate a certain template shape (e.g., first frame) over time to match all the frames, incorporating a non-rigid registration with estimated correspondences. Without any additional work, this limits tracking only to surfaces with a constant topology equivalent to that of the template shape.

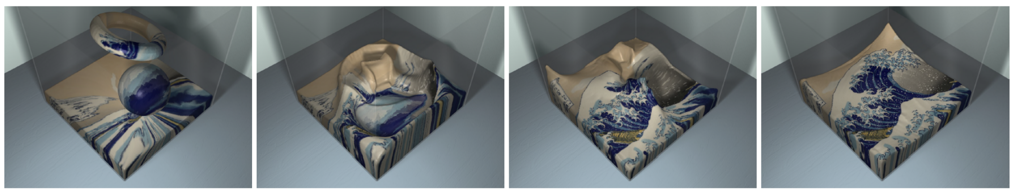


Figure 6.4: Backwards texture propagation. Source: [12]

This was, however, addressed by Bojsen-Hansen et al. [12]. Their method also uses the first frame as a template. It is, however, frequently refined to better match the data. For non-rigid registration, the method uses the bi-resolution approach proposed by Li et al. [60]. On the coarse level, the deformation is represented by an embedded graph, which is formed by sub-sampling vertices of the template and connected using k-nearest neighbors considering geodetic distance. Only the vertices of the graph are fitted to the target frame; for the rest, the deformation is computed using linear blend skinning of the nearest graph vertices. On the fine level, all the vertices are considered in fitting, however, the process is much simpler than on the coarse level. Instead of estimating the correspondences from features, the method simply projects the points onto the target surface in the normal direction, which is much more robust to topology change. Since the deformation might introduce irregularities into the template shape, it is improved by subdivision of long edges and large triangles and contraction of short edges and small triangles.

The second and most important refinement is in the sense of topology. To this end, the authors propose a *constrainTopology* operation, which takes the mesh to be refined and a signed distance function (SDF) of a certain surface as an input. The SDF is sampled on a regular grid. In grid cells where the mesh and isosurface of SDF have a different local topology, the mesh is resampled using a marching cubes reconstruction of the isosurface in the cell. This operation is applied twice to the template, first using the SDF of the template itself to remove topologically complex parts (e.g., self-intersections), and second using the SDF of the target frame. To preserve the consistency, the method stores a record for each update of the template. The record contains set of input and output vertices and mappings of values between them. The method by Bojsen-Hansen et al. [12] allows, for example, propagation of texture (see Figure 6.4), or application of displacement maps to liquid simulation results without requiring one to rerun the simulation. However, it cannot be used in compression since the tracking results require a larger storage capacity than the original data itself.

Closely similar to and sometimes interchangeable with tracking is 4D reconstruction, which focuses on reconstructing evolving shapes from incomplete observations of data, e.g., color or depth images, point clouds, or incomplete meshes. Although it is not the main objective, most of the methods aim to obtain a temporally coherent representation. Describing all the interesting work in this field is out of the scope of this thesis, and we refer the reader to the report by Zollhöfer et al. [112], for example, for more information.

Inspired by the optical flow in video, scene flow might be considered an orthogonal approach to tracking since it does not aim to find the deformed shape but to find a flow field representing its movement.

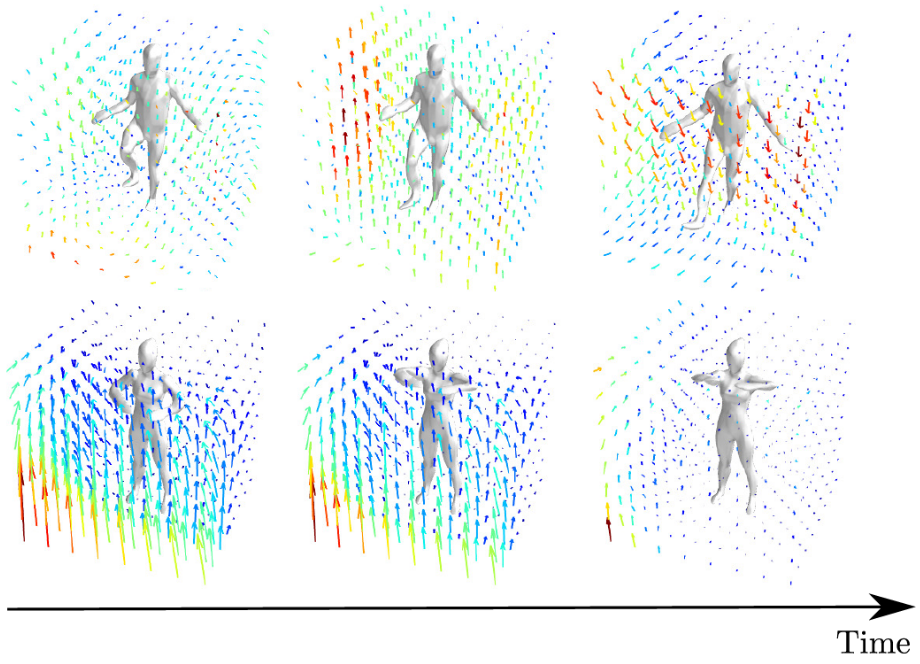


Figure 6.5: Two reconstruction examples from [77]. Source: [77]

Recently, Niemeyer et al. [77] proposed a method that combines 4D reconstruction and scene flow. Their method represents the data by an occupancy function and a vector field that deforms it (see Figure 6.5), both parameterized by a neural network. To train the networks, the authors use two loss functions. The first one, denoted $\mathcal{L}_{recon}$, treats the problem of obtaining the occupancy function at a certain time as a classification problem, in which the ground truth occupancy at a point $\mathbf{p}$ in time $t = \tau$ is matched with predicted occupancy at a position obtained by backward propagation through the vector field to a time $t = 0$. The second loss function, $\mathcal{L}_{corr}$, is used only for training the vector field, is optional, and can be used when prior correspondence information is known. It calculates the Euclidean distance between a point $\mathbf{s}$ in time $t = 0$ propagated to time $t = \tau$ and its corresponding point at that time. After the training process is finished, the isosurface mesh is extracted from the occupancy function at time $t = 0$. To obtain the positions in a specific frame, only the vertices of the mesh are propagated to the corresponding time. This is much faster than extracting the isosurface in all the frames and correspondences are explicitly encoded in the data. Unfortunately, this limits the obtained surface only to constant topology influenced mostly by the first frame. However, this does not affect the vector field, which, in theory, should be able to represent even the movement of the surfaces of time-varying topology.

Chapter 7

Our Contribution

We have published four papers relevant to the topic of this thesis. For the sake of simplicity, we will discuss only the crucial parts of each work. For more details, we refer the reader to the original papers.

7.1 Error Propagation Control in Laplacian Mesh Compression

Our first contribution relevant to the topic of this thesis was concerned with improving performance under the mechanistic distortion metrics of high-pass coding [86], a method first proposed for compressing static meshes but later adapted to dynamic meshes as well [99, 101].

High-pass coding, often called Laplacian mesh compression, is a compression strategy based on encoding differential coordinates obtained by applying a discrete Laplace operator to the input geometry. Assuming the geometry is represented by a matrix $\mathbf{X} \in \mathbb{R}^{|V| \times 3}$, the differential coordinate matrix $\mathbf{D} \in \mathbb{R}^{|V| \times 3}$ is obtained as follows:

$$\mathbf{D} = \mathbf{M}\mathbf{X},$$

where $\mathbf{M} \in \mathbb{R}^{|V| \times |V|}$ is a Laplacian matrix corresponding to the chosen Laplace operator discretization. Naïvely, one could try to quantize and encode the values in $\mathbf{D}$ and reconstruct the geometry by solving the Poisson equation:

$$\mathbf{M}\hat{\mathbf{X}} = \hat{\mathbf{D}},$$

where $\hat{\mathbf{D}}$ is the matrix of differential coordinates distorted by quantization. However, $\mathbf{M}$ is a singular matrix (differential coordinates are translation invariant) and the problem cannot be solved trivially. To this end, the original authors proposed to select one or more vertices per connected component of the mesh as *anchor vertices* [86]. For each anchor vertex, a row with a single nonzero unit element at a position corresponding to the index of such vertex is appended to $\mathbf{M}$, and a rectangular Laplacian matrix $\mathbf{M}_+$ is obtained.

Using the rectangular Laplacian matrix, an extended differential coordinate matrix $\mathbf{D}_+$ is obtained. The values that are added with respect to $\mathbf{D}$ are actually the positions of anchor vertices. With the linear system extended by anchor data, $\bar{\mathbf{X}}$ can be reconstructed using the least squares method:

$$\mathbf{M}_+^T \mathbf{M}_+ \bar{\mathbf{X}} = \mathbf{M}_+^T \bar{\mathbf{D}}_+.$$

Due to the properties of the normal matrix $\mathbf{M}_+^T \mathbf{M}_+$ (symmetric, sparse, PSD), the problem can be solved using Cholesky factorization.

Mechanistic approaches (e.g., parallelogram prediction [95]), which work directly on Euclidean coordinates of vertices, tend to produce high-frequency artifacts at low data rates that are easily visible, especially in flat or generally smooth parts of input surfaces. The high-pass coding approach, on the other hand, tends to produce low-frequency distortion, which is much less likely to be detected visually. However, it does not perform well under mechanistic measures since it is not able to control the upper bound of the absolute coordinate error – while the positions of anchor points are distorted only by the quantization, the distortion of positions of other vertices is influenced by the distortion of all the vertices lying between the given vertex and nearby anchor point. This issue was already addressed in [5, 64], however, while both approaches improve the mechanistic properties, neither of them brings the performance on par with parallelogram prediction.

