CHAPTER

Coding of dynamic 3D meshes

1

Jean-Eudes Marvie[a], Maja Krivokuća[a], and Danillo Graziosi[b]
[a]InterDigital INC., [b]Sony Corporation of America

ABSTRACT
This chapter presents the state-of-the-art in the coding (or compression) of dynamic 3D mesh models. Section 1.1 introduces the motivations behind using 3D meshes for volumetric video. In Section 1.2, some fundamental mesh concepts are explained, which are required in order to understand the state-of-the-art that follows. Section 1.3 briefly revisits the topic of static mesh compression, while sections 1.4 and 1.5 provide an extensive review of the state-of-the-art in dynamic mesh compression. Finally, Section 1.6 concludes this chapter and offers some opinions on potential future directions of dynamic mesh compression.

1.1 Introduction

When it comes to *volumetric video coding* (i.e., the compression of three-dimensional (3D) videos obtained from real-life captures of moving subjects), we often see solutions working on *point clouds* or *voxel data*, as discussed in previous chapters. This is due to the fact that the existing devices that are available to capture real-life animated subjects are based on discrete sensors (e.g., cameras, depth sensors, etc.), so a point-based 3D representation is a natural output of such sensors. The signal coming from the acquisition-reconstruction step of these sensors therefore offers a convenient trade-off between the quality of the captured subject (in terms of sufficiently representing its surface details and photometry) and the reconstruction cost, which is relatively low compared to other 3D representations that require additional reconstruction steps. Therefore, using the reconstructed signal in this form for its consumption can be an effective approach in the various use cases of *volumetric videos*.

However, 3D *point clouds* are not so convenient to render, especially when targeting real-time playback by leveraging graphics hardware. Indeed, obtaining high quality renders of point clouds requires accurate *splatting* methods [1] to produce continuous surface projections of the model (which is originally discrete) without holes between the rendered points. Splatting usually requires additional pre-generated information per point, such as the splat radius or stretch vector, normal vector, or other information, which massively increases the cost of storage. Some point cloud rendering approaches also make use of hierarchical representations such as *octrees* to find

point neighborhoods and compute accurate splat stretches at runtime. This reduces the required additional volume of data, but increases the processing complexity for rendering. In the case of *voxel data*, which are equivalent to point clouds aligned within a 3D grid (where each grid element is set to 1 to express the presence of a point or 0 otherwise), rendering can require less additional data than for raw point clouds, thanks to the implicit knowledge of point neighborhoods. However, such grid structures require the storage of points that are regularly organized, and high-resolution grids are needed to accurately represent models that contain many fine (high-frequency) geometric details. Here again, some hierarchical data structures such as octrees can be used to lower the data volume, but at the cost of complexity at rendering.

Another approach for 3D object representation is to reconstruct a surface made of triangles, by connecting neighboring points from point clouds (using a *graph* relationship - see Section 1.2), and to encode this additional data, called the *mesh connectivity*, which is used to describe the *topology* of the 3D mesh model.[1] The ensemble of point positions and triangles (described by indices into the array of point positions) is called a *mesh*. The so-connected points are called the *vertices* of the mesh, and any colors associated with the points are called simply the *vertex colors*, or more generally the *attributes* of the points. In the context of volumetric video, where real-life captures are used and not computer-generated 3D models, the raw point clouds are usually very dense. Hence, the meshes obtained from such point clouds are also very dense. Throughout this chapter, we will refer to such meshes as *dense meshes with color per vertex*. These meshes can also be generated from *voxel data* by using surface reconstruction algorithms such as *marching cubes* [2]. In both cases, the mesh is then used at rendering to produce continuous 2D surfaces in the final 2D image that will be displayed on screen for the user, by performing the projection of inter-connected *triangles* instead of the set of points. Colors per vertex are usually linearly interpolated to produce colors inside the triangle projections. The projection of these triangles can be easily performed by using *ray-tracing* or *rasterization* algorithms. Even though recent 3D graphics hardware is able to execute both of these methods in real-time, the most widely used one is rasterization. By leveraging such modern graphics hardware, which is optimized to work with triangle meshes, it is very easy to render 3D meshes comprising thousands of triangles in real time at 30 frames per second, even on mobile phones. Another strong advantage of using mesh models is the possibility to zoom in at will on the mesh without introducing holes in the projected (rendered) version of the mesh. There also exist many *multi-resolution representations* of meshes, permitting to adapt the quality on demand at the rendering stage. Meshes are also very convenient to reconstruct normal vectors of the 3D object surface on the fly, whereas point clouds require complex neighborhood searches to do so. In contrast to point clouds, renderers can easily apply lighting to mesh models by using shaders, and generate shadows on and from the meshes, which is important

[1] Note that many different *connectivities* can be used to describe the same surface *topology*, as long as the *genus* of the mesh does not change (see Section 1.2.6).

for Virtual Reality (VR) and Augmented Reality (AR) applications. More generally, meshes can also be used to model watertight surfaces, which is a useful property for physics simulations. Furthermore, collisions are easy to compute for meshes, which is very convenient for interactions in AR, VR, and game environments (game engines mostly use *textured mesh* representations - explained in the following paragraph).

Even though dense meshes are very good for rendering the original dense signal (captured 3D object) while preserving the captured signal quality, the cost of storage for the vertex positions and connectivity is non-negligible, and such meshes can quickly saturate the renderer with many millions of triangles per second. Preserving such high-resolution models can be of interest in some cases, for instance in the scientific visualization context to render very fine geometric details coming from physics simulations. However, such high geometric precision is usually not needed in most use cases of volumetric videos, such as telepresence or integration within VR or AR environments. In these latter use cases, the geometric precision can be lower, while the photometric precision (the color details, such as the skin pores) must be preserved. Therefore, most industrial volumetric video captures today use *textured mesh* representations instead of *dense meshes with color per vertex* for the 3D content representation. In the case of *textured meshes*, the dense mesh is also usually *simplified* - that is, the original mesh surface is approximated by decimating it to obtain a lower-resolution mesh with fewer triangles, which represents the original surface by *piecewise linear approximation*. The vertex colors of the dense mesh are also projected into a *texture map* made of one or several 2D images. Some mapping coordinates, usually noted uv, are assigned to the vertices of the *simplified* mesh in order to associate the surface of its triangles to different parts of the texture map. At rendering time, the simplified mesh with uv coordinates and its associated texture map are used to fill the projected triangles with highly detailed photometry extracted from the texture map. This *textured mesh* representation presents many advantages: for instance, graphics hardware can easily handle meshes with texture maps up to 4 million pixels with good quality filtering using *mipmapping* [3]. The mesh resolution can be set appropriately (at the production or encoding stage) to a desired resolution, without introducing heavy distortions at the photometric level. In terms of compression, the encoding of the image texture can leverage classical 2D image coding schemes, such as PNG (lossless) or JPEG (lossy). Moreover, texture map *sequences* can be encoded using existing 2D video compression schemes, such as HEVC or VVC. For all these reasons, and many others such as easy editing, the *textured mesh* is the de facto representation for 3D models. All of these advantages can also be leveraged in the context of volumetric video.

Historically, before the recent trend of *sensor*-based acquisition of animated subjects, the only animated meshes that were available were purely synthetic ones, produced using 3D modeling software and eventually human pose scan and motion capture, mostly by the Gaming, Visual Effects (VFX), and animated movies industries. Such 3D content, from the Computer-Generated Image (CGI) field, is slightly (but importantly) different from real-life acquisitions. CGI sequences are generally made of a topology, as well as a texture map and uv coordinates, that are common to

all the frames of the sequence (i.e., static topology, texture map, and *uv*s), with only the vertex positions evolving over time. These *mesh sequences with constant connectivity* are commonly called *Animated Meshes* (AMs). In contrast, *sensor*-captured 3D sequences are usually made up of positions, topologies, vertex *uv*s and texture maps that can all vary for each frame. These *mesh sequences with variable connectivity* are commonly called *Time-Varying Meshes* (TVMs). Note that with TVMs, even the *number of vertices* can vary at each frame, which makes those meshes even more challenging to compress.

The rest of this chapter is organized as follows. After an introduction to some important mesh fundamental concepts in Section 1.2, we present a review of existing techniques for static mesh compression in Section 1.3. We then explain, in Section 1.4, how mesh sequences with *variable connectivity* (i.e. TVMs) can be converted into mesh *sub-sequences* with *constant connectivity* (i.e. sub-sequences of AMs) through *tracking* and *re-meshing*, or directly constructed as sub-sequences of AMs. We review, in this same section, the different AM compression methods currently existing in the literature, as well as those that have been standardized by international standards organizations. We finally present the existing methods for the coding of TVMs (without any tracking or pre-processing for conversion to AMs) in Section 1.5. Section 1.6 concludes the chapter. In Figure 1.1, we summarize the different publications related to the compression of AMs and TVMs, in the taxonomy that will be used to present these methods in this chapter.

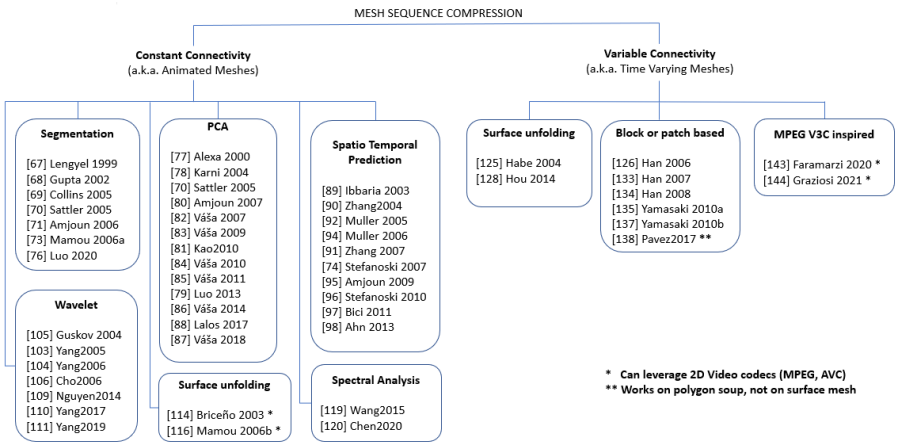

FIGURE 1.1

A taxonomy of dynamic 3D mesh compression techniques.

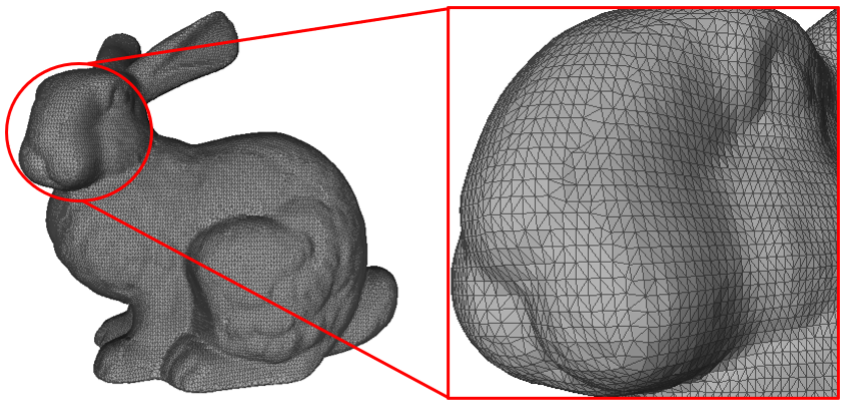

FIGURE 1.2

(a) Triangle mesh representation of the Stanford Bunny; (b) Zoomed-in region of the Bunny's head from (a), to show the triangular faces more clearly.

1.2 Mesh Fundamentals

A 3D *polygonal mesh* model (sometimes also referred to as a *surface mesh*, as it explicitly defines a 3D object's surface but not its interior volume) consists of a set of planar polygons that are defined by three types of elements: *vertices*, *edges*, and *faces* (or *facets*). The *vertices* constitute a set of points in 3D Euclidean space $\mathbb{R}^3$, which are defined by their (x, y, z) coordinate values, similarly to the *point cloud* representation. These vertices are linked by straight *edges* to form polygonal *faces*. The faces are most commonly triangles, but they can also be other simple convex polygons, e.g., quadrilaterals. Each vertex or face of a mesh may also have additional *attributes* associated with it. These attributes are most commonly (R, G, B) colours (or textures), but there could also be surface normals or other per-vertex or per-face attributes. An example of a triangular surface mesh is shown in Figure 1.2. Due to the planarity of the mesh faces, a polygonal mesh can only *approximate* a curved (smooth) surface, and a more accurate approximation of this surface can be achieved by increasing the density of vertices and faces in (the corresponding region of) the mesh. Therefore, obtaining an increasingly more accurate representation of a smooth surface using a polygonal mesh can easily require a very large amount of data. This is why efficient mesh compression algorithms are crucial to enable the use, storage, and distribution of 3D mesh models. On the other hand, since modern 3D graphics hardware is optimized to deal with triangle mesh representations, such meshes are relatively easy to render and manipulate. Triangle meshes are also easily derived from other surface representations, which makes them a very flexible and portable model type [4]. Another reason for the popularity of triangle meshes (and

polygonal meshes in general) is that they are capable of modeling any complex object of arbitrary topology (provided that there is enough computer memory available). For these reasons, the vast majority of 3D models created and used today are triangular surface meshes.

