ELSEVIER

Signal Processing: *Image Communication* 15 (2000) 321–345

SIGNAL PROCESSING: IMAGE COMMUNICATION

www.elsevier.nl/locate/image

MPEG-4's binary format for scene description

Julien Signès[a,*], Yuval Fisher[b], Alexandros Eleftheriadis[c]

[a]*France Telecom R&D, 1000 Marina Blvd., Suite 300, Brisbane, CA 94005, USA*
[b]*Institute for Nonlinear Science, University of California, San Diego, La Jolla, CA 92093-0402, USA*
[c]*Columbia University, Dept. of Electrical Engineering, 500 West 120th Street, Mail Code 4712, New York, NY 10027, USA*

Abstract

The new MPEG-4 standard provides a suite of functionalities under one standard: streaming multimedia content, good compression, and user interactivity. This paper provides an introduction to the use and internal mechanisms of these functions.

Keywords: MPEG-4; Streaming multimedia; Interactivity; Animation; Scene description

1. Introduction

MPEG-4 is a digital bit stream format and associated protocols for representing multimedia content consisting of natural and synthetic audio, visual, and object data. It is the third (and not fourth) in a series of MPEG specifications that have a history of wide acceptance and use in the marketplace. Like its predecessors, it is ambitious in its scope. It provides coding/compression capability for a rich set of functionalities, including the manipulation of audio and video data, as well as the synthesis of audio, two- and three-dimensional graphics, sophisticated scripting, interactivity, face and body animation, and texture and geometric coding. It allows data to be streamed and provides for interaction between the content server and the browser. It can be used in a broadcast scenario, or in one-on-one interaction both in "push" and "pull" modes. In short, it is extremely flexible and attempts to provide a basis for a wide variety of uses.

All this functionality does not come cheap. The resulting specification is voluminous, to put it kindly. This paper consists of a tutorial of one portion of the MPEG-4 tool set – the Binary Format for Scenes, BIFS. BIFS is the compressed format in which scenes are defined and modified. It is encapsulated on a streaming mechanism that we will not discuss in detail, and it is transported using a protocol that we will almost completely ignore. Even with these omissions, this paper can serve as only a brief introduction to the capabilities and intricacies of BIFS.

While an MPEG-4 BIFS scene has, for the most part, a structure inherited from the Virtual Reality Modeling Language (VRML 2.0 [4]), its explicit bit stream representation is completely different. Moreover, MPEG-4 adds several distinguishing

* Corresponding author.
E-mail addresses: Julien.Signes@thinkone.com (J. Signès), yfisher@ucsd.edu (Y. Fisher), eleft@ee.columbia.edu (A. Eleftheriadis)

0923-5965/00/$ - see front matter
PII: S0923-5965(99)00052-1

mechanisms to VRML: data streaming, scene updates and compression.

MPEG-4 uses a client–server model. An MPEG-4 client (or player, or browser) contacts an MPEG-4 server, asks for content, receives the content, and renders the content. This "content" can consist of video data, audio data, still images, synthetic 2D or 3D data, or all of the above. The way all these different data is to be combined at the receiver for display on the user's screen or playback in the user's speakers is determined by the *scene description*. The content, as well as the scene description, is typically streamed, which means that the client gets little bits of data and renders them as needed. For example, a video can be played as it arrives from the server. This is a notable contrast with VRML where a scene is first completely transferred from the server to the client and only then rendered, leading to long latency. MPEG-4's streaming capability, thus, allows faster rendering of content.

Once a scene is in place, the server can further modify it by using MPEG-4's scene update mechanism, called BIFS-Command. This powerful mechanism can be used to do many things: for example, an avatar can be manipulated by the server within a 3D scene, a video channel can be switched from the server side, news headlines can be updated on a virtual reality marquis, etc. This mechanism can also be used to progressively transmit large scenes, thus reducing bandwidth requirements.

Finally, by combining the updating and streaming mechanisms, BIFS allows scene components to be animated. BIFS animation (or BIFS-Anim) consists of an arithmetic coder that sends a stream of differential steps to almost any scene component. This functionality is not different from what can be achieved with scene updates, but it has a better coding efficiency when the equivalent BIFS commands would be numerous.

The next section contains background information on how BIFS fits into the rest of the MPEG-4 universe as well as on BIFS's ancestor, VRML. The sections that follow describe how BIFS encodes various scene components and how BIFS can be used effectively.

2. How BIFS fits into MPEG-4

An MPEG-4 system can be conceptually decomposed into several components, as shown in Fig. 1. Starting at the bottom of the figure, we can identify the following layers:

- *Deliver (or transport) layer*: The delivery layer is media unaware and delivery aware. MPEG-4 does not define any specific deliver layer. Rather, MPEG-4 media can be transported on existing transport layers such as RTP, TCP, MPEG-2 Transport Streams, H-323 or ATM.
- *Sync or elementary stream layer*: This component of the system is in charge of the synchronization and buffering of compressed media. The sync layer deals with elementary streams. Elementary streams are a key notion in MPEG-4. A complete MPEG-4 presentation transports each medium/object in a different elementary stream. Elementary streams are composed of *access units* (e.g., a video object frame), packetized into Sync Layer (SL) packets. Some media may be transported in several elementary streams, for instance if scalability is involved. This layer is media-unaware and delivery-unaware, and talks to the Deliver layer through something called the DMIF application interface (DAI) [7]. The DAI is only a non-normative conceptual abstraction of the deliver layer and, in addition to the usual session set up and stream control functions, also enables setting the quality of service requirements for each stream. The DAI is network independent.
- *Media or Compression layer*: This is the component of the system performing the decoding of the media: audio, video, etc. Media are extracted from the Sync Layer through the elementary stream interface. Note that BIFS itself also has to go through the compression layer and then be decoded. Another special MPEG-4 medium type is the Object Descriptor (OD). An OD is a structure similar to a URL, containing pointers to elementary streams. Typically, however, these pointers are not to remote hosts, but to elementary streams that have already been received by the client. ODs also contain additional information such as Quality of Service parameters. This