Our approach is based on two proposals. The first is to encode the anchor data separately and use a reduced Laplacian matrix $\mathbf{M}'$ instead of $\mathbf{M}_+$. It is obtained by removing all the rows and columns corresponding to anchor vertices from the original matrix and updating $\mathbf{D}$ so that it contains known values. If at least one anchor point was selected per connected component, $\mathbf{M}'$ has full rank and the reconstruction can be done by solving a simple linear system. If the original Laplacian matrix $\mathbf{M}$ was symmetric and positive semi-definite, the Cholesky decomposition can be applied to the reduced matrix as well. This approach was already studied in the original work; however, the authors pointed out that it led to higher error accumulation and caused spikes around anchor points. Nevertheless, we show that when it is combined with the following process, these issues are mitigated.

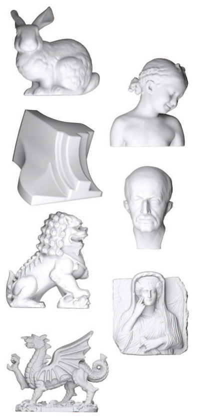
The key novelty of our approach is based on deeper exploration of the reconstruction process. The decoder first constructs the factor $\mathbf{L}$ such that $\mathbf{L}\mathbf{L}^T = \mathbf{M}'$. Then, it solves $\mathbf{L}\bar{\mathbf{Y}} = \bar{\mathbf{D}}$ by forward substitution. Finally, $\bar{\mathbf{X}}$ is obtained by solving $\mathbf{L}^T\bar{\mathbf{X}} = \bar{\mathbf{Y}}$. Our focus is on the forward substitution. An i-th row of $\bar{\mathbf{Y}}$ is computed as follows:

$$\hat{\mathbf{y}}_i = \frac{\hat{\mathbf{d}}_i - \sum_{j=1}^{i-1} L_{i,j}\hat{\mathbf{y}}_j}{L_{i,i}}.$$

The error accumulation is caused by the influence of the distortion introduced in the previous $i-1$ rows. However, the encoder knows the ground-truth value of $\mathbf{Y} = \mathbf{L}^T\mathbf{X}$. We thus propose adjusting the value of $\hat{\mathbf{d}}_i$ on the fly so that $\hat{\mathbf{y}}_i$ is as close as possible to its original value. While some error accumulation is also present in backward substitution, the total effect is substantially reduced by having an error-limited $\bar{\mathbf{Y}}$.

Table 7.1: Static mesh compression results in comparison with high-pass coding (HPC) [86], error diffusion [5], and weighted parallelogram (WP) [98].

		DAME				MSE			
	rate[bpv]	proposed	HPC	Diffusion	WP	proposed	HPC	Diffusion	WP
bunny	10	4.046E-08	3.507E-08	4.368E-08	8.894E-08	9.877E-09	6.819E-07	3.758E-08	8.874E-09
35946	15	1.166E-08	1.063E-08	1.291E-08	2.596E-08	7.318E-10	4.203E-08	3.541E-09	7.490E-10
bimba	10	9.975E-06	8.628E-06	1.060E-05	3.118E-05	1.094E-05	5.962E-04	5.643E-05	7.449E-06
8857	15	6.056E-07	5.965E-07	6.807E-07	8.626E-06	9.278E-07	4.329E-05	5.063E-06	5.919E-07
fandisk	10	1.520E-04	1.141E-04	1.573E-04	5.909E-04	3.748E-05	1.278E-02	1.639E-04	4.143E-05
6475	15	3.756E-05	3.018E-05	4.066E-05	1.221E-04	1.893E-06	8.643E-04	1.058E-05	2.975E-06
maxplanck	10	1.645E-05	1.605E-05	1.877E-05	3.347E-05	4.820E-06	3.367E-04	3.137E-05	4.322E-06
25445	15	5.411E-06	5.348E-06	6.073E-06	9.277E-06	4.795E-07	3.062E-05	2.408E-06	3.396E-07
chindragon	10	9.577E-04	8.346E-04	1.038E-03	2.315E-03	4.843E-04	2.319E-02	2.212E-03	5.310E-04
585018	15	3.622E-04	3.251E-04	3.866E-04	7.176E-04	5.463E-05	2.384E-03	2.522E-04	5.072E-05
palmyra	10	1.476E-05	1.450E-05	1.635E-05	2.757E-05	8.002E-06	3.893E-04	4.888E-05	7.350E-06
492465	15	4.749E-06	4.541E-06	5.031E-06	7.240E-06	7.950E-07	3.563E-05	4.421E-06	5.322E-07
welshdragon	10	1.086E-02	1.180E-02	1.187E-02	2.083E-02	6.320E-03	3.391E-01	3.332E-02	5.401E-03
291892	15	3.446E-03	3.529E-03	3.599E-03	5.188E-03	5.810E-04	3.039E-02	3.065E-03	3.568E-04



Not only does our method achieve better control over the propagation of error, which yields results similar to state-of-the-art methods considering both mechanistic and perception criteria (see Tables 7.1 and 7.2), but it also leads to a performance increase in decoding. The reduced Laplacian matrix has lower fill-in than the originally used normal matrix, and obtaining its factorization can be done much faster. Additionally, the performance increase is a result of the fact that the normal matrix is not computed at all. $\mathbf{M}'$ is also better conditioned. Although it may seem that our method requires the decoder to use Cholesky factorization for geometry reconstruction, it is actually independent of the technique used for solving a linear system.

Table 7.2: Dynamic mesh compression results in comparison with respect to different Laplace operator discretizations. Comparison was done with high-pass coding [86] and error diffusion [5].

	Kirchhoff	Diffusion Kirchhoff	Proposed Kirchhoff	Cotan	Diffusion Cotan	Proposed Cotan
dataset	STED error					
samba	0.0236	0.0247	0.0215	0.0145	0.0141	0.0125
march	0.0221	0.0223	0.0200	0.0148	0.0147	0.0138
handstand	0.0376	0.0405	0.0345	0.0358	0.0502	0.0246
jump	0.0220	0.0213	0.0209	0.0199	0.0193	0.0197
	KG error					
samba	1.0474	0.5165	0.2643	0.8621	0.4093	0.1968
march	0.8566	0.4777	0.2943	0.5882	0.3659	0.2665
handstand	1.5786	0.8550	0.4320	1.7947	1.3516	0.3721
jump	0.6632	0.3213	0.2901	0.4956	0.2991	0.2879

On the other hand, the encoding process is slower since the encoder must additionally compute the factorization. The proposed modification is applicable only to Laplace-Beltrami discretizations that yield a symmetric Laplacian matrix. The most significant limitation is, however, its performance on highly regular meshes, where the differential coordinates are usually of small magnitude. On such surfaces, the method tends to overcompensate in a zigzag pattern, raising the entropy to higher values than those of the original method.

7.2 Predictive Compression of Molecular Dynamics Trajectories

We have also developed a compression method for dynamic geometry of molecular structures that exploits the characteristic movement of atoms. While such data is not directly related to TVMs, some of its properties are actually similar to dynamic meshes.

Molecular dynamics trajectories capture the dynamic behavior of molecular structures. They are represented by atoms with positions evolving over time and bonds between them that are usually assumed not to change during the captured phenomenon. When considering compression, bonds are usually omitted from the compressed data since they represent only a fraction of the overall size. However, they must be taken into consideration as an input since they constrain movement significantly.

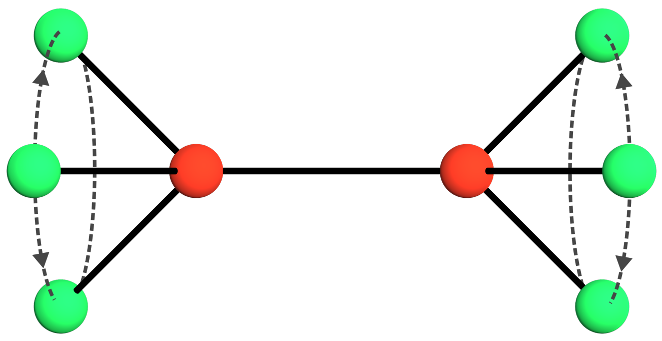


Figure 7.1: Molecular movement is mainly constrained to so-called *dihedral angles*.

For molecular data, mechanistic approaches are more desirable (yet there are few methods based, for example, on PCA, e.g., the PCZ format) since the compressed data is expected to be further analyzed after decompression. Most of the compression formats in use are being developed as a part of various molecular dynamics simulation tools (e.g., Gromacs XTC [1] and TNG [88]). HRTC [46], the most recent compression format, promises data rates below 1 bit per coordinate (bpc), however, we were unable to reproduce such results in our experiments. The current limitation of the previous work, in our opinion, is that the atom bond information is not exploited to its full potential.

Our key observations are that the molecular movement is quite constrained (e.g., distance between two atoms forming a bond does not change much throughout the sequence) and that the greatest variance in atom positions over time occurs in so-called dihedral angle movement (see Figure 7.1).

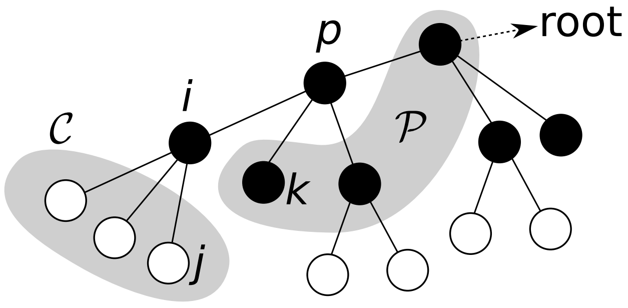


Figure 7.2: The situation during one step of a DFS traversal in predictive molecule compression. Already encoded atoms are black, atoms to be encoded are white, and the current atom is i.

