INTERNATIONAL ORGANISATION FOR STANDARDISATION  
ORGANISATION INTERNATIONALE DE NORMALISATION  
ISO/IEC JTC 1/SC 29/WG 11  
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 N16662  
Genève, CH, January 2017

**Title:** White paper on Three-Dimensional Graphics Compression (3DGC)

**Source:** Communication

**Editor:** Francisco Morán Burgos (Universidad Politécnica de Madrid, ES)

Contents

1. Introduction 1

2. 3DG-related Standards 2

2.1. Location of 3DG Tools in MPEG‑4 3

3. AFX: Technology and Key Features 3

3.1. Shape Compression Tools 3

3.1.1. Single-Rate Coding 4

3.1.2. Multi-Resolution Coding 4

3.2. Appearance Compression Tools 5

3.2.1. Image Coding 6

3.2.2. Attribute Mapping Coding 6

3.3. Animation Compression Tools 6

3.3.1. Generic Animation Coding 6

3.3.2. Avatar Animation Coding 7

4. Ongoing Developments 7

5. Annex: 3DG Coding Basics 8

5.1. Static Object Coding 8

5.2. Dynamic Object Coding 10

5.3. Scene Coding 10

6. References 11

# Introduction

This white paper focuses on the Three-Dimensional Graphics Compression (3DGC) tools contained in the MPEG‑4 international standard (formally, ISO/IEC 14496), which was designed by the Moving Picture Experts Group (MPEG; formally, ISO/IEC JTC 1/SC 29/WG 11) as its first specification for the compression of truly multimedia content. Unlike the previous standards developed by this ISO committee, MPEG[‑1] and MPEG‑2, which only dealt with the coding of natural video and audio (and some related systems information), MPEG‑4 was aimed at compressing audio-visual scenes which not only were to be interactive but also have different kinds of multimedia objects, including synthetic ones. Synthetic audio tools have been developed by MPEG’s Audio Subgroup almost since the MPEG‑4 was conceived, but synthetic visual objects were dealt with first within the Video Subgroup and later within the 3D Graphics (3DG) Subgroup, specifically created for this purpose.

MPEG‑4 already featured, in its first two versions finalised in 1999 and 2000, efficient compression techniques for: *i)* low bit-rate coding of Face and Body Animation (FBA) for virtual humanoids modelled in a specific way (to mimic the hierarchy of the joints of a human skeleton); and *ii)* 3D Mesh Coding (3DMC) for generic, but static, 3D objects represented by first-order (i.e., polygonal) approximations of their surfaces. Since 2000, MPEG’s 3DG Subgroup has been developing and maintaining the Animation Framework eXtension (AFX), a set of tools for efficiently coding the shape, texture and animation of interactive synthetic 3DG objects. AFX is specified in Part 16 of MPEG‑4, whose first Edition was published by ISO in February 2004 and whose fourth (and most recent one at the time of writing) was released in October 2011 as ISO/IEC 14496‑16:2011 [1].

This white paper is mostly devoted to the results yielded by finalised activities of the 3DG Subgroup which have been published within AFX, but contains as well a short Section on the current activity regarding Point Cloud Compression (PCC).

Finally, it is worth noting that other MPEG standards such as MPEG‑7 and MPEG‑V (formally, ISO/IEC 15938 and ISO/IEC 23005, respectively) contain as well 3DG-related tools, but are not reviewed below because they are not aimed at the *coding* of 3DG content.

# 3DG-related Standards

The computer graphics field is a relatively young one which started in the early 1960’s, but has known since an exponential growth similar to that of the semiconductors supporting it, mostly driven by the need for special effects and 3DG content from the film and videogames industries. The strategy for developing 3DG solutions has always been based on three requirements: faster, cheaper, better (image quality). The 3DG chain is usually focused on its two ends, production and consumption, each having its own strong requirements. In many applications such as games, dealing with complex 3D environments, there is an iterative loop involving content creators and programmers or beta-testers, to make sure that the content fits a specific end-user terminal configuration. Besides, the complexity of creating 3DG content leads to the coexistence of several authoring tools, each specialized in a few particular tasks, which typically makes it necessary to use several tools to create one single 3DG asset. Interoperability is thus an issue, so either a unique representation standard is used by all tools or data format converters must be provided in each of them, and for all others in the ideal case.

In the last decades, several efforts have been made to develop a unique data format for interchanging assets between authoring tools. In the category of open standards, X3D [2] (which is the evolution of VRML97 [3]) and COLLADA [4] are the best known, the latter probably being the most adopted by current tools. All these standards specify textual formats for both graphics primitives, to represent the characteristics of isolated 3D objects (their shape, appearance, animation, etc.), and the scene graph, to describe the space-time relations between different objects and between them and the user. While COLLADA concentrates on representing (static) 3D objects or scenes, X3D pushes the standardization further by addressing as well user interaction thanks to an event model in which scripts, possibly external to the main file containing the 3D scene, may be used to control the behaviour of its objects.

On the consumption side, data should be as “flat” (i.e., close to the hardware) as possible to enable easy and rapid processing by the graphics cards. A lot of effort is spent on optimizing the rendering step, one of the bottlenecks in the 3DG pipeline. Better, faster algorithms are continuously proposed and many are directly implemented in hardware. To ease the access to different graphics cards, low level 3DG Application Programming Interface (APIs) like OpenGL and Microsoft Direct3D were created and keep evolving. Such APIs hide the different capabilities of the many and wildly diverse graphics cards in the market (with and without specific hardware acceleration for 3D visualization, with more or less texture memory, etc.) and offer the programmer a single set of rendering functions.

