An MPEG-4 Tool for Composing 3D Scenes

Petros Daras, Ioannis Kompatsiaris, and Theodoros Raptis
Informatics and Telematics Institute

Michael G. Strintzis
Aristotle University of Thessaloniki

MPEG-4's complicated format makes developing scenes from scratch all but impossible for novice users. By converting MPEG-4's text-based description into graphical form, the authors' proposed tool exploits all of MPEG-4's 3D functionalities while easing the authoring burden.

In addition to efficient compression, MPEG-4 offers multimedia content authors many novel capabilities, such as letting them code audio-visual objects rather than frames. It also integrates 2D and 3D content and human face- and body-specific features and separates elementary stream transmission for individual audio-visual objects. Implementing these capabilities is complicated, requiring that authors use several functionalities, from encoding audio and visual scene descriptions to implementing different delivery scenarios.

History teaches us that however powerful the underlying technologies, the success of multimedia computing systems depends on their ease of authoring. As the "MPEG-4 Tools" sidebar explains, existing MPEG-4 authoring tools help users create 2D scenes, but if authors want to create 3D scenes, they have few options.

To address this, we've developed an authoring tool that fully exploits MPEG-4's 3D functionalities, integrating unique features such as update commands and facial animation. Our tool helps even MPEG-4 novices create scenes that are totally MPEG-4 compliant, which are almost impossible for nonexperts to build from scratch using only text. Using our tool, authors can insert basic 3D objects, such as boxes, spheres, cones, cylinders, and text, and modify their attributes. Our tool also lets authors:

- create or insert and modify generic 3D models using the `IndexedFaceSet` node;
- control object behavior using various sensors (time, touch, cylinder, sphere, and plane) and interpolators (color, position, and orientation);
- texture map static images and video on 3D objects;
- modify the scene's temporal behavior by adding, deleting, or replacing nodes over time using the Update commands; and
- add and animate synthetic faces.

Our tool is based on an open and modular architecture that can progress with MPEG-4 versions and is easily adaptable to newly emerging, higher-level authoring features. The tool is available for download at http://media.iti.gr/MPEG4/.

MPEG-4 overview

MPEG-4 is an audio-visual representation standard with two primary objectives: to support new ways of communication, access, and interaction with digital audio-visual data, and to offer a common technical solution to various service paradigms—such as telecommunications, broadcast, and interactive applications—whose borders are rapidly disappearing. MPEG-4 builds on digital television's success,[1] as well as that of synthetic content development from interactive graphics applications[2] and content distribution and access methods from interactive multimedia such as the World Wide Web.[3]

MPEG-4 addresses application needs in several fields, including Internet video, multimedia broadcasting, content-based audio-visual database access, games, advanced audio-visual communications (notably over mobile networks), and remote monitoring and control.

MPEG-4 audio-visual scenes are composed of several media objects organized hierarchically. At the hierarchy's leaves, we find primitive media objects including still images (a fixed background, for example), video objects (a talking person without the background), audio objects (the voice associated with this person), and so on. Apart from natural objects, MPEG-4 lets authors code objects that are 2D and 3D, synthetic and hybrid, and audio and visual. As Figure 1 shows, this object coding enables content-based interactivity and scalability.[4]

In MPEG-4 systems, these audio-visual objects

1070-986X/04/$20.00 © 2004 IEEE

Published by the IEEE Computer Society

MPEG-4 Tools

MPEG-4 authoring is undoubtedly a challenge. Far from the relative simplicity of MPEG-2's one-video-plus-two-audio-streams, MPEG-4 lets content creators spatially and temporally compose numerous objects of many different types, including rectangular video, arbitrarily shaped video, still image, speech synthesis, voice, music, text, and 2D and 3D graphics. MPEG-Pro,[1] the most well-known MPEG-4 authoring tool, includes a user interface, binary format for scenes (BIFS) update, and a timeline, but can handle only 2D scenes. Another MPEG-4-compliant authoring tool[2] also composes only 2D scenes.

Other MPEG-4-related algorithms segment and generate video objects, but don't provide a complete MPEG-4 authoring suite.[3-6] Commercial multimedia authoring tools, such as IBM Hotmedia (http://www-4.ibm.com/software/net.media) and Veon (http://www.veon.com), are based on proprietary formats rather than widely acceptable standards.

References

1. S. Boughoufalah, J.C. Dufourd, and F. Bouilhaguet, "MPEG-Pro, an Authoring System for MPEG-4," *IEEE Int'l Symp. Circuits and Systems* (ISCAS 2000), IEEE Press, 2000, pp. 175-178.
2. V.K. Papastathis, I. Kompatsiaris, and M.G. Strintzis, "Authoring Tool for the Composition of MPEG-4 Audiovisual Scenes," *Int'l Workshop on Synthetic Natural Hybrid Coding and 3D Imaging*, 1999; available at http://uranus.ee.auth.gr/IWSNHC3DI99/proceedings.html.
3. H. Luo and A. Eleftheriadis, "Designing an Interactive Tool for Video Object Segmentation and Annotation," *ACM Multimedia*, ACM Press, 1999, pp. 265-269.
4. P. Correia and F. Pereira, "The Role of Analysis in Content-Based Video Coding and Interaction," *Signal Processing J.*, 1998, vol. 26, no. 2, pp. 125-142.
5. B. Erol and F. Kossentini, "Automatic Key Video Object Plane Selection Using the Shape Information in the MPEG-4 Compressed Domain," *IEEE Trans. Multimedia*, vol. 2, no. 2, June 2000, pp. 129-138.
6. B. Erol, S. Shirani, and F. Kossentini, "A Concealment Method for Shape Information in MPEG-4 Coded Video Sequences," *IEEE Trans. Multimedia*, vol. 2, no. 3, Aug. 2000, pp. 185-190.

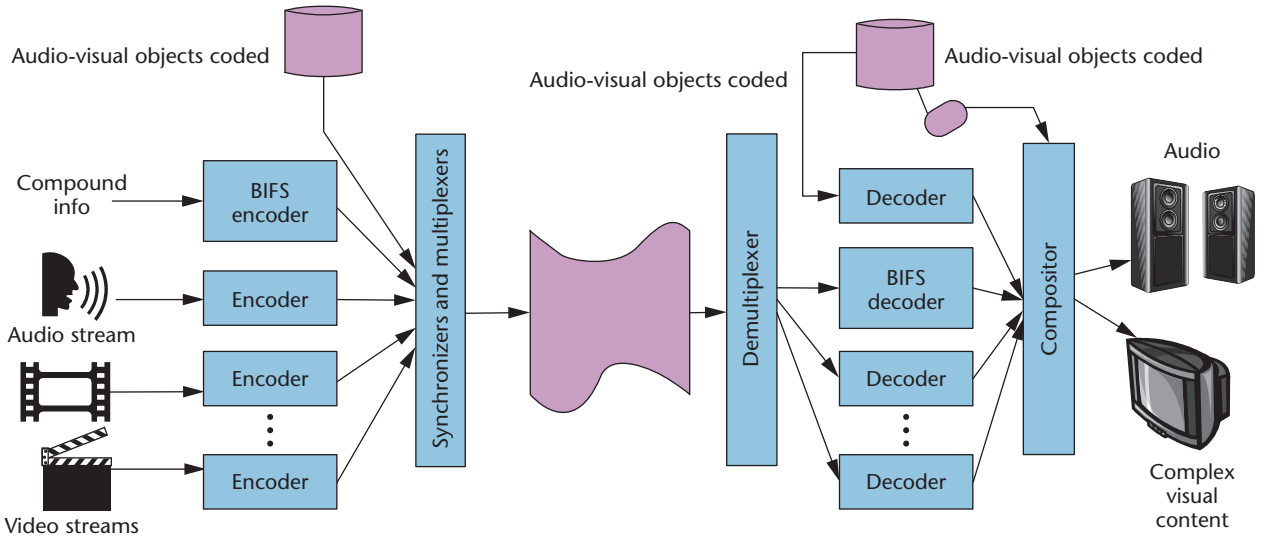


Figure 1. MPEG-4 systems overview.

are decoded from elementary streams and organized into a presentation.[5] The coded stream describing the spatial-temporal relationships between the coded audio-visual objects is called the *scene description*, or binary format for scenes (BIFS) stream. MPEG-4 extends Virtual Reality Modeling Language (VRML)[6] scene description to include coding and streaming, timing, and 2D and 3D object integration. Furthermore, the Extensible MPEG-4 Textual format (XMT)[7] provides an exchangeable format among content authors, while also preserving the authors' intentions in a

high-level text format. In addition to providing a suitable, author-friendly abstraction of the underlying MPEG-4 technologies, XMT also respects existing author practices including HTML and Web3D's Extensible 3D (X3D). Other 3D scene description and authoring frameworks, such as the X3D Graphics specification (http://www.web3d.org), are in development.