We first construct a canonical frame that captures the general features common in all the input frames in an iterative process, where the neighborhood of an atom is aligned with the canonical representation and the representation is refined to better match the original positions. It is then encoded using a simple prediction method based on a depth-first search (DFS) traversal, in which the position of an atom is predicted by the position of its predecessor in the traversal. The molecular dynamics trajectories are also encoded during a DFS traversal; however, the canonical frame is used for prediction. Edges representing a bond between a currently processed atom a_i and its predecessor in the traversal a_p in current and canonical frames are transformed so that a_i lies at the origin of the coordinate system and the edge is aligned with the z-axis. Next, the local neighborhood (see Figure 7.2) in the current frame is rotated around the z-axis so that the preceding neighborhood $\mathcal{P}$ is aligned with the canonical molecule. After that, we find an angle α parameterizing a rotation matrix $\mathbf{R}_\alpha$ around the z-axis, which achieves the best alignment between the current and canonical frames for the currently coded neighborhood $\mathcal{C}$. The angle is quantized and encoded. Due to the dihedral angle movement, a temporal coherence in encoded angles is expected and we can further reduce the entropy of the data by differential coding. The predicted positions of coded neighbors in $\mathcal{C}$ are then achieved by rotating their canonical positions using $\mathbf{R}_{\hat{\alpha}}$, where $\hat{\alpha}$ is the rotation angle distorted by quantization. The corrections are computed simply as differences between the predictions and the original values. An entropic coder is used to compress the quantized angles and corrections.

Table 7.3: Comparison of the proposed PMC against other formats in terms of relative size of compressed data (lower values are better). H5 denotes the HDF5 format with MAFISC [45] compression filter for general multi-dimensional data. The maximal allowed error limit was ≈ 0.00866 Å. HRTC and PCZ may exceed this limit.

	Relative size of compressed data [bpc]						Max. error [Å]	
	H5	XTC	TNG	HRTC	PCZ	PMC	HRTC	PCZ
p53	13.4	10.1	8.0	10.2	9.4	5.2	0.00875	2.54
p53-0.05	11.9	10.2	6.4	8.2	9.7	5.1	0.00880	1.55
ARID	14.1	10.2	9.0	16.5	5.6	5.2	0.00050	3.72
DhaA31	14.7	10.1	8.1	12.9	6.6	5.2	0.00906	3.56
ethanol	15.5	9.4	9.4	14.6	7.5	7.0	0.00883	25.79
water	15.4	8.8	8.8	13.8	14.7	7.5	0.00876	11.91

Our results are substantially better than the results obtained with other state-of-the-art methods and allow either saving 1.3–3.8 bits per coordinate at the same precision or providing up to 10× better precision at the same data rate. (see Table 7.3). The biggest increase in performance can be observed for large molecular structures (e.g., proteins). The method is relatively simple, with the most complex operation being the rotational alignment of an atom neighborhood. However, we show that this problem can actually be solved in closed form using a simple formula and does not require computing an SVD of any matrix, as is usual in solving an alignment problem. While the atoms must be processed in a sequential order determined from the DFS, the method can be parallelized in terms of frame positions of single atoms as long as the differential coding of angles is performed when all the frames are processed. The method does not support streaming, since the data is processed in two passes (canonical molecule construction and encoding).

7.3 Towards Understanding Time-Varying Triangle Meshes

To address the lack of versatility of models used in time-varying mesh compression, we have presented a tracking pipeline that outputs a representation of the underlying movement with interesting properties that the TVM compression could benefit from.

Our idea follows from the fact that in many practical scenarios where a surface is captured from multiple viewpoints, the overall enclosed volume changes negligibly – it does not suddenly appear or disappear. This, however, does not hold for the surface itself, where a part might disappear due to self-contact (see Figure 7.3). Instead of points on a surface, we propose to track a fixed number of volumetric centers: points representing a small volume surrounding them. Our objective is to find the positions of such centers that evolve over time so that they cover all parts of each frame, are evenly distributed over the volume, and move consistently with their surrounding centers.

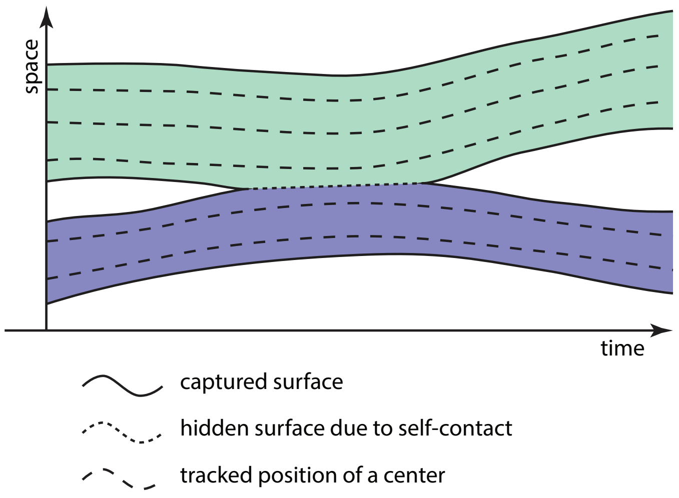

space
time
captured surface
hidden surface due to self-contact
tracked position of a center

Figure 7.3: Schematic of volume tracking in 1D space: during the sequence, two objects (green and blue) touch and then separate.

Our method is applied to each frame as follows. First, we sample an indicator (occupancy) function in a regular grid over the frame's bounding box. Then we find an initial distribution of centers. This is done by initializing the centers in randomly selected occupied voxels and performing Lloyd's algorithm [63], which repeatedly computes a Voronoi partitioning of the enclosed volume with centers as generators and moves the centers into centroids of corresponding cells. Since it is quite difficult to evaluate the centroid position analytically, we approximate it as an average position of centers of occupied voxels inside the corresponding Voronoi cell. For the first frame, the method already terminates. For any subsequent frame, we next apply the Kuhn-Munkers algorithm [55] to estimate correspondences between the centers of the previous and current frames, using squared distances as a cost function. However, we penalize any correspondence outside the volume (see Figure 7.4). Finally, we apply an optimization process ensuring the consistence of the represented movement. Our energy E consists of uniformity and smoothness terms: $E = E_u + \beta E_s$. The uniformity term

$$E_u = \frac{1}{2} \sum_{\mathbf{c}_i \in C} \|\mathbf{c}_i - \mathbf{m}_i\|^2,$$

where $\mathbf{c}_i$ is a position of i-th center and $\mathbf{m}_i$ is a position of the centroid of its corresponding Voronoi cell, ensures uniform distribution of centers inside the volume. The movement between frames can be represented by a vector field $\mathbf{v}$ sampled at the centers in the previous frame. We measure the smoothness of the movement by a squared length of Laplacian $\Delta\mathbf{v}$ using the Laplace operator discretization proposed by Belkin [8]:

$$E_s = \sum_{\hat{\mathbf{c}}_i \in C} \|\Delta\mathbf{v}(\hat{\mathbf{c}}_i)\|^2 = \frac{1}{|C|} \sum_{\hat{\mathbf{c}}_i \in C} \| \sum_{\hat{\mathbf{c}}_j \in C} H^t(\hat{\mathbf{c}}_i, \hat{\mathbf{c}}_j)(\mathbf{v}_j - \mathbf{v}_i)\|^2,$$

where $H^t(\mathbf{x}, \mathbf{y}) = \frac{1}{(4\pi t)^{\frac{5}{2}}} e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4t}}$ is a Gaussian kernel with parameter t, and $\mathbf{v}(\hat{\mathbf{c}}_i) = \mathbf{c}_i - \hat{\mathbf{c}}_i = \mathbf{v}_i$ is a displacement vector between the position of the center in the previous frame $\hat{\mathbf{c}}_i$ and its current position $\mathbf{c}_i$. We solve the optimization using a gradient descent.

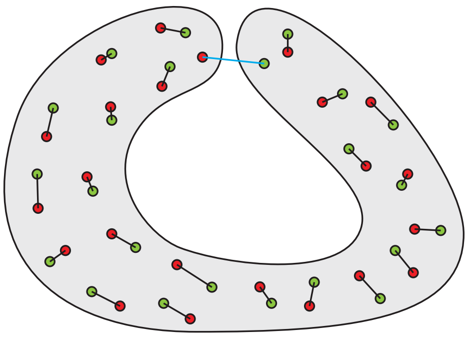


Figure 7.4: Correspondences outside the volume are inherently wrong and lead to a transfer of centers between separate parts of the object.

The resulting centers capture, to some extent, the underlying motion (see Figure 7.5). We show that a simple implicit function can be derived from the tracking results that has an isosurface approximating the original data. Even in an uncompressed form, the tracked centers have much lower storage requirements than the original data. Additionally, since the center ordering is consistent in time, we can further reduce the data size using PCA. We show that for a selected human performance dataset, 90% of the information is already contained in the first 50 out of 1638 principal components.

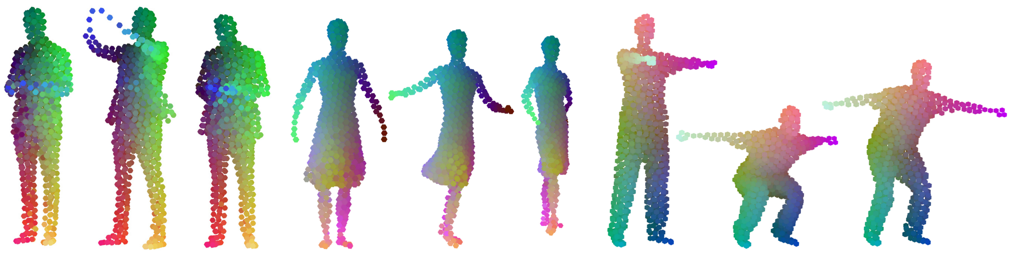


Figure 7.5: Results of the original volume tracking method on selected human performance datasets.