In the sub-sections below, we introduce some fundamental concepts that are necessary in order to more fully understand 3D mesh models and therefore the work on mesh compression that will be presented in later sections in this chapter. Note that the material in the following sub-sections is largely based on the work in [5].

1.2.1 Manifold vs Non-Manifold Meshes

A 3D mesh, being a *boundary* (or *shell*) representation of a solid three-dimensional object, can be viewed as a two-dimensional surface embedded in $\mathbb{R}^3$. The mesh can then be characterized as a *2-manifold* (or simply *manifold*) if every point on its surface has a neighbourhood that is *homeomorphic* to an *open disc* of $\mathbb{R}^2$. Two objects are said to be *homeomorphic* if one of the objects can be stretched or bent, without tearing, to form the other object. Intuitively, this means that, at every point on the surface of a 2-manifold, the surface locally looks like the 2D plane. This implies that one edge must be shared by only two faces and not more, and each vertex must have only one ring of connected faces around it. Thus for each vertex, all the faces incident to this vertex must form a *closed fan* (or *disc*). Figure 1.3 shows a 2D example of such a closed fan (or disc-shaped) neighbourhood, along with two examples of non-manifold mesh connectivities.

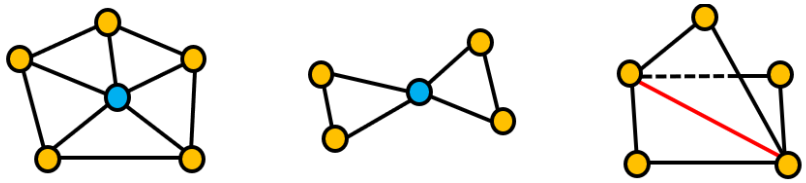

FIGURE 1.3

Examples of manifold and non-manifold mesh connectivities: (a) Manifold (notice the disc-shaped neighbourhood around the central, blue vertex); (b) Non-manifold, since the neighbourhood around the central vertex is not a closed fan; (c) Non-manifold, since the edge in red is shared by 3 faces, not 2.

1.2.2 Meshes With and Without Boundaries

It is possible for a 3D mesh to have one or more *boundaries*, so that it represents an *open* instead of a *closed* mesh. In this case, the mesh can be said to be a *manifold with boundary* if every point on the boundary has a neighbourhood that is homeomorphic to a *half-disc* in $\mathbb{R}^2$, while all the points that are not on the boundary

have neighbourhoods that are homeomorphic to an open disc (as explained in section 1.2.1, above). A boundary edge is incident to only one face instead of two. For a vertex on the boundary, all the faces incident to this vertex thus form an *open* fan instead of a closed one. Figure 1.4 illustrates an example of an open fan (or half-disc) neighbourhood around a vertex, and it shows an example of a manifold mesh with and without boundaries.

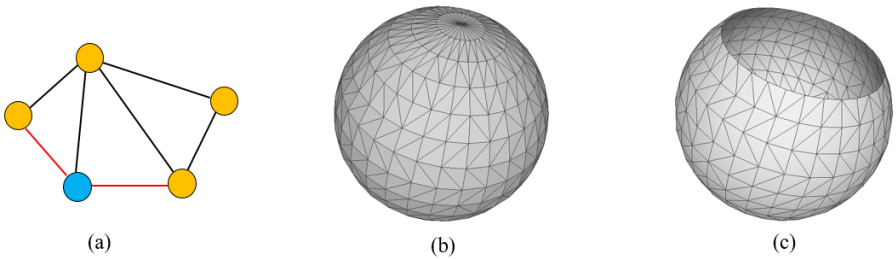

FIGURE 1.4

(a) Example of a half-disc neighbourhood around a vertex (blue) on the boundary of a mesh – the boundary edges are marked in red; (b) Example of a mesh without boundaries (i.e., a closed mesh); (c) Example of a mesh with boundaries (i.e., an open mesh).

1.2.3 Mesh Genus

For a manifold mesh, we can also define its *genus*. The *genus* is the number of "handles" that this mesh has. For example, the Torus mesh in Figure 1.5(b) has a genus of 1, and the Eight mesh in Figure 1.5(c) has a genus of 2. A mesh that has no handles (i.e., a genus of 0) and no boundary edges is called a *simple mesh*, e.g., see Figure 1.5(a). A simple mesh is topologically equivalent to a sphere, i.e., it is *homeomorphic* (see Section 1.2.1) to a sphere, as it can be moulded into the shape of a sphere without tearing the mesh's surface.

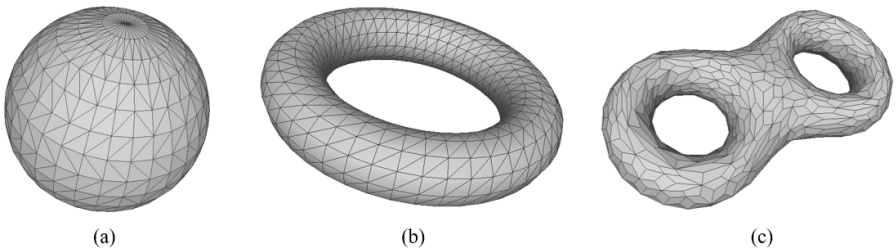

FIGURE 1.5

Examples of meshes with different topologies: (a) Genus 0; (b) Genus 1; (c) Genus 2.

1.2.4 Types of Connectivity in a Mesh

We can define the vertex *degree* (or *valence*) for each vertex of a mesh, as the number of edges incident to that vertex. We say that a polygon mesh has a *regular* connectivity if all of its vertices have the same degree (typically 6 for a triangle mesh); an *irregular* connectivity if the vertices have varying degrees; and a *semi-regular* connectivity if all of the vertices have a regular connectivity except for a few "extraordinary" vertices, which can have an irregular connectivity. Figure 1.6 shows some examples of regular, irregular, and semi-regular connectivities for a triangular mesh model (note that these examples are shown in 2D for the sake of simplicity, so the vertices on the boundaries appear as if they have fewer edges connected to them than they actually do).

Most 3D mesh models in practice have an *irregular* connectivity, but they are sometimes remeshed to obtain a regular or semi-regular connectivity in order to facilitate mesh compression.

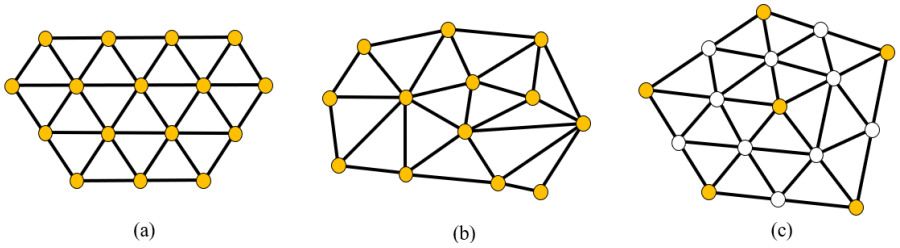

FIGURE 1.6

Examples of different connectivity types for a triangular mesh: (a) Regular connectivity; (b) Irregular connectivity; (c) Semi-regular connectivity (the "extraordinary" vertices are shown in yellow here, while the vertices with regular connectivity are shown in white).

1.2.5 Representing a Mesh as a Graph

The *connectivity* of a mesh can be represented as a *planar graph* [6] $G = (V, E)$, where V denotes the set of mesh vertices and E denotes the set of edges making up the mesh. Each node of the graph represents one vertex, and the links between the different nodes represent the edges that connect these vertices together. In the simplest (and most convenient) case, the connectivity graph is *simple*. This means that: (i) the links between the different nodes are *undirected* (i.e., the edges have no orientation), (ii) there are *no loops* around any node (i.e., each edge connects two *different* vertices, not any one vertex to itself), (iii) there are *no multiple links* between any pair of nodes (i.e., there can only be *one edge* connecting any given pair of vertices), and (iv) the graph links are *unweighted* (i.e., the edges have no weights associated with them, as they are all considered equally important). Figure 1.7 illustrates an example of a simple graph with five nodes, and two examples of non-simple graphs: one with multiple edges, and one with loops. In practice, non-simple graphs could result in

degenerate mesh models, e.g., a loop around a vertex (such as in Figure 1.7(c)) would mean that the edge there is not a straight line (but all edges must be straight lines in a polygonal mesh), and multiple edges between vertices (such as in Figure 1.7(b)) would create degenerate polygons as the extra edge would just represent a line segment rather than a polygon.

Unfortunately, in practice, particularly for meshes generated from scans of real objects (such as in the case of Time-Varying Meshes, covered in Section 1.5), the produced mesh models rarely have *simple* connectivity, and sometimes must be "cleaned up" prior to being processed.

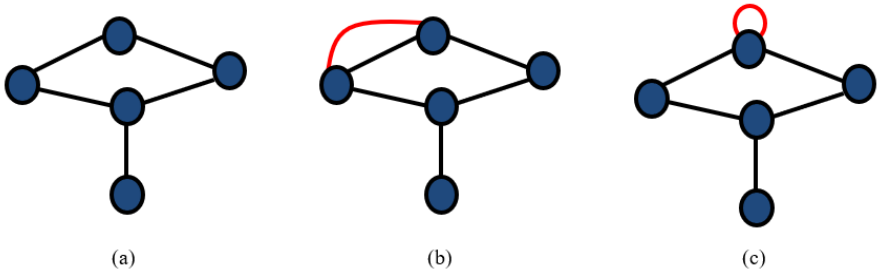

FIGURE 1.7

(a) Example of a simple graph; (b) Example of a non-simple graph with multiple edges between two nodes (the extra edge is shown in red); (c) Example of a non-simple graph with a self-loop around one node (shown in red).

A mesh represented as a graph may also have one or more *connected components*. Connected components are essentially sub-meshes, which are disconnected from each other, but are used together to describe a 3D object or scene.

1.2.6 Euler-Poincaré Characteristic

Considering a 2-manifold mesh with f faces, v vertices, e edges, genus g, and δ boundary polygons, the Euler-Poincaré characteristic can be used to determine if two 2-manifold meshes are homeomorphic. The Euler-Poincaré characteristic is defined as:

$$f + v - e = 2 - 2g - \delta. \tag{1.1}$$

We can say that two 2-manifold meshes *without boundary* ($\delta = 0$) are homeomorphic to each other *if and only if* they have the same Euler-Poincaré characteristic, i.e., if the left-hand side (or right-hand side) of equation (1.1) evaluates to the same value for both meshes. We can see from (1.1) that this is only possible in the case where both meshes have the *same genus* g. Equation (1.1) also tells us that two 2-manifold meshes *with boundary* ($\delta > 0$) can only be homeomorphic if, in addition to having the same genus g, they also have the *same number of boundary polygons* δ. From

these observations, we can conclude that in order for two 2-manifold meshes to be considered *topologically equivalent*, it does not matter how many vertices, faces, or edges they each have, as long as they have the same number of "handles" (in the case of meshes with genus $g > 0$) and the same number of boundary polygons (for meshes with boundaries). Indeed, this is an intuitive conclusion, because any discrete 3D model of a real-world object, such as a mesh or a point cloud, represents only one possible sampling of the real surface of the 3D object; therefore, many such samplings are possible to represent the same surface topology.

In a triangle manifold mesh, we know that each face is constructed from 3 edges. If we then assume that there is a sufficiently large number of edges and triangles and that the ratio of the number of boundary edges (if these exist) to the number of non-boundary edges is negligible [7], we can say that each edge is generally shared by two triangles (as is the case for a manifold mesh without boundary), and thus the number of edges in this mesh is approximately $e \approx 3f/2$. We can then substitute this value for e into the Euler-Poincaré formula in (1.1), to obtain:

$$f + v - 3f/2 \approx 2 - 2g - \delta. \tag{1.2}$$

Rearranging (1.2) to solve for v, we get:

$$v \approx f/2 + 2 - 2g - \delta. \tag{1.3}$$

Since $f/2$ is much larger than $2-2g-\delta$, we can simplify (1.3) to obtain $v \approx f/2$. This tells us that a typical triangle mesh has approximately twice as many faces as vertices. Furthermore, since we have approximated the number of edges as $e \approx 3f/2$ and we also have $v \approx f/2$, this gives us the approximate relationship between the number of edges and the number of vertices as being $e \approx 3v$. From Euler's *degree-sum formula* for graphs [8], we can conclude that the sum of the degrees of all the vertices in a mesh is equal to twice the number of edges in that mesh; thus we have:

$$\sum degrees = 2e \approx 6v. \tag{1.4}$$

This tells us that in a typical triangle mesh, the average vertex *valence* (see Section 1.2.4) is 6, which means that, on average, one vertex will be shared by 6 different faces.