MPEG-4 provides a large, rich toolset for coding audio-visual objects.[8] To ensure that the standard is effectively implemented, subsets of the MPEG-4 systems, visual, and audio toolsets have been identified for use with specific applications. These *profiles* limit the tool set a decoder must implement. For each profile, the standard sets one or more levels that restrict the computational complexity. Profiles exist for various types of media content (audio, visual, and graphics) and for scene descriptions. Our tool is compliant with the following profile types: the simple facial animation visual profile, the scalable texture visual profile, the hybrid visual profile, the natural audio profile, the complete graphics profile, the complete scene graph profile, and the object descriptor profile, which includes the object descriptor tool.

Binary format for scenes (BIFS)

The BIFS description language[9] extends the VRML 2.0 specification,[6] which was designed for use on the Internet, intranets, and local client systems. Authors can use VRML in various application areas, including engineering and scientific visualization, multimedia presentations, entertainment and educational titles, Web pages, and shared virtual worlds. Advanced BIFS (version 2.0, which will be included in MPEG-4 2.0) will be a VRML superset that authors can use to compress VRML scenes. MPEG-4 Systems 2.0 supports all VRML nodes.

BIFS extends the base VRML specification in several ways. First, it offers new media capabilities in the scene. Among these are 2D nodes containing 2D graphics and 2D scene graph description, and the ability to mix 2D and 3D graphics. Also, its new audio nodes support advanced audio features including source mixing, streaming audio interface, and synthetic audio content creation. BIFS also offers face- and body-specific nodes to link to specific animation streams and nodes that link to the streaming client-server environment, including media time sensors and back channel messages.

Second, BIFS offers binary scene encoding, which permits efficient scene transmission. Finally, as we describe in more detail below, BIFS supports two specific protocols to stream scene and animation data:

- The *BIFS-Command protocol* lets authors send synchronized scene modifications with a stream.
- The *BIFS-Anim protocol* permits continuous streaming of the scene's animation.

BIFS information types

BIFS contains the following four information types:

- media object attributes that define an object's audio-visual properties;
- the structure of the scene graph that contains these objects;
- the objects' predefined spatio-temporal changes, independent of user input; and
- the spatio-temporal changes that user interaction triggers.

Audio-visual objects have both temporal and spatial extent. Temporally, all objects have a single dimension: time. Spatially, objects can be 2D or 3D. Each object has a local coordinate system, in which it has a fixed spatio-temporal location and scale (size and orientation). Authors position objects in the scene by specifying a coordinate transformation from the object-local coordinate system into another coordinate system defined by a parent node. The coordinate transformation that locates an object in a scene is a scene attribute, rather than an object attribute. Therefore, the system must send the scene description as a separate *elementary* stream.

Elementary streams are a key notion in MPEG-4. A complete MPEG-4 presentation transports each media/object in a different elementary stream. Such streams are composed of access units (such as a video object frame) that are packetized into sync layer packets. Some objects might be transported in several elementary streams, such as when scalability is involved. This is an important feature for bitstream editing, which is one of MPEG-4's content-based functionalities.

Scene description structure

As Figure 2 shows, we can represent the scene

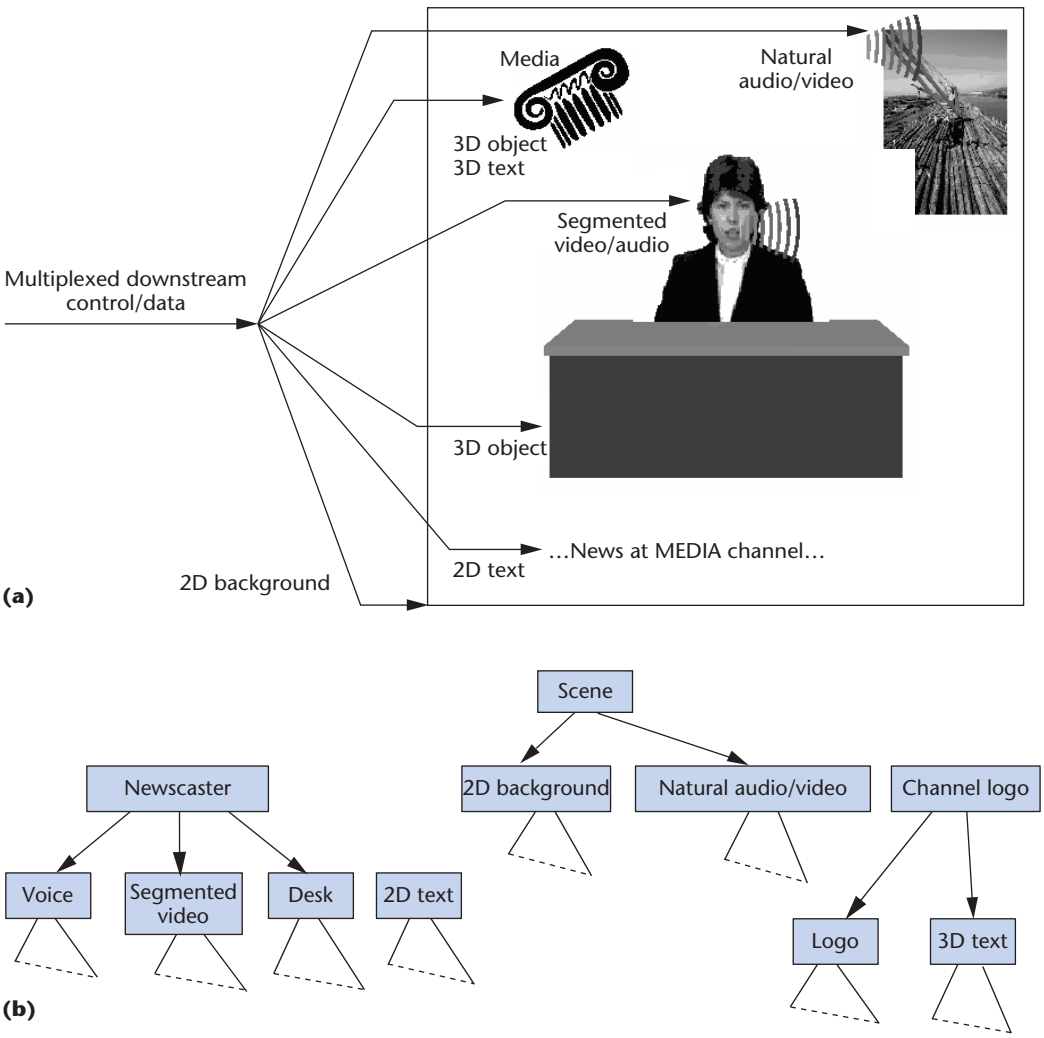


Figure 2. Scene description structure. (a) An example MPEG-4 scene, and (b) the corresponding scene tree.

description's hierarchical structure as a tree. Each node of the tree is an audio-visual object. Complex objects are constructed using appropriate scene description nodes.