Recent developments of collaborative sharing platforms are efficient vehicles for pushing media over the Internet. By potentially transforming every end user in a content producer, Web2.0 platforms lead to the proliferation of media content, therefore introducing an even greater need of compactness in data representation. Even in the case of games, which have traditionally been distributed in DVDs containing all assets, openness to new distribution channels is a must, either for entire game download before playing, or for the streaming of assets during the match in the case of on-line games. Neither the formats used in the content creation phase, nor the ones used for rendering are appropriate for transmission, since they usually satisfy none of the three requirements that are most important in this context: *i) compression*, to reduce the size of the transmitted data; *ii) streamability*, to enable players to start using the content before having downloaded it entirely; *iii) scalability*, to ease real-time adaptation of a single version content to different specific networks and terminals.

AFX offers both compact high-level representation and compression of low-level components, meant to be used in combination with typical 3DG rendering APIs such as OpenGL or Microsoft DirectX. Besides a more compact representation, high-level abstract models like the ones used in AFX often provide other functionalities such as scalability; and, in particular, view-dependent transmission and automatically managed hierarchical local refinements. These functionalities typically require large amounts of computation, but clever implementations may provide only crude capabilities for a limited resource (typically mobile) terminal, and nevertheless full support for a powerful (desktop) terminal; the rendering will not have the same quality but the content will be only one.

## Location of 3DG Tools in MPEG‑4

To model and later compress an interactive, dynamic, synthetic 3DG scene containing several objects, and their evolution over time, and how the user may interact with them, two separate problems have to be solved: the representation of each 3D object and that of the whole 3D scene. All standards for synthetic 3D (or even 2D) scene modelling and compression segregate those two problems, and MPEG‑4 is no exception. Annex I (Section 5) introduces several basic concepts necessary to understand the 3DG compression tools of MPEG‑4, which have been published in several of its Parts:

* Most compression tools for individual 3DG objects belong in MPEG‑4 Part 16, AFX [1], and are reviewed in Section 3. However, as already mentioned, some old tools such as 3DMC and FBA were published in the first Editions of Part 2, “Visual”, which is currently at its third [5].
* Tools related to scene graph representation are mostly in MPEG‑4 Part 11, “Scene Description and Application Engine” [6], which is the home of the BInary Format for Scene (BIFS) toolset, originally published in Part 1, “Systems” [7]. BIFS did not only inherit from VRML97 [3] the scene graph concept, but also many scene nodes and generic animation mechanisms. Part 25, “3D Graphics Compression Model” [8], offers an architectural model able to accommodate a third party XML-based description of the scene graph with potential binarization tools and with MPEG‑4 3DG compression tools.
* Finally, like all others in MPEG‑4 (and other MPEG standards), 3DG-related tools have reference software [9] and conformance bitstreams [10] associated to them.

# AFX: Technology and Key Features

## Shape Compression Tools

As in the cases of image or video coding, the concepts of resolution scalability and of *single- vs. multi-rate coding* are important for 3D mesh coding. In some scenarios, a single-rate coding of a 3D mesh may be enough, but its progressive compression may be desirable, especially if it is a complex mesh to be transmitted over a network with a restricted bandwidth, or to terminals with limited processing power. In such cases, it is useful to represent and code the original, fine mesh as a sequence (or a hierarchy) of refinements applied to a simple, coarse mesh. During decoding, connectivity and geometry are reconstructed incrementally from the bitstream until the original mesh is rendered at its full resolution or the transmission is cancelled by the user. Progressive compression thus allows transmission and rendering of different Levels Of Detail (LODs).

### Single-Rate Coding

The compression efficiency of the tools reviewed in this Section is measured against the following estimate of the raw size (before compression), *Sraw*, of a generic mesh with *V* vertices and *T* triangles: *Sraw* = 3 (16 *V* + ⎡log2 (*V*)⎤ *T*) bits ≅ 3 (16 + 2 ⎡log2 (*V*)⎤) bits per vertex (bpv). This estimate is based on the Indexed Face Set (IFS) paradigm described in Section 5.1.A, and on the assumptions that 16 bits are used for each coordinate in the geometry table, that ⎡log2 (*V*)⎤ bits are used for each index in the topology table, and that *T* ≅ 2 *V*, because typical triangle meshes, and especially large ones, have a vast majority of “regular” vertices (which are connected to another six). For a relatively large mesh with, say, 50 k vertices (more precisely, up to 216 = 65 536), and hence around 100 k triangles (217 = 131 072), *Sraw* = 144 bpv, of which one third (3·16 bpv) is devoted to geometry and two thirds (96 bpv) to topology.

#### Topological Surgery

The problem of compactly encoding the connectivity information of a 3D polygon mesh was studied extensively in the second half of the 20th century, starting with the theory of planar polygon graphs. From results established much earlier by Tutte and Turán, Taubin and Rossignac invented in 1998 their Topological Surgery (TS) [11] scheme for 3D mesh connectivity coding, which starts by cutting the mesh along a selected set of edges to create a vertex tree and a triangle tree. The mesh connectivity is then represented by both trees, yielding 1 bpv for very regular meshes and 4 bpv on average otherwise — vs. the raw size estimate of 96 bpv! TS was the basis for the old 3D Mesh Coding (3DMC) toolset of MPEG‑4 Part 2, and also for the 3DMC extension (3DMCe) [12] tool of AFX, which adds support for efficient texture coordinate compression, and for mesh animation/editing.