The tracking pipeline, as proposed in this original paper, still has some limitations. Mainly the smoothness term does not capture a global rigid movement (see Figure 7.6b) and the weighting based on proximity of centers actually leads to a transfer of centers between separate parts of the tracked object. The transfer is, however, caused by multiple other factors. For example, while the correspondence estimation penalizes the correspondence between different parts, it does not prevent such assignment. Besides that, the selected optimizing strategy requires a large number of iterations and does not guarantee a convergence.

7.4 As-Rigid-As-Possible Volume Tracking For Time-Varying Surfaces

We have presented improvements to the volume-tracking pipeline that address some of the issues that arose in the original method described in Section 7.3.

One of the most problematic parts of the original pipeline was the correspondence estimation. Combined with the initial distribution of centers, its goal was to provide an initial configuration for the optimization process. In our experiments, however, it turned out that much better results were obtained when the initial configuration was created by extrapolating the centers from the previous frame. Not only does this adjustment make the tracking more stable, but it also does not require the application of Lloyd's algorithm to each frame – only to the first one.

The biggest improvements were achieved by changes in the optimization step. We incorporated a different smoothing term E_s, inspired by the as-rigid-as-possible (ARAP) approach already used in various similar optimization scenarios [6]. The ARAP energies ensure that points move rigidly or nearly rigidly with their neighborhood. We evaluate E_s as a squared distance between the current center position and a prediction achieved by an estimated rigid movement:

$$E_s = \frac{1}{2} \sum_{\mathbf{c}_i \in C} \|\mathbf{c}_i - \mathbf{p}_i\|^2,$$

where $\mathbf{p}_i = \mathcal{A}_i(\hat{\mathbf{c}}_i) = \mathbf{R}_i\hat{\mathbf{c}}_i + \mathbf{t}_i$, $\mathbf{R}_i$ is a rotation matrix and $\mathbf{t}_i$ is a translation vector, such that

$$(\mathbf{R}_i, \mathbf{t}_i) = \operatorname*{arg\,min}_{\mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3} \sum_{w_{ij} \geq \mu} w_{ij} \left\|\mathbf{c}_j - (\mathbf{R}\hat{\mathbf{c}}_j + \mathbf{t})\right\|^2,$$

where w_{ij} is an affinity weight of an ordered pair of centers (c_i, c_j) and μ is a weight threshold. The alignment problem yielding the rigid transformation has a closed form solution obtained using a weighted Kabsch algorithm [87].

Given that both energy terms can now be interpreted as a squared distance between the actual center position and some prediction (Voronoi cell centroid $\mathbf{m}_i$, rigid movement prediction $\mathbf{p}_i$), we can utilize a different optimization strategy. If we ignore that both predictions depend on the optimized center positions and consider them fixed, the optimal center position can be simply computed as a weighted average:

$$\mathbf{c}_i = \frac{\mathbf{m}_i + \beta \mathbf{p}_i}{1 + \beta}.$$

Similarly to update step in Lloyd's algorithm, we can alternate between fixing the centers and updating predictions, and fixing the predictions and updating the centers. Our experiments show that such an optimization strategy requires many fewer iterations to obtain satisfying results, however, we still have no guarantee of convergence.

Basing the weight w_{ij} in the smoothness term only on the proximity turned out to be problematic in the original method. To this end, we propose combining this information with similarity of motion:

$$w_{ij} = a^p_{ij} \tilde{a}^f_{ij},$$

where $a^p_{ij} = \exp(-\sigma_p \cdot \|\hat{\mathbf{c}}_i - \hat{\mathbf{c}}_j\|^2)$ is the proximity term and $\tilde{a}^f_{ij}$ is the filtered motion affinity at current frame f, which is computed as follows:

$$\begin{aligned} \tilde{a}^0_{ij} &= a^p_{ij}, \\ \tilde{a}^{f \neq 0}_{ij} &= \alpha a^m_{ij} + (1-\alpha)\tilde{a}^{f-1}_{ij}, \end{aligned}$$

where the parameter $0 < \alpha < 1$ controls the response falloff. We employ such an infinite impulse response (IIR) filter to accumulate the motion information from previous frames. The unfiltered motion affinity a^m_{ij} for a given frame is evaluated using the dissimilarity of rigid transformations:

$$\begin{aligned} a^m_{ij} &= \exp(-\sigma_m \cdot d_i(\hat{\mathcal{A}}_i, \hat{\mathcal{A}}_j)^2), \\ d_i(\mathcal{A}, \mathcal{B})^2 &= \frac{1}{|V_i|} \sum_{\mathbf{v}_k \in V_i} \|\mathcal{A}(\mathbf{v}_k) - \mathcal{B}(\mathbf{v}_k)\|^2, \end{aligned}$$

where $\hat{\mathcal{A}}_i$ and $\hat{\mathcal{A}}_j$ are the final estimated rigid transformation from the optimization process in the last frame and V_i is a set of all the positions of voxels in the Voronoi cell generated by the center $\hat{\mathbf{c}}_i$ [43].

To quantify the quality of tracking results, we have proposed two metrics: PCA compactness (PCAC) and deviation from uniformity (DFU). The PCAC measures the complexity of the tracked center trajectories:

$$PCAC = \sum_{i=0}^{3F-1} i \sum_{j=0}^{N-1} |c^j_i|,$$

where c_i^j is a PCA projection coefficient of the j-th center trajectory onto the i-th principal direction, N is the number of centers, and F is the number of frames. The complexity of the tracked trajectories is given by the complexity of the input movement itself, together with the redundant complexity caused by tracking errors. The DFU measures the average relative standard deviation of Voronoi cell sizes:

$$\bar{v}^f = \frac{1}{N}\sum_{i=0}^{N} v_i^f,$$

$$DFU_f = \sqrt{\frac{1}{N}\sum_{i=0}^{n-1}(v_i^f - \bar{v}_f)^2},$$

$$DFU = \frac{1}{F}\sum_{f=0}^{F-1}\frac{DFU_f}{\bar{v}_f},$$

where v_i^f is the number of voxels in the Voronoi cell generated by a center $\mathbf{c}_i$ in the frame f. To show the superiority of one method over another, the better one must provide better results in one criterion while also achieving better or comparable results in the other.

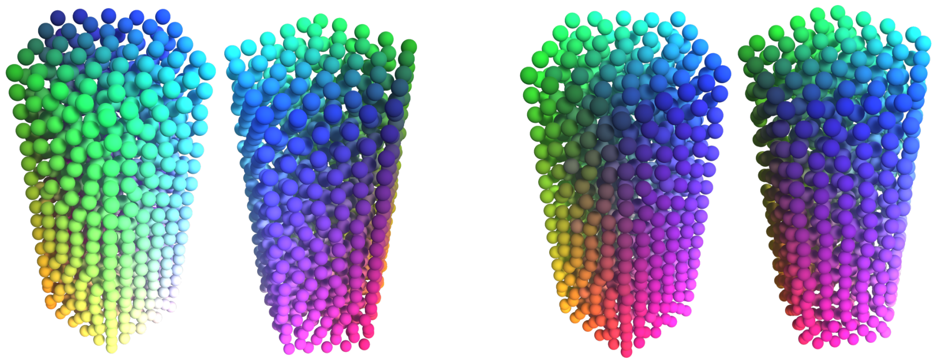

(a) Improved (b) Original

Figure 7.6: Tracking results of a rotating pentagonal prism. The original method fails to recognize such motion.

When compared with the original method, not only do we achieve better results in the proposed metrics, but the tracking results are also much more visually pleasing. The transfer of centers between different parts is significantly reduced, but unfortunately still not completely eliminated. Our method is also able to detect large global rigid motions (see Figure 7.6). ARAP-based tracking also shows better overall performance when applied to noisy datasets, although it may spawn a local motion in cases where the global movement is ambiguous.

Chapter 8

Future Work

As we already stated, we have decided to focus our future work only on the compression of time-varying meshes since dynamic mesh compression is already a well-studied area. We plan to work on a compression method with a similar overall structure to the method proposed by Doumanoglou et al. [23] in the sense that it will encode the geometry first, which will then be used to drive the connectivity coding. Geometry encoding of a single frame will consist of 1) propagating the geometry of the previous frame using a certain *temporal model* to obtain a reference shape and 2) applying a certain data reduction technique given the redundant data with respect to such a reference shape. The structure is outlined in Figure 8.1. At the end of this chapter, we will also discuss some goals that should be addressed in a more distant future.

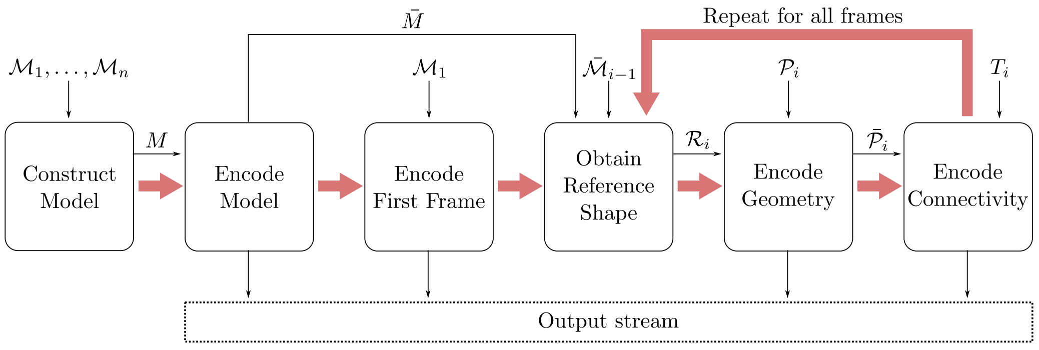


Figure 8.1: Outline of the structure of a TVM compression with a versatile temporal model M. Thin arrows represent the data flow, while bold arrows represent the order of operations. $\mathcal{M}_i$ denotes a mesh at the i-th frame, $\mathcal{R}_i$ denotes its reference shape, $\mathcal{P}_i$ is a point cloud consisting of its vertices, and T_i is its connectivity. The bar above a symbol denotes data distorted by compression.