1.2.7 Mesh data structures

Several kinds of data structures [9] can be used to represent meshes. Some, like the *Indexed Face Set*, are more suitable for rendering, whereas some others such as the *Half Edge*, the *Corner Table*, or the *Adjacency Matrix*, are more adapted for geometry processing. A review of these data structures can be found in [9], and Kettner also discusses the different Half Edge variants in [10]. The Half Edge and the Corner Table consume more storage space than the Indexed Face Set, since they store more data to permit efficient traversals of the mesh graph, such as requesting one-ring

neighbors of a vertex and other similar operations. Note that both the Half Edge and Corner Table cannot represent non-manifold meshes. The Adjacency Matrix is quite a compact representation, but it does not encode the entire mesh representation and is generally used as an intermediate data structure. Many common file formats for 3D mesh models, such as VRML/X3D [11], OBJ [12], PLY [13], etc., rely on the Indexed Face Set representation.

We know from Section 1.2.6 that in a typical triangle mesh with v vertices and f faces, we can approximate $v \approx f/2$. Thus, such a mesh stored as an explicit list of independent triangles, where each triangle is described by the three 32-bit floating-point coordinates for its vertices, requires about $2 \times 3 \times 96v = 576v$ bits. Furthermore, this representation provides no explicit information regarding the adjacency between neighboring triangles or vertices. An Indexed Face Set representation theoretically requires $\log_2(v)$ bits to encode each vertex index, and since we know from Section 1.2.6 that in a typical triangle mesh each vertex will be referenced by 6 different faces on average, this means that around $6v \log_2(v)$ bits will be required for the connectivity data in total. This must be accompanied by a vertex table, which requires a further $96v$ bits, if 32-bit floats are used. As stated in [14], the Corner Table requires about $12v \log 2(v)$ bits and must be accompanied (as for the Indexed Face Set) by a vertex table, which requires $96v$ bits. However, it was found in [15] that for large, arbitrary surface meshes (e.g., $v > 1000$), the theoretical upper bound for the number of bits required to describe the mesh connectivity is around $3.24v$.

Therefore, the Indexed Face Set representation, and the other representations mentioned above, are far from optimal in terms of mesh compression. For this reason, many different mesh compression algorithms have been proposed in the literature to date, which attempt to encode the mesh connectivity and geometry in a more compact manner; some of these compression techniques will be mentioned in the following sections.

1.3 Static meshes

The topic of static mesh compression has been extensively studied in the past few decades [7,16,17], and several static mesh compression implementations are freely available [18–21]. Since many of the AM and TVM compression methods (see Sections 1.4 and 1.5) use static mesh compression approaches in parts of their algorithms, in the current section we will briefly review some of the most well-known static mesh compression techniques. We will categorize our review into three sub-sections: connectivity compression, vertex (geometry) coding, and standards and software.

1.3.1 Connectivity Compression

Connectivity compression methods are usually divided into two categories: *single-rate* and *progressive* compression. *Single rate* methods were traditionally designed to reduce the storage or transmission load between a CPU and a graphics card. These methods encode the entire mesh model (its geometry, connectivity, and attributes) as

a whole, so the graphics card cannot render the reconstructed model before the entire bitstream of mesh data has been wholly received. Meanwhile, *progressive* methods are used when a highly detailed mesh is transmitted over bandwidth-restricted channels or consumed by devices with different decoding and rendering capabilities. These methods allow a 3D mesh to be decomposed into a number of different "resolution" or quality levels, so that it can be reconstructed incrementally from coarse to fine levels of detail, or from worse to better quality, by the decoder. In general, *single-rate* compression methods perform a traversal of the mesh elements and identify configurations that can be easily encoded, while *progressive* compression methods use mesh simplification tactics to achieve the necessary hierarchical representation.

An efficient way to represent the list of triangles in a mesh is to arrange them into *triangle strips* or *triangle fans*. Instead of three indices per triangle, a triangle strip sends a sequence of triangles connected by one edge and can describe n triangles using $n - 2$ indices. In the case of a triangle fan, a series of triangles share a single vertex, and similarly n triangles can be described using $n - 2$ indices. Due to their efficiency and simplicity, triangle strips and triangle fans are primitives often used in computer graphics description languages like openGL [22] and DirectX [23]. Deering [24], Chow [25], and Bajaj et al. [26] are examples of *single-rate* compression algorithms based on triangle strips and triangle fans.

Meshes can also be represented by graphs such as tree structures. For instance, in a *vertex spanning tree*, the nodes represent the mesh vertices, while the branches represent which vertices are connected to each other. In the case of a *face spanning tree*, the nodes correspond to a triangle, and the connected nodes indicate which triangles share an edge. These concepts were used by Taubin and Rossingac [27] in their *Topological Surgery* algorithm, and by Diaz-Gutierrez et al. [28] in their *Hand-and-Glove* algorithm.

Some methods use a border line to divide the mesh into two parts: an inner part and an outer part. At first, the face of a single triangle defines the inner part, while all the other triangles belong to the outer part. Then triangles are iteratively assimilated by the inner part in a *region-growing* fashion. The growing operations are described by symbols. Once the mesh has been entirely covered by the growing region, processing the symbols in reverse order at the decoding stage reconstructs the connectivity of the mesh. When the region growing operation takes into account the vertices and their respective valences, it is called *valence encoding*, and if the region growing operation takes into account the neighboring triangles, it is known as a *triangle conquest approach*.

The pioneering valence encoding algorithm from Touma and Gotsman [29] is considered to be one of the most efficient *single-rate* mesh compression algorithms, but it is only suitable for oriented manifold 3D meshes. Mamou et al. [30] proposed TFAN, a triangle fan-based compression method that encodes non-manifold and non-oriented triangle meshes by partitioning the mesh into a set of triangle fans. Then, TFAN encodes the configuration of triangle fans using symbols that describe 10 different arrangements and the degree of vertices. In the case of triangle conquest approaches, the *Cut-Border machine* [31] and *Edgebreaker* [32] are examples of

coding algorithms that use symbols to indicate the presence of neighboring triangles and also grow the number of traversed triangles. In the *Edgebreaker* case, using the valence of the vertices to specify a different entropy context improves the encoding of the symbols [33]. Furthermore, efficient implementations can use data structures such as the Corner-Table [34].

Usually, *single-rate* compression methods preserve the connectivity of the mesh. But when the lossless criteria is not mandatory, algorithms can use *remeshing* techniques to improve compression performance. The new connectivity is often *regular* and *uniform*, and can therefore achieve higher compression gains, as exemplified in the proposals from Szymczak et al. [35] and Attene et al. [36], among others.

In *progressive* compression methods, a *simplification* operation iteratively modifies the original mesh. The *base mesh*, which results from all the accumulated simplification operations, is then used together with the simplification operations to describe (reconstruct) the original mesh in a progressive manner. By using a reduced set of simplification operations, the resulting coarser mesh can also be used to render a lossy representation of the original mesh. One example of a mesh simplification is the *vertex split* and the *edge collapse* operations used by Hoppe [37] in his *Progressive Mesh* method. It is also possible to group the vertex split operations together to generate more efficient intermediate representations of the *progressive meshes*, as was done by Taubin et al. [38] in their *Progressive Forest Split* proposal.

Another method for simplifying a triangular mesh and generating a hierarchical representation is to use the *vertex decimation* technique, first introduced by Schroeder et al. [39]. The vertex decimation approach simplifies a mesh by removing a vertex and all its adjacent edges, and then retriangulating the resulting hole with fewer triangles than were originally present in that location. Cohen-Or et al. [40] also used the vertex decimation approach in their patch coloring algorithm for progressive mesh compression.

1.3.2 Vertex (Geometry) Compression

The mesh vertex coordinates (x, y, z) are usually represented, by default, as IEEE 32-bit floating-point values, but lower resolutions can be used without any visual impairment (for a human observer). A common approach for geometry compression is to first quantize the floating-point values to integers, whereby the quantization resolution is typically 8- to 16-bit. Even though *vector quantization* and *non-uniform quantization* have been used and demonstrated superior performance [5,16], most existing algorithms simply apply *scalar uniform quantization* due to its simplicity and generally adequate performance. Uniform scalar quantization is equivalent to creating a 3D grid inside the bounding box of the mesh, and then snapping the vertices to the nearest grid position, in order to convert the vertex positions to integer values within a chosen integer range.

Following quantization, many methods apply *prediction* strategies to reduce the entropy of the quantized signal, which can then be losslessly encoded with an *entropy encoder* such as Huffman or an arithmetic encoder [41]. In the case of *single-rate*

mesh compression, the mesh traversal produced by coding the connectivity usually influences the prediction, since it imposes a decoding order for the vertices and determines the available reconstructed values. Usually, a combination of previously decoded vertices predicts the current position. *Delta* prediction [24,25] uses the difference between the current and latest decoded vertex in a Differential Pulse-Code Modulation (DPCM) fashion. The K previous coefficients of a vertex spanning tree can be used for *Linear* prediction [27] of the next vertex position. One of the most popular methods is the *Parallelogram* prediction [29]. The latter predicts by using the vertices connected to the predicted position and the vertex opposite to that position - that is, the vertex from the triangle that shares the edge. The prediction is the vertex of a parallelogram formed with the three vertices, which are assumed to be coplanar. To overcome the limitation of coplanar vertices, other methods propose variations of the *Parallelogram* prediction by analyzing the angles between triangles [42,43].

Apart from the somewhat standard geometry compression framework of quantization, prediction, and entropy coding, other researchers have proposed alternative methods for geometry compression, which consider more the mesh shape. For example, Karni and Gotsman [44] used the concepts of a graph representation of a mesh, and the Laplacian matrix, for mesh geometry compression. For a mesh with n vertices, the Laplacian matrix (sometimes referred to as the *Tutte* Laplacian) is a $n \times n$ matrix with ones on its main diagonal and $-1/deg(v_i)$ in the positions (i, j), where $deg(v_i)$ is the degree of a vertex v_i adjacent to a vertex v_j. The eigenvalues of this matrix can be used as basis functions to obtain a spectral decomposition of the mesh surface, and can thereby be used to compress the mesh geometry. Karni and Gotsman [44] used this principle to propose a *progressive* mesh compression approach for the geometry component. The Laplacian matrix can be generated with the connectivity only, then coefficients resulting from projecting the geometry information onto the eigenvectors of the Laplacian matrix can be sent in a progressive manner (from high to low magnitude) to obtain a progressively better mesh shape reconstruction.

Based on a similar principle of sending coefficients of a transform, Gu et al. [45] proposed *Geometry Images*, which maps the mesh surface to a regular squared image and uses wavelets to compress and transmit the image. This prioritizes the encoding of the mesh geometry, but the connectivity is converted to a semi-regular mesh. Other algorithms that prioritize geometry over connectivity in a progressive manner use tree structures to create hierarchical representations of the vertices. For instance, Devillers and Gandoin [46] use the *kd-tree* to encode the mesh geometry, while Peng and Kuo [47] use an *octree* instead. Both approaches are able to reconstruct the original mesh's connectivity.

Other mesh properties, like normals and texture coordinates, are usually represented using an array of floating-point values, and similarly to vertex positions, can be encoded by using quantization, prediction and entropy coding. Since in most cases the connectivity and reconstructed vertex positions are available before compression of other mesh attributes, they can be used when compressing these other mesh properties. For the *Parallelogram* prediction of texture coordinates, Isenburg

and Snoeyink [48] proposed four different rules considering the presence of texture discontinuities. They can be identified in the mesh connectivity by noticing that one edge for geometry transforms into two edges for the texture connectivity (also known as a *crease* edge). Váša and Brunet [49] also proposed a *Parallelogram* prediction modification, but in their case they explicitly use the geometry information to improve the prediction of UV coordinates.. Those are two examples of *single-rate* approaches, but *progressive* methods for textured meshes have also been proposed, such as the method from Caillaud et al. [50], which creates the hierarchical representation of the mesh by taking into account the texture seams as well. For per-vertex color compression, Ahn et al. [51] noted that a simple first-order predictor for colors is enough, and they used mapping tables to encode RGB values. In the case of normals, the authors proposed to use a unit sphere representation divided into 6 parts, and to quantize each part into a 4×4 matrix, different from other schemes that usually use the octahedral representation [52] for normal compression. The quantized normals are then predicted by using an average of the normals from vertices in three neighboring triangles.

1.3.3 Standards and Software

The Motion Picture Experts Group (MPEG) is well-known for producing international ISO standards for video and image compression [53]. In the early 2000's, MPEG published the MPEG-4 standard [54], aimed at compressing multimedia audio-visual scenes, including interactivity with different kinds of multimedia objects, such as synthetic textured meshes. This standard includes static 3D mesh coding tools, like the *Topological Surgery* algorithm [27], and several progressive mesh compression tools, for instance the *Progressive Forest Split* [38], the *Wavelet Subdivision Surface* [55], *MeshGrid* [56], and *Footprint* [57].