As for geometry information, to achieve higher compression rates than the 3·16 bpv mentioned above, both 3DMC and 3DMCe feed the vertex coordinates to a *quantization* step, whose resulting values are then compressed by entropy coding after some *prediction* (relying on some smoothness assumptions) is applied. Both steps contribute to the compactness of the final result, but quantization is intrinsically and irreversibly lossy, whereas prediction is a perfectly reversible and lossless transformation of the signal to make it fit for a more efficient subsequent entropy coding. Vertex coordinates are typically uniformly quantized with 8‑14 bits each and prediction is usually linear: at most three vertices adjacent to the one being decoded, and already decoded themselves, according to the vertex ordering imposed by TS, are used for the prediction known as the “parallelogram rule”. This yields bitrates of some 13‑18 bpv for 9‑12 bits per coordinate, or 13 bpv at 8‑bit quantization resolution.

#### Scalable Complexity 3D Mesh Compression

The Scalable Complexity 3D Mesh Compression (SC3DMC) toolset of AFX specifies a way to fine tune the trade-off between compression efficiency (bpv) and computational resources (CPU/GPU and memory) needed in both encoder and decoder by choosing among three 3D mesh coding techniques: Quantization-Based Compact Representation (QBCR), Shared Vertex Analysis (SVA) and Triangle FAN (TFAN). The main idea behind the SC3DMC scheme is that, in some application scenarios, especially the ones involving mobile devices with reasonable network connections, the minimization of bitstream size may not be as important as that of computational resources. Current implementations of SC3DMC yield encoding performances of 17-70 bpv (for both topology and geometry) with associated decoding speeds of millions of vertices per second on an ordinary PC.

### Multi-Resolution Coding

The concept of Progressive Mesh (PM) [13] allows to code a mesh with a total of ~35 bpv. A PM is a base mesh plus a sequence of vertex split records, each specifying which vertex and pair of edges incident to it must be split, and the local geometry changes. From such a representation, it is easy to extract an LOD of the mesh with any desired number of triangles by simply choosing the adequate prefix of the vertex split sequence, which is streamed after the base mesh has been transmitted. The Progressive Forest Split (PFS) [14] technique is based on the PM and TS ideas, and was included in the old 3DMC toolset of MPEG‑4 Part 2. PFS is able to reduce the bitrates of PMs at the expense of reduced granularity: two successive LODs of a PFS set differ by a group of vertex splits, instead of only one. Logically enough, the highest compression ratios are achieved by minimizing the number of LODs but, typically, it is possible to remain slightly below 30 bpv for medium size meshes coded with several LODs.

#### Wavelet Subdivision Surfaces

Wavelet Subdivision Surfaces (WSSs) [15][16] are the basis for the best known truly *hierarchical* (as opposed to merely progressive) 3D mesh coding technique. As explained in Section 5.1.A, connectivity information can be coded extremely efficiently thanks to remeshing. Besides, the geometry information contained in the hierarchical set of 3D details (representing the prediction errors between successive LODs) can also be very compactly coded as follows: *i)* a space-frequency wavelet transform is applied to the 3D details to obtain a zero-centred set of coefficients; *ii)* since the magnitude of those wavelet coefficients decays at finer levels with a rate related to the smoothness of the original surface, they are particularly well suited for zerotree coding. In fact, the coding efficiency results of the WSS tool in AFX are over four times better than those of PFS (in terms of reconstruction error for a given bitrate).

#### MeshGrid

Another hierarchical 3D mesh compression tool of AFX is MeshGrid [17], a compact representation attaching a description of the connectivity between the vertices on the surface, called “connectivity wireframe”, to a regular 3D grid of points, called “reference grid”. MeshGrid spends very few bpv for topology information thanks to a 3D extension of Freeman’s chain-code and to the particular properties of the connectivity wireframe, which allows the decoder to derive the triangulation unambiguously without having encoded it explicitly à la IFS. As for geometry information, the reference grid is a smooth vector field defined on a regular discrete 3D space, and is also efficiently compressed by using an embedded 3D wavelet-based multi-resolution intra-band coding algorithm, not unlike the one used for 3D details in WSS-based coders.

#### FootPrint

Finally, AFX’s FootPrint toolset is yet another solution for the multi-resolution representation of 3D objects based on footprints, of which the most notable example are buildings in virtual cities [18]. Indeed, the on-line, interactive navigation over a large and densely built urban environment raises difficult problems due to the scene size and complexity, and its hierarchical representation allows to build client-server applications in which only the details of particular Regions Of Interest (ROIs) must be streamed by the server to each client, depending on the specific viewpoint chosen in real time by the corresponding user (note that WSSs and MeshGrid are also suited for view-dependent, ROI refinement). FootPrint efficiently exploits the fact that most automated urban environment modelling techniques are essentially extrusion-based and proposes a procedural representation for roofs and façades.