8.1 Time-Varying Mesh Compression with a Versatile Temporal Model

The most crucial part of designing such a compression method will be selecting an appropriate temporal model. Although, for example, [48] showed it is possible to use the previous frame itself as a reference shape for inter-coding of the current frame geometry, the compression performance can be improved upon if a temporal model is used to adjust the reference shape with respect to the underlying motion because such reference, in theory, should be closer to the coded shape. This is, however, based on the assumption that the model sufficiently captures the dynamic behavior of the sequence. To achieve higher versatility of the method, the model must also handle topological changes. Not only must it be able to represent surfaces with varying topology, but it should also be able to adjust the genus of the reference shape in cases, when the topology change occurs between consecutive frames. While this adjustment is not required for the method to be applicable to such data, the compression performance might be negatively affected if this is not addressed.

When selecting the model, it is also important to consider its data footprint since it will be encoded alongside the data as well. Note that the amount of information present in the model is indirectly connected to the efficiency of the inter-coding. For example, given a model that basically outputs the correct triangle mesh as a reference shape, little to no information is required to be transmitted during the inter-coding, but the model itself may take the majority of the resulting data size. While this can be, to some extent, addressed by lossless compression of the model, it is better to allow for a loss of information, preferably with intuitive control over the size of the model and the quality of the prediction.

With the improvements proposed in our latest work, we believe that the volume tracking results adequately represent the temporal information present in data. One additional advantage of the tracked centers approach is that it was designed with the data footprint in mind. Since the center ordering is consistent throughout the sequence, the size of the model can be compressed through the PCA. Additionally, we can directly control the quality of the model through the encoding process by choosing the number of principal directions to preserve. To allow the applicability of tracked centers as a temporal model in time-varying mesh compression, we must still derive a process that relates the motion of the centers to the motion of coded vertices to be able to deform the previous frame to obtain the reference shape. This can be done, for example, by using an *embedded deformation* approach [90] in which the corresponding position of a point $\mathbf{p}$ is obtained by linear blending of rigid transformations assigned to tracked centers:

$$\bar{\mathbf{p}} = \sum_{i=1}^{n} w_i(\mathbf{p}) \left[\mathbf{R}_i(\mathbf{p} - \mathbf{c}_i) + \mathbf{c}_i + \mathbf{t}_i \right],$$

where $w_i(\mathbf{p})$ are weight functions usually based on the Euclidean distance in $\mathbb{R}^3$ between the point $\mathbf{p}$ and center $\mathbf{c}_i$. The main issue of embedded deformation, and of similar approaches as well, is that they ignore the interior-exterior information. This means that different parts of the shape influence each other even when not directly connected, and that there is no mechanism for handling self-intersections. While it is desirable for some parts of the surface to come into contact, self-intersections are not desired. We are currently working on incorporating the geodesic distance in the volume into the weights, which might solve the problem of influence by separate parts, but the latter issue remains unsolved. Addressing these issues is, in our opinion, the most challenging part of the future work, on which we should focus with the highest priority.

Another temporal model that can be considered is the vector field neural network of Niemeyer et al. [77]. However, similarly to the approach presented above, the vector field does not consider interior-exterior information. Given that the vector field is continuous and is defined outside of the surface volume, it has difficulties with representing closely located motions of opposite direction. This indicates that no self-intersections occur. However, not only it is impossible for two parts of the surface to come into contact, but when attempting to do so, these parts are also undesirably deformed (see Figure 8.2). In our opinion, it is much more desirable to allow self-intersections than not to allow self-contact. While the neural network is itself a reduced representation of a dense time-dependent vector field, it is much more complicated to reduce the size further. Nevertheless, we will also be closely monitoring other newly proposed methods in 4D reconstruction and related problems since this area currently receives a lot of attention.

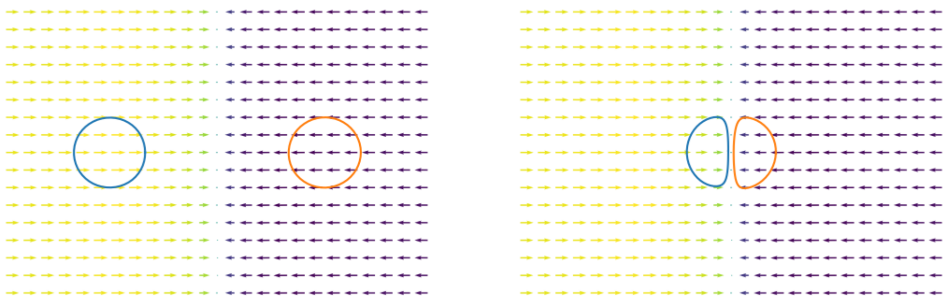


Figure 8.2: Collision of objects propagated by continuous vector is impossible. Instead, the objects are undesirably deformed.

Another challenge in obtaining the reference shape, independently of which approach is selected as a base for the temporal model, will be addressing the change of topology between consecutive frames. If we apply the deformation directly to the vertices of the previous frame, the reference shape will have a different topology from the coded frame. We believe this could be alleviated by deforming an implicit representation of the shape instead, followed by surface extraction. However, while this approach solves the merging of parts, other adjustments are required to address their separation.

Another important part of the compression pipeline is the geometry coding given the reference shape obtained by the temporal model. The simplest approach is the prediction present in the work by Doumanoglou et al. [23]: each encoded vertex is predicted by its nearest neighbor in the reference shape. We will also consider dynamic point cloud approaches based on the coherence of spatial data structures. By incorporating the reference shape in an XOR prediction of octrees [48], a method equivalent to the one in [93] is obtained with an alternative mechanism for motion compensation. It is also possible to consider the Silhouette 4D approach based on context modeling. However, since all the listed point cloud approaches work on voxelized point clouds, there is a possibility of multiple vertices falling inside a single voxel. If this is not signaled to the decoder, such vertices are to be replaced by a single vertex when reconstructing the point cloud from a voxelized representation, which destroys the isomorphism between the original and reconstructed data. Signaling such information, however, introduces additional data to be encoded, and thus the objective is to perform the voxelization process with caution to reduce the occurrence of this phenomenon.

One possible challenge we might need to address is the dependence of all the listed geometry coding approaches on the relative sampling density between the reference and the coded shape. Compared to cases, when the sampling density is similar, in a case when the sampling density of the reference is much higher, the prediction by Doumanoglou et al. [23] will have to encode much more symbols representing that a reference vertex is not used to predict any coded vertex, and conversely, in a case when it is much lower, the length of correction vectors will increase. In the case of point cloud approaches, the sampling density directly influences the structure of voxelized data. This might not be an issue when the temporal model propagates the mesh of the previous frame since we expect that all the frames were obtained in an identical process. However, if we choose to propagate a different (e.g., implicit) representation of a surface, such an assumption cannot be made. While this problem can be solved by simplifying or remeshing the reference shape, we plan to explore the possibilities of prediction more oblivious to this issue. Mourycová – Váša [75] have already conducted some experiments in this area, but their method is only efficient for very low data rates and is currently limited in terms of the required quality of the reference shape.

Since all the listed geometry coding approaches reorder the vertices, it will be important to preserve the order during the connectivity coding. As discussed in Section 6.1, this can be done by driving the connectivity coding using the already reconstructed geometry. Of the existing methods, the most suitable for our scenario is the one proposed by Marais et al. [72] since it performs best on the general manifold meshes we expect as input. Nevertheless, we believe, that their approach can still be significantly improved. For example, it traverses the connectivity similarly to Edgebreaker [82], which strongly limits the order of processed triangles and the edges through which they will be discovered. This in turn negatively influences the resulting data rate, since a triangle can be discovered through an edge, where the prediction obtains a larger number of candidate vertices. We are currently experimenting with a different approach that determines the order from the certainty of the prediction.

When evaluating the compression performance, one might consider comparing the results with those of the method by Doumanoglou et al. [23]. However, since our goal is to be able to efficiently compress more general data rather than only human TVMs, this comparison is actually irrelevant. More relevant for such a comparison should be the approach based on the MPEG V-PCC codec [27], which, however, is outperformed by a static mesh approach. For this reason, it is thus most preferable to compare with static mesh compression algorithms, e.g., a weighted parallelogram [98]. The comparison should be done on various types of datasets, including, for example, human performances or liquid simulations. For now, our goal is to achieve better compression rates with comparable distortion but without considering the computation times, since even obtaining the temporal model of the data is already a slow process.

8.2 Distant Goals

In a more distant future, we plan to study how to achieve real-time or near-real-time performance without sacrificing the versatility to allow the application of our method in tele-immersion. Almost certainly, some sacrifices will have to be made, for example, by discarding a fraction of the frames and replacing them with the deformed previous frame. This would lead to a loss of isomorphism between the original and decoded vertices, which would also make it more difficult to measure the distortion. The temporal model should also be heavily altered so that it can be obtained as fast as possible. Additionally, in real-time scenarios, we do not have access to all the frames of the sequence at any time. Thus, instead of constructing a complete model at the beginning, it must be updated before encoding each frame, and this update must be encoded as well. Unfortunately, for now, the tracked centers temporal model is unusable in such a scenario due to the computational complexity, and even if it were usable, we would have to consider different ways of encoding it since the PCA coding expects the center positions in all the frames to already be available.

To date, the performance of all the TVM compression methods has been evaluated using only the mechanistic error measures. To the best of our knowledge, there is currently no perceptual metric designed specifically to work with the geometry of time-varying meshes, and the only option for now is to render the frames from multiple viewpoints and use perceptual metrics for videos. For this reason, we believe it is important to turn our future attention to this research area as well.