Acknowledging the importance of a trade-off between compression and computational resources, especially with the proliferation of graphics cards in mobile systems, MPEG developed the Animation Framework eXtension (AFX), specified in Part 16 of MPEG 4 [58]. The Scalable Complexity 3D Mesh Compression (SC3DMC) toolset of AFX can choose among three 3D mesh coding techniques: Quantization-Based Compact Representation (QBCR), Shared Vertex Analysis (SVA), and Triangle FAN (TFAN). Connectivity is not compressed in QBCR, while SVA and TFAN apply the proposals from Jang et al. [59] and Mamou et al. [30]. Note that QBCR and SVA maintain the original order of vertices/faces, while TFAN reorders them (although one can code the mapping between input and compressed meshes). The geometry is quantized, predicted and entropy encoded in all three options. Six different predictions are possible (including the *Parallelogram* prediction [29]) and five different entropy coding modes may be selected as well. Normals, texture coordinates, color per vertex and generic attributes can also be encoded, whereby normals are converted to the octahedral representation [50] and texture coordinates are quantized according to the texture map dimension. An open-source and royalty-free implementation of the AFX standard [19] has been included in the glTF, the standard file format for

three-dimensional scenes and models from the Khronos group [60].

The glTF standard has also included an extension for mesh compression based on Draco [18], the point cloud and mesh compression tool from Google. Draco has three modes for connectivity compression: a sequential encoder, which encodes the indices directly, the efficient *Edgebreaker* algorithm [34], and also an *Edgebreaker Valence Encoding* method [33]. Draco then reorders the vertices and encodes their attributes (geometry, texture coordinates, normals, etc.) following the traditional quantizion, prediction, and entropy encoding steps. For the prediction of vertex positions, Draco can either use the difference from the last decoded position, the *Parallelogram* predictor [29], or the *Multi-Parallelogram* predictor, which uses all the triangular faces opposite to the predictor vertex. For texture coordinates, Draco also includes the *Constrained Multi-Parallelogram* predictor, which explicitly selects which parallelograms to use for prediction by marking the *crease* edges between triangles. For normal coding, Draco uses the octahedral representation [50], and performs prediction by weighting the normals from neighboring triangles by the triangles' areas.

1.4 Constant-connectivity mesh sequences

A so-called *animated mesh* is a sequence of static meshes, where each frame represents the dynamic mesh at a given point in time. The connectivity, topology, and colours of the animated mesh remain constant across all the frames; it is only the geometry (vertex positions) that changes over time. As mentioned in the Introduction of this chapter, such animated meshes are usually computer-generated (not obtained from real-life captures) and are most commonly used in the gaming and film (VFX) industries, but also for medical and scientific visualizations. However, animated meshes are not suited to volumetric video coding, since they do not support varying photometry or topology over time. Nevertheless, we will see in Section 1.4.1 that it is possible to produce or transform some *variable-connectivity mesh sequences* (see section 1.5) into sub-sequences of meshes with constant connectivity. Once obtained, each sub-sequence (or group of frames) can then be encoded using *animated mesh* compression techniques.

In the current section, we first give a quick overview of possible solutions to obtain *constant-connectivity mesh sequences* from *variable-connectivity mesh sequences*. We then provide a summary of work to date that proposes compression algorithms for constant-connectivity mesh sequences. We will base our presentation of animated mesh compression techniques on the work in [16] from 2015, and extend their taxonomy with more recent publications, using the following categories (see Figure 1.1): segmentation, Principal Component Analysis (PCA), spatio-temporal prediction, wavelets, surface unfolding, and spectral analysis.

1.4.1 Tracking and re-meshing

At the production stage of real-life volumetric video captures, generating meshes independently for each frame of a mesh sequence produces meshes with variable connectivity. However, in order to leverage existing AM compression schemes, a consistent connectivity over frames is required. The process of transforming a *variable-connectivity* mesh sequence into a *constant-connectivity* mesh sequence is called mesh *tracking*.

As an example of variable-connectivity mesh sequence conversion into constant-connectivity sequences, Collet et al. [61] present a complete tool chain for acquisition, reconstruction, tracking and compression. After reconstruction, they obtain a set of frames of variable connectivity. They first subdivide the sequences into sub-sequences of similar topology (recall that different connectivities can define a similar topology). To do so, they compute which frames are the most promising to be used as *keyframes* (i.e., frames whose meshes are well representative of their neighbor frames). They search for frames of higher surface area, of lower *genus* (see Section 1.2.3), and with a higher number of *connected components* (see Section 1.2.5). Once keyframes are found, they use the state-of-the-art non-rigid *Iterative Closest Point* (ICP) algorithm of Li et al. [62] to perform mesh registration, though other approaches [63,64] can be adapted as well. Following this re-meshing, the frames of each sub-sequence have the same connectivity and a stable texture *uv* atlas (i.e., a temporally consistent parameterization). The sub-sequence can then be coded by any AM coding method (in [62] the authors use their own, prediction-based solution). Furthermore, the frame texture atlases are stable over the entire sub-sequence, which leads to higher quality for fixed compression bitrates of the texture stream by using MPEG H.264 or other standard 2D video coding schemes. In [65] Prada et al. extend this tracking method using local re-meshing, to permit tracking over long sequences containing significant deformations or topological changes, which further enhances texture atlas stability. Extracting a spatio-temporally coherent mesh out of a 4D capture is an active area of research, and one can refer to [61–66] for further references.

It should be noted that tracking is a time-consuming process that implies some re-meshing and some texture re-projections, which in turn implies inevitable distortions from the original, non-tracked animated sequence. Thus, tracking is better performed at production time during the reconstruction stage, where all the parameters are controlled, rather than at the compression stage. The tracking solution is thus not very practical in the case of a standalone TVM coder. We will see in Section 1.5 other approaches to encode TVMs that do not rely on tracking. But first, let us review the existing AM coding approaches.

1.4.2 Methods based on segmentation

The existing segmentation-based approaches for animated mesh compression consist in partitioning the vertices of the dynamic mesh into groups of vertices, called *clusters*, where each cluster represents a section of the dynamic mesh that has similar movement

over time. In 1999, Lengyel proposed the first animated mesh compression algorithm (of all the categories) based on this approach [67]. The clusters in [67] are defined with respect to a reference frame and determined using a heuristic where a set of seed triangles is chosen at random. The movement of each cluster, estimated by a rigid transformation, is used as a predictor to extrapolate the positions of vertices in the current frame from the reference one. The animation is then encoded using the set of motion parameters for each cluster, as well as the prediction errors associated with each vertex.

Gupta et al. [68] improved the solution in [67] by using the Iterative Closest Point (ICP) algorithm to compute the displacement of the vertices. In their solution, the initial segmentation is based on a topology partitioning algorithm and is then refined based on some motion coherency criteria. The authors presents a compression ratio of 45:1, compared to 27:1 for the Lengyel solution on an animated chicken model.

Later, Collins et al. [69] improved Lengyel's solution by using only rigid transforms without the encoding of residual errors. The authors in [69] introduce a new distortion bound segmentation algorithm based on a weighted least-squares approach that minimizes the number of generated clusters according to the distortion criterion. This solution, however, produces geometric seams between patches, and requires a low-pass filtering of vertex displacements that is performed as a post-processing to attenuate the artifacts.

Sattler et al. [70] also proposed an improved segmentation using clustered vertex trajectories, by integrating a combination of Lloyd's algorithm (also known as Voronoi iteration) and *Principal Component Analysis* (PCA). Each generated cluster is compressed independently using PCA.

Amjoun et al. [71] partitioned mesh vertices into clusters by applying *k-means* clustering [72], where vertex motions can be described by unique 3D affine transforms. The resulting clusters are then encoded using PCA. The algorithm segments the animated mesh into clusters by using a region-growing algorithm, and transforms the original vertex coordinates into the local coordinate frame of their segment. However, the results are seriously dependent on, and affected by, the choice of initial seed vertices.

Mamou et al. [73] also consider a segmentation into almost rigid parts, by performing a hierarchical decimation, which privileges the simplification of neighboring vertices with similar affine motion. The motion of each vertex is then expressed as a weighted linear combination of the cluster motions using a skinning approach adapted from skeletal animation techniques. Motion compensation errors are finally compressed using the *Discrete Cosine Transform* (DCT), which makes the stream spatially scalable if the DCT coefficients are ordered. This approach was later combined with [74] to define the MPEG FAMC [75] standard (see Section 1.4.8).

In [76], Luo et al. make extensive use of a spatio-temporal approach. They first compute an initial temporal cut on the input mesh sequence to obtain a small subsequence by detecting the temporal boundary of dynamic behavior. Then, they apply a two-stage vertex clustering on the resulting sub-sequence to classify the vertices into groups with optimal intra-affinities. After that, they perform a temporal segmentation

step based on the variations of the principal components within each vertex group. The obtained sub-sequences of clusters are compressed using PCA. They finally perform a lossless compression of the PCA bases and coefficients using ZLib. Their solution generates geometric artifacts at the cluster boundaries, so they generate clusters of a larger size to overlap sibling clusters. They present results ranging from 0.63 to 7 bits per vertex per frame (bpvf) on a range of test models.

1.4.3 Methods based on Principal Component Analysis

Principal Component Analysis (PCA) methods find a new orthogonal basis to describe the motion of an animated mesh, and achieve compression by using a reduced set of basis eigenvectors. The frames of a mesh sequence are then represented by coefficients obtained from projecting the mesh onto the new (reduced) basis. PCA can be used in two different ways: either by exploiting the temporal correlation of frames and finding the average shape of the sequence (*Eigenshapes*), or by exploiting the spatial correlation of the vertices' trajectories and finding the average trajectory (*Eigentrajectories*).

Alexa and Muller [77] proposed the first method that applied PCA to obtain the *Eigenshapes*. In [77], the vertices of the mesh are arranged in a matrix of size $3v \times f$, where v is the number of vertices and f is the number of frames. Note that the vertex positions are not the coordinate positions in the corresponding frame, but actually the residual positions after global motion compensation, in order to decouple the elastic and rigid motion components. By decomposing the matrix using *Singular Value Decomposition* (SVD), the first matrix contains the eigenshapes of the sequence, the second matrix contains the eigenvalues, and the third one contains the coefficients of the frames in the new basis formed by the eigenvectors (i.e., eigenshapes). The animation is compressed by sending the basis vectors for the animation, and the coefficients and global motion estimation per frame. To improve the coding of the coefficients, Karni and Gotsmann [78] proposed a linear predictor based on the *Parallelogram* predictor. To better adapt the PCA solution to the motion, Sattler et al. [70] proposed spatial clustering of vertices, while Luo et al. [79] proposed temporal clustering of the frames. Amjoun and Straßer [80] also used spatial clustering of vertices, but additionally proposed a rate-distortion allocation by having more eigenvectors at clusters that underwent extreme deformations. A *progressive* animated compression using PCA was also proposed by Kao et al. [81]. With the latter method, the decoder can choose any combination of mesh or motion resolution.

The challenge with PCA methods is that the dimension of the auto-correlation matrix used in the SVD decomposition is dependent on the number of vertices, which can be quite large, usually compared to the sequence of frames that is commonly a couple of seconds long. Furthermore, the PCA eigenvectors need to be precisely encoded and there is little correlation between them. Therefore, Váša and Skala [82] proposed CODDYAC. This codec uses PCA in the vertex trajectories instead of their shape - that is, the vertices are arranged in a matrix of size $3f \times v$ instead, which leads to smaller auto-correlation matrices and smaller eigenvectors. The coefficients

are then traversed using the *Edgebreaker* algorithm, and encoded using the *Parallelogram* prediction. Subsequent publications from the same authors [83,84] improved the performance of CODDYAC by compressing the Eigentrajectories using motion models and the coefficient predictors by using local neighborhoods. Váša [85] further optimized the mesh traversal, and with that achieved one of the best performances for CODDYAC. Next, Váša et al. [86,87] used geometric Laplacian and mesh averaging techniques to improve the original CODDYAC performance, as judged by a perceptual metric.

Even though the compression performance of PCA methods is better than other proposed methods for animated meshes, these are global methods that use *all* the frames to compute the optimal basis. Therefore, it is challenging to use PCA in streaming applications. Furthermore, the computation cost of calculating the SVD can be prohibitive as well, for large (dense) mesh models or long sequences. To reduce the computation time of the SVD calculation for both Eigentrajectories and Eigenshapes, Lalos et al. [88] proposed an efficient method to update the SVD using adaptive orthogonal iterations. They also use intervals of 10 frames, which can be used in low-delay applications, but the compression performance is reduced.

1.4.4 Methods based on spatio-temporal prediction

Due to the fact that the topology of animated mesh sequences is constant over time, the animated vertices generally exhibit strong redundancies and correlations between frames. Differently from global PCA-based methods presented in the previous subsection, prediction methods for animated mesh compression exploit local coherences and are thus computationally efficient and more suited for real-time streaming. These methods extend the spatial prediction approaches, such as the parallelogram ones [29,43], to exploit these temporal properties, by either interpolating between spatial or temporal surrounding positions, or by extrapolating from previous frames. Ibarria et al. [89] present two spatio-temporal predictors in their Dynapack framework: Extended Lorenzo Predictor (ERP) and REPLICA. ERP directly extends the parallelogram method originally introduced by [29] for static mesh compression, while REPLICA extends ERP to make it more robust to rotation and scale transformations.