## Appearance Compression Tools

As explained in Section 5.1.B, an appearance may be linked at modelling time to a 3D shape through the process of attribute mapping. In the particular but most common case of texture mapping, this enables graphic artists to simulate the wrapping of each 3D object in one or more textures, i.e., 2D images, which have of course to be stored or transmitted along with the information corresponding to the 3D shape and the attribute/texture mapping itself. This is why we devote the first Subsection below to image coding standards related to MPEG‑4, although the most important part of this Section, conceptually, is the following Subsection, which deals with the *attribute mapping coding* tools of AFX (and other MPEG‑4 Parts).

### Image Coding

There are many *image compression standards* to choose from, the most commonly used being probably JPEG [19], which can achieve a very high compression ratio (in the order of 50:1 or more) without significant visual quality loss for most natural images. JPEG is not natively supported by MPEG‑4, which has its own image coding tool defined in its Part 2, namely Visual Texture Coding (VTC), but may be used as the format for a texture in an MPEG‑4 scene as an “external” elementary bitstream. On the other hand, JPEG 2000 [20] is supported natively by MPEG‑4, and is based, like VTC, on a wavelet-based space-frequency transform, which makes it inherently suitable for multi-resolution decoding of embedded bitstreams, and hence view-dependent, ROI refinement of the 2D texture mapped onto a 3D object [21].

### Attribute Mapping Coding

In the old IFS tool from MPEG‑4 Part 11, which was inherited from VRML97 and initially published in Part 2, attribute mapping data benefits from exactly the same compression sources as the geometry data, i.e., coordinate quantization and entropy coding. It may be specified on a per-face (i.e., per-triangle, if all faces/polygons are triangles), per-vertex or per-corner fashion, but the choice is a global one for the entire IFS. Another limitation inherited from VRML97 is that, when using texture mapping, all texture coordinates of an IFS must refer to a single image.

The Indexed Region Set (IRS) tool contained in the first Amendment to the fourth Edition of AFX, “Efficient representation of 3D meshes with multiple attributes” [22], gets rid of these limitations by introducing the concept of (mesh texturing) *regions*. Regions in an IRS may be used, for instance, to group sets of faces of (what would have been) an IFS in such a way that each group is assigned a different texture image. Then, region border vertices may be assigned different texture coordinates on a per-corner basis, whereas vertices which are completely interior to a region (i.e., the ones which do not sit on texture borders) need not. This flexibility is not present in the regular IFS node, for which the per-corner specification turns out to be quite expensive.

## Animation Compression Tools

### Generic Animation Coding

As briefly summarised in Section 5.2, *key frame animation* (and implicitly the corresponding value interpolation between key frames) is one of the most popular methods for computer animation. The interpolated data type determines the dimension of the value in each (key; value) pair: for vertex coordinates and object positions, 3D values have to be coded/interpolated, whereas orientations are represented with 4D vectors called quaternions (an axis plus an angle). The old Interpolator Compression (IC) [23] tool of MPEG‑4 Part 11 was designed to compress generic interpolated data, and can achieve compression ratios of up to 30:1 (vs. textual representation of the same data) for coordinates, normals, translations and rotations. However, the IC tool is unable to exploit any spatial redundancy present in the animation data.

Instead, AFX’s Frame-based Animated Mesh Compression (FAMC) [24] tool does cluster vertices based on their motion properties. FAMC encodes the time-varying positions, normals, etc. associated to the vertices of a mesh, both for deformations and rigid motions. The data in a FAMC stream is structured into segments of several frames that can be decoded individually. Within a segment, a temporal prediction model, used for motion compensation, is represented. Each decoded animation frame updates the geometry and possibly the attributes of the 3D object that FAMC is referred to. Once the mesh is segmented with respect to the motion of each cluster, three kinds of data are obtained and encoded. The standardized implementation of the FAMC encoder [24] achieves compression ratios of up to 45:1 (vs. textual representation of the same data).

### Avatar Animation Coding

Concerning avatar animation, possibly the most used animation content (e.g., in games), the old Face and Body Animation (FBA) [25] tool of MPEG‑4 Part 2, which was highly inspired on video coding algorithms, and targeted at the compression of virtual humanoids at very low bit-rate, had limitations in terms of realism.

AFX includes, since its first Edition, a more powerful and generic tool, called Bone-Based Animation (BBA) [26], featuring a multilayer modelling of the avatar and mesh deformation driven by a set of 1D controllers (bones and muscles). BBA is a compression tool for geometric transforms of bones (used in skinning-based animation) and weights (used in morphing animation). These elements are defined in the scene graph and must be uniquely identified so that the BBA stream may then refer to them. The BBA stream contains, for each frame, the new transforms of bones (expressed as Euler angles or quaternions) and the new weights of the morph targets.

Analogously to what was done in the FBA framework, two compression schemes were developed in BBA to encode the Skeleton, Muscles and Skin (SMS) animation parameters, namely predictive-based and DCT-based. The compression ratios obtained with both encoding schemes depend strongly on the motion and skeleton complexity, and on the number of muscles used. Typically, when considering a human-like virtual character with natural motion (obtained from a motion capture system), a bitrate of 25 kb/s ensures a good motion quality, although some optimization mechanisms allow achieving better results. Without an a prioriknowledge about the range of bone and muscle motion, due to the fact that any kind of skeleton is allowed, the MPEG‑4 standard cannot directly address an efficient arithmetic coding. For this reason, the standard supports the transmission of this range within the animation stream. An optimized implementation [27] of the BBA encoder yields compression ratios of up to 70:1 (vs. textual representation of the same data).