The tree structure is not necessarily static. The relationships can evolve over time and authors can delete, add, or modify nodes. Individual scene description nodes expose a set of parameters that let users control several aspects of the nodes' behavior. Examples include a sound's pitch, a synthetic visual object's color, and a video sequence's speed. The scene description structure makes a clear distinction between the audio-visual object itself, the attributes that enable control of its position and behavior, and any elementary streams that contain coded information representing object attributes.

The scene description doesn't directly refer to elementary streams when specifying a media object; instead it uses the concept of *object descriptors* (ODs). The OD framework identifies and properly associates elementary streams with the scene description's media objects. This often requires elementary stream data to point to an OD using a numeric identifier—an OD ID. Typically, however, these pointers aren't to remote hosts, but to elementary streams that the client is receiving. ODs also contain additional information, such as quality-of-service parameters.

Each OD contains one or more *ES descriptors*, which describe the elementary streams comprising a single media object. An ES descriptor identifies a single stream with a numeric identifier—an ES ID. In the simplest case, an OD contains just one ES descriptor that identifies, for example, an audio stream that belongs to the `AudioSource` node by which this OD is referenced.[10] That same OD could be referenced from two distinct scene description nodes. A single OD might also contain

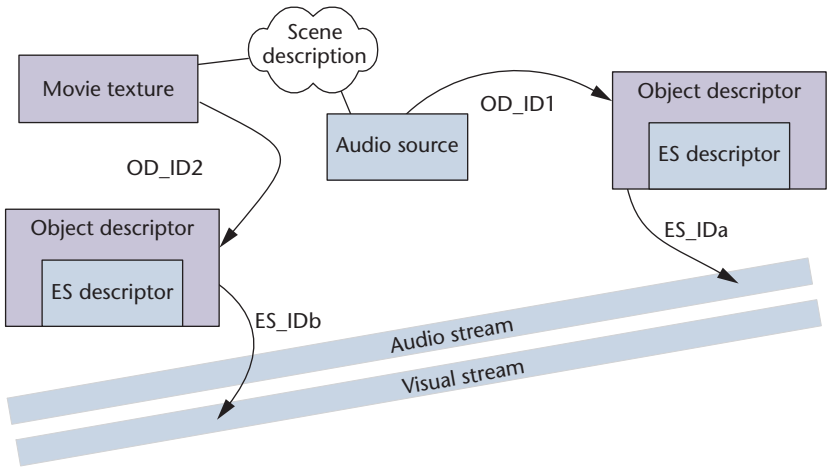


Figure 3. Two scene description nodes and their object descriptors. Because the `AudioSource` and `MovieTexture` nodes are of different types, they must use distinct object descriptors.

two or more ES descriptors, such as one identifying a low bit-rate audio stream and another identifying a higher bit-rate audio stream for the same content. In such cases, the user can choose between the two audio qualities. With audio, it's also possible to have multiple audio streams with different languages for users to choose from. Generally, a single OD can advertise all kinds of different resolution or bit-rate streams representing the same audio or visual content, thus offering a choice of quality. In contrast, streams that represent different audio or visual content must be referenced through distinct ODs. For example, as Figure 3 shows, the **`AudioSource`** and **`MovieTexture`** nodes refer to different elementary streams and must use two distinct ODs.

Nodes and fields

Every MPEG-4 scene is constructed as a directed acyclic graph of nodes:

- Grouping nodes construct the scene structure.
- Children nodes are offspring of grouping nodes and represent the scene's multimedia objects.
- Bindable children nodes are a type of children node; only one bindable children node can be active in a scene at any given time. In a 3D scene, for example, multiple viewpoints can be contained, but only one viewpoint (or camera) can be active at a time.
- Interpolator nodes are another children nodes subtype that represents interpolation data to perform key frame animation. These nodes generate a values sequence as a function of time or other input parameters.
- Sensor nodes sense the user and environment changes for interactive scene authoring.

BIFS and VRML scenes are both composed of a node collection arranged in a hierarchical tree. Each node represents, groups, or transforms a scene object and consists of a list of fields that define the node's particular behavior. A **`Sphere`** node, for example, has a radius field that specifies the sphere's size. MPEG-4 has roughly 100 nodes with 20 basic field types that represent the basic field data types: Boolean, integer, floating point, 2D and 3D vectors, time, normal vectors, rotations, colors, URLs, strings, images, and other more arcane data types such as scripts. The "MPEG-4 Nodes" sidebar lists the most common MPEG-4 nodes, including those that our tool supports.

Routes and dynamical behavior

BIFS' event model uses VRML's *route* concept to propagate events between scene elements. Routes are connections that assign the value of one field to another field. As with nodes, authors can assign routes a name so that they can identify specific routes for modification or deletion. Routes combined with interpolators can animate a scene. For example, the system could route an interpolator's value to a **`Transform`** node's rotation field, causing the nodes in the **`Transform`** node's children field to rotate as the values in the interpolator node's corresponding field change over time. Figure 4 (on page 64) shows an implementation of this, which lets authors add interactivity and animation to the scene.

BIFS-Command

MPEG-4 is designed for use in broadcast applications as well as interactive and one-to-one communication applications. To meet this requirement, MPEG-4 BIFS contains a new concept in which the application itself is a temporal stream. This means that the presentation (the

MPEG-4 Nodes

BIFS and VRML scenes are both composed of a collection of nodes arranged in a hierarchical tree. Each node represents, groups, or transforms an object in the scene and consists of a list of fields that define the node's particular behavior. For each node category here, we've included a list of the binary format for scenes (BIFS) nodes, and put the nodes that our MPEG-4 authoring tool fully supports in bold-faced type.

- *Children nodes* are direct children of grouping nodes. They can represent a geometry (shape), sound nodes, lighting parameters, interpolators, sensors, and grouping nodes. This category contains all grouping, sensor, and interpolator nodes, and all bindable children nodes, as well as the BIFS nodes below. In addition, valuator children nodes are classified in three categories: nodes usable for both 2D and 3D children, 2D-specific nodes, and 3D-specific nodes. BIFS nodes in this category include `AnimationStream`, `Conditional`, **`Face`**, `QuantizationParameter`, `Script`, `Shape`, `TermCap`, `WorldInfo`, `Sound2D`, `DirectionalLight`, `PointLight`, **`Sound`**, and `SpotLight`.

- *Dynamic-content-related nodes* enable the inclusion of media in the scene: audio, video, animation, or scene updates. BIFS nodes in this category include `Anchor`, `AnimationStream`, `AudioClip`, `AudioSource`, **`Background`**, `Background2D`, **`ImageTexture`**, `Inline`, and **`MovieTexture`**.

- *FBA nodes* are those related to face and body animation. They contain one child node (Face), and the rest are attributes for the Face node. BIFS nodes in this category include **`Face`**, `FaceDefMesh`, `FaceDefTables`, `FaceDefTransform`, `FDP`, `FIT`, and `Viseme`.

- *Miscellaneous attributes* are features of the children nodes that are represented by specific nodes, except FBA-, media-, or geometry-specific attributes. Attributes nodes are classified in three categories: those usable for both 2D and 3D attributes, 2D-specific nodes, and 3D-specific nodes. BIFS nodes in this category include **`Appearance`**, **`Color`**, `FontStyle`, `PixelTexture`, `Coordinate2D`, `Material2D`, `Coordinate`, **`Material`**, `Normal`, `TextureCoordinate`, and `TextureTransform`.