In [90,91] Zhang proposes an alternative approach based on a segmentation method using an octree based motion representation for each frame. Two consecutive frames are used to generate a small set of motion vectors that represent the motion from the first frame to the other. Quantization and an adaptive arithmetic coder are used to achieve further data reduction. An optimized version of this approach was introduced in [92] by Müller et al. In the solution in [92], called Dynamic 3D Mesh Coder (D3DMC), the authors extract only one representative for a cluster of difference vectors, which provides a significant reduction in the data rate. A *context-adaptive binary arithmetic coder* (CABAC) [93] is finally used to code the representative of the clusters, which have been previously scaled and quantized. Müller et al. [94] later refined their solution with a rate-distortion approach.

Amjoun and Straßer [95] encode delta vectors in local coordinate systems. How-

ever, the encoding performance is strongly dependent of the seeds selected for the surface segmentation that they perform.

Stefanoski et al. [74] introduce the decomposition into layers, which are basically mesh levels-of-detail. This approach has become a foundation for several other proposed methods. It was refined by Stefanoski et al. with scalable predicitive coding (SPC) in [96], which was the first solution to provide spatio-temporal scalability, and provides an excellent compression ratio as well (between 3.5 and 8 bpfv). The solution in [96] was then further extended by Bici and Akar [97] through novel prediction approaches. Finally, Ahn et al. [98] managed to obtain a 30% gain in performance (compression ratio between 2 and 6 bpvf) compared with SPC, still using the same layered approach.

1.4.5 Methods using wavelets

Wavelet-based mesh compression methods first became popular with the introduction of *subdivision wavelets* by Lounsbery et al. in the mid- to late-1990s [99,100]. In these seminal papers, the authors established a theoretical basis for extending the concept of *multiresolution analysis* (specifically, wavelets) to surfaces of arbitrary topological type, by defining a wavelet-like basis for a mesh surface using the subdivision rules from Loop [101]. The main idea behind subdivision wavelets is to decompose a high-resolution input mesh into a very coarse representation called a *base mesh*, and a set of detail coefficients termed *wavelet coefficients*. At the decoder, the wavelet coefficients can then be used to progressively refine the base mesh at multiple levels of detail (resolution). Because the connectivity of the mesh can be refined in a predictable manner using a set of standard subdivision rules known to both the encoder and decoder, the only connectivity data that needs to be transmitted is the connectivity of the base mesh, which is usually negligible. However, the main restriction with this approach is that the input mesh must have a *semi-regular* connectivity (see Section 1.2.4). This means that the connectivity of the input mesh must be able to be achieved by repeatedly subdividing each face of a coarse base mesh into 4 sub-faces until the resolution of the input mesh is obtained.

Many wavelet-based mesh compression algorithms that exist today are based on a similar principle as the original subdivision wavelets algorithm [99,100]. Since with this technique the mesh connectivity does not need to be encoded separately at each resolution level, the priority of such compression algorithms is on the coding of the mesh *geometry*. Geometry compression is usually achieved by either discarding small, negligible wavelet coefficients at various resolution levels, and/or by quantizing and entropy coding the wavelet coefficients that are chosen for transmission to the decoder. While subdivision wavelets were originally used for *static* mesh compression [55,99, 100,102], they have also been applied to animated mesh compression [103,104]. In both [103] and [104], the hierarchical mesh subdivision is performed on the first frame of the sequence, and then hierarchical motion estimation is used to map the same topology to the other frames. As well as decomposing each frame of the mesh sequence into a base mesh and wavelet coefficients, the wavelet subdivision is applied

temporally, along the motion trajectories. The wavelet coefficients are encoded using SPIHT [55].

Although semi-regular remeshing for the subdivision wavelet transform has the obvious advantage that the mesh connectivity data needs no encoding (apart from the base mesh connectivity), in some applications it might be important to preserve the original mesh connectivity. For this reason, several researchers have proposed methods for compressing constant-connectivity animated mesh sequences that have an *irregular* connectivity, without requiring a prior remeshing to a semi-regular connectivity, e.g., [105,106]. In [105], a more compact version of the multiresolution representation presented in [107] (which is based on the *non-uniform* subdivision method of [108]) is introduced. The wavelet coefficients in [105] are computed by using the mesh *geometry* rather than just the mesh connectivity as in earlier subdivision wavelet schemes. More specifically, the wavelet coefficients are computed based on the geometry of a *parametric* mesh. The first frame of the animated mesh sequence is used as a parametric mesh, and all the other frames are transformed with wavelet coefficients computed from this parametric frame. The parametric frame is encoded separately using a static mesh compression technique. The method in [106] is similar to [105], but it allows lossless compression as well as lossy.

More recent approaches for animated mesh compression that make use of wavelets consider wavelets defined on *graphs* [109–111]. In [109], *Graph Wavelet Filter Banks* (GWFB) [112] are used to compress the geometry and colour of animated mesh sequences representing moving human bodies. Both [110] and [111] make use of *Spectral Graph Wavelet Transforms* (SGWT) [113].

1.4.6 Methods based on surface unfolding

Inspired by Gu's *Geometry Images* [45], Briceño extends the principle of unfolding the mesh into an image to animated sequences [114]. The solution is called *Geometry Videos*. In this solution, the surface is cut and unfolded using stretch minimization [115]. It is first re-sampled and re-organized so that it becomes highly compressible. A strong advantage of the solution is that it can leverage classical 2D video compression techniques (such as MPEG HEVC or VVC) to encode the geometry signal (images). Its drawback resides in the fact that the surface re-sampling and the regular re-meshing introduces some non-negligible distortions in the reconstructed model and the original mesh connectivity is not preserved (unless the mesh happens to have a regular connectivity already, which is rare).

In [116], Mamou proposes the use of Multi-Chart Geometry Video, which, similarly to a UV texture atlas, unwraps the surface into several components instead of only one as in [114]. It then leverages *rigid transforms* as described in [67], to enhance compression, but using a fixed number of patches. The prediction errors are represented as Multi-Chart Geometry Images that can be encoded using standard 2D video encoders. The proposed solution preserves the original topology of the mesh, thus reducing distortions. The use of a piecewise affine predictor leads to better compression than [114]. Finally, the solution in [116] makes use of a low-distortion

atlas of parameterization [117], which leads to lower distortion than when using a simple mapping on a 2D square domain.

1.4.7 Methods based on spectral analysis

Karni and Gotsman [44] have shown that spectral methods can be applied to static mesh compression by using the combinatorial graph Laplacian matrix to extract from the mesh's connectivity a basis to encode the geometry. The Laplacian matrix is formed by considering only a one-ring neighborhood of the vertices, which leads to a sparse matrix whose eigenvectors can be easily extracted. However, this dependence on the mesh's connectivity makes the eingenvectors dependent on the quality of the meshing, which may vary for objects even with the same topology. In order to circumvent this problem, Vallet and Lévy [118] proposed the *Manifold Harmonic Basis* (MHB), derived as eingenvectors from the Laplace-Beltrami operator. The proposed framework is independent of the meshing and generates an orthogonal basis that can be used in several different applications, from mesh filtering to parameterization and even compression.

In [119], Wang et al. use spectral analysis to compress animated meshes by projecting on an MHB the field of deformation gradients defined on the surface mesh. The deformation gradient (that is, how the vertices of each triangle rotate and stretch from one frame to another) can be represented by a second-order tensor, or a 3×3 matrix, which can be decomposed into two matrices (rotation and stretching) using polar decomposition. Both matrices are then transformed into coefficients by projecting them onto the MHB. The coefficients are then quantized and arithmetically encoded. Note that the MHB is defined per vertex, while the deformation gradient is defined per triangle, so the authors use the average of the functions' values on the vertices.

Another approach that uses spectral analysis for the compression of animated meshes is from Chen et al. [120]. In their proposal, the mesh sequence is divided into clusters of frames with similar pose by using *K-medoids*. Then, for each cluster, the representative frame (i.e. the *keyframe*) is encoded using a static mesh compression techhnique and transformed into the MHB coefficients. The other frames are projected onto the new basis defined by the keyframe, and the MHB coefficients are encoded with *Linear Prediction Coding* (LPC), which generate values that are then quantized and entropy encoded. At the decoder side, the coefficients are inverse-transformed to generate a low-resolution representation of the non-keyframe. Then the deformation of the low-resolution keyframe to the full-resolution keyframe is determined, and this deformation is transferred from the keyframe to the non-keyframe to recover the high-frequency details.

Both [119] and [120] compare their approaches with PCA-based method CODDYAC [82]. For higher bitrates, spectral analysis has a better performance, since it does not need to send the eingenvectors (they can be derived from the stored keyframes). Furthermore, it preserves better the shape of the mesh even if its geometry (vertex positions) is not reconstructed exactly, which is believed to have a higher

impact in terms of perceptual quality. However, for lower bitrates PCA has a better performance, since it is able to achieve a better quality reconstruction than MHB when using the same number of basis vectors. Moreover, frames with sharp protrusions also require a significant number of MHB coefficients for better representation.

1.4.8 The MPEG framework

The first standard to encode animations by the MPEG group, the Face and Body Animation (FBA) standard [53], targeted human avatars, but was limited to fixed feature points, like eyes and mouth corners. This standard was later extended to a more generic framework with the Bone-Based Animation (BBA) [58], which includes geometric transform of bones (used in skinning-based animation) and weights (used in morphing animations). For a more generic animation, MPEG first issued the Interpolator Compression (IC) [121], which is used to compress *key frame animations*, defined by a pair of *keys* indicating the frame index, and *values* indicating, for instance, new positions of vertices. In 2009, MPEG added the Frame-based Animation Mesh Compression (FAMC) to the AFX set of animation compression tools. FAMC does not depend on how the animation is obtained (deformation or rigid motion) and compresses an animated mesh by encoding on a time basis the attributes (positions, normal vectors, etc.) of the mesh's vertices. It encodes the first frame with any static mesh compression algorithm, then applies skinning-based motion compensation [73] and layered decomposition [74]. The skinning-based motion compensation is composed of the following steps: global motion encoding (using the barycenter of the meshes to remove the global motion), then vertex partitioning (separating the vertices into clusters using the k-means method), followed by weighted motion compensation (obtaining the prediction by a weighted combination of the K affine transforms for each cluster). The motion-compensated residue is then transformed (using the DCT or Lifting transform) and once again the coefficients are predicted by neighboring frames, but using the layered representation [75]. Improvements over the FAMC standard have also been reported in [96,122].

1.5 Variable-connectivity mesh sequences

In contrast to the animated meshes discussed in Section 1.4, *time-varying meshes* (or TVMs) do not have a fixed topology or connectivity across all frames. They are also likely to contain different numbers of vertices in different frames. While this can make TVMs easier to generate from real-world captures (e.g., from images or 3D scans of real-world objects [123,124], where the mesh for each frame can be generated independently of other frames), it makes the compression problem much more difficult than for animated meshes, as there are usually no explicit correspondences between the vertices or connections across different frames. Furthermore, in a TVM, meshes in successive frames are not necessarily *homeomorphic* (see Section 1.2.1), as the surface reconstruction may be different in different frames. There is also no guarantee that the meshes in a TVM sequence will be manifold, or that the manifold property will

continue across different frames.

In this section, we aim to provide an overview of the existing literature on time-varying mesh compression, followed by a discussion on the recent MPEG activities in this area. The problem of compressing time-varying meshes began to be addressed in the literature more than a decade ago (e.g., [125,126]), but it has not progressed much further since then. This is due to both the complexity of the problem and the fact that the production of TVM content has only begun to gain traction relatively recently as a result of improvements in volumetric capture systems and increasing interest in using real data captures instead of only computer-generated models (e.g., see [127]).

We categorize the existing literature on TVM compression by the following methods: mesh surface unfolding, subdivision of meshes into blocks, and video-based coding (solutions that leverage standard MPEG V-PCC encoders for compression).

1.5.1 Methods based on mesh surface unfolding

In some of the earliest work on TVM compression [125], the authors propose to cut open the 3D mesh, then flatten the surface by projecting it onto 2D images, similarly to the *geometry images* [45] and *geometry videos* [114] ideas proposed earlier. Conventional 2D video coding methods can then be applied to encode the geometry and associated textures in the 2D images. A notable difference between the method in [125] and geometry images [45] is that the cut path for unfolding the 3D mesh with geometry images is selected so that it passes through high-curvature areas in the mesh, while in [125] the cut passes where no significant texture information exists.