# Ongoing Developments

Advanced 3D representations of the real world are enabling more immersive forms of interaction and communication to better understand and navigate it. 3D point clouds have recently emerged as such an enabling representation, so MPEG’s 3DG Subgroup, in collaboration with other Subgroups, has already identified a number of use cases associated with point cloud data. Clouds of 3D points with associated attributes (colour, material properties, etc.) can also be generated synthetically, but are typically captured using multiple cameras and depth sensors in various setups, and may have to contain from thousands up to billions of points in order to realistically represent the sampled objects or scenes, which can later be reconstructed/rendered thanks to those point clouds.

Point Cloud Compression (PCC) technologies are thus obviously needed: lossy PCC is likely to be desirable for use in real-time communications, and lossless PCC might be necessary in the contexts of dynamic mapping for autonomous driving, six Degrees of Freedom (6 DoF) virtual reality, cultural heritage applications, etc. Recent research on compression tools for both static and dynamic 3D point clouds has shown evidence that it is possible to improve the coding efficiency of already existing solutions. MPEG’s 3DG Subgroup has therefore developed a set of requirements for PCC, addressing compression of geometry and attributes, scalable/progressive coding, coding of sequences of clouds captured over time, etc., which are included in the corresponding Call for Proposals (CfP) for PCC issued in January 2017. As is usual in MPEG’s competitive process, companies and organizations will respond to this CfP by submitting their PCC technologies, that will be evaluated against each other based upon objective metrics and subjective tests.

# Annex: 3DG Coding Basics

As any good book on computer graphics [28] explains, to model an interactive, dynamic, synthetic 3DG scene containing several objects, two separate problems have to be solved: the representation of *each 3D object* and that of the *whole 3D scene*.

In turn, to model and compress a single interactive 3D object, even if it is *static* (it does not move by itself and its shape remains the same over time), two very different kinds of information must be specified to describe its *shape* and *appearance*, and this is done again separately in most cases. Furthermore, if the object is *dynamic*, its *animation* over time must be specified as well. Finally, the same goes for *interaction*: the scene creator must describe at modelling time whether, and in what way, the user may interact with the object, and what the result of that interaction will be, e.g., through scripts to be executed at rendering time.

Representing and coding an interactive scene means specifying the interrelations, over space and time, of its objects, both between themselves and with the user. The essential concept here is that of *scene graph*.

## Static Object Coding

Few 3D object modelling and coding techniques pay attention to the inner volume, especially if what is most important is the visual result once the object is decoded and rendered. Since only the object surface shape and appearance are taken into account by most rendering engines, “truly 3D” models describing solids are much less frequent than “merely 2,5D” ones describing surfaces (embedded in 3D space, but with two degrees of freedom). This Section thus focusses on 3D *surface* modelling[[1]](#footnote-1)⬃.

As for how the *shape* and *appearance* of a surface are described, the traditional approach, which deals with them separately, remains the most common by far. There do exist hybrid Image-Based [Modelling and] Rendering (IB[M]R) techniques to represent simultaneously those two kinds of information, and they do have certain advantages over the traditional approach, but they are not as mature and widespread as the latter, which is still the only one obviously supported in hardware by current Graphics Processing Units (GPUs). This Section thus focusses on traditional surface representation techniques, in which the shape is explicitly described first, and the appearance linked to it later⬃.

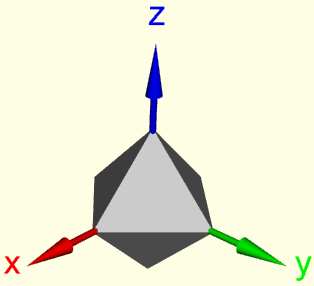
#### Shape

This leads to another taxonomy concerning the shape approximation degree. Higher-order, polynomial or rational patches, e.g., Bézier’s or Non-Uniform Rational B-Spline (NURBS) ones, are very used in Computer-Aided Design (CAD) [29], and greatly simplify the edition of synthetic 3D models. But in the “wider 3DG world”, especially if the videogame industry is included, the most common surface approximation is still the piecewise linear one: indeed, unsophisticated as it may sound, tiling the (likely smooth) desired surface with a mesh of (flat, by definition) polygons is the preferred option. The main reason for this is again that the current rendering engines and GPUs provide specific support for polygon meshes. This Section thus focusses on the representation and compression of *triangle meshes*, since triangles are the simplest polygons, and all other can be triangulated⬃.

A 3D triangle mesh is almost invariably represented with two tables, as illustrated by Figure 1: one for *geometry*, in which the three coordinates (*x*, *y*, *z*) of each vertex are given; and another for *topology* (or *connectivity*), listing the indices (*i*, *j*, *k*) in the previous table of the three vertices forming each triangle. This pair of tables is known as an Indexed Face Set (IFS), which is precisely the name of a VRML97 node (except for the spaces). Given the clear distinction between the geometry information and the topology one, most static 3D triangle mesh compression techniques [30] treat them separately.