Chapter 9

Conclusions

We have studied the problem of compression of triangle mesh and point cloud sequences. Investigation of the current state of the art has shown high potential for future improvements in the field of time-varying mesh compression, even simply by experimenting with modern and more effective alternatives to some of the key parts of current methods. In contrast, there is currently little to no potential for improvement in the field of compression of dynamic meshes. It is also very unlikely we will make any breakthrough in dynamic point cloud compression; however, this field is highly relevant to TVM compression as it can be incorporated into TVM geometry coding.

Although we have not yet presented any novel TVM compression method, we have already focused on designing a mathematical model that captures the temporal coherence of data. The proposed model is not only designed to be used in geometry prediction, but may also be beneficial in different general mesh sequence processing tasks (e.g., in temporally coherent texture mapping).

The main goal of the doctoral thesis is to compile the knowledge we have collected so far in proposing a novel TVM compression method. It combines geometry coding based on a carefully selected temporal model and connectivity coding which exploits the already compressed geometry. We have also set more distant goals that might not be addressed in this Ph.D. study. Those are: 1) adapting the proposed method for real-time performance to allow its usage in tele-immersion, and 2) studying the possibility of measuring the distortion of general mesh sequences correlating with human perception.

Bibliography

[1] GROMACS File Formats, 2018. `http://manual.gromacs.org/documentation/2018/user-guide/file-formats.html`.

[2] G-PCC Codec Description v9. *ISO/IEC JTC1/SC29/WG7 N0011*. October 2020.

[3] Call for proposals for point cloud compresssion V2. *ISO/IEC JTC1/SC29/WG11 N16763*. April 2017.

[4] V-PCC Codec Description v8. *ISO/IEC JTC1/SC29/WG11 N18892*. November 2019.

[5] Alexa, M. – Kyprianidis, J. E. Error diffusion on meshes. *Computers & Graphics*. 2015, 46, pages 336–344. ISSN 0097-8493. doi: 10.1016/j.cag.2014.09.010. Online: `https://www.sciencedirect.com/science/article/pii/S0097849314000983`. Shape Modeling International 2014.

[6] Alexa, M. – Cohen-Or, D. – Levin, D. As-Rigid-as-Possible Shape Interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 157–164, USA, 2000. ACM Press/Addison-Wesley Publishing Co. doi: 10.1145/344779.344859. Online: `https://doi.org/10.1145/344779.344859`. ISBN 1581132085.

[7] Alexiadis, D. S. – Zarpalas, D. – Daras, P. Real-time, full 3-D reconstruction of moving foreground objects from multiple consumer depth cameras. *IEEE Transactions on Multimedia*. 2012, 15, 2, pages 339–358.

[8] Belkin, M. – Sun, J. – Wang, Y. Discrete laplace operator on meshed surfaces. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 278–287. ACM, 2008.

[9] Besl, P. J. – McKay, N. D. Method for registration of 3-D shapes. In *Sensor fusion IV: control paradigms and data structures*, 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[10] Biswas, S. et al. MuSCLE: Multi Sweep Compression of LiDAR using Deep Entropy Models. *arXiv preprint arXiv:2011.07590*. 2020.

[11] Bogo, F. et al. Dynamic FAUST: Registering Human Bodies in Motion. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[12] Bojsen-Hansen, M. – Li, H. – Wojtan, C. Tracking Surfaces with Evolving Topology. *ACM Trans. Graph.* July 2012, 31, 4. ISSN 0730-0301. doi: 10.1145/2185520.2185549. Online: `https://doi.org/10.1145/2185520.2185549`.

[13] Briceño, H. M. et al. Geometry videos. In *Eurographics/SIGGRAPH symposium on computer animation (SCA)*. Eurographics Association, 2003.

[14] Cao, C. et al. Skeleton-based motion estimation for Point Cloud Compression. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2020. doi: 10.1109/MMSP48831.2020.9287165.

[15] Cao, C. – Preda, M. – Zaharia, T. What's new in Point Cloud Compression? *Global Journal of Engineering Sciences.* 03 2020, 4. doi: 10.33552/GJES.2020.04.000598.

[16] Chaine, R. – Gandoin, P.-M. – Roudet, C. Reconstruction Algorithms as a Suitable Basis for Mesh Connectivity Compression. *IEEE Transactions on Automation Science and Engineering.* 2009, 6, 3, pages 443–453. doi: 10.1109/TASE.2009.2021336.

[17] Chen, C. et al. High-Fidelity Compression of Dynamic Meshes with Fine Details Using Piece-Wise Manifold Harmonic Bases. In *Proceedings of Computer Graphics International 2018*, CGI 2018, pages 23–32, New York, NY, USA, 2018. Association for Computing Machinery. doi: 10.1145/3208159.3208163. Online: `https://doi.org/10.1145/3208159.3208163`. ISBN 9781450364010.

[18] Corsini, M. et al. Perceptual Metrics for Static and Dynamic Triangle Meshes. *Computer Graphics Forum.* 2013, 32, 1, pages 101–125. doi: 10.1111/cgf.12001. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12001`.

[19] Daribo, I. et al. Efficient rate-distortion compression of dynamic point cloud for grid-pattern-based 3D scanning systems. *3D Research.* 2012, 3, 1, pages 2.

[20] De Queiroz, R. L. – Chou, P. A. Compression of 3d point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing.* 2016, 25, 8, pages 3947–3956.

[21] Queiroz, R. L. – Chou, P. A. Motion-compensated compression of dynamic voxelized point clouds. *IEEE Transactions on Image Processing.* 2017, 26, 8, pages 3886–3895.

[22] Dou, M. et al. 3D scanning deformable objects with a single RGBD sensor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 493–501, 2015.

[23] Doumanoglou, A. et al. Toward real-time and efficient compression of human time-varying meshes. *IEEE Transactions on Circuits and Systems for Video Technology*. 2014, 24, 12, pages 2099–2116.

[24] Dvořák, J. – Vaněček, P. – Váša, L. Towards Understanding Time Varying Triangle Meshes. In Paszynski, M. et al. (Ed.) *Computational Science – ICCS 2021*, pages 45–58, Cham, 2021. Springer International Publishing. doi: 10.1007/978-3-030-77977-1_4. ISBN 978-3-030-77977-1.

[25] Dvořák, J. – Maňák, M. – Váša, L. Predictive compression of molecular dynamics trajectories. *Journal of Molecular Graphics and Modelling*. 2020, 96, pages 107531. ISSN 1093-3263. doi: 10.1016/j.jmgm.2020.107531. Online: `https://www.sciencedirect.com/science/article/pii/S1093326319306564`.

[26] Dvořák, J. et al. As-rigid-as-possible volume tracking for time-varying surfaces. *Computers & Graphics*. 2022, 102, pages 329–338. ISSN 0097-8493. doi: 10.1016/j.cag.2021.10.015. Online: `https://www.sciencedirect.com/science/article/pii/S0097849321002284`.

[27] Faramarzi, E. – Joshi, R. – Budagavi, M. Mesh Coding Extensions to MPEG-I V-PCC. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5, 2020. doi: 10.1109/MMSP48831.2020.9287057.

[28] Feng, X. et al. A perceptual quality metric for 3D triangle meshes based on spatial pooling. *Frontiers of Computer Science*. 2018, 12, 4, pages 798–812. ISSN 2095-2236. doi: 10.1007/s11704-017-6328-x. Online: `https://doi.org/10.1007/s11704-017-6328-x`.

[29] Feng, Y. – Liu, S. – Zhu, Y. Real-Time Spatio-Temporal LiDAR Point Cloud Compression, 2020.

[30] Furukawa, R. et al. One-shot Entire Shape Acquisition Method Using Multiple Projectors and Cameras. In *2010 Fourth Pacific-Rim Symposium on Image and Video Technology*, pages 107–114, 2010. doi: 10.1109/PSIVT.2010.25.

[31] Galligan, F. et al. Google/Draco: a library for compressing and decompressing 3D geometric meshes and point clouds, 2018.

[32] Gao, Z. et al. [G-PCC][New proposal] Predictive Geometry Coding. *ISO/IEC JTC1/SC29/WG11 MPEG2019/m51012*. October 2019.

[33] Garcia, D. C. – Queiroz, R. L. Context-based octree coding for point-cloud video. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 1412–1416. IEEE, 2017.

[34] Garcia, D. C. et al. Geometry coding for dynamic voxelized point clouds using octrees and multiple contexts. *IEEE Transactions on Image Processing.* 2019, 29, pages 313–322.

[35] Garland, M. – Heckbert, P. S. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[36] Graziosi, D. et al. An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC). *APSIPA Transactions on Signal and Information Processing.* 2020, 9, pages e13. doi: 10.1017/ATSIP.2020.12.

[37] Gupta, S. – Sengupta, K. – Kassim, A. Registration and partitioning-based compression of 3-D dynamic data. *IEEE transactions on circuits and systems for video technology.* 2003, 13, 11, pages 1144–1155.

[38] Hammond, D. K. – Vandergheynst, P. – Gribonval, R. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis.* 2011, 30, 2, pages 129–150.

[39] Han, S.-R. – Yamasaki, T. – Aizawa, K. Time-varying mesh compression using an extended block matching algorithm. *IEEE Transactions on Circuits and Systems for Video Technology.* 2007, 17, 11, pages 1506–1518.

[40] Han, S.-R. – Yamasaki, T. – Aizawa, K. Geometry compression for time-varying meshes using coarse and fine levels of quantization and run-length encoding. In *2008 15th IEEE International Conference on Image Processing*, pages 1045–1048. IEEE, 2008.