Similarly to [125], in [128] the authors also make use of the concept of geometry images [45] and geometry videos [114] to compress TVMs. More specifically, they use the extension of geometry videos proposed in [129], called *conformal geometry videos* (CGVs), which aim to more efficiently represent 3D articulated motion (e.g., human motion) than the traditional geometry videos [114]. As in [129], in [128] salient feature points for the mesh in each frame are first detected (e.g., head, feet, hands, etc.), and corresponding feature points are found in successive frames. Then the marked TVMs are mapped to the *polycube* [130] domain as in [129], the 3D polycubes are cut open, flattened, and reparameterized onto a regular rectangular 2D domain to obtain the CGV representation. Since the CGVs have a regular structure, the original mesh connectivity is not encoded and therefore cannot be reconstructed losslessly. To compress the CGVs, in [128] the mesh vertices in each frame (2D image) of the CGV are placed as column vectors in a matrix that contains one column vector per frame. Next, *low-rank approximation* (i.e., truncated *singular value decomposition* (SVD)) [131] is applied on this matrix of vertex positions (separately for the X, Y, and Z positions), and the resulting singular values are reshaped back into frames and are named *EigenGVs* by the authors. These EigenGVs are by their nature much more compact than the original CGVs, and they are further compressed in [128] by using a standard 2D video encoder such as H.264/AVC [132]. The results have been shown to significantly outperform the original geometry videos [114] method in terms of

rate-distortion performance and visual quality. Furthermore, the method in [128] naturally offers the possibility of a progressive mesh reconstruction, as the user can choose how many of the EigenGV frames to reconstruct.

1.5.2 Methods based on subdivision of meshes into blocks

In [126,133], the authors propose to extend the idea of 2D block matching from conventional 2D video coding, to 3D time-varying mesh coding. Matching blocks across frames are found by comparing the directions of the mean surface normal vectors (SNVs) in those blocks, and the surface normal vectors across the best-matching blocks represent the inter-frame motion vectors. The motion vectors are encoded by using a differential pulse code modulation (DPCM) to obtain predictions of motion vectors between adjacent blocks (which were found to be highly correlated for the mesh data used in [126,133]). Residual values are computed in matching blocks as the minimum sum of the differences of the vertex positions in those blocks. The residual values are decorrelated by using a 1D Discrete Cosine Transform (DCT), and the transform coefficients are uniformly quantized, truncated at the higher frequencies, and entropy coded using Huffman coding.

In [134], the 3D meshes across different frames are first registered (aligned), then the bounding box of the largest mesh in the sequence is subdivided into sub-blocks, which the authors call a *coarse-level quantization* operation. The binary occupancy information for these sub-blocks is encoded using run-length encoding (RLE). The binary occupancy bitstreams are further compared (using an exclusive-OR operation between bitstreams) across different frames to obtain the motion information, and the resulting difference vectors are further encoded using RLE. To compress the vertex positions inside each sub-block, uniform quantization is applied and all the vertices inside a sub-block are quantized to one representative point. The representative points are then also converted to a sequence of occupancy (1 or 0) bits on a regular grid and encoded using RLE. Then the runs that have the same lengths are considered "super-symbols" and further encoded using arithmetic coding. The method in [134] has been shown to outperform the authors' earlier work in [133], in terms of rate-distortion performance.

Contrary to the work in [133,134], where only the geometry data is considered for inter-frame coding, in [135] the authors propose inter- and intra-frame coding approaches that consider the geometry (vertex positions), the connectivity, the colour textures, and any other data that is attached to the vertices. In [135], each 3D mesh in the TVM sequence is first subdivided into patches of approximately equal surface area and small enough that they can be considered flat discs. Principal Component Analysis (PCA) is then applied to place the patches' centres of gravity at the origin of the world coordinate system and to adjust the orientation of the patches so as to align them. These centres of gravity and the rotation parameters for the patches need to be encoded and transmitted to the decoder. For intra-frame coding of the vertex positions, a *spectral compression* [44,136] method is applied since the vertex positions are highly correlated spatially. Only around 10-50% of the lowest-frequency

spectral coefficients are kept, with the other values being set to 0, and Huffman coding [41] is used to encode the quantized set of coefficients. To compress the colour data per vertex, the authors propose to use either a vector quantization (VQ) on the (R, G, B) colour vectors, or a simple scalar quantization, and not spectral compression, since the colour values are not usually strongly correlated spatially with other colour values. For connectivity compression, the authors employ existing high-performing static 3D mesh connectivity algorithms, such as Edgebreaker [32]. For the inter-frame coding, a patch matching approach is proposed to remove the temporal redundancies. Patch matching is achieved by finding the minimum sum of Euclidean distances between a patch in the target frame and all the patches in the reference frame. The residuals between the position and colour vectors in the target patch and their corresponding vectors in the matched reference patch are then encoded by a vector or scalar quantization (VQ has generally been found to perform better). Correspondence data between vertices in the target and reference frames must also be encoded, but compared to the bitrate for encoding this data in [133], in [135] the bitrate is very small. Compared to [133,134], for inter-frame geometry coding the method in [135] has been shown to offer significant rate-distortion improvements.

In [137], the authors present a study on a better bit allocation strategy for their method in [135]. For the mesh models used in [135], the authors demonstrate in [137] that for a good visual quality for the reconstructed 3D meshes, as many bits as possible should be assigned to the vertex positions, while for colour 8-10 bpv per frame seems sufficient. This is because the quality of the colour reconstruction is highly dependent on the quality of the geometry reconstruction, as the colour must be coded on top of lossy geometry. The study in [137] also considers the trade-off in the bit allocations between the target frames and reference frames for the method in [135]. The authors conclude that as many bits as possible should be allocated to the reference frames, while the target frames can be allocated a smaller number of bits (e.g., half the bitrate of the reference frames) to achieve the same overall visual quality for inter-frame coding.

More recently, Pavez and Chou [138] propose an alternative representation to surface meshes: a so-called *polygon soup*. They claim that such an unstructured set of triangles, where the triangles are not connected and can overlap, are better at describing surfaces from real-life captures than point clouds or meshes are on their own. *Polygon soups* can be seen as a trade-off between point-clouds and surface meshes. The authors use an octree encoding to represent the vertices of the reference frames (i.e., the keyframes). They achieve this by quantizing the vertices (i.e., by voxelization) and reordering the quantized vertices by using Morton Codes [139]. Using the Morton order, they apply the *Region-Adaptive Hierarchical Transform* [140] (RAHT), which is a sequence of orthonormal transforms applied to attribute data living on the leaves of an octree. The output transform coefficients are sorted by decreasing magnitude of weight, quantized by uniform scalar quantization, and entropy coded by using a *Run-Length-Golomb-Rice* [141] (RLGR) entropy coder. The authors claim that compared to static polygon clouds and a fortiori static point clouds, dynamic polygon clouds can improve color compression by up to 2-3 dB in fidelity, and can

improve geometry compression by up to a factor of 2-5 in bitrate. It should be noted that the method in [138] assumes that time-consistent dynamic polygon clouds can be constructed in real time, and therefore that the triangles of the polygon soup are consistent across frames so that there is a correspondence between colors and vertices within each *Group of Frames*.

1.5.3 Methods inspired by MPEG V-PCC

In more recent work on TVM compression, solutions that are based on the MPEG Video-based Point Cloud Coding (V-PCC) standard from the V3C framework [142] have begun to appear.

Faramarzi et al. [143] propose extending the V-PCC framework to encode dense meshes with per-vertex colour data. More specifically, they propose to use V-PCC to encode the mesh geometry and textures, and to encode the connectivity by using Edgebreaker [32] and TFAN [30]. Since the order of the reconstructed vertices produced by V-PCC is different to the order of the vertices in the input mesh before compression, but both Edgebreaker and TFAN rely on having the same vertex ordering at input and output so that they can traverse the mesh losslessly, the encoder also needs to encode and transmit the vertex reordering information. Due to this additional coding burden, the proposed solution performs much worse compared to when using Draco to encode the same mesh. The authors thus propose an alternative solution, where vertices and colours are linearly packed into standard video frames, so-called *raw patches* in V-PCC. With this second approach, some V-PCC coding steps such as patch generation, as well as geometry and attribute smoothing, are skipped; therefore, this solution does not leverage the regular patch packing for more efficient coding as proposed in V-PCC. In both the proposed frameworks, V-PCC + Edgebreaker is shown to have better compression performance than V-PCC + TFAN; however, Draco usually performs better (on average) than either of the proposed solutions using V-PCC. Moreover, the results in [143] have been demonstrated only for a single frame for each of the meshes in the chosen test set, so it is not yet clear how the proposed solutions might work for time-varying mesh *sequences*.

In [144], the author also proposes an extension of the MPEG V-PCC framework [142] to encode TVMs. Similarly to the case for 3D point clouds, in [144] each vertex position of a 3D mesh is projected onto a pixel position in a 2D patch image, but additionally the *surface* of the 3D mesh is also projected onto the 2D patch projection plane by using rasterization. This produces a dense image representing the mesh connectivity, which is suitable for video coding. Occupancy, geometry, and attributes are encoded as usual by V-PCC. Due to the subdivision into patches and the lossy compression of vertex positions (e.g., quantization), the reconstructed mesh may have gaps between patches. To fix this problem, the author uses an algorithm similar to the mesh zippering approach used in [145], where triangle vertices on the borders of neighbouring patches are merged. Note that for both [144] and [143], the input mesh vertex positions must be *voxelized* (converted to integers) before they can be processed by the proposed methods. For geometry compression, the method in [144] has been

shown to be outperformed by Draco in terms of rate-distortion performance, while for colour compression the performance is comparable (or in some cases a little better) to colour encoded using HEVC on top of a reconstructed geometry using Draco. The method in [144] has been designed to work both on meshes with per-vertex colours and textured meshes.

1.5.4 The MPEG V-Mesh Call for Proposals

The TVM compression methods inspired by MPEG V-PCC were part of Exploration Experiments that were realized during the standardization of the video-based point cloud compression standard, V-PCC [142]. These experiments applied the new point cloud compression standard to the coding of meshes, since both point clouds and meshes are commonly used in the case of VR/AR and volumetric video. It was noted that in order to encode the connectivity of the meshes, modifications to the new standard would be recommended. Due to the interest of several companies in the mesh data format, the MPEG group decided to issue a Call for Proposals [146] for TVMs in October 2021. The responses are due in April 2022 and the new standard is expected to be concluded in October 2023.

1.6 Conclusion and future directions

In this chapter, we first motivated the use of 3D triangular *mesh* models to represent animated three-dimensional subjects in the *volumetric videos* that are fast emerging as the newest form of multimedia. Unlike earlier animated mesh sequences, which were generated purely synthetically by computers, the emerging *volumetric videos* are produced from *real-life* captures of moving subjects or objects. With volumetric videos, as for all previous forms of multimedia (1D audio signals, 2D images, and 2D videos), naturally there is a need for efficient compression algorithms in order to enable such multimedia to be used in a practical manner. In this chapter, we provided an overview of the history of compression techniques that have been proposed over the past few decades for *static* (single-frame) 3D meshes, followed by compression algorithms for synthetically-generated *dynamic* mesh sequences (so-called *animated meshes*), and finally for dynamic mesh sequences produced from real-life captures (so-called *time-varying meshes*, or TVMs). While we have seen many similarities and patterns in the compression algorithms proposed across these different categories, it is clear that the time-varying meshes are still the most difficult to compress. Indeed, even though research on the compression of such time-varying meshes started more than a decade ago, it is still rather in its infancy. The most difficult problem in compressing such data is the lack of correspondences in the connectivity and geometry of the mesh models across different frames. Tracking algorithms that attempt to find such correspondences are often costly and time-consuming, and therefore not practical to use in the case of a standalone TVM codec. More recently, researchers have begun to propose compression algorithms for TVMs inspired by the recent MPEG V-PCC standard, but the results are still far from the compression rates achievable

for animated meshes. Perhaps the recent MPEG Call for Proposals (CfP) on TVM compression will inspire a flurry of new and creative research in this direction.

Aside from the inherent challenges of designing new compression algorithms, we also have a few other roadblocks that stand in the way of progress, most importantly: (i) the lack of a sufficiently large and variable set of TVM datasets to test on (few such datasets currently exist, and they are often proprietary to the companies or institutions that generate them and so can be difficult to acquire); (ii) the fact that different papers in the literature do not all present results on the same datasets, which makes it difficult to compare different compression algorithms; (iii) the frequent lack of sufficient algorithm details in the literature, and/or the lack of available source code for the proposed algorithms, which often makes it difficult or impossible to reproduce the results; and (iv) the lack of agreement on common, reliable error metrics that can be used for the quality assessment of decompressed mesh models and for the fair comparison of different mesh compression algorithms, and which consistently correlate well with human perception of error. In order to have a more accurate idea of where the research on compression of dynamic 3D meshes truly stands, and therefore to be able to make useful progress, we must also enforce more rigorous presentations of new algorithms that are published in the literature. Indeed, if we cannot accurately reproduce a method, or fairly compare different methods, or even accurately measure the performance of a new method, the usefulness of these methods is limited as we do not have full control over them.

Perhaps the work presented in this chapter will serve as inspiration for new insights and valuable contributions to the field of dynamic mesh compression, and will therefore enable us to reproduce increasingly richer multimedia representations of our world.

Bibliography

[1] M. Zwicker, H. Pfister, J. Van Baar, M. Gross, Surface splatting, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001, pp. 371–378.