|  |  |  |  |
| --- | --- | --- | --- |
| **Geometry information** | | **Topology information** | |
| **Vertex nr.** | **(*x*, *y*, *z*)** | **Triangle nr.** | **(*i*, *j*, *k*)** |
| 🄌 | (-1, 0, 0) | 🄋 | (➌, ➎, ➍) |
| ➊ | (0, -1, 0) | ➀ | (➌, ➍, 🄌) |
| ➋ | (0, 0, -1) | ➁ | (➌, 🄌, ➊) |
| ➌ | (0, 0, +1) | ➂ | (➌, ➊, ➎) |
| ➍ | (0, +1, 0) | ➃ | (➋, ➍, ➎) |
| ➎ | (+1, 0, 0) | ➄ | (➋, 🄌, ➍) |
|  |  | ➅ | (➋, ➊, 🄌) |
|  |  | ➆ | (➋, ➎, ➊) |

Figure 1: IFS representation of an octahedron



🄌

➊

➋

➌

➍

➎

🄋

Traditional 3D mesh encoding and decoding schemes aim to eventually recover the exact original connectivity. For small meshes with carefully laid out connectivity and sample locations, this is very appropriate, but the situation is different for highly detailed, densely sampled meshes coming from 3D scanning: since distortion is measured as geometric distance[[2]](#footnote-2)⬃, the sample locations and connectivity can be treated as additional degrees of freedom to improve the rate-distortion performance. As long as the final result has a geometric error similar to that of the original approximation, the actual sample locations and connectivity do not matter.

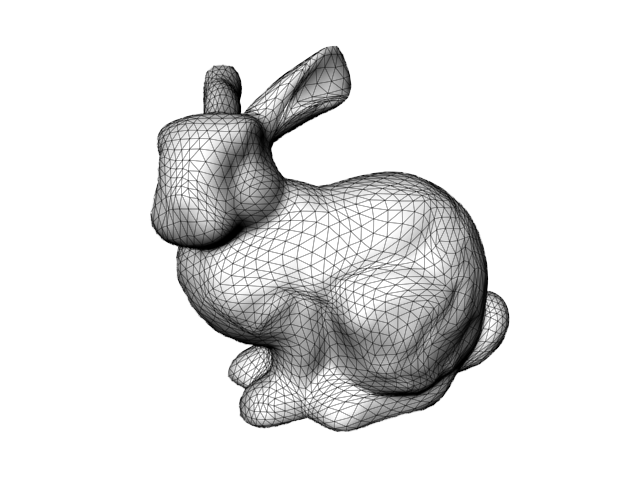
 

Figure 2: Remeshing of the original, arbitrary connectivity “Stanford bunny” mesh of *T* = 69451 triangles into a semi-regular connectivity one of *T’* = *t* · 4*s* = 69376 triangles (*t* = 271 is the number of triangles of the base mesh, recursively subdivided *s* = 4 times into four each)

This is why some 3D mesh coding techniques, especially multi-resolution ones, start with a *remeshing* step: the original mesh, which typically results from an arbitrary sampling of a smooth surface, and typically has an equally arbitrary connectivity, can be remeshed to approximate that surface with the same LOD. For instance, Figure 2 illustrates how it is possible to transform a given original mesh into one with “semi-regular” connectivity, whose topology can be efficiently encoded, as it only depends on that of its underlying base mesh and number of recursive subdivisions. The geometry information is also derived from that of the base mesh, in this case by adding a hierarchical set of 3D details representing the differences (prediction errors) between successive LODs.

#### Appearance

As already stated above, in traditional surface representation techniques, the shape is described first, and then an appearance is linked to it. For a triangle mesh, this is done by assigning *attributes* to its elements (triangles, vertices, etc.: see next paragraph). Examples of such attributes are a colour, hence a 3D vector, e.g., in the Red, Green, Blue (RGB) colour space; or a local normal vector to the surface, hence again a 3D vector, although in the usual (*x*, *y*, *z*) space; or, most typically, a pair of texture coordinates, hence a 2D vector in the (*u*, *v*) texture space, pointing to a pixel (hence an RGB colour) in some texture/image that is to be wrapped around the shape at rendering time through indirections.

*Attribute mapping* is also specified at modelling time, and may follow a per-triangle approach: for instance, a single colour or normal vector may be assigned to an entire triangle and then it will be used, at rendering time, to fill all the pixels covered by that triangle after performing the necessary lighting calculations, if any. However, the resulting “faceted” rendering style is not photorealistic, so attributes are much more often mapped onto the mesh on a per-vertex basis, or even on a per-corner one, a corner being a vertex considered within a particular triangle. In particular, texture coordinates are always mapped on a per-vertex/corner fashion. At rendering time, and more specifically during the rasterizing stage, some kind of interpolation is performed to fill the pixels covered by each triangle after performing the lighting calculations (at its vertices only, in most cases).

Coding the appearance of a 3D triangle mesh involves of course image compression techniques if textures are used, as is often the case, to achieve a more photo-realistic result. But it involves as well, even if textures are not used, attribute coding techniques, which are not too different from the geometry coding ones when attributes are mapped on a per-vertex basis: a vertex has then, on top of the obvious (*x*, *y*, *z*) 3D vector giving its position, other information attached to it, such as an (*r*, *g*, *b*) or (*nx*, *ny*, *nz*) 3D vector, or a (*u*, *v*) 2D one.

## Dynamic Object Coding

Some of the mesh features mentioned above may be continuously updated, therefore transforming the static object into a *dynamic* one. Most commonly, two kinds of features are animated over time, and both concern the geometry data: either the position and/or orientation of the whole set of vertices is changed globally, which means that the object moves and/or rotates inside the scene, or the locations of some vertices are altered relative to that of others, which yields a shape deformation. On the other hand, mesh connectivity typically remains consistent over time, as do attributes and textures.