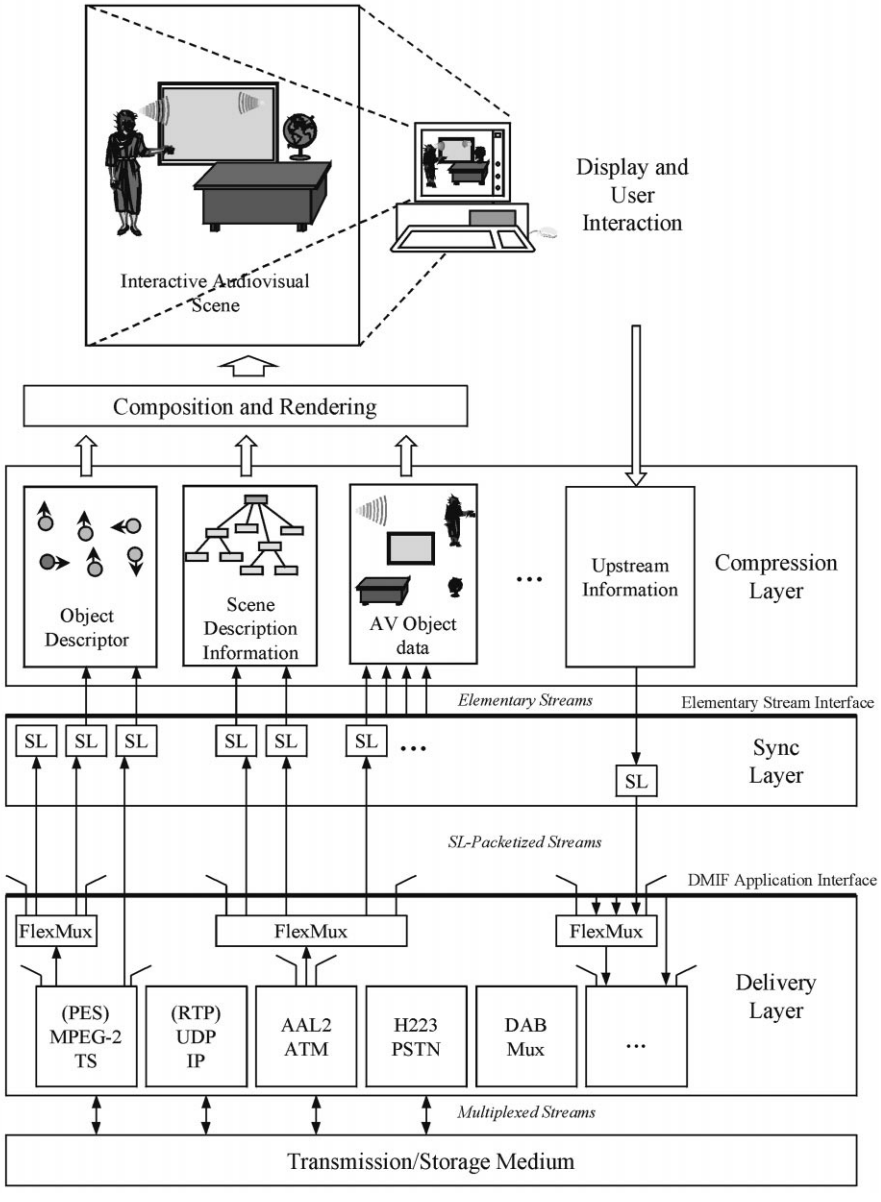

Fig. 1. The architecture of a typical MPEG-4 terminal. BIFS is the scene description information.

layer is media aware, but delivery unaware. The BIFS content is delivered in an elementary stream and is shown in the figure as the "scene description information."

MPEG-4 is a large and complex standard because it addresses a huge problem space: the creation and efficient delivery of compelling interactive audiovisual content across a variety of networks. MPEG-4, however, defines profiles in order to best match specific application areas with subsets of its specifications. The same approach has been used very effectively with MPEG-2. Not all applications need the entire set of MPEG-4 tools,

and it would be a burden to force terminals to fully implement features that they do not require. MPEG-4 thus defines a limited number of subsets of Visual, Audio and Systems tools, to which specific applications can conform. Within each profile, levels may be defined to match the varying complexity of different terminals.

Within the context of this toolkit approach, BIFS can also be seen as a separate tool. While full MPEG-4 implementations would get the maximum benefit from the standard in terms of features and capabilities, it is certainly possible for a given application to comply with certain Scene Graph profiles and levels without using other parts of the MPEG-4 specification. The details of the Scene Graph profiles are beyond the scope of this article, but in broad terms they consist of an audio profile, a simple 2D profile, a complete 2D profile, and a complete profile.

3. BIFS scene description features

The BIFS scene model is very similar to VRML. (In fact, BIFS version 2 will be a super set of VRML and can be used as an effective tool for compressing VRML scenes.) Where VRML is a bit short on functionality, for example, two-dimensional content, MPEG-4 extends it in a VRML-esque way. But whereas VRML is concerned only with scene description, MPEG-4 is concerned also with compact representation and optimized delivery of multimedia content. This section contains a terse introduction to VRML concepts and their extension in BIFS. The reader who wishes to understand the intricacies of MPEG-4 and BIFS is urged to review VRML concepts (see, for example, [4]).

3.1. Scene structure

An MPEG-4 scene is constructed as a direct acyclic graph of nodes. The following types of nodes exist:

- *Grouping nodes* construct the scene structure;
- *Children nodes* are the children of grouping nodes that represent the multimedia objects in the scene.
- *Bindable children nodes* are a specific type of *children nodes* that represent the node types for which only one instance of the node type can be active at a time in the scene (a typical example of that is the Viewpoint for a 3D scene. A 3D scene may contain multiple viewpoints or "cameras", but only one can be active at a time).
- *Interpolator nodes* are another subtype of *children nodes* that represent interpolation data to perform key frame animation. These nodes generate a sequence of values as a function of time or other input parameters.
- *Sensor nodes* sense the user and environment changes for authoring interactive scenes.

These nodes are summarized in Section 3.9.

3.2. Nodes and fields

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 particular behavior of the node. For example, a Sphere node has a radius field that specifies the size of the sphere. MPEG-4 has roughly 100 nodes with 20 basic field types for representing the basic field data types: boolean, integer, floating point, two- and three-dimensional vectors, time, normal vectors, rotations, colors, URLs, strings, images, and other more arcane data types such as scripts. Table 1 shows the list of the more common MPEG-4 types.