[2] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, ACM siggraph computer graphics 21 (4) (1987) 163–169.

[3] L. Williams, Pyramidal parametrics, in: Proceedings of the 10th annual conference on Computer graphics and interactive techniques, 1983, pp. 1–11.

[4] G. Lavoué, 3D Object Processing - Basic Background in 3D Object Processing, John Wiley&Sons, Ltd, 2008, Ch. 1, pp. 5–43. doi:https://doi.org/10.1002/9780470510773.ch1.

[5] M. Krivokuća, PhD Thesis, Progressive Compression of 3D Mesh Geometry Using Sparse Approximations from Redundant Frame Dictionaries, Department of Electrical and Computer Engineering, The University of Auckland, New Zealand, 2015.

[6] M. Fiedler, Algebraic connectivity of graphs, Czechoslovak mathematical journal 23 (2) (1973) 298–305.

[7] J. Peng, C.-S. Kim, C.-C. J. Kuo, Technologies for 3d mesh compression: A survey, Journal of visual communication and image representation 16 (6) (2005) 688–733.

[8] S. Jendrol', H.-J. Voss, Light subgraphs of graphs embedded in the plane—a survey, Discrete Mathematics 313 (4) (2013) 406–421. doi:https://doi.org/10.1016/j.disc.2012.11.007.

[9] M. Ben-Chen, A. Lai Lin, Course on geometry processing algorithms - chapter 2 mesh data structures (2010).
URL https://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/02_Mesh_Data_Structures.pdf

[10] L. Kettner, Using generic programming for designing a data structure for polyhedral surfaces, Computational Geometry 13 (1) (1999) 65–90.

[11] W. Consortium, X3d and vrml, the most widely used 3d formats (2000).
URL https://www.web3d.org/x3d-vrml-most-widely-used-3d-formats

[12] P. Bourke, Obj - polygonal file format.
URL http://paulbourke.net/dataformats/obj/

[13] P. Bourke, Ply - polygonal file format.
URL http://paulbourke.net/dataformats/ply/

[14] J. Rossignac, A. Safonova, A. Szymczak, Edgebreaker on a corner table: A simple technique for representing and compressing triangulated surfaces, in: Hierarchical and geometrical methods in scientific visualization, Springer, 2003, pp. 41–50.

[15] W. T. Tutte, A census of planar triangulations, Canadian Journal of Mathematics 14 (1962) 21–38.

[16] A. Maglo, G. Lavoué, F. Dupont, C. Hudelot, 3d mesh compression: Survey, comparisons, and emerging trends, ACM Computing Surveys (CSUR) 47 (3) (2015) 1–41.

[17] P. Alliez, C. Gotsman, Recent advances in compression of 3d meshes, Advances in multiresolution for geometric modelling (2005) 3–26.

[18] Google, Draco 3d data compression.
URL https://google.github.io/draco/

[19] K. Mammou, Open 3d graphics compression (2013).
URL https://github.com/KhronosGroup/glTF/wiki/Open-3D-Graphics-Compression

[20] M. Geelnard, Openctm (2010).
URL http://openctm.sourceforge.net/

[21] V. Vidal, E. Lombardi, M. Tola, F. Dupont, G. Lavoué, Mepp2: a generic platform for processing 3d meshes and point clouds, in: EUROGRAPHICS 2020 (Short Paper), 2020.

[22] openGL, opengl primitives (2000).
URL https://www.khronos.org/opengl/wiki/Primitive#Triangle_primitives

[23] Microsoft, Directx primitives (2000).
URL https://docs.microsoft.com/en-us/windows/win32/direct3d9/primitives

[24] M. Deering, Geometry compression, in: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995, pp. 13–20.

[25] M. M. Chow, Optimized geometry compression for real-time rendering, IEEE, 1997.

[26] C. L. Bajaj, V. Pascucci, G. Zhuang, Single resolution compression of arbitrary triangular meshes with properties, Computational Geometry 14 (1-3) (1999) 167–186.

[27] G. Taubin, J. Rossignac, Geometric compression through topological surgery, ACM Transactions on Graphics (TOG) 17 (2) (1998) 84–115.

[28] P. Diaz-Gutierrez, M. Gopi, R. Pajarola, Hierarchyless simplification, stripification and compression of triangulated two-manifolds, in: Computer Graphics Forum, Vol. 24, Blackwell Publishing, Inc Oxford, UK and Boston, USA, 2005, pp. 457–467.

[29] C. Touma, C. Gotsman, Triangle mesh compression, in: Proceedings-Graphics Interface, Canadian Information Processing Society, 1998, pp. 26–34.

[30] K. Mamou, T. Zaharia, F. Prêteux, Tfan: A low complexity 3d mesh compression algorithm, Computer Animation and Virtual Worlds 20 (2-3) (2009) 343–354.

[31] S. Gumhold, W. Straßer, Real time compression of triangle mesh connectivity, in: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998, pp. 133–140.

[32] J. Rossignac, Edgebreaker: Connectivity compression for triangle meshes, IEEE transactions on visualization and computer graphics 5 (1) (1999) 47–61.
[33] A. Szymczak, Optimized edgebreaker encoding for large and regular triangle meshes, The Visual Computer 19 (4) (2003) 271–278.
[34] J. Rossignac, 3d compression made simple: Edgebreaker with zipandwrap on a corner-table, in: Proceedings International Conference on Shape Modeling and Applications, IEEE, 2001, pp. 278–283.
[35] A. Szymczak, J. Rossignac, D. King, Piecewise regular meshes: Construction and compression, Graphical Models 64 (3-4) (2002) 183–198.
[36] M. Attene, B. Falcidieno, M. Spagnuolo, J. Rossignac, Swingwrapper: Retiling triangle meshes for better edgebreaker compression, ACM Transactions on Graphics (TOG) 22 (4) (2003) 982–996.
[37] H. Hoppe, Progressive meshes, in: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996, pp. 99–108.
[38] G. Taubin, A. Guéziec, W. Horn, F. Lazarus, Progressive forest split compression, in: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998, pp. 123–132.
[39] W. J. Schroeder, J. A. Zarge, W. E. Lorensen, Decimation of triangle meshes, in: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, 1992, pp. 65–70.
[40] D. Cohen-Or, D. Levin, O. Remez, Progressive compression of arbitrary triangular meshes, in: IEEE visualization, Vol. 99, 1999, pp. 67–72.
[41] R. Gonzalez, R. Woods, Digital Image Processing, Chapter 8: Image Compression, Upper Saddle River, New Jersey: Pearson Education, Inc., 2008, pp. 525–626.
[42] H. Lee, P. Alliez, M. Desbrun, Angle-analyzer: A triangle-quad mesh codec, in: Computer Graphics Forum, Vol. 21, Wiley Online Library, 2002, pp. 383–392.
[43] L. Vasa, G. Brunnett, Exploiting connectivity to improve the tangential part of geometry prediction, IEEE transactions on visualization and computer graphics 19 (9) (2013) 1467–1475.
[44] Z. Karni, C. Gotsman, Spectral compression of mesh geometry, in: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000, pp. 279–286.
[45] X. Gu, S. J. Gortler, H. Hoppe, Geometry images, in: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 2002, pp. 355–361.
[46] O. Devillers, P.-M. Gandoin, Geometric compression for interactive transmission, in: Proceedings Visualization 2000. VIS 2000 (Cat. No. 00CH37145), IEEE, 2000, pp. 319–326.
[47] J. Peng, C.-C. J. Kuo, Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition, in: ACM SIGGRAPH 2005 Papers, 2005, pp. 609–616.
[48] M. Isenburg, J. Snoeyink, Compressing texture coordinates with selective linear predictions, in: Proceedings Computer Graphics International 2003, IEEE, 2003, pp. 126–131.
[49] L. Váša, G. Brunnett, Efficient encoding of texture coordinates guided by mesh geometry, in: Computer Graphics Forum, Vol. 33, Wiley Online Library, 2014, pp. 25–34.
[50] F. Caillaud, V. Vidal, F. Dupont, G. Lavoué, Progressive compression of arbitrary textured meshes, in: Computer Graphics Forum, Vol. 35, Wiley Online Library, 2016, pp. 475–484.
[51] J.-H. Ahn, C.-S. Kim, Y.-S. Ho, Predictive compression of geometry, color and normal data of 3-d mesh models, IEEE Transactions on Circuits and Systems for Video Technology 16 (2) (2006) 291–299.
[52] Z. H. Cigolle, S. Donow, D. Evangelakos, M. Mara, M. McGuire, Q. Meyer, A survey of efficient representations for independent unit vectors, Journal of Computer Graphics Techniques 3 (2) (2014).
[53] L. Chiariglione, The MPEG representation of digital media, Springer Science & Business Media, 2011.
[54] ISO/IEC JTC 1/SC 29/WG 11, ISO/IEC 14496 2:2004, Information technology – Coding of audio-visual objects – Part 2: Visual (2004).

URL https://www.iso.org/standard/73025.html

[55] A. Khodakovsky, P. Schröder, W. Sweldens, Progressive geometry compression, in: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000, pp. 271–278.

[56] I. A. Salomie, A. Munteanu, A. Gavrilescu, G. Lafruit, P. Schelkens, R. Deklerck, J. Cornelis, Meshgrid-a compact, multiscalable and animation-friendly surface representation, IEEE transactions on circuits and systems for video technology 14 (7) (2004) 950–966.

[57] J. Royan, R. Balter, C. Bouville, Hierarchical representation of virtual cities for progressive transmission over networks, in: Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), Citeseer, 2006, pp. 432–439.

[58] ISO/IEC JTC 1/SC 29, Information technology — Coding of audio-visual objects — Part 16: Animation Framework eXtension (AFX) (2011).
URL https://www.iso.org/standard/57367.html?browse=tc

[59] E. S. Jang, S. Lee, B. Koo, D. Kim, K. Son, Fast 3d mesh compression using shared vertex analysis, ETRI journal 32 (1) (2010) 163–165.

[60] K. Group, Directx primitives (2000).
URL https://www.khronos.org/gltf/

[61] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, S. Sullivan, High-quality streamable free-viewpoint video, ACM Transactions on Graphics (ToG) 34 (4) (2015) 1–13.

[62] H. Li, B. Adams, L. J. Guibas, M. Pauly, Robust single-view geometry and motion reconstruction, ACM Transactions on Graphics (ToG) 28 (5) (2009) 1–10.

[63] E. De Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, S. Thrun, Performance capture from sparse multi-view video, in: ACM SIGGRAPH 2008 papers, 2008, pp. 1–10.

[64] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, et al., Real-time non-rigid reconstruction using an rgb-d camera, ACM Transactions on Graphics (ToG) 33 (4) (2014) 1–12.

[65] F. Prada, M. Kazhdan, M. Chuang, A. Collet, H. Hoppe, Spatiotemporal atlas parameterization for evolving meshes, ACM Transactions on Graphics (TOG) 36 (4) (2017) 1–12.

[66] J. Dvořák, P. Vaněček, L. Váša, Towards understanding time varying triangle meshes, in: International Conference on Computational Science, Springer, 2021, pp. 45–58.

[67] J. E. Lengyel, Compression of time-dependent geometry, in: Proceedings of the 1999 symposium on Interactive 3D graphics, 1999, pp. 89–95.

[68] S. Gupta, K. Sengupta, A. A. Kassim, Compression of dynamic 3d geometry data using iterative closest point algorithm, Computer Vision and Image Understanding 87 (1-3) (2002) 116–130.

[69] G. Collins, A. Hilton, A rigid transform basis for animation compression and level of detail, in: Vision, Video, and Graphics, 2005, pp. 21–28.

[70] M. Sattler, R. Sarlette, R. Klein, Simple and efficient compression of animation sequences, in: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2005, pp. 209–217.

[71] R. Amjoun, R. Sondershaus, W. Straßer, Compression of complex animated meshes, in: Computer Graphics International Conference, Springer, 2006, pp. 606–613.

[72] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu, An efficient k-means clustering algorithm: Analysis and implementation, IEEE transactions on pattern analysis and machine intelligence 24 (7) (2002) 881–892.

[73] K. Mamou, T. Zaharia, F. Prêteux, A skinning approach for dynamic 3d mesh compression, Computer Animation and Virtual Worlds 17 (3-4) (2006) 337–346.

[74] N. Stefanoski, X. Liu, P. Klie, J. Ostermann, Scalable linear predictive coding of time-consistent 3d mesh sequences, in: 2007 3DTV Conference, IEEE, 2007, pp. 1–4.

[75] K. Mamou, T. Zaharia, F. Prêteux, N. Stefanoski, J. Ostermann, Frame-based compression of animated meshes in mpeg-4, in: 2008 IEEE International Conference on Multimedia and Expo, IEEE, 2008, pp. 1121–1124.

[76] G. Luo, Z. Deng, X. Zhao, X. Jin, W. Zeng, W. Xie, H. Seo, Spatio-temporal segmentation based adaptive compression of dynamic mesh sequences, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 16 (1) (2020) 1–24.