*Mesh deformation coding* is addressed in two ways: one can either code the vertex positions themselves as a function of time, or use some deformation controller to influence the mesh geometry, and code the controller parameters as a function of time. In both cases, classic signal processing techniques such as space-frequency transforms or predictive and entropy coding can be used to reduce the data size. Usually, the animation data is highly redundant, so compressing it may easily reduce the size by two orders of magnitude.

This redundancy is typically exploited in two (not mutually exclusive) ways: *i)* *temporal interpolation* is a technique that only codes (instant; value) pairs of the considered variable (position, orientation, colour, etc.) for some relevant time instants, called “key frames”, and relies on linear or higher order interpolation to reconstruct the variable values at other instants; *ii)* *spatial interpolation* is also used, for instance skeleton-based animation, to cluster vertices and attach to each group a unique transform.

## Scene Coding

The *scene graph* is the general data structure commonly used by vector-based graphics editing applications and modern computer games. This structure arranges the logical and spatial representation of a multimedia scene and corresponds to a collection of nodes organized as a *hierarchical graph* or tree, whose nodes may have many children but only one single parent. An operation applied to a parent node automatically propagates its effect to all of its children. In many scene graph implementations, associating a geometrical transformation matrix at each level and concatenating such matrices together is an efficient and natural way to perform animation. A common feature is the ability to group related objects into a compound object which can then be transformed, selected, etc., as easily as a single object.

Historically, the scene graph concepts are inherited from techniques enabling to optimize 3DG rendering. To avoid processing invisible objects (i.e., objects outside the window chosen by the user), a logic organization of relations between them is created. Additionally, common properties such as textures or lighting conditions may be used to group together different objects; consequently, loading and unloading operations may be performed per group of objects instead of per object.

Because of the heterogeneity of the scene graph, encoding it will not allow high compression rates (10:1 is common), only small redundancy being exploitable, especially thanks to predictive and entropy coding. Quantization is difficult to use without knowing the semantics of the data to be encoded and the requirements with respect to the reconstruction accuracy.