- *Top nodes* are nodes that can be put at the top of an MPEG-4 scene. BIFS nodes in this category include **`Group`**, `Layer2D`, `Layer3D`, and `OrderedGroup`.

- *Grouping nodes* have a field containing a list of children nodes. Each grouping node defines a coordinate space for its children that is relative to the coordinate space of the group node's parent node. Transformations thus accumulate down the scene graph hierarchy. Grouping nodes are classified in four subcategories: nodes usable for both 2D and 3D grouping, 2D-specific nodes, 3D-specific nodes, and audio-specific nodes. BIFS nodes in this category include **`Group`**, `Inline`, `OrderedGroup`, `Switch`, `Form`, `Layer2D`, `Layout`, `Transform2D`, `Anchor`, `Billboard`, `Collision`, `Layer3D`, `LOD`, **`Transform`**, `AudioBuffer`, `AudioDelay`, `AudioFX`, `AudioMix`, and `AudioSwitch`.

- *Interpolator nodes* perform linear interpolation for key frame animation. They receive as input a key and output a value interpolated according to the key value and the reference points' value. Interpolator nodes are classified in three categories: nodes usable for both 2D and 3D interpolation, 2D-specific nodes, and 3D-specific nodes. BIFS nodes in this category include **`ColorInterpolator`**, `ScalarInterpolator`, **`PositionInterpolator`**, `CoordinateInterpolator`, `NormalInterpolator`, **`OrientationInterpolator`**, and `Position Interpolator2D`.

- *Sensor nodes* detect events in their environment and fire events. A `TouchSensor`, for example, detects a mouse click, and a `ProximitySensor` detects the user's entry into a particular region. Sensor nodes are classified in three categories: nodes usable for both 2D and 3D sensors, 2D-specific nodes, and 3D-specific nodes. BIFS nodes in this category include `Anchor`, **`TimeSensor`**, **`TouchSensor`**, `DiscSensor`, `PlaneSensor2D`, `ProximitySensor2D`, `Collision`, **`CylinderSensor`**, **`PlaneSensor`**, `ProximitySemsor`, **`SphereSensor`**, and `VisibilitySensor`.

- *Geometry nodes* represent a geometry object and are classified in nodes usable for 2D- or 3D-specific scenes (all 2D geometry can also be used in 3D scenes). BIFS nodes in this category include `Bitmap`, `Circle`, `Curve2D`, `IndexedFaceSet2D`, `IndexedLineSet2D`, `PointSet2D`, `Rectangle`, **`Text`**, **`Box`**, **`Cone`**, **`Cylinder`**, `ElevationGrid`, `Extrusion`, **`IndexedFaceSet`**, `IndexedKineSet`, `PointSet`, **`Sphere`**, and `Backgound2D`.

- *Bindable children nodes* represent scene features for which exactly one node instance can be active at any instant. For example, in a 3D scene, exactly one `Viewpoint` node is always active. For each node type, a stack of nodes is stored, with the active node on top of the stack. Events can trigger a particular node. 2D-specific nodes are listed first, followed by 3D-specific nodes. BIFS nodes in this category include **`Background`**, `Fog`, `ListeningPoint`, `NavigationInfo`, and `Viewpoint`.

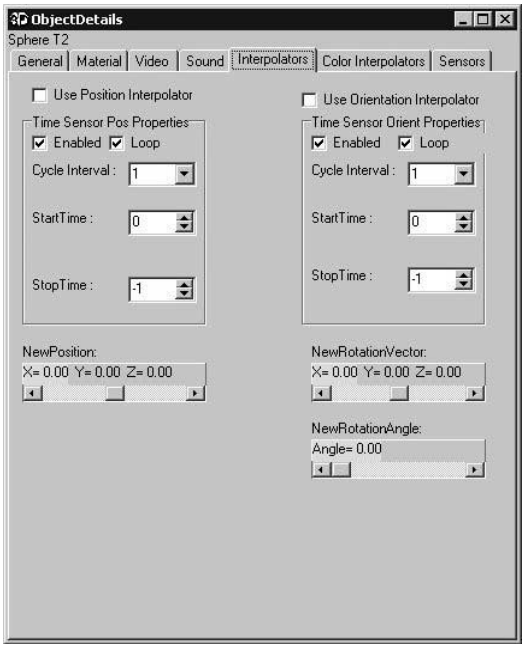


Figure 4. The interpolators panel lets authors add interactivity and animation using routes and interpolators.

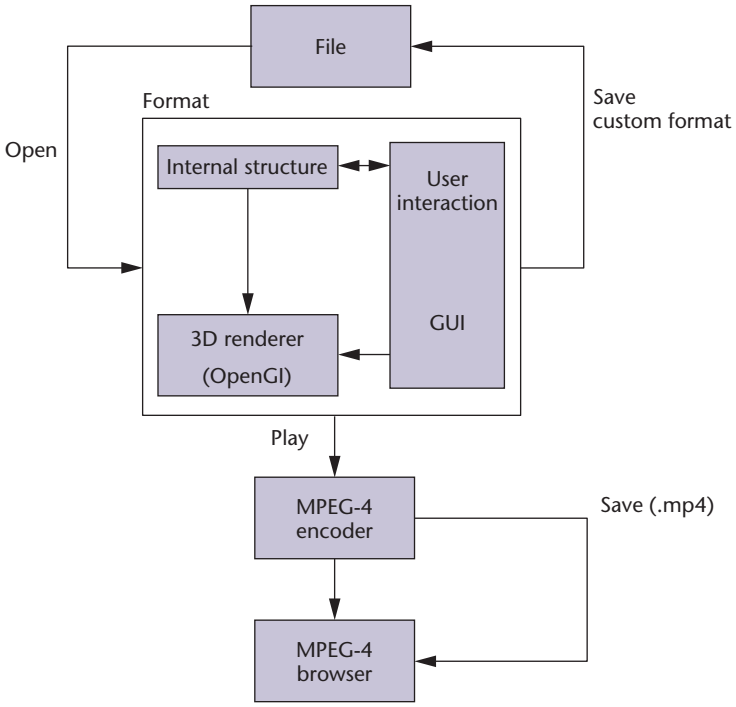


Figure 5. System architecture. Creating content involves four basic stages: open, format, play, and save.

scene itself) has a temporal dimension. The Web's model for multimedia presentations is that users download a scene description (such as an HTML page or VRML scene) once, and then play it locally. In the MPEG-4 model, a BIFS presentation (which describes the scene itself) is delivered over time: the initial scene is loaded and updates follow. In fact, the initial scene loading itself is considered an update. MPEG-4's scene concept therefore encapsulates the elementary streams that convey it over time.

Because the mechanism that provides BIFS information to the receiver over time comprises the BIFS-Command protocol (also known as BIFS-Update), the elementary stream that carries it is called the *BIFS-Command stream*. BIFS-Command conveys commands for scene replacement, node addition or deletion, field modification, and so on. A `ReplaceScene` command, for example, becomes a BIFS stream's entry (or random access) point, just as an Intra frame serves as a video's random access point. BIFS commands come in four main functionalities: scene replacement, node/field/route insertion, node/value/route deletion, and node/field/value/ route replacement. We implemented the BIFS-Command protocol to let users temporally modify the scene using the authoring tool's user interface.

Facial animation

BIFS facial and body animation nodes let authors render an animated face. The facial definition parameters (FDP) and the facial animation parameters (FAP) control the face's shape, texture, and expressions. Initially, the face object contains a generic face with a neutral expression. Authors can change definition parameters to alter the face's appearance from something generic to something with a particular shape and (optionally) texture. They can also download a complete face model via the FDP set. In addition to rendering the face, authors can use the bitstream's animation parameters to animate it with expression, speech, and so on. Our application implements the `Face` node using the generic MPEG-4 3D face model, which lets users insert a synthetic 3D animated face and its associated FAP files. Although researchers have presented several FAP extraction[11-13] and 3D motion-estimation algorithms,[14] no other authoring suite integrates those synthetic faces into a complete scene.

The MPEG-4 authoring tool

As Figure 5 shows, we can characterize MPEG-4 content creation as a development cycle with four stages: open, format, play, and save. As we describe in more detail below, after opening a file, the author can format an existing scene or create a new scene by

- clicking the appropriate icon to insert 3D objects, such as spheres, cones, cylinders, text, boxes, and background (see Figure 6);
- deleting objects or modifying their attributes, such as 3D position, size, color, and so on;
- adding realism to the scene by associating image and video textures with the inserted objects;
- duplicating inserted objects using the copy-and-paste functionality;
- grouping objects to simultaneously change their attributes;
- inserting sound and video streams;
- adding interactivity using interpolators (for object motion, periodic color changes, and so on) and sensors (for interactivity between objects; for example, when one object is clicked, a new one is inserted);
- controlling the scene dynamically using the BIFS-Command protocol (such as indicating that a specific scene segment, or object group, appears 10 seconds after the initial scene loads); and
- using the `IndexedFaceSet` node to create or insert generic 3D models and modify them.

During the creation process, authors store the BIFS-defined object attributes and commands in an internal program structure, which is continuously updated depending on the user's actions. At the same time, the author can view a real-time, 3D scene preview in an integrated OpenGL window (see Figure 6). OpenGL (http://www.opengl.org) is a software interface to graphics hardware that renders 2D and 3D objects into a frame buffer. OpenGL describes these objects as sequences of vertices (for geometric objects) or pixels (for images) and converts the data to pixels to create the final desired image in the buffer.

Once the scene is formatted, the tool plays the created content by interpreting the commands issued in the editing phase. This lets the author check the current description's final presentation. The author can then save the file in either a custom format or, after encoding/multiplexing and packaging it, in an MP4 file[8] (the standard MPEG-4 file format). The MP4 file format stores an MPEG-4 presentation's media information in a flexible, extensible format that facilitates the media's interchange, management, editing, and presentation.

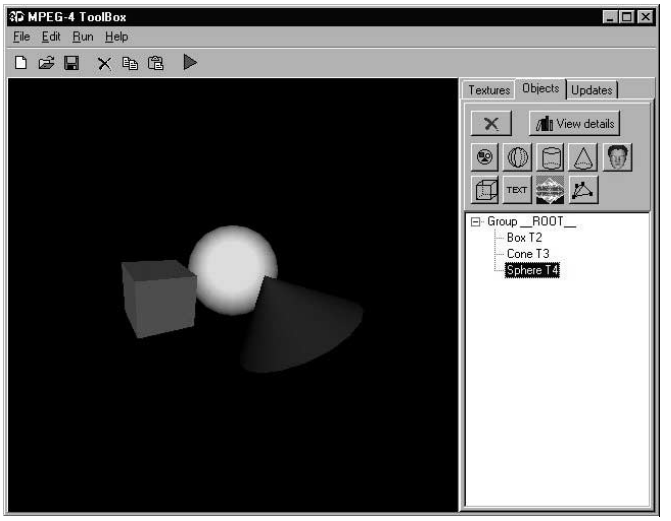


Figure 6. The tool's main window. The buttons on the right let authors insert 3D objects (shown), change texture, and update commands.

User interface

To improve the creation process, authors need powerful tools.[15] Multimedia applications' temporal dependence and variability can hinder authors from obtaining an accurate view of what they're editing. To address this, we used OpenGL to create an environment with multiple, synchronized views. As Figure 6 shows, the interface is composed of three main views: edit/preview, scene tree, and object details.

The *edit/preview* window integrates the presentation and editing phases in the same view, which presents authors with a partial result of their created objects in an OpenGL window. If they insert a new object into the scene, the window displays it immediately in the exact 3D position specified. However, if authors assign a particular behavior to an object (such as assigning a video a texture), the preview window shows only the first frame (they can see the full video only when they play the scene). If an object already has one texture (such as a video texture) and an author tries to add another texture (such as an image texture), a warning message appears.

The *scene tree* view (the bottom right box in Figure 6) provides a structural view of the scene

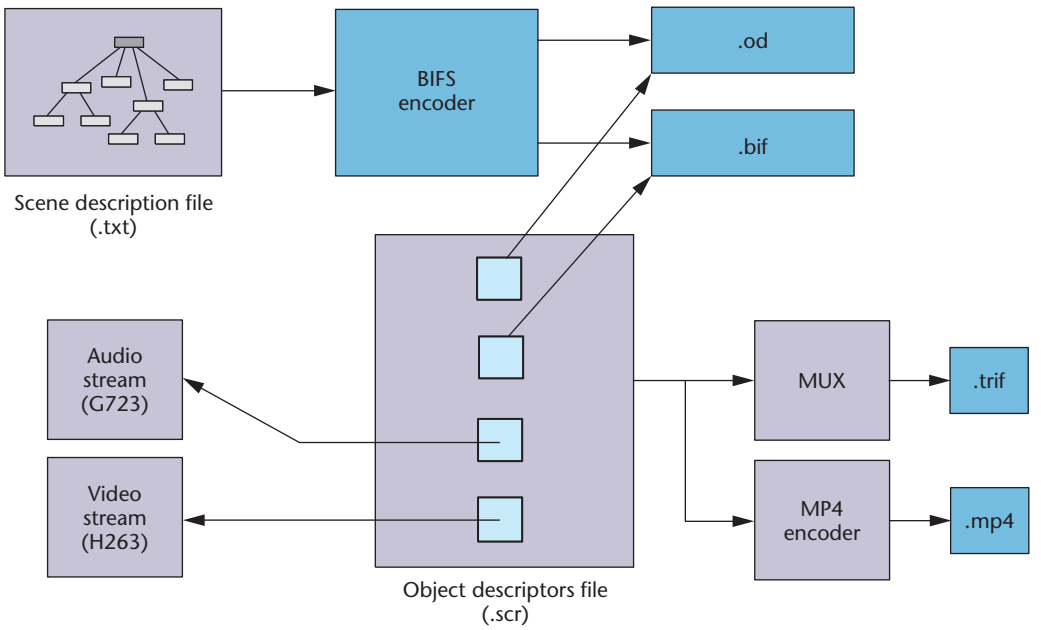


Figure 7. Scene generation and the MPEG-4 IM1 Implementation Group's reference software. The tool automatically creates scene description files (.txt files) and object-descriptor list files (.scr files) while authors work. To create binary files, the tool uses the BIFS encoder and the tiled raster interchange format (TRIF) multiplexer (MUX).

as a tree (although a BIFS scene is a graph, we display it as a tree for ease of presentation). The scene tree view gives authors more detailed information on object behaviors, which can't be displayed in the edit view. Authors can also use drag-and-drop and copy-paste operations in this view.

In the *object details* window (the top right box in Figure 6), authors can use object properties to assign values to an object beyond those offered by default. These properties include 3D position, rotation, and scale; color (diffuse, specular, and emission); shine and texture; video and audio streams (transmitted as two separate elementary streams, according to the OD mechanism); cylinder and cone radius and height; text style and fonts; sky and ground background and texture; and, to add interactivity and animation, interpolators (color, position, orientation) and sensors (sphere, cylinder, plane, touch, and time).

Furthermore, this view also lets authors insert, create, and manipulate generic 3D models using the `IndexedFaceSet` node. Authors can easily insert simple VRML files and, using the `Face` node, synthetically animated 3D faces. The author must provide a FAP file[15] and the corresponding encoder parameter file (EPF), which gives the FAP encoder all the information related to the corresponding FAP file (such as I and P frames, masks, frame rate, quantization scaling factor, and so on). The tool then creates a binary format for animation (BIFA) file for the scene description and OD files to use.

Scene building

As Figure 7 shows, while authors are continuously changing a particular node's fields using the tool's dialogue boxes, the program automatically creates two files:

- *Scene description file* (.txt file). This file is similar to VRML because developers used VRML's nodes set as MPEG-4's initial composition nodes set.
- *OD list file* (.scr file). This file identifies and names elementary streams that it can then refer to in a scene description and attach to individual audio-visual objects.

Once authors have created these two text files, they must construct suitable binary files, which authors can process locally or transmit to the receiver side via the network. To create such files, authors can use software provided by the MPEG-4 Implementation Group (IM1): to construct the BIFS/binary file (.bif file) from the BIFS/text file, authors use the BifsEncoder (BifsEnc), and then use the MP4Enc multiplexer to create the final MPEG-4 file. Authors can now save or view the scene on the MPEG-4 player.

Implementation

We developed the 3D MPEG-4 authoring tool using C/C++ for Windows—specifically, Builder C++ 5.0 and OpenGL—interfaced with IM1's software platform's core module and tools (see

Figure 7). IM1's 3D player is a software implementation of an MPEG-4 systems player.[16] The player is built on top of IM1's core framework, which includes tools to encode and multiplex test scenes. The player aims to be compliant with the complete 3D profile.

The core module provides the infrastructure for full implementation of MPEG-4 players.[17] It includes support for all functionalities (demultiplexing, BIFS and OD decoding, scene construction, update, and so on). The modules use decoding and composition buffers to manage synchronized data flow between the multiplexer, the decoders, and the compositor. The module supports plug-ins for the decoder's API, the Delivery Multimedia Integration Framework (the MPEG layer that delivers content over various networks and media), and intellectual property management and protection (IPMP). It also provides functionality for `MediaObject`, which is the base class for all specific node types.

The core module is the foundation layer for customized MPEG-4 applications. It contains hooks for plugging all kinds of decoders (including JPEG, Advanced Audio Coding, H.263, and G.723) and customized compositors. Written in C++, the core module's code is platform independent and developers have used it as the infrastructure for applications that run on Windows or Unix. The module includes a Windows "console" test application that reads a multiplexed file—output by the tiled raster interchange format (TRIF) multiplexer, described below—which contains scene description and media streams. The test application then produces two files. One shows each composition unit's presentation time—that is, the time when a plug-in compositor would receive the composition unit (CU) for presentation, compared to the composition time stamp attached to the encoded unit. The other file shows textual presentation of the decoded BIFS and OD. The software tools include a BifsEnc and TRIF multiplexer.

The BifsEnc reads a textual scene description, scene updates, and OD stream commands (which might include OD and IPMP objects), and produces two binary files—BIFS file and an OD stream.[18] We used the BifsEnc to encode the textual output of our authoring tool. It also produces two files; both have the same name as the input file, but one has the .bif extension and the other has the .od extension. It also produces a text file with the input file's name and the .lst extension. This file lists all the input lines, each followed by error descriptions (if any) and a textual description of the binary encoding.

The TRIF multiplexer is a software tool developed by Zvi Lifshitz that reads a set of files containing an MPEG-4 elementary stream, and multiplexes them into one bitstream according to TRIF specifications. The TRIF multiplexer can also encode a bootstrap OD (`InitialObjectDescriptor`) and place it at the beginning of the multiplexed file. The MP4Enc multiplexer reads MPEG-4 elementary streams and multiplexes them into a single MP4 file, which is based on the TRIF multiplexer and the MP4 file format API `libisomp4.lib`. To verify compliance with MPEG-4 bitstreams,[19] we use Im1Player (for 2D scenes) and 3D player (for 3D scenes). These tools input MP4 files and produce text files that describe the file's content. The tools' output includes full text descriptions of all elementary streams (BIFS, OD) that the tool processes.

Our authoring tool provides a front-end user interface to the MPEG-4 IM1 software and produces the .txt and .scr files that IM1 uses as inputs to the BIFS and MPEG-4 (and MUX) encoders, respectively (see Figure 7).

Examples: Creating scenes

To show how our tool works, we'll create two scenes: an ancient Greek temple and a virtual studio.

Building a temple

Figure 8 (next page) shows the construction of an example scene: an ancient Greek temple—made up of several groups of cylinders and boxes—that continuously rotates around its *y*-axis. We can create this temple through several relatively simple steps.

Creating the temple's facade. We first create a vertical cylinder, changing its position and scaling to make it similar to a temple column. We then use a copy-paste operation in the scene tree view to create a second identical column and reposition it in the desired place (we'll create more columns later, when the temple's facade is complete). Next, we build the temple's roof: we create a box and reposition it on top of the two columns, making the box's *z*-dimension equal to the column's diameter. We then create a second box, resize and rotate it, and place it on top of the first box. We rotate this second box about 45 degrees around its *z*-axis, then duplicate it; by changing its *z*-axis rotation vector with a symmetric negative value, we create two similar anti-

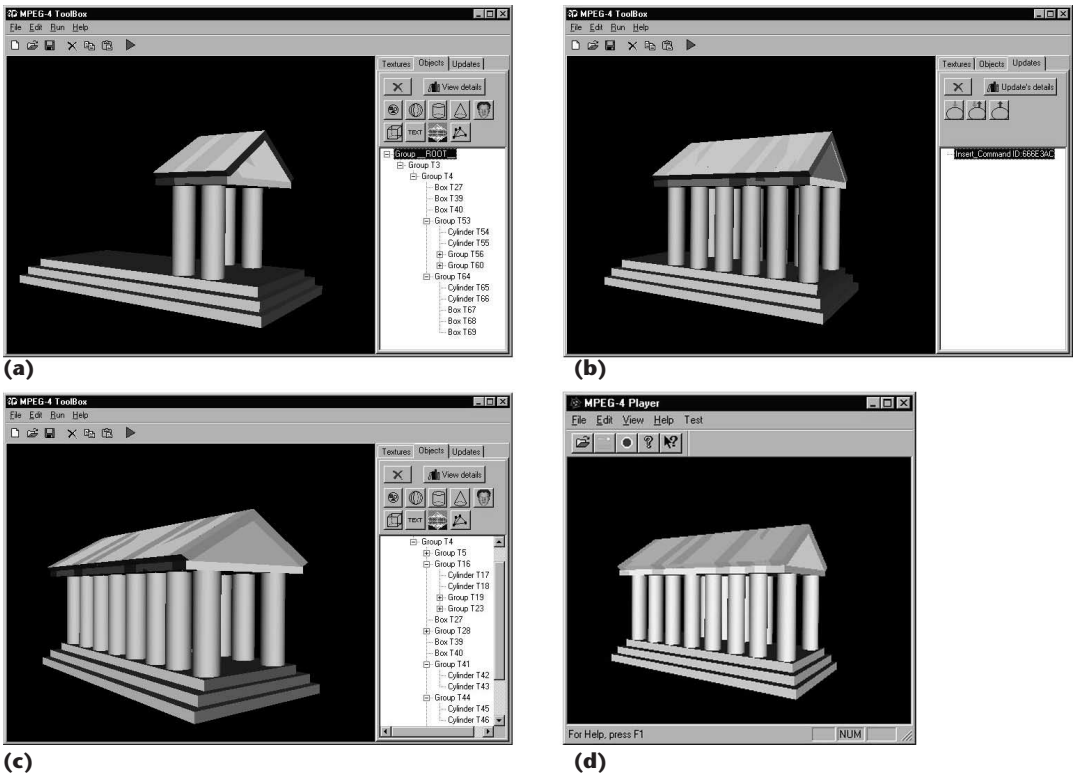


Figure 8. An ancient Greek temple. (a) The temple's facade is created using copy-paste operations in the scene tree view. (b) To achieve a gradual presentation, the author selects the Insert command on the Updates tab, then, in the scene tree's main window (c) selects the object groups defined earlier and copies them. (d) The temple as viewed through the MPEG player.

symmetric boxes. The roof now looks like the extrusion of an isosceles triangle. The front of the temple is complete (see Figure 8a).

Duplicating scene portions. The temple's facade is an important part of the building's geometry, and by duplicating it twice we can create the temple's back and middle sections. Given this, we create a group object using a drag-and-drop operation in the scene tree view, including all items from the temple's facade. Grouping the objects makes them easier to manipulate as a set. We can then easily create the remaining temple portions by copying and pasting the object group several times, taking care to adjust the groups' *z*-positions so that the *z*-values of the front and back sections are symmetrical.

Final details. At this point, we must fill in the roof's gaps. To do this, we create identical boxes and place them between the roof's front and middle portions and its middle and back portions. We can either do this from scratch or by slightly duplicating parts of the roof (we could duplicate the roof's front part, for example, and then reposition and scale it toward its *z*-aspect).

The temple now needs only a floor, which we create using a stretched box. We can then add more specific details to the created objects, including texture and color.

Adding gradual presentation. To demonstrate the historic process of temple construction, we want to use a gradual presentation. To achieve this, we use BIFS-Commands (updates) as follows:

1. We select the Insert command on the Updates tab (see Figure 8b).

2. In the scene tree's main window, we select the object groups that we defined earlier for gradual presentation and copy them (see Figure 8c).

3. We select the General tab on the Update Command Details panel (see Figure 9) and paste the object group. We then specify the group using the Set Target button and specify the timing using the Time of Action button (in this case, 500 milliseconds).

4. Finally, we press Play to see the results in a 3D MPEG-4 player.

Adding movement. We can animate the scene using interpolators. First, we group all the

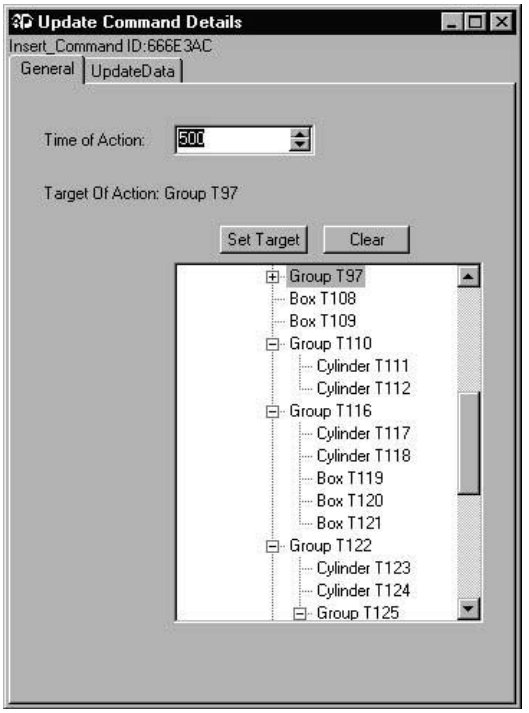


Figure 9. The General tab on the Updates Command Details Panel lets authors select an object group and specify its time of play.

objects in a global group object, then set it in motion by activating its interpolator properties. In the Interpolators menu, we check the Orientation Interpolator property and then make selections to rotate the object around its *y*-axis. We can achieve more complicated movements as needed by placing group nodes inside each other and activating each one's interpolator properties.

To view our results, we can either select the Preview/Play button in the interface's toolbar, or save the scene so we can view it externally with a MPEG-4 player. Every scene that our tool produces is fully compatible with the MPEG-4 BIFS standard and can be presented by any MPEG-4 player capable of reproducing BIFS.

Building a virtual studio

Our second scene represents a virtual studio. The scene contains several groups of synthetic objects, including a synthetic face, boxes with textures, text objects, and **`indexedfacesets`** (see Figure 10). The **`logo`** group, located in the studio's upper left corner, is comprised of a rotating box and a text object that describes the channel's name. The background contains four boxes with image textures (the left and right sides, the floor, and the back side). We created the desk

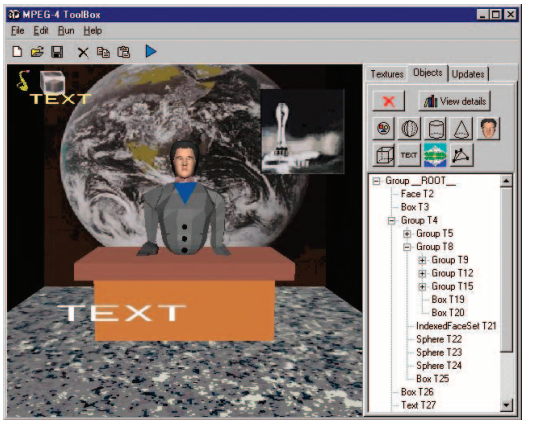


Figure 10. The virtual studio scene in the authoring tool. The scene includes several object groups, including textured boxes and a synthetic face.

using another two boxes. A box with video texture is in the scene's upper right corner. We loaded an H.263 video on this box. The newscaster's body is an **`indexedfaceset`** imported from a VRML 3D model; we inserted the 3D face using the Insert button. Finally, we inserted a rolling text of headlines. After we selected an FAP file and audio stream (for the saxophone in the upper left corner), we configured the face to animate according to the FAP file. The tool transmits the video stream (H.263) and audio stream (G.723) as two separate elementary streams according to the OD mechanism.

We implemented all the animation (except the face) using interpolator nodes. Figure 11 (next page) shows a major part of the scene description text file.

The **`AnimationStream`** node reads the FAP file from an external source. We inserted a **`Transform`** node before the **`Face`** node to control the animated face's position in the scene. The **`Face`** node inserts the animated face and connects it with the FAP file defined earlier. To create the logo in the upper left corner (and, more specifically, the textured rotating box) we first define the box's position (**`Transform`** node) and then the texture image (appearance and texture fields). We then define the object's geometry and dimensions (**`Geometry`** node). In our case, the object is a box. To create the rotating motion, we first define the motion period (how fast the box will rotate) and whether the rotation speed will be constant. This is controlled by the **`TimeSensor`** node and the loop and cycleInterval fields. The **`Orientation-Interpolator`** node defines the motion's intermediate positions. Finally, the **`Route`** nodes

```
    DEF ID_014     AnimationStream      #fap animation stream
    {
        url 50
    }
Transform {
        translation 0.000 1.529 1.690
        rotation 0.000 0.000 0.000 0.000
        scale 0.013 0.013 0.013
    Children Face                       #face node
    {
        fap DEF ID_104 FAP{}
        renderedFace
    }
}

DEF T120661744 Transform {
    translation 0.000 0.000 0.000
    rotation 1.786 1.014 0.000 0.911
    children Shape {
        appearance Appearance {
        texture ImageTexture {
            url 10
    }
        textureTransform TextureTransform {
    }
}
        geometry Box {                  #box with image texture
            size 0.796 0.796 0.694
        }
    }
}

DEF OrientTS120658180                   TimeSensor {
    stopTime -1
    startTime 0
    loop TRUE                           # time sensor for interpolation
                                        # purposes
    cycleInterval 15
}

DEF ORI120658180     OrientationInterpolator {
    key0, 1
    keyValue0.000 0.000 0.000 0.000 ,0.000 0.200 0.000 3.143
}
. . .
ROUTE OrientTS120658180 .fraction_changed TO ORI120658180.set_fraction
ROUTE ORI120658180 .value_changed TO T120661744 .rotation
```

Figure 11. The scene description text file for the virtual studio.

connect the movement's defined parameters to the textured object. The tool's definition (`DEF`) nodes uniquely characterize the object. For example, the texture box is object T120661744. Figure 12 shows the completed scene.

Conclusion

We found that while content developers were satisfied with the tool's efficiency and effectiveness, users who were unfamiliar with MPEG-4 had problems understanding the terminology we used. We thus need to further develop and refine the tool for large-scale deployment.

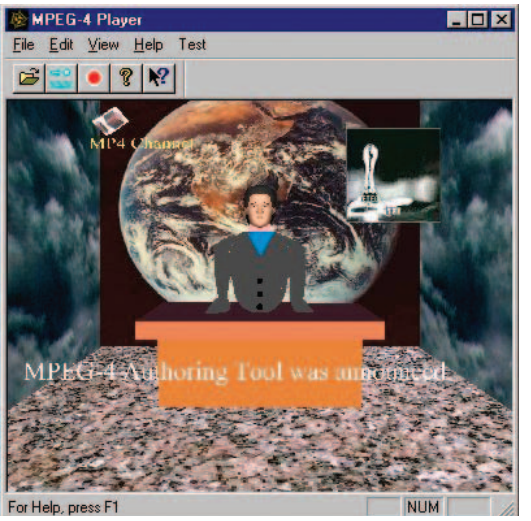


Figure 12. The complete virtual studio, viewed through an IM1 3D player.

Because our authoring tool produces MPEG-4-compliant scenes, users can visualize them using the IM1-3D player without modifications. Users can thus use the tool to create MPEG-4-compliant applications without introducing proprietary features.

Our tool can help MPEG-4 algorithm and system developers integrate their algorithms and make them available through a user-friendly interface. It can also serve as a beginning for developing new tools. Finally, our authoring tool can be a benchmark for comparing other or proprietary authoring tools with a tool that has MPEG-4 system capabilities.

In all, 2,000 visitors have accessed our tool's Web page (averaging eight visitors a day). More than 60 visitors have also registered to use the tool and have provided useful feedback about its functionality. Most visitors seem to be associated with research laboratories that deal with MPEG-4, including Deutsche Telecom, Norway's Telenor, France Telecom, British Telecom, Telecom Italia Lab, Sony Semiconductors and Devices Europe, and Philips Digital Broadcast Systems Group. **MM**

References

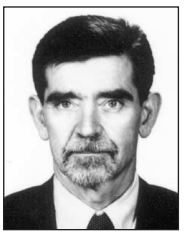
1. C. Herpel and A. Eleftheriadis, "Tutorial issue on MPEG-4," *Signal Processing: Image Communication*, vol. 15, nos. 4-5, 2000.
2. R. Koenen, "MPEG-4 Multimedia for our Time," *IEEE Spectrum*, vol. 36, no. 2, Feb. 1999, pp. 26-33.
3. L. Chiariglione, "MPEG and Multimedia Communications," *IEEE Trans. Circuits and Systems for Video Tech-*

nology, vol. 7, no. 1, Feb. 1997, pp. 5-18.
4. F. Pereira, "MPEG-4: Why, What, How, and When," *Signal Processing: Image Communication*, vol. 15, 2000, pp. 271-279.
5. MPEG-4 Systems, "ISO/IEC 14496-1: Coding of Audio-Visual Objects: Systems, Final Draft International Standard," ISO/IEC JTC1/SC29/WG11 N2501, Oct. 1998.
6. ISO/IEC 14472-1, "The Virtual Reality Modeling Language," http://www.vrml.org/, 1997.
7. M. Kim, S. Wood, and L.T. Cheok, "Extensible MPEG-4 Textual Format (XMT)," *ACM Multimedia-2000*; available at http://www.acm.org/sigs/sigmm/MM2000/ep/michelle/index.html.
8. R. Koenen, "MPEG-4 Overview (V.16 La BauleVersion)," ISO/IEC JTC1/SC29/WG11 N3747, Int'l Standards Organization, Oct. 2000.
9. J. Signæes, Y. Fisher, and A. Eleftheriadis, "MPEG-4's Binary Format for Scene Description," *Signal Processing: Image Communication*, vol. 15, nos. 4-5, 2000, pp. 321-345.
10. E.D. Shreirer, R. Vaananen, and J. Huopaniemi, "AudioBIFS: Describing Audio Scenes with the MPEG-4 Multimedia Standard," *IEEE Trans. Multimedia*, vol. 1, no. 3, June 1999, pp. 237-250.
11. F. Lavagetto and R. Pockaj, "The Facial Animation Engine: Toward a High-Level Interface for the Design of MPEG-4 Compliant Animated Faces," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, no. 3, Mar. 1999, pp. 277-289.
12. G.A. Abrantes and F. Pereira, "MPEG-4 Facial Animation Technology: Survey, Implementation, and Results," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, no. 3, Mar. 1999, pp. 290-305.
13. H. Tao et al., "Compression of MPEG-4 Facial Animation Parameters for Transmission of Talking Heads," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, no. 3, Mar. 1999, pp. 264-276.
14. I. Kompatsiaris, D. Tzovaras, and M.G. Strintzis, "3D Model-Based Segmentation of Videoconference Image Sequences," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 8, no. 5, Sept. 1998, pp. 547-561.
15. B. MacIntyre and S. Feiner, "Future Multimedia User Interfaces," *Multimedia Systems*, vol. 4, no. 5, Oct. 1996, pp. 250-268.
16. Z. Lifshitz, *Status of the Systems Version 1, 2, 3 Software Implementation*, tech. rep., ISO/IEC JTC1/SC29/WG11 N3564, Int'l Standards Organization, July 2000.
17. Z. Lifshitz, *Part 5 Reference Software Systems* (ISO/IEC 14496-5 Systems), tech. rep., ISO/IEC JTC1/SC29/WG11 MPEG2001, Int'l Standards Organization, Mar. 2001.
18. Z. Lifshitz, *BIFS/OD Encoder*, tech. rep., ISO/IEC JTC1/SC29/WG11, Int'l Standards Organization, Mar. 2001.
19. Z. Lifshitz, "IM1 Player: A Bitstream Verification Tool," tech. rep., ISO/IEC JTC1/SC29/WG11, Int'l Standards Organization, Mar. 2001.

Petros Daras is an associate researcher at the Informatics and Telematics Institute, Thessaloniki, Greece, and is pursuing a PhD in multimedia from the Aristotle University of Thessaloniki. His main research interests include the MPEG-4 standard, streaming, 3D object search and retrieval, and medical informatics applications. He is a member of the Technical Chamber of Greece.

Ioannis Kompatsiaris is a senior researcher with the Informatics and Telematics Institute, Thessaloniki. His research interests include computer vision, 3D-model-based monoscopic and multiview image sequence analysis and coding, medical image processing, standards (MPEG-4, MPEG-7), and content-based indexing and retrieval. He is a member of the IEEE and the Technical Chamber of Greece.

Theodoros Raptis is an MBA postgraduate student at the Economic University of Athens. His scientific interest is centered around investment analysis in the field of free energy markets. Raptis received a diploma in electrical engineering from Aristotle University of Thessaloniki in 2001; his thesis was on MPEG-4 and the development of the MPEG-4 authoring tool. He is a member of the Technical Chamber of Greece.



Michael G. Strintzis is a professor of electrical and computer engineering at the University of Thessaloniki, and director of the Informatics and Telematics Research Institute, Thessaloniki. His current research interests include 2D and 3D image coding, image processing, biomedical signal and image processing, and DVD and Internet data authentication and copy protection. In 1984, he was awarded one of the Centennial Medals of the IEEE.