[77] M. Alexa, W. Müller, Representing animations by principal components, in: Computer Graphics Forum, Vol. 19, Wiley Online Library, 2000, pp. 411–418.

[78] Z. Karni, C. Gotsman, Compression of soft-body animation sequences, Computers & Graphics 28 (1) (2004) 25–34.

[79] G. Luo, F. Cordier, H. Seo, Compression of 3d mesh sequences by temporal segmentation, Computer Animation and Virtual Worlds 24 (3-4) (2013) 365–375.

[80] R. Amjoun, W. Straßer, Efficient compression of 3d dynamic mesh sequences, journal of WSCG (2007).

[81] C.-K. Kao, B.-S. Jong, T.-W. Lin, Representing progressive dynamic 3d meshes and applications, in: 2010 18th Pacific Conference on Computer Graphics and Applications, IEEE, 2010, pp. 5–13.

[82] L. Váša, V. Skala, Coddyac: Connectivity driven dynamic mesh compression, in: 2007 3DTV Conference, IEEE, 2007, pp. 1–4.

[83] L. Váša, V. Skala, Cobra: Compression of the basis for pca represented animations, in: Computer Graphics Forum, Vol. 28, Wiley Online Library, 2009, pp. 1529–1540.

[84] L. Váša, V. Skala, Geometry-driven local neighbourhood based predictors for dynamic mesh compression, in: Computer Graphics Forum, Vol. 29, Wiley Online Library, 2010, pp. 1921–1933.

[85] L. Váša, Optimised mesh traversal for dynamic mesh compression, Graphical Models 73 (5) (2011) 218–230.

[86] L. Váša, S. Marras, K. Hormann, G. Brunnett, Compressing dynamic meshes with geometric laplacians, in: Computer Graphics Forum, Vol. 33, Wiley Online Library, 2014, pp. 145–154.

[87] L. Váša, J. Dvořák, Error propagation control in laplacian mesh compression, in: Computer Graphics Forum, Vol. 37, Wiley Online Library, 2018, pp. 61–70.

[88] A. S. Lalos, A. A. Vasilakis, A. Dimas, K. Moustakas, Adaptive compression of animated meshes by exploiting orthogonal iterations, The Visual Computer 33 (6) (2017) 811–821.

[89] L. L. Ibarria, J. R. Rossignac, Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity, Tech. rep., Georgia Institute of Technology (2003).

[90] J. Zhang, C. B. Owen, Octree-based animated geometry compression, in: Proceedings of the IEEE Data Compression Conference, 2004, pp. 508–520.

[91] J. Zhang, C. B. Owen, Octree-based animated geometry compression, Computers & Graphics 31 (3) (2007) 463–479.

[92] K. Muller, A. Smolic, M. Kautzner, P. Eisert, T. Wiegand, Predictive compression of dynamic 3d meshes, in: IEEE International Conference on Image Processing 2005, Vol. 1, IEEE, 2005, pp. I–621.

[93] D. Marpe, H. Schwarz, T. Wiegand, Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard, IEEE Transactions on circuits and systems for video technology 13 (7) (2003) 620–636.

[94] K. Müller, A. Smolic, M. Kautzner, P. Eisert, T. Wiegand, Rate-distortion-optimized predictive compression of dynamic 3d mesh sequences, Signal Processing: Image Communication 21 (9) (2006) 812–828.

[95] R. Amjoun, W. Straßer, Single-rate near lossless compression of animated geometry, Computer-Aided Design 41 (10) (2009) 711–718.

[96] N. Stefanoski, J. Ostermann, Spc: fast and efficient scalable predictive coding of animated meshes,

in: Computer Graphics Forum, Vol. 29, Wiley Online Library, 2010, pp. 101–116.

[97] M. O. Bici, G. B. Akar, Improved prediction methods for scalable predictive animated mesh compression, Journal of Visual Communication and Image Representation 22 (7) (2011) 577–589.

[98] J.-K. Ahn, Y. J. Koh, C.-S. Kim, Efficient fine-granular scalable coding of 3d mesh sequences, IEEE Transactions on Multimedia 15 (3) (2012) 485–497.

[99] J. Lounsbery, PhD Thesis, Multiresolution Analysis for Surfaces of Arbitrary Topological Type, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, USA, 1994.

[100] M. Lounsbery, T. D. DeRose, J. Warren, Multiresolution analysis for surfaces of arbitrary topological type, ACM Transactions on Graphics (TOG) 16 (1) (1997) 34–73.

[101] C. Loop, Master of Science Thesis, Smooth Subdivision Surfaces Based on Triangles, Department of Mathematics, The University of Utah, Salt Lake City, UT, USA, 1987.

[102] A. Khodakovsky, I. Guskov, Compression of normal meshes, in: Geometric modeling for scientific visualization, Springer, 2004, pp. 189–206.

[103] J.-H. Yang, C.-S. Kim, S.-U. Lee, Progressive coding of 3d dynamic mesh sequences using spatiotemporal decomposition, in: 2005 IEEE International Symposium on Circuits and Systems, IEEE, 2005, pp. 944–947.

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

[105] I. Guskov, A. Khodakovsky, Wavelet compression of parametrically coherent mesh sequences, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2004, pp. 183–192.

[106] J.-W. Cho, M.-S. Kim, S. Valette, H.-Y. Jung, R. Prost, 3-d dynamic mesh compression using wavelet-based multiresolution analysis, in: 2006 International Conference on Image Processing, IEEE, 2006, pp. 529–532.

[107] I. Guskov, W. Sweldens, P. Schröder, Multiresolution signal processing for meshes, in: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999, pp. 325–334.

[108] I. Guskov, Multivariate subdivision schemes and divided differences, Preprint, Princeton University 1 (1998) 998.

[109] H. Q. Nguyen, P. A. Chou, Y. Chen, Compression of human body sequences using graph wavelet filter banks, in: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2014, pp. 6152–6156.

[110] B. Yang, Z. Jiang, Y. Tian, J. Shangguan, C. Song, Y. Guo, M. Xu, A novel dynamic mesh sequence compression framework for progressive streaming, in: 2017 International Conference on Virtual Reality and Visualization (ICVRV), IEEE, 2017, pp. 49–54.

[111] B. Yang, Z. Jiang, J. Shangguan, F. W. Li, C. Song, Y. Guo, M. Xu, Compressed dynamic mesh sequence for progressive streaming, Computer Animation and Virtual Worlds 30 (6) (2019) e1847.

[112] S. K. Narang, A. Ortega, Compact support biorthogonal wavelet filterbanks for arbitrary undirected graphs, IEEE transactions on signal processing 61 (19) (2013) 4673–4685.

[113] J. D. J. G. Leandro, R. M. Cesar Jr, R. S. Feris, Shape analysis using the spectral graph wavelet transform, in: 2013 IEEE 9th International Conference on e-Science, IEEE, 2013, pp. 307–316.

[114] H. M. Briceno, P. V. Sander, L. McMillan, S. Gortler, H. Hoppe, Geometry videos, in: Eurographics/SIGGRAPH symposium on computer animation (SCA), Eurographics Association, 2003.

[115] P. V. Sander, S. Gortler, J. Snyder, H. Hoppe, Signal-specialized parameterization, in: Proceedings of the Thirteenth Eurographics Workshop on Rendering, Eurographics Association/Association for Computing Machinery, 2002.

[116] K. Mamou, T. Zaharia, F. Prêteux, Multi-chart geometry video: A compact representation for 3d animations, in: Third International Symposium on 3D Data Processing, Visualization, and

Transmission (3DPVT'06), IEEE, 2006, pp. 711–718.

[117] K. Zhou, J. Synder, B. Guo, H.-Y. Shum, Iso-charts: stretch-driven mesh parameterization using spectral analysis, in: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2004, pp. 45–54.

[118] B. Vallet, B. Lévy, Spectral geometry processing with manifold harmonics, in: Computer Graphics Forum, Vol. 27, Wiley Online Library, 2008, pp. 251–260.

[119] C. Wang, Y. Liu, X. Guo, Z. Zhong, B. Le, Z. Deng, Spectral animation compression, Journal of Computer Science and Technology 30 (3) (2015) 540–552.

[120] C. Chen, Q. Xia, S. Li, H. Qin, A. Hao, Compressing animated meshes with fine details using local spectral analysis and deformation transfer, The Visual Computer 36 (5) (2020) 1029–1042.

[121] E. S. Jang, J. D. Kim, S. Y. Jung, M.-J. Han, S. O. Woo, S.-J. Lee, Interpolator data compression for mpeg-4 animation, IEEE transactions on Circuits and Systems for Video Technology 14 (7) (2004) 989–1008.

[122] O. Petňík, L. Váša, Improvements of mpeg-4 standard famc for efficient 3d animation compression, in: 2011 3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), IEEE, 2011, pp. 1–4.

[123] T. Matsuyama, X. Wu, T. Takai, T. Wada, Real-time dynamic 3-d object shape reconstruction and high-fidelity texture mapping for 3-d video, IEEE Transactions on Circuits and Systems for Video Technology 14 (3) (2004) 357–369. doi:10.1109/TCSVT.2004.823396.

[124] J. Starck, A. Hilton, Surface capture for performance-based animation, IEEE Computer Graphics and Applications 27 (3) (2007) 21–31. doi:10.1109/MCG.2007.68.

[125] H. Habe, Y. Katsura, T. Matsuyama, Skin-off: representation and compression scheme for 3d video, in: Picture Coding Symposium, 2004, pp. 301–306.

[126] S.-R. Han, T. Yamasaki, K. Aizawa, 3d video compression based on extended block matching algorithm, in: 2006 International Conference on Image Processing, IEEE, 2006, pp. 525–528.

[127] ISO/IEC JTC 1/SC 29/WG 7, Use cases for Mesh Coding (April 2021).

[128] J. Hou, L.-P. Chau, Y. He, N. Magnenat-Thalmann, A novel compression framework for 3d time-varying meshes, in: 2014 IEEE International Symposium on Circuits and Systems (ISCAS), 2014, pp. 2161–2164. doi:10.1109/ISCAS.2014.6865596.

[129] D. T. Quynh, Y. He, X. Chen, J. Xia, Q. Sun, S. C. Hoi, Modeling 3d articulated motions with conformal geometry videos (cgvs), in: Proceedings of the 19th ACM International Conference on Multimedia, MM '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 383–392. doi:10.1145/2072298.2072349.

[130] J. Xia, I. Garcia, Y. He, S.-Q. Xin, G. Patow, Editable polycube map for gpu-based subdivision surfaces, in: Symposium on Interactive 3D Graphics and Games, Association for Computing Machinery, New York, NY, USA, 2011, p. 151–158. doi:10.1145/1944745.1944771.

[131] N. Halko, P.-G. Martinsson, J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM review 53 (2) (2011) 217–288.

[132] T. Wiegand, G. J. Sullivan, G. Bjontegaard, A. Luthra, Overview of the h.264/avc video coding standard, IEEE Transactions on circuits and systems for video technology 13 (7) (2003) 560–576.

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

[134] S.-R. Han, T. Yamasaki, K. Aizawa, 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, 2008, pp. 1045–1048. doi:10.1109/ICIP.2008.4711937.

[135] T. Yamasaki, K. Aizawa, Patch-based compression for time-varying meshes, in: 2010 IEEE International conference on image processing, IEEE, 2010, pp. 3433–3436.

[136] . R. Ohbuchi, . A. Mukaiyama, . S. Takahashi, A frequency-domain approach to watermarking 3d shapes, in: Computer Graphics Forum, Vol. 21, Wiley Online Library, 2002, pp. 373–382.

[137] L. Yamasaki, K. Aizawa, Bit allocation of vertices and colors for patch-based coding in time-varying meshes, in: 28th Picture Coding Symposium, 2010, pp. 162–165. doi:10.1109/PCS.2010.5702449.

[138] E. Pavez, P. A. Chou, Dynamic polygon cloud compression, in: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 2936–2940. doi:10.1109/ICASSP.2017.7952694.

[139] G. M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, Tech. rep. (1966).

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

[141] H. S. Malvar, Adaptive run-length/golomb-rice encoding of quantized generalized gaussian sources with unknown statistics, in: Data Compression Conference (DCC'06), IEEE, 2006, pp. 23–32.

[142] ISO/IEC JTC 1/SC 29, Information technology — Coded representation of immersive media — Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC) (2021).
URL https://www.iso.org/standard/73025.html

[143] E. Faramarzi, R. Joshi, M. Budagavi, Mesh coding extensions to mpeg-i v-pcc, in: 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP), 2020, pp. 1–5. doi:10.1109/MMSP48831.2020.9287057.

[144] D. B. Graziosi, Video-based dynamic mesh coding, in: 2021 IEEE International Conference on Image Processing (ICIP), IEEE, 2021, pp. 3133–3137.

[145] G. Turk, M. Levoy, Zippered polygon meshes from range images, in: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, Association for Computing Machinery, New York, NY, USA, 1994, p. 311–318. doi:10.1145/192161.192241.

[146] ISO/IEC JTC 1/SC 29/WG 7, CfP for Dynamic Mesh Coding (2021).