The node fields are labeled as being of type *field, eventIn, eventOut* or *exposedField*. The *field* label is used for values that are set only when instantiating the node. Fields that can receive incoming events have the *eventIn* label, whereas fields that emit events are labeled as *eventOut*. Finally, some fields may set values but also receive or emit events, in which case they are labeled as *exposedField*. An additional important feature of the node and field structure is that node fields receive default values. When instantiated, the default values are assumed if not specified explicitly in the bit stream.

Node fields sometimes represent one value, as in the radius of the Sphere node, or many values, as in a list of vertices that define a polygon. Thus, each

Table 1
Common MPEG-4 field types

Basic data type	Example	Single-field type	Multiple-field type
Floating point value	1.0	SFFloat	MFFloat
Integer value	1	SFInt32	MFInt32
Time value (64-bit float)	0.1	SFTime	MFTime
Boolean value	TRUE	SFBool	*None*
Color (rgb triplet)	1 0 0 (represents red)	SFColor	MFColor
3-Vector	0.5 1 0	SFVec3f	MFVec3f
2-Vector	0.25	SFVec2f	MFVec2f
Rotation (direction + angle)	0 0 1 3.1415	SFRotation	MFRotation
Image	See VRML spec	SFImage	*None*
String values	"Hello World"	SFString	MFString
Nodes	Sphere { … }	SFNode	MFNode

basic data type can be represented as a single-value type, denoted by type names that begin with an "S", or as a multiple-value type, denoted by type names that begin with an "M". For example, the floating point radius of the Sphere node has type SFFloat, whereas the collection of points with three-dimensions in a Coordinate node that can be used to define a polygon has the type MFVec3f. Nodes are defined using a semantic declaration that contains the node name, followed by a list of the node's fields. The list specifies each field's label, type, name and default value. For example, the Viewpoint node has the following semantic declaration.

Viewpoint {

EventIn	SFBool	**set_bind**	
ExposedField	SFFloat	**FieldOfView**	0.785398
ExposedField	SFBool	**Jump**	TRUE
ExposedField	SFRotation	**Orientation**	0, 0, 1, 0
ExposedField	SFVec3f	**Position**	0, 0, 10
Field	SFString	**Description**	""
EventOut	SFTime	**BindTime**	
EventOut	SFBool	**IsBound**	

}

Many nodes have fields that hold other nodes – this is what gives the scene its tree structure. Whereas VRML has only two node types, SFNode and MFNode, MPEG-4 has a rigidly typed collection of nodes that specifies exactly which nodes can be contained within which other nodes. A node may have more than one type, however, when it can be contained as a child node in different contexts.

For example, the Shape node is used to include a geometric shape in the scene. It has a geometry field that holds any node with type SFGeometryNode, of which there are 18, and an appearance field that holds nodes of type SFAppearanceNode, of which there is only one. An example is shown in Fig. 2. In this example, the geometry field holds a Cube node which is of size 1 in each direction; the appearance field of Shape node holds an Appearance node, which in turn holds a Material node that defines the color associated with the Cube (in this case red).

The example in Fig. 2 gives a textual description of a portion of a scene that is based on VRML. MPEG-4, however, only specifies a binary description of a scene. The translation from a textual to a binary description is almost direct. As an example, consider the simple scene portion in Fig. 3, and the following explanation of its BIFS representation.

```
Shape {
      geometry Cube { size 1 1 1}
      appearance Appearance {
            material Material {
                  diffuseColor 1 0 0
            }
      }
}
```

Fig. 2. A Shape node with two fields, appearance and geometry, containing other nodes.

```
Transform {
  Translation 1 0 0
  Children [
        Shape {
              geometry Cube { size 1 1 1}
              appearance Appearance {
                    material Material {
                          diffuseColor 1 0 0
                    }
              }
        }
  ]
}
```

Fig. 3. A simple VRML scene consisting of a red cube centered at (1, 0, 0).

The BIFS representation of the scene in Fig. 3 would consist of the following:

1. a header that contains some global information about the encoding;
2. a binary value representing the Transform node;
3. a bit specifying that the fields of the Transform node will be specified by their index, rather than in an exhaustive list;
4. the index for the "translation" field;
5. a binary encoding of the SFVec3f value 1 0 0 (since there is no quantization defined here, this encoding consists of three 32-bit values; during decoding, the decoder knows the type of the field it is reading and thus knows how many bits to read and how to interpret them);
6. the index of the children field of the Transform node;
7. the binary representation of the Shape node, which is:
   - 7.1. a binary value for the Shape node;
   - 7.2. a bit specifying that all of the fields of the Shape node and their values will be listed sequentially rather than by index/value pairs;
   - 7.3. a binary representation for the Cube node which is:
     - 7.3.1. a binary value for the Cube node;
     - 7.3.2. a bit specifying that the fields of the Cube will be specified by index;
     - 7.3.3. the index of the "size" field;
     - 7.3.4. a binary encoding of the SFVec3f value 1 1 1;
     - 7.3.5. a bit specifying that no more fields for the Cube node will be sent;
   - 7.4. a binary value for the Appearance node, followed by its encoding, omitted here;
8. a bit terminating the list of fields for the Transform node.

VRML and BIFS both have a mechanism for reusing nodes. For example, once a wheel is defined as a collection of geometric nodes collected inside a Group node, it is possible to reuse the wheel elsewhere in the scene, rather than copying it explicitly wherever it is to appear. In VRML, this is done using the DEF and USE keywords. A node is given a DEF name, and it is inserted into the scene graph wherever the same name appears after a USE statement. In BIFS, a bit is used to determine if the node has an ID, and when this bit is set, it is followed by a certain number of bits that hold an integer ID value. The number of bits used in the scene to specify IDs is given in a header that is sent at the beginning of the scene (in fact, contained in the decoder configuration information within the Object Descriptor through which the BIFS stream was accessed).

3.3. ROUTEs and dynamical behavior

The event model of BIFS uses the VRML concept of ROUTEs to propagate events between scene elements. ROUTEs are connections that assign the value of one field to another field. As is the case with nodes, ROUTEs can be assigned a "name" in order to be able to identify specific ROUTEs for modification or deletion.

ROUTEs combined with interpolators can cause animation in a scene. For example, the value of an interpolator is ROUTEd to the rotation field in a Transform node, causing the nodes in the Transform node's children field to be rotated as the values in the corresponding field in the interpolator node change with time.

Recall that the labels Field, ExposedField, EventIn, EventOut correspond, respectively, to private fields that cannot be ROUTEd at all, fields that can be both input and output in a ROUTE, fields that can only accept input from a ROUTE, and fields that can only serve as output for

```
Group {
    DEF position Transform {
          translation 1 0 0
          children [
                Shape {
                geometry DEF MyHeadline Text {
                      string " news headline here"
                      }
                }
          ]
    }
}
DEF PosInterp PositionInterpolator {
    Key [0 1]
    KeyValye [1 0 0, -1 0 0] # move from x=1 to x=-1
}
DEF MyTime TimeSensor {
    loop TRUE
    stopTime -1
}
ROUTE MyTime.fraction_changed TO PosInterp.key
ROUTE PosInterp.keyValue TO position.translation
```

Fig. 4. A VRML scene portion in which a Text node is moved across the screen using a ROUTE. A TimeSensor node generates successive values that are ROUTEd to the key field of a PositionInterpolator, which converts these input values to interpolated SFVec3f values in the keyValue field. This field is then ROUTEd to the translation field of Transform node, causing the children in the Transform node to be repositioned.

a ROUTE. The VRML ROUTE syntax has the form

ROUTE⟨OutNodeID⟩.outFieldName TO

⟨InNodeID⟩.inFieldName

This syntax means that values from the field with name "outFieldName" in the node with DEF ID ⟨OutNodeID⟩ will be mapped to the field with name "inFieldName" in the node with DEF ID ⟨InNodeID⟩. In MPEG-4, the same data is sent to specify a ROUTE, except that the IDs are integer values and the fields are indexed numerically also, rather than referenced by name.

Fig. 4 shows a VRML scene in which a Text node is moved across the screen repeatedly. This might be part of a scene in which news headlines are scrolled across the bottom of a larger scene. While the ROUTE mechanism has no problem in causing the text to scroll nicely across the screen, the scene is otherwise static. If the news needs to be updated, it is not possible to change it from the server side. However, the BIFS update mechanism allows precisely this. With BIFS update, it is possible to send commands to insert, delete or replace nodes or field values in the scene. The specific syntax of BIFS update is discussed below, but we can think of it textually, as shown in Fig. 5. After this command is sent to the MPEG-4 client, the scene would look almost identical, except with the recent news headline replacing the previous field in the Text node. (In fact, it is not necessary to send a whole new text node. The same effect can be achieved by replacing only the string field of the text node.)

3.4. Streaming scene description updates: *BIFS-Command*

MPEG-4 is designed to be used in broadcast applications and in interactive and one-on-one communication applications. To fit this requirement, an important concept developed within MPEG-4 BIFS is that the application itself can be seen as a temporal stream. This means that the presentation, or the scene itself, has a temporal

```
Update {
      time 10
      action " Replace"
      id " MyHeadline"
      data Text  {
            string " More recent news headline here"
            }
}
```

Fig. 5. A textual representation of a BIFS update that replaces the node with ID "MyHeadline" by a new node with the same ID by with different text content.

dimension. On the web, the model used for multimedia presentations is that a scene description (for instance an HTML page or a VRML scene) is downloaded once, and then played locally. In the MPEG-4 model, a BIFS presentation, which describes the scene itself, is delivered over time. The basic model is that an initial scene is loaded and can then receive further updates. In fact, the initial scene loading itself is considered an update. The concept of a scene in MPEG-4, therefore, encapsulates the elementary stream(s) that convey it over time.

The mechanism with which BIFS information is provided to the receiver over time comprises the *BIFS-Command* protocol (also known as BIFS-Update), and the elementary stream that carries it is thus called BIFS-Command stream. BIFS-Command conveys commands for the replacement of a scene, addition or deletion of nodes, modification of fields, etc. For example, a "ReplaceScene" command becomes the entry (or random access) point for a BIFS stream, exactly in the same way as an Intra frame serves as a random access point for video. A BIFS-Command stream can be read from the Web as any other scene, potentially containing only one "ReplaceScene" command, but it can also be broadcast as a "push" stream, or even exchanged in a communications or collaborative application.

BIFS commands come in four main functionalities: scene replacement, node/field/route insertion, node/value/route deletion, and node/field/value/route replacement. The commands enable the following operations:

- *Replacing the entire current scene with a new one*. When a BIFS Replace Scene command is received, the whole context is reset, and a new scene graph corresponding to the new BIFS scene is constructed.
- *Insertion command:* This command comes in three subtypes:
  - *Node insertion:* Nodes can be inserted into the children field of grouping nodes. This command allows the insertion of a node at the beginning, end, or indexed position in the list of children nodes of an already existing node.
  - *Indexed field insertion:* With this update command, a generic field is inserted into a specified position in a multiple value field.
  - *ROUTE insertion:* This can be used to enable user interaction or other dynamic functionality in the scene by linking event source and sink fields in the scene.
- *Deletion command:* This command also comes in three subtypes, analogous to the insertion command:
  - *Node deletion:* Simply deletes the identified node. Note that it is possible to delete a node that does not have an ID (i.e., it has not been DEF-ed) by using the IndexedValue deletion command (see below).
  - *IndexedValue deletion:* This command allows the deletion of a specified entry in a multiple value field.
  - *ROUTE deletion:* Simply deletes a ROUTE.
- *Replacement command:* This command also comes in the standard three flavors, plus one more for replacing single value field values:
  - *Node replacement:* Replaces an existing node in the scene with a new one.
  - *Field replacement:* Changes the value of a field within a node. This command can be used, for example, to change a color, a position, the vertices of a mesh, or to switch an object on and off.
  - *IndexedValue replacement:* This command is very similar to the field replacement command, except that the field referred to is a multiple valued one, and hence the command also includes indexing information in order to identify the particular value that should be replaced.
  - *ROUTE replacement:* Replaces an existing ROUTE with a new one.