# References

1. Moving Picture Experts Group (MPEG; formally ISO/IEC JTC 1/SC 29/WG 11): “ISO/IEC 14496‑16:2011, Information technology – Coding of audio-visual objects – Part 16: Animation Framework eXtension (AFX)”, ISO/IEC International Standard (IS), October 2011. This is the fourth and most recent Edition of this IS, including all Corrigenda and Amendments to its previous Editions, of which the first was published in February 2004 (the technical committee work and the balloting process had finished in December 2002).
2. Web3D Consortium and ISO/IEC JTC 1/SC 24, “ISO/IEC 19775‑1:2013, Information technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D), Part 1: Architecture and base components”, ISO/IEC IS, November 2013. This is the third and most recent Edition of this IS, of which the first was published in November 2004.
3. Virtual Reality Modeling Language (VRML) Consortium (currently Web3D Consortium) and ISO/IEC JTC 1/SC 24: “ISO/IEC 14772‑1:1997, Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language – Part 1: Functional specification and UTF‑8 encoding”, ISO/IEC IS, December 1997. This is the first and most recent Edition of this IS, commonly known as “VRML97”.
4. R. Arnaud and M. C. Barnes, “COLLADA: Sailing the Gulf of 3D Digital Content Creation”, A. K. Peters, 2006.
5. MPEG: “ISO/IEC 14496‑2:2004, Information technology – Coding of audio-visual objects – Part 2: Visual”, ISO/IEC IS, May 2004. This is the third and most recent Edition of this IS, of which the first was published in May 1999.
6. MPEG: “ISO/IEC 14496‑11:2015, Information technology – Coding of audio-visual objects – Part 11: Scene description and application engine”, ISO/IEC IS, November 2015. This is the second and most recent Edition of this IS, of which the first was published in December 2005.
7. MPEG: “ISO/IEC 14496‑1:2010, Information technology – Coding of audio-visual objects – Part 1: Systems”, ISO/IEC IS, May 2010. This is the fourth and most recent Edition of this IS, of which the first was published in May 1999.
8. MPEG: “ISO/IEC 14496‑25:2011, Information technology – Coding of audio-visual objects – Part 25: 3D Graphics Compression Model”, ISO/IEC IS, May 2011. This is the second and most recent Edition of this IS, of which the first was published in March 2009.
9. MPEG: “ISO/IEC 14496‑5:2001, Information technology – Coding of audio-visual objects – Part 5: Reference software”, ISO/IEC IS, December 2001. This is the second and most recent Edition of this IS, of which the first was published in May 1999. It is worth noting that the electronic attachments of the textual specification of this IS and all its (many!) Amendments and Corrigenda published to date are freely available at <http://standards.iso.org/ittf/PubliclyAvailableStandards/>: look for all table entries starting with “ISO/IEC 14496‑5:2001”.
10. MPEG: “ISO/IEC 14496‑27:2009, Information technology – Coding of audio-visual objects – Part 27: 3D Graphics conformance”, ISO/IEC IS, December 2009. This is the first and most recent Edition of this IS. It is worth noting that the electronic attachments of the textual specification of this IS and its three Amendments published to date are freely available at <http://standards.iso.org/ittf/PubliclyAvailableStandards/>: look for all table entries starting with “ISO/IEC 14496‑27:2009”.
11. G. Taubin and J. Rossignac: “Geometric Compression Through Topological Surgery”, ACM Trans. Graphics, vol. 17, nr. 2, p. 84-115, April 1998.
12. E.-Y. Chang, N. Hur and E. S. Jang: “3D model compression in MPEG”, Proc. IEEE ICIP 2008, p. 2692-2695, October 2008.
13. H. Hoppe: “Progressive Meshes”, Proc. ACM SIG­GRAPH 1996, p. 99-108, August 1996.
14. G. Taubin, A. Guéziec, W. Horn and F. Lazarus: “Progressive Forest Split Compression”, Proc. ACM SIG­GRAPH 1998, p. 123-132, July 1998.
15. A. Khodakovsky, P. Schröder and W. Sweldens: “Progressive Geometry Compression”, Proc. ACM SIG­GRAPH 2000, p. 271-278, July 2000.
16. F. Morán and N. García: “Comparison of Wavelet-Based Three-Dimensional Model Coding Techniques”, IEEE Trans. Circuits and Systems for Video Technology, vol. 14, nr. 7 (special issue on AFX), p. 937-949, July 2004.
17. I. A. Salomie, A. Munteanu, A. Gavrilescu, G. Lafruit et al.: “MeshGrid – A Compact, Multiscalable and Animation-Friendly Surface Representation”, IEEE Trans. Circuits and Systems for Video Technology, vol. 14, nr. 7, p. 950-966, July 2004.
18. J. Royan, R. Balter and C. Bouville: “Hierarchical Representation of Virtual Cities for Progressive Transmission over Networks”, Proc. Intl. Symp. 3D Data Processing, Visualization and Transmission (3DPVT) 2006, p. 432-439, June 2006.
19. Joint Photographic Experts Group (JPEG; formally ISO/IEC JTC 1/SC 29/WG 1): “ISO/IEC 10918‑1:1994, Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines”, ISO/IEC IS, February 1994. This is the first and most recent Edition of this IS, commonly known as “JPEG”.
20. JPEG: “ISO/IEC 15444‑1:2016, Information technology – JPEG 2000 image coding system: Core coding system”, ISO/IEC IS, October 2016. This is the third and most recent Edition of this IS, of which the first was published in December 2000.
21. G. Lafruit, E. Delfosse, R. Osorio, W. van Raemdonck et al.: “View-Dependent, Scalable Texture Streaming in 3D QoS with MPEG-4 Visual Texture Coding”, IEEE Trans. Circuits and Systems for Video Technology, vol. 14, nr. 7, p. 1021-1031, July 2004.
22. MPEG: “ISO/IEC 14496‑16:2011/Amd 1:2011, Efficient representation of 3D meshes with multiple attributes”, ISO/IEC IS, November 2011. This is the first Amendment to the fourth Edition of AFX, and will eventually be integrated in its fifth.
23. E. S. Jang, J. D. K. Kim, S. Y. Jung, M.-J. Han et al.: “Interpolator data compression for MPEG‑4 animation”, IEEE Trans. Circuits and Systems for Video Technology, vol. 14, nr. 7, p. 989-1008, July 2004.
24. K. Mamou, T. Zaharia and F. Prêteux: “A skinning approach for dynamic 3D mesh compression”, Computer Animation and Virtual Worlds, vol. 17, nr. 3-4, p. 337-346, July 2006.
25. T. K. Capin, I.-S. Pandzic, N. Magnenat Thalmann and D. Thalmann: “Virtual Human Representation and Communication in VLNet”, IEEE Computer Graphics and Applications, vol. 17, nr. 2, p. 42-53, March 1997.
26. M. Preda, I. A. Salomie, F. Prêteux and G. Lafruit: “Virtual character definition and animation within the MPEG‑4 standard”, p. 27-69 in M. Strintzis and N. Sarris (eds.): “3D modeling and animation: Synthesis and analysis techniques for the human body”, IRM Press, 2004.
27. M. Preda, B. Jovanova, I. Arsov and F. Prêteux: “Optimized MPEG‑4 animation encoder for motion capture data”, Proc. ACM Intl. Conf. 3D Web Technology (Web3D), p. 181-190, April 2007.
28. J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes: “Computer Graphics: Principle and Practice in C (2nd ed.)”, Addison-Wesley, 1997. There is a 3rd ed. of this book from 2014, but its author list and table of contents differ substantially from those of the first two.
29. G. Farin: “Curves and Surfaces for CAGD: A Practical Guide (5th ed.)”, Morgan Kauffman, 2001.
30. D. Berjón, F. Morán and S. Manjunatha: “Objective and subjective evaluation of static 3D mesh compression”, Elsevier Signal Processing: Image Communication, vol. 28, nr. 2, p. 181-195, February 2013.

1. ⬃ AFX contains tools for the efficient coding of solids, and of surfaces approximated by Bézier’s or NURBS patches, and for the joint description of the shape and appearance of a 3D object, such as the ones contained in the Depth Image Based Representation (DIBR) and DIBR version 2 toolsets, but they are not addressed in this white paper. [↑](#footnote-ref-1)
2. ⬃ 3D mesh reconstruction errors are typically measured with both the L2 (i.e., RMS) and L∞ (i.e., Hausdorff’s or maximum) distances, and expressed in sub-units (10-4 is common) of the original mesh bounding box diagonal. [↑](#footnote-ref-2)