[41] Hanocka, R. et al. MeshCNN: A Network with an Edge. *ACM Transactions on Graphics (TOG).* 2019, 38, 4, pages 90.

[42] Hou, J. et al. A novel compression framework for 3D time-varying meshes. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2161–2164. IEEE, 2014.

[43] Hruda, L. – Dvořák, J. – Váša, L. On evaluating consensus in RANSAC surface registration. *Computer Graphics Forum.* 2019, 38, 5, pages 175–186. doi: 10.1111/cgf.13798. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13798`.

[44] Hruda, L. – Dvořák, J. Estimating approximate plane of symmetry of 3D triangle meshes. In *Proc. Central European Seminar on Computer Graphics*, 2017.

[45] Hübbe, N. – Kunkel, J. Reducing the hpc-datastorage footprint with mafisc—multidimensional adaptive filtering improved scientific data compression. *Computer Science-Research and Development.* 2013, 28, 2, pages 231–239.

[46] J., H. et al. Compressing molecular dynamics trajectories: Breaking the one-bit-per-sample barrier. *Journal of Computational Chemistry.* 2016, pages 1897 – 1906. doi: 10.1002/jcc.24405.

[47] Jain, J. – Jain, A. Displacement measurement and its application in interframe image coding. *IEEE Transactions on communications.* 1981, 29, 12, pages 1799–1808.

[48] Kammerl, J. et al. Real-time compression of point cloud streams. In *2012 IEEE International Conference on Robotics and Automation*, pages 778–785. IEEE, 2012.

[49] Karni, Z. – Gotsman, C. Spectral compression of mesh geometry. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286, 2000.

[50] Karni, Z. – Gotsman, C. Compression of soft-body animation sequences. *Computers & Graphics.* 2004, 28, 1, pages 25–34. ISSN 0097-8493. doi: 10.1016/j.cag.2003.10.002. Online: `https://www.sciencedirect.com/science/article/pii/S0097849303002267`.

[51] Katajainen, J. – Mäkinen, E. Tree compression and optimization with applications. *International Journal of Foundations of Computer Science.* 1990, 1, 04, pages 425–447.

[52] Kathariya, B. et al. Embedded binary tree for dynamic point cloud geometry compression with graph signal resampling and prediction. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2017. doi: 10.1109/VCIP.2017.8305130.

[53] Kawasaki, H. et al. Dynamic scene shape reconstruction using a single structured light pattern. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. doi: 10.1109/CVPR.2008.4587702.

[54] Kim, J. et al. 3D Motion Estimation and Compensation Method for Video-Based Point Cloud Compression. *IEEE Access.* 2020, 8, pages 83538–83547. doi: 10.1109/ACCESS.2020.2991478.

[55] Kuhn, H. W. The Hungarian method for the assignment problem. *Naval research logistics quarterly.* 1955, 2, 1-2, pages 83–97.

[56] Lasserre, S. – Flynn, D. Point Cloud Compression in MPEG and beyond, May 2019. Online: `http://clim.inria.fr/workshop/LasserrePCC.pdf`.

[57] Lavoué, G. A Multiscale Metric for 3D Mesh Visual Quality Assessment. *Computer Graphics Forum.* 2011, 30, 5, pages 1427–1437. doi: 10.1111/j.1467-8659.2011.02017.x. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.02017.x`.

[58] Lengyel, J. E. Compression of Time-Dependent Geometry. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, I3D '99, pages 89–95, New York, NY, USA, 1999. Association for Computing Machinery. doi: 10.1145/300523.300533. Online: https://doi.org/10.1145/300523.300533. ISBN 1581130821.

[59] Lewiner, T. et al. GEncode: Geometry-driven compression for General Meshes. *Computer Graphics Forum.* 2006, 25, 4, pages 685–695. doi: 10.1111/j.1467-8659.2006.00990.x. Online: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2006.00990.x.

[60] Li, H. et al. Robust Single-View Geometry and Motion Reconstruction. *ACM Trans. Graph.* December 2009, 28, 5, pages 1–10. ISSN 0730-0301. doi: 10.1145/1618452.1618521. Online: https://doi.org/10.1145/1618452.1618521.

[61] Li, L. et al. Advanced 3D Motion Prediction for Video-Based Dynamic Point Cloud Compression. *IEEE Transactions on Image Processing.* 2020, 29, pages 289–302. doi: 10.1109/TIP.2019.2931621.

[62] Lien, J.-M. – Kurillo, G. – Bajcsy, R. Multi-camera tele-immersion system with real-time model driven data compression. *The Visual Computer.* 2010, 26, 1, pages 3.

[63] Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory.* 1982, 28, 2, pages 129–137. doi: 10.1109/TIT.1982.1056489.

[64] Lobaz, P. – Váša, L. Hierarchical Laplacian-based compression of triangle meshes. *Graphical Models.* 2014, 76, 6, pages 682–690. ISSN 1524-0703. doi: 10.1016/j.gmod.2014.09.003. Online: https://www.sciencedirect.com/science/article/pii/S1524070314000502.

[65] Luo, G. et al. 3D Mesh Animation Compression Based on Adaptive Spatio-Temporal Segmentation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '19, New York, NY, USA, 2019. Association for Computing Machinery. doi: 10.1145/3306131.3317017. Online: https://doi.org/10.1145/3306131.3317017. ISBN 9781450363105.

[66] Luo, G. et al. Spatio-Temporal Segmentation Based Adaptive Compression of Dynamic Mesh Sequences. *ACM Trans. Multimedia Comput. Commun. Appl.* March 2020, 16, 1. ISSN 1551-6857. doi: 10.1145/3377475. Online: https://doi.org/10.1145/3377475.

[67] Luo, G. et al. Dynamic data reshaping for 3D mesh animation compression. *Multimedia Tools and Applications.* 2021, pages 1–18.

[68] Maeda, T. – Yamasaki, T. – Aizawa, K. Model-Based Analysis and Synthesis of Time-Varying Mesh. In Perales, F. J. – Fisher, R. B. (Ed.) *Articulated Motion and Deformable Objects*, pages 112–121, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-70517-8.

[69] Maglo, A. et al. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Comput. Surv.* February 2015, 47, 3. ISSN 0360-0300. doi: 10.1145/2693443. Online: `https://doi.org/10.1145/2693443`.

[70] Malvar, H. S. Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian sources with unknown statistics. In *Data Compression Conference (DCC'06)*, pages 23–32. IEEE, 2006.

[71] Mammou, K. – Zaharia, T. – Prêteux, F. TFAN: A low complexity 3D mesh compression algorithm. *Computer Animation and Virtual Worlds.* 2009, 20, 2-3, pages 343–354.

[72] Marais, P. – Gain, J. – Shreiner, D. Distance-Ranked Connectivity Compression of Triangle Meshes. *Computer Graphics Forum.* 2007, 26, 4, pages 813–823. doi: 10.1111/j.1467-8659.2007.01026.x. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01026.x`.

[73] Mekuria, R. – Blom, K. – Cesar, P. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology.* 2016, 27, 4, pages 828–842.

[74] Milani, S. – Polo, E. – Limuti, S. A Transform Coding Strategy for Dynamic Point Clouds. *IEEE Transactions on Image Processing.* 2020, 29, pages 8213–8225. doi: 10.1109/TIP.2020.3011811.

[75] Mourycová, E. – Váša, L. Geometry Compression of Triangle Meshes using a Reference Shape. In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP,*, 2022.

[76] Nakagawa, S. – Yamasaki, T. – Aizawa, K. Deformation-based data reduction of Time-Varying Meshes for displaying on mobile terminals. In *2010 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 1–4, 2010. doi: 10.1109/3DTV.2010.5506509.

[77] Niemeyer, M. et al. Occupancy Flow: 4D Reconstruction by Learning Particle Dynamics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[78] Orts-Escolano, S. et al. Holoportation: Virtual 3D Teleportation in Real-Time. UIST '16, pages 741–754, New York, NY, USA, 2016.

Association for Computing Machinery. doi: 10.1145/2984511.2984517. Online: `https://doi.org/10.1145/2984511.2984517`. ISBN 9781450341899.

[79] Peixoto, E. Intra-Frame Compression of Point Cloud Geometry Using Dyadic Decomposition. *IEEE Signal Processing Letters.* 2020, 27, pages 246–250. doi: 10.1109/LSP.2020.2965322.

[80] Peixoto, E. – Medeiros, E. – Ramalho, E. Silhouette 4d: An Inter-Frame Lossless Geometry Coder Of Dynamic Voxelized Point Clouds. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2691–2695, 2020. doi: 10.1109/ICIP40778.2020.9190648.

[81] Pinkall, U. – Polthier, K. Computing Discrete Minimal Surfaces and Their Conjugates. *Experimental Mathematics.* 1993, 2, 1, pages 15–36. doi: 10.1080/10586458.1993.10504266. Online: `https://doi.org/10.1080/10586458.1993.10504266`.

[82] Rossignac, J. Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics.* 1999, 5, 1, pages 47–61. doi: 10.1109/2945.764870.

[83] Rubner, Y. – Tomasi, C. – Guibas, L. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 59–66, 1998. doi: 10.1109/ICCV.1998.710701.

[84] Said, A. – Pearlman, W. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology.* 1996, 6, 3, pages 243–250. doi: 10.1109/76.499834.

[85] Schwarz, S. et al. Video coding of dynamic 3D point cloud data. *APSIPA Transactions on Signal and Information Processing.* 2019, 8, pages e31. doi: 10.1017/ATSIP.2019.24.

[86] Sorkine, O. – Cohen-Or, D. – Toledo, S. High-pass quantization for mesh encoding. In *Symposium on Geometry Processing*, 42, 2003.

[87] Sorkine-Hornung, O. – Rabinovich, M. Least-Squares Rigid Motion Using SVD, 2016. Technical note.