3.5. Animation using interpolators

VRML scenes can display a wide range of complicated dynamical behavior. One way this is achieved is by the use of interpolator nodes that convert an increasing sequence of time events into a sequence of interpolated positions. Fig. 6 shows a scene in which a ball bounces along the steps repeatedly. The ball position is encoded by using

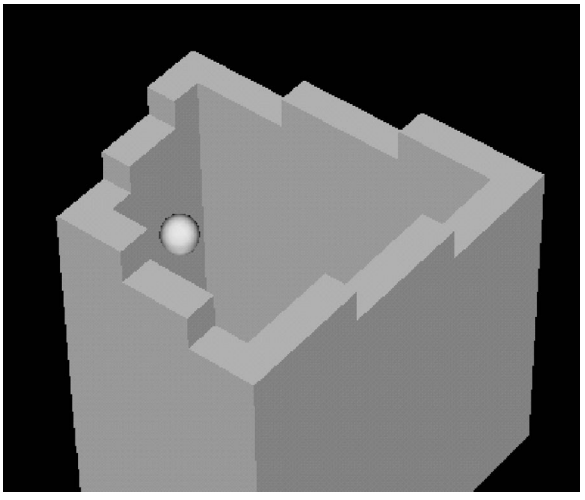

Fig. 6. An animated scene in which a ball bounces on the steps. The animation is generated using interpolators that modify the position of the ball in time. (Scene composed by Diganta Saha.)

the position interpolator shown in Fig. 7. The position interpolator interpolates each ROUTEd time value into a position using the values in its key and keyValue fields. When the motion is complicated, these fields must contain many floating point numbers and thus require a lot of memory.

While this is one way of introducing animation into the scene, it suffers from several drawbacks. First, the animation is fixed – it is part of the scene and cannot be modified. Second, the whole scene has to be downloaded, including the lengthy interpolator fields, before it can be rendered. MPEG-4 combines streaming and updates to create an animation functionality that overcomes these drawbacks.

3.6. *Streaming animations*: *BIFS-Anim*

In the previous section, we described how it is possible to trigger a key frame animation by

```
DEF Ball-Tform-Move Transform {
  children [
.
.
.
        DEF Ball-Move-Timer TimeSensor {loop TRUE  . . . . . },
        Shape {
            appearance Appearance {
                material Material {
                    specularColor      .2 .2 .2
                    diffuseColor       1 1 0
                }
            }
            geometry Sphere{}
        }
    ]
}
.
.
.

DEF Ball-Mover PositionInterpolator {
  key [ 0, .03125, .0625, .09375, . . . . .  ]
  keyValue [ 0 0 0, 0 0 2, 0 -2 4, 0 -2 6, 0 -4 8, . . . . . ]


.
.
.
ROUTE Ball-Move-Timer.fraction_changed TO Ball-Mover.set_fraction
ROUTE Ball-Mover.value_changed TO Ball-Tform-Move.set_translation
```

Fig. 7. A portion of the scene (writen in VRML) shown in Fig. 6 containing the PositionInterpolator node that causes the ball to move around the stairs. By ROUTing the Ball-Mover output values to the translation of the Ball-Tform-Move node, the ball is animated in the scene.

downloading interpolator data in the scene along with an appropriate TimeSensor. This can be used, for example, to modify the fields of a Transform node in order to move its children around. An alternative method is to use BIFS-Command to update the fields of a Transform node. These methods are well suited for small animations. However, when the source of animation is a live one, or when better compression is sought, MPEG-4 provides an alternative way to stream animation to the scene with the *BIFS-Anim* tool. BIFS-Anim enables optimal compression of the animation of all parameters of a scene: 2D and 3D positions, rotations, normals, colors, scalar values, etc.

The BIFS-Anim framework works as follows:

- The BIFS scene is loaded with objects that have been DEF'ed, i.e., assigned a unique node ID.
- An Animation mask is loaded, containing the list of nodes and fields to be animated. When multiple fields are animated, it is possible to select the field to be animated.
- The animation stream itself is streamed, containing animation frames in time-stamped access units.

The field values in the stream can consist of initial values (Intra-frames), used to set or reset the field values, and difference values (Predictive-frames), used to successively modify the previous field value. The Intra- and Predictive-frames are further arithmetically coded to give a highly compressed representation of the animation values.

3.7. Programmability

MPEG-4 Systems supports scripting in the same way that VRML does. However, whereas VRML allows both inlined JavaScript (recently standardized as ECMAScript) as well as URLs pointing to externally available Java class files, MPEG-4 currently only allows JavaScript to be used. Scripts in MPEG-4 have three primary uses:

- Scripts can manipulate the scene directly, for example by routing output values of the script to other parts of the scene.
- Scripts can store state information in local variables. For example, if the last mouse click position is needed, it can be routed to a script that will hold the value until it is required (perhaps by another script).
- Scripts can do type conversion. For example, if only the red component of a color is needed for some operation, a script can extract this value and convert it from a component of an SFColor to an SFFloat.

```
DEF SomeNode Transform { }

Script {
      field SFNode node USE SomeNode
      eventIn SFVec3f pos
      url "javascript:
        function pos(value) {
          node.set_translation = value;
        }"
}
```

Fig. 8. A sample Script node that can access the fields of a Transform node with DEF SomeNode (because a reference to it is included in the Script node). The Script node also has a defined field called pos whose value is copied into the Transform node's translation field.

A script has user-definable fields (with the standard MPEG-4 types and label) that are used for routing values into and out of the script, as well as for storing state information. Scripts can access the fields of any other DEFd node in the scene, as long as a reference to the node is passed to the Script node. Fig. 8 shows a sample script node.

MPEG-4 is currently (as of this writing) defining a Java layer, called MPEG-J, in order to provide additional programmatic control of the terminal to application developers. The model enhances the External Application Interface (EAI) used in VRML, and is expected to be included in Version 2 of MPEG-4. In addition to controlling the scene, an MPEG-J program can access and assess system resources, and act on the network and decoder interfaces. The full details of MPEG-J and MPEG-4's scripting capabilities are beyond the scope of this chapter.

3.8. Name space rules

As mentioned earlier, nodes and ROUTEs in BIFS can receive unique identifiers or IDs. In

MPEG-4 terminals, the following scoping rules apply for the ID name spaces:

- A BIFS-Command stream shares the name space across the whole stream.
- An Inlined BIFS-Command stream (i.e., a BIFS-Command stream accessed through an Inline node[1]) opens a new name space.
- Multiple BIFS-Command streams referenced by the same Object Descriptor structure share the same name space.

The last rule allows building multiple streams of content sharing the same name space. The Object Descriptor hence gathers streams that share the same name space. By grouping streams under the same Object Descriptor, the content creators can signal their intention to allow the pieces of content to share the name space. Obviously, the implication is that the nodes receive a unique identifier across all BIFS-Command elementary streams sharing the same Object Descriptor. This functionality is very useful for building large content packages.

3.9. Summary of MPEG-4 Version 1 BIFS capabilities

The BIFS Node are grouped in semantic categories (Table 2).

3.10. MPEG-4 Version 2 BIFS Extensions

In this section, we list some of the features that are expected to be offered by MPEG-4 Version 2.

3.10.1. Server interaction

One of the key features of Version 2 MPEG-4 BIFS is enabling the transmission of data from the terminal presentation back to the server. This may be useful in particular for many applications such as stream control, initiating transactions, etc. To provide this functionality, MPEG-4 Version 2 will add a ServerCommand node that enables the interactive triggering of messages back to the server. The content of the command is completely application-dependent and is carried back to the server using a normative syntax.

3.10.2. Enhanced sound

In Version 1, MPEG-4 provides the ability to considerably enhance the basic VRML 2.0 sound model. In Version 2, MPEG-4 will go further and provide new sound features:

1. Enhanced physical sound rendering model, so that more natural sound source and sound environment modeling is made possible in BIFS. The functionality of the new sound source includes modeling of air absorption and more natural distance dependent attenuation, as well as sound source directivity modeling. In Version 2, BIFS also takes into account the response of the environment, so that the sound can be rendered to correspond to the visual parts of the scene. For example, polygons representing walls cause sound to be attenuated.
2. An enhanced sound-rendering model, based on geometry-independent perceptual parameters. These perceptual parameters provide high level interfaces to control the "aural aspect" of the sound rendering. Parameters such as the source presence, heaviness, envelope, brilliance and warmth can be controlled.

3.10.3. Scene extensions: prototypes

The BIFS Version 2 specification will provide ways to encode PROTO and EXTERNPROTO. These scene constructs enable the definition of new interfaces to user-constructed scene components. For example, a button PROTO can be constructed which accepts a string label as an input parameter. The body of the PROTO consists of a scene portion that draws a button (using, say, a Box node) and renders the string parameter on the button. The EXTERNPROTO is similar to the PROTO, except that its definition is not part of the scene. Instead, its definition is referenced using a URL. This enables the construction and use of on-line libraries of PROTOs.

Additionally, by assigning an InterfaceCoding Table to the PROTO or EXTERNPROTO nodes, BIFS Version 2 allows the addition of parameters

[1] The Inline node allows an MPEG-4 scene to be included (or "inlined") within another scene.

Table 2
Summary of BIFS Version 1 capabilities

Type of nodes	Function	BIFS Nodes
Grouping Nodes	Grouping nodes have a field that contains a list of children nodes. Each grouping node defines a coordinate space for its children. This coordinate space is relative to the coordinate space of the node of which the group node is a child. Such a node is called a parent node. This means that transformations accumulate down the scene graph hierarchy. Grouping nodes are ordered in four sub categories: nodes usable for both 2D and 3D grouping, 2D specific nodes, 3D specific nodes and Audio specific nodes.	Group Inline OrderedGroup Switch Form Layer2D Layout Transform2D Anchor Billboard Collision Layer3D LOD Transform AudioBuffer AudioDelay AudioFX AudioMix AudioSwitch
Interpolator nodes	Interpolator nodes perform linear interpolation for key frame animation. They receive as an 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	ColorInterpolator ScalarInterpolator PositionInterpolator2D CoordinateInterpolator2D CoordinateInterpolator NormalInterpolator OrientationInterpolator PositionInterpolator
Sensor nodes	Sensor nodes detect events in their environment and fire events. For instance, a TouchSensor detects a click of a mouse, a ProximitySensor detects that the user entered a region of the space. Sensor nodes are classified in three categories: nodes usable for both 2D and 3D sensors, 2D specific nodes, and 3D specific nodes. Interpolator, Sensor and ROUTE statements enable the design of interactive scenes	Anchor TimeSensor TouchSensor DiscSensor PlaneSensor2D ProximitySensor2D Collision CylinderSensor PlaneSensor ProximitySensor SphereSensor VisibilitySensor
Geometry nodes	Geometry nodes represent a geometry object. Geometry nodes are classified in three categories. Nodes usable for 2D specific scenes, and 3D specific scenes. Note that all 2D geometry can also be used in 3D scenes	BitMap Circle Curve2D IndexedFaceSet2D IndexedLineSet2D PointSet2D Rectangle Text Box Cone Cylinder ElevationGrid Extrusion IndexedFaceSet IndexedLineSet PointSet Sphere

Table 2 (Continued)

Type of nodes	Function	BIFS Nodes
Bindable children nodes	These nodes represent features of the scene for which exactly one instance of a node 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. The active node is put on the top of the stack. Activating a particular node can be triggered through events. 2D specific nodes are listed followed by 3D specific nodes	Background2D Background, Fog ListeningPoint NavitationInfo, Viewpoint
Children nodes	Children nodes are direct children of grouping nodes. They can represent a geometry (Shape), sound nodes, lighting parameters, interpolators, sensors, grouping nodes. This category contains: • all grouping nodes, • all sensor nodes, • all interpolator nodes, • all bindable children nodes, as well as the nodes listed to the right Children nodes are classified in three categories: nodes usable for both 2D and 3D children, 2D specific nodes, and 3D specific nodes	AnimationStream Conditional Face QuantizationParameter Script Shape TermCap Valuator WorldInfo Sound2D DirectionalLight PointLight Sound SpotLight
Dynamic content related nodes	These nodes enable the inclusion of media in the scene: Audio, Video, Animation or update of scenes	Anchor AnimationStream AudioClip AudioSource Background Background2D ImageTexture Inline MovieTexture
FBA nodes	FBA nodes are nodes related to face and body animation. They contain one child node (Face), and the rest are attributes for the Face node	Face FaceDefMesh FaceDefTables FaceDefTransform FDP FIT Viseme
Miscellaneous attributes	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. Nodes usable for both 2D and 3D attributes, 2D specific nodes, and 3D specific nodes	Appearance Color FontStyle PixelTexture Coordinate2D Material2D Coordinate Material Normal TextureCoordinate TextureTransform
Top Nodes	Top nodes are the nodes that can be put at the top of an MPEG-4 scene	Group Layer2D Layer3D OrderedGroup

for controlling these new interfaces with BIFS-Anim and BIFS-Command. This way, it is possible to control these interfaces using a stream. It is also possible to assign quantization parameters to the prototype fields, in order to allow efficient compression.

3.10.4. Body animation

Whereas Version 1 allows only the animation of facial models, Version 2 will add body animation capability. The body will be defined in the scene by H-Anim[2] compliant definitions, and will use a specific, optimized encoding algorithm. Furthermore, as for the facial animation, the animation stream will be integrated in the BIFS-Anim framework.

3.10.5. MPEG-J

MPEG-J is a set of Application Programming Interfaces that allow Java code to communicate with an MPEG-4 player engine. By combining MPEG-4 media with safe executable code, content creators may embed complex control and data processing mechanisms with their media data to intelligently manage the operation of the audio-visual session.

MPEG-J defines the following interfaces:

- Scene graph API (similar to the VRML EAI), that enables scene control.
- Network API, to control the up- or down-loading of content.
- Decoder API, to control media decoders.
- Devices API, to control various input devices.
- System capability APIs, in order to be able to assess the status of the system in terms of memory, CPU load, etc.

When downloading an MPEG-J script (or "MPEGlet"), the MPEG-J system can use the interfaces to the system or other application-specific Java APIs to reflect the changes to the scene through the Scene Graph API. For instance, the MPEGlet can query systems capabilities in real time and reflect necessary changes on the scene graph. Another typical example may include data coming from the external application (typically a GUI, a data base interface, etc.) and modifying the scene graph.

3.11. Example of a scene and corresponding scene graph

In this section we show a complete example of a complex 2D/3D scene and its corresponding scene graph. The example illustrates some of the 2D and 3D mixing capabilities of MPEG-4 through the concept of Layers as well as the corresponding scene graph. Layers can be used to determine a region of the screen in which a scene is drawn.

In the example, shown in Figs. 9 and 10, a 2D scene and 3D scene are displayed simultaneously. The 3D scene is displayed with two different viewpoints, using 3Dlayer-1 and 3Dlayer-2. The 2D scene is overlaid on top of the 3D scenes. The 2D scene is also displayed as a map in the 3D scene (in 3Dobj-3). This scene could represent a 3D scene seen from 2 different viewpoints and a 2D map of the world displayed both facing the viewer and in the 3D scene.

4. BIFS compression and streaming

4.1. The BIFS format design goals

The binary format for MPEG-4 scene description attempts to balance several considerations. Compression is, of course, the ultimate goal, but compression conflicts with extensibility, ease of parsing, and a simple specification. The BIFS protocol is a compromise between these goals.

For compression, BIFS uses a compact representation for the scene components. For example, when a scene parameter is specified, the minimal number of bits needed to distinguish that parameter from others is used. This scheme is used for specifying the scene contents also. For example, there are 18 different components that are used to specify the geometry of a scene: Spheres, Cones, Polygons, etc. These are indexed and specified using 5 bits. While this is not as efficient as possible, it makes parsing and specification simpler.

[2] H-Anim is a VRML working group that has defined a set of face and body animation specifications.

```
OrderedGroup{   children [
      DEF 2Dlayer-1 Layer2D { children [
         DEF 2DScene-1 Transform2D { children[
            DEF 2DObj-1 Transform2D { children [
               DEF 2Dobj-3 …
               DEF 2Dobj-4…..
            ]}
            DEF 2Dobj-2 Shape{….}
         ]}
      ]}
      Transform2D { # Transforms to translate and scale the layer
         children [
            DEF 3DLayer-1 Layer3D {
               viewpoint DEF v1 Viewpoint {…}
               children [
                  DEF 3Dscene-1 Transform2D { children [
                     DEF 3DObj-1 Transform { children [
                        DEF 3Dobj-3 Transform{ children USE 2Dscene-1}
                        DEF 3Dobj-4…..
                     ]}
                     DEF 3Dobj-2 Shape {……}
                  ]}
               ]
            }
         ]
      }
      Transform2D {
      # Another set of transforms to translate and scale the layer3D-2
         children [
            DEF 3Dlayer-2 Layer3D {
               Viewpoint DEF v2 Viewpoint {…….}
               children [
                  DEF 3Dscene-2 USE 3DScene-1
               ]
            }
         ]
      }
]}
```

Fig. 9. An example of a combined 2D/3D scene.

The actual parameter values associated with the scene parameters have a different quantization scheme, that is actually part of the scene and thus under the control of the scene author. By default, scene parameter values are not quantized at all; they are stored in their native format (e.g., 32 bits for floats and integers, 1 bit for Booleans, etc.). The scene parameters are classified into different categories, and the values in each category can be linearly quantized using quantization parameters (maximum and minimum values, along with the number of quantization bits) that are specified locally in the scene. The categories consist of parameters that should have similar values, e.g., scaling values, 3D coordinate values, etc. This scheme balances the utility of having local quantization control with the cost of specifying the quantization parameters.

BIFS uses a different quantization scheme in the case of animation. Animation allows scene parameter values to be modified as a function of time. The scene author specifies which node parameters

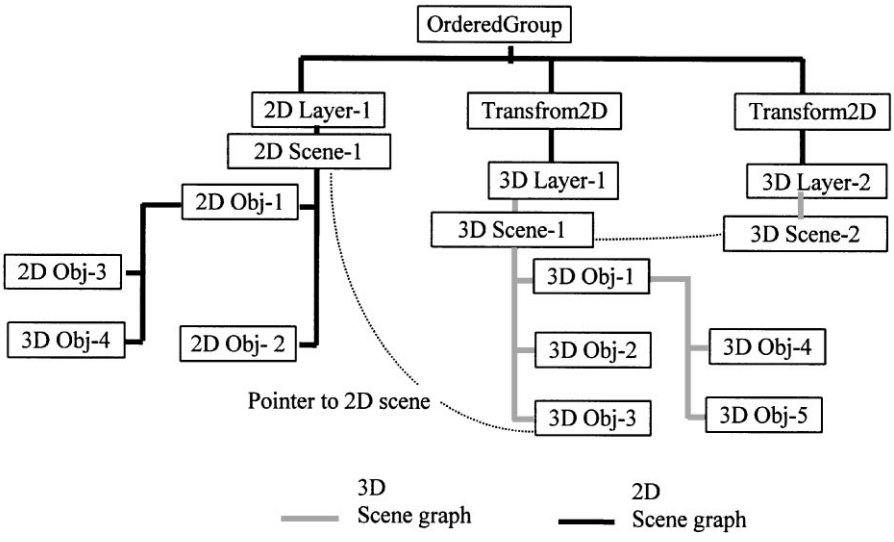

Fig. 10. The scene graph corresponding to the description in Fig. 9.

should be animated by associating these parameters with a stream of input values. The stream consists of a sequence of initial values (I-frames) and successive difference values (P-frames) that are arithmetically encoded. This scheme allows highly efficient encoding of animation values, which can typically compose a large part of a scene. The following sections discuss these notions in more detail.

4.2. BIFS compression

While the BIFS protocol is efficient, it represents a trade-off between bit-stream efficiency on the one hand, and parsing complexity, (relatively) simple specification, and extensibility on the other. In fact, a compressed scene can sometimes be further compressed using existing data compression tools. This is true because some types of data, e.g., Strings, contain redundancy that is not eliminated using BIFS coding (BIFS encodes strings as a collection of characters).

The following sections discuss the components of scene coding in MPEG-4.

4.2.1. Context dependency

BIFS takes advantage of context dependency in order to compress the scene efficiently at the node and the field level. Whereas VRML treats all nodes as being of type SFNode or MFNode, BIFS makes use of the fact that only certain nodes can appear as children of other nodes. This allows the binary specification of the children nodes to be more efficient.

BIFS introduces the concept of a Node Data Type (NDT). There are some 30 different NDTs, and each node belongs to one or more of them. The NDT table consists of a list of nodes for each NDT.[3] In each node data type, the node receives a fixed-binary-length local ID value that corresponds to its position in the NDT table. For example, the Shape node can appear in both a 3D context and a 2D context. It thus has both the SF3DNode and SF2DNode node data type (in fact, as with all nodes, it also has the global SFWorldNode NDT). In a 3D context, the Shape node can appear in the children field of a Transform node – this field has the type MF3Dnode and it accepts nodes of type SF3DNode. In a 2D

[3] Note that the NDT is not related to the MPEG-4 field types. The field types are the atomic values in the scene – integer, floats, etc. A Node Data Type is a collection of nodes that can be children of certain nodes.

context, the Shape node can appear in the children field of a Transform2D node – this field has type MF2DNode. The binary ID code for a Shape node thus depends on the context in which it appears. When it is an SF3DNode, it has a 6-bit binary ID value of 100100 (decimal value 36), because it occupies position number 36 in the list of 48 total SF3DNodes. In the 2D case, its ID is represented by 5-bits with value 10111. There are only 30 different SF2DNodes, requiring only 5 bits to specify them all.

This node representation is quite efficient. It does not make sense to apply entropy-coding techniques based on the probabilistic distribution of nodes in scenes – such data does not currently exist. Moreover, as the ID value 0 is reserved as an escape value for future extensions, and because a fractional bit arises from the fact that the number nodes in each category is not a power of 2, the coding would be slightly sub-optimal regardless.

At the field level, BIFS introduces types that do not exist in VRML. These exist only for coding purposes and are cast into some of the basic field types (see Section 3.2) at runtime:

- The SFURL type is used for representing URLs. VRML uses a string to encode URLs, but BIFS requires a finer resolution and allows URLs to be encoded either as strings or as references to stream IDs (object descriptors).
- The SFScript type is used to encode scripts. VRML uses URLs to hold scripts, which are typically either externally accessible Java class files or in-lined JavaScript (or, more precisely, ECMAScript). In BIFS, only JavaScript is supported and it is represented using a binary format. The SFScript type allows either an MPEG-4 URL to be encoded (which allows full compatibility with VRML) or a binary script representation.
- The SFBuffer type is used to encode buffers in the MPEG-4 Conditional node. This special field holds a binary representation of a BIFS-Update command that can be triggered using the ROUTE mechanism.

4.2.2. The Node Coding Tables

Each node is associated with a Node Coding Table (NCT). Each node's NCT holds information related to how the fields of the node are coded. The NCT specifies the type of values that each field can hold. Thus, for example, the NCT for the Transform2D node specifies that its children field has type MF2DNode and thus can hold a list of SF2DNode nodes. The NCT also specifies the other field types that can occur, for example, SFFloat, SFInt32, etc.

The NCT also specifies an index for each field according to one of four usage categories. These categories are:

- DEF – used for defining field values when a node is transmitted. This corresponds to the Field and ExposedField modes, since these are the only modes that have values that can be specified.
- IN – used for data that can be modified using BIFS updates or ROUTEs. This corresponds to the EventIn and ExposedField modes.
- OUT – is used for the EventOut and ExposedField modes. That is, fields that can be used as input values for ROUTEs.
- DYN – is a subset of the IN category, used for BIFS-Animation, discussed in Section 4.4.

By creating these four categories, a BIFS scene always references fields with the minimal number of bits needed. For example, the IndexedFaceSet node is used to hold a collection of polygons that form a 3D object. It has 13 fields that can be defined when it is created, and thus each field requires 4 bits when it is indexed. However, this node only has four fields that can output values to a ROUTE, so that only two bits are needed to specify which field is being used when this node is routed from. The IndexedFaceSet node has 8 fields that can accept ROUTEd values or be updated using BIFS-Command, and thus these protocols need only 3 bits to specify the modified field. The Node Coding Table for the Fog node is shown in Fig. 11.

The NCT also defines a quantization type for each field that holds numerical values. This is the topic of the next section.

4.2.3. Quantization of node fields

Quantizing BIFS scenes is achieved using the QuantizationParameter node. This node affects the quantization of field values for all nodes that appear hierarchically after or beneath it (i.e., its

Fog	SFWorldNode SF3Dode SFFogNode					0101011 001111 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m,M]***	***Q***	***A***
color	SFColor	00	00	00	0	[0, 1]	4	4
fogType	SFString	01	01	01				
visibilityRange	SFFloat	10