[88] Spångberg, D. – Larsson, D. S. D. – Spoel, D. Trajectory NG: portable, compressed, general molecular dynamics trajectories. *Journal of Molecular Modeling.* 2011, 17, 10, pages 2669–2685. doi: 10.1007/s00894-010-0948-5.

[89] Sullivan, G. J. et al. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology.* 2012, 22, 12, pages 1649–1668. doi: 10.1109/TCSVT.2012.2221191.

[90] Sumner, R. W. – Schmid, J. – Pauly, M. Embedded Deformation for Shape Manipulation. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, pages 80–es, New York, NY, USA, 2007. Association for Computing Machinery. doi: 10.1145/1275808.1276478. Online: `https://doi.org/10.1145/1275808.1276478`. ISBN 9781450378369.

[91] Tevs, A. et al. Animation Cartography—Intrinsic Reconstruction of Shape and Motion. *ACM Trans. Graph.* apr 2012, 31, 2. ISSN 0730-0301. doi: 10.1145/2159516.2159517. Online: `https://doi.org/10.1145/2159516.2159517`.

[92] Thanou, D. – Chou, P. A. – Frossard, P. Graph-based motion estimation and compensation for dynamic 3D point cloud compression. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 3235–3239. IEEE, 2015.

[93] Thanou, D. – Chou, P. A. – Frossard, P. Graph-based compression of dynamic 3D point cloud sequences. *IEEE Transactions on Image Processing.* 2016, 25, 4, pages 1765–1778.

[94] Torkhani, F. – Wang, K. – Chassery, J.-M. A Curvature-Tensor-Based Perceptual Quality Metric for 3D Triangular Meshes. *Machine GRAPHICS & VISION.* 2014, 23, 1/2, pages 59–82.

[95] Touma, C. – Gotsman, C. Triangle Mesh Compression. In *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, pages 26–34, June 1998. Online: `http://graphicsinterface.org/wp-content/uploads/gi1998-4.pdf`.

[96] Vallet, B. – Lévy, B. Spectral Geometry Processing with Manifold Harmonics. *Computer Graphics Forum.* 2008, 27, 2, pages 251–260. doi: https://doi.org/10.1111/j.1467-8659.2008.01122.x. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01122.x`.

[97] Váša, L. – Dvořák, J. Error propagation control in Laplacian mesh compression. *Computer Graphics Forum.* 2018, 37, 5, pages 61–70. doi: 10.1111/cgf.13491. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13491`.

[98] Váša, L. – Brunnett, G. Exploiting Connectivity to Improve the Tangential Part of Geometry Prediction. *IEEE Transactions on Visualization & Computer Graphics.* sep 2013, 19, 09, pages 1467–1475. ISSN 1941-0506. doi: 10.1109/TVCG.2013.22.

[99] Váša, L. – Petřík, O. Optimising Perceived Distortion in Lossy Encoding of Dynamic Meshes. *Computer Graphics Forum.* 2011, 30, 5, pages 1439–1449. doi: 10.1111/j.1467-8659.2011.02018.x. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.02018.x`.

[100] Váša, L. – Petřík, O. Optimising Perceived Distortion in Lossy Encoding of Dynamic Meshes. *Computer Graphics Forum.* 2011, 30, 5, pages 1439–1449. doi: 10.1111/j.1467-8659.2011.02018.x. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.02018.x`.

[101] Váša, L. et al. Compressing dynamic meshes with geometric laplacians. *Computer Graphics Forum.* 2014, 33, 2, pages 145–154. doi: 10.1111/cgf.12304. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12304`.

[102] Váša, L. – Rus, J. Dihedral Angle Mesh Error: a fast perception correlated distortion measure for fixed connectivity triangle meshes. *Computer Graphics Forum.* 2012, 31, 5, pages 1715–1724. doi: 10.1111/j.1467-8659.2012.03176.x. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03176.x`.

[103] Váša, L. – Skala, V. CODDYAC: Connectivity Driven Dynamic Mesh Compression. In *2007 3DTV Conference*, pages 1–4, 2007. doi: 10.1109/3DTV.2007.4379408.

[104] Váša, L. – Skala, V. A Perception Correlated Comparison Method for Dynamic Meshes. *IEEE Transactions on Visualization and Computer Graphics.* 2011, 17, 2, pages 220–230. doi: 10.1109/TVCG.2010.38.

[105] Vlasic, D. et al. Articulated Mesh Animation from Multi-View Silhouettes. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery. doi: 10.1145/1399504.1360696. Online: `https://doi.org/10.1145/1399504.1360696`. ISBN 9781450301121.

[106] Wall, J. et al. REVERIE: Natural human interaction in virtual immersive environments. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2165–2167. IEEE, 2014.

[107] Wang, K. – Torkhani, F. – Montanvert, A. A fast roughness-based approach to the assessment of 3D mesh visual quality. *Computers & Graphics.* 2012, 36, 7, pages 808–818. ISSN 0097-8493. doi: 10.1016/j.cag.2012.06.004. Online: `https://www.sciencedirect.com/science/article/pii/S0097849312001203`. Augmented Reality Computer Graphics in China.

[108] Wang, S. et al. Deep Parametric Continuous Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[109] Yamasaki, T. – Aizawa, K. Patch-based compression for time-varying meshes. In *2010 IEEE International Conference on Image Processing*, pages 3433–3436. IEEE, 2010.

[110] Yang, B. et al. Motion-Aware Compression and Transmission of Mesh Animation Sequences. *ACM Trans. Intell. Syst. Technol.* April 2019, 10, 3. ISSN 2157-6904. doi: 10.1145/3300198. Online: `https://doi.org/10.1145/3300198`.

[111] Yang, J.-H. – Kim, C.-S. – Lee, S.-U. Semi-regular representation and progressive compression of 3-D dynamic mesh sequences. *IEEE Transactions on Image Processing.* 2006, 15, 9, pages 2531–2544.

[112] Zollhöfer, M. et al. State of the Art on 3D Reconstruction with RGB-D Cameras. *Computer Graphics Forum.* 2018, 37, 2, pages 625–652. doi: https://doi.org/10.1111/cgf.13386. Online: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13386`.

Appendix A

Activities

A.1 Publications in International Conferences

- DVOŘÁK, J. VANĚČEK, P. VÁŠA, L. Towards Understanding Time-Varying Triangle Meshes. In PASZYNSKI, M. et al. (Ed.) *Computational Science – ICCS 2021*, pages 45–58, Cham, 2021. Springer International Publishing. doi: 10.1007/978-3-030-77977-1_4. ISBN 978-3-030-77977-1. [24] (40%)

A.2 Publications in Impacted Journals

- VÁŠA, L. DVOŘÁK, J. Error propagation control in Laplacian mesh compression. *Computer Graphics Forum*. 2018, 37, 5, pages 61–70. doi: 10.1111/cgf.13491. [97] (50%)

- DVOŘÁK, J. MAŇÁK, M. VÁŠA, L. Predictive compression of molecular dynamics trajectories. *Journal of Molecular Graphics and Modelling*. 2020, 96. ISSN 1093-3263. doi: 10.1016/j.jmgm.2020.107531. [25] (60%)

- DVOŘÁK, J. KÁČEREKOVÁ, Z. VANĚČEK, P. HRUDA, L. VÁŠA, L. As-rigid-as-possible volume tracking for time-varying surfaces. *Computers & Graphics*. 2022, 102, pages 329–338. ISSN 0097-8493. doi: 10.1016/j.cag.2021.10.015 [26] (50%)

A.3 Participation in Scientific Projects

- SGS-2019-016, Synthesis and Analysis of Geometric and Computing Models, Ministry of Education, Youth and Sports

- 17-07690S, Methods of Identification and Visualization of Tunnels for Flexible Ligands in Dynamic Proteins, Czech Science Foundation

- 20-02154S, Representation and processing methods for three dimensional dynamic shapes, Czech Science Foundation

A.4 Teaching Activities

2018/2019

- Polygon Mesh Processing (KIV/ZPOS), Czech, tutor
- Computers and Programming 2 (KIV/PPA2), Czech, tutor of 2 student groups

2019/2020

- Polygon Mesh Processing (KIV/ZPOS), Czech, tutor

2020/2021

- Polygon Mesh Processing (KIV/ZPOS), Czech, tutor
- Polygon Mesh Processing (KIV/ZPOSE), English, for Erasmus students, tutor

A.5 Non-related Publications

- Hruda, L. Dvořák, J. Estimating approximate plane of symmetry of 3D triangle meshes. In *Proc. Central European Seminar on Computer Graphics*, 2017. [44] (50%)
- Hruda, L. Dvořák, J. Váša, L. On evaluating consensus in RANSAC surface registration. *Computer Graphics Forum.* 2019, 38, 5, pages 175–186. doi: 10.1111/cgf.13798. [43] (5%)

A.6 Stays Abroad

- Research stay at *TU Munich*, Computer Vision Group, Feb. 2020 - May 2020, first month in person, continued online due to the COVID-19 pandemic.

A.7 Oral Presentations

- Towards Understanding Time-Varying Triangle Meshes, 18. 6. 2021, English, *ICCS 2021*, online
- As-rigid-as-possible volume tracking for time-varying surfaces, 15. 11. 2021, English, *SMI 2021*, online

Appendix B

Links to Full Versions of Related Published Work

Error Propagation Control in Laplacian Mesh Compression

- `https://doi.org/10.1111/cgf.13491`

Predictive compression of molecular dynamics trajectories

- `https://doi.org/10.1016/j.jmgm.2020.107531`

Towards Understanding Time-Varying Triangle Meshes

- `https://doi.org/10.1007/978-3-030-77977-1_4`

As-rigid-as-possible volume tracking for time-varying surfaces

- `https://doi.org/10.1016/j.cag.2021.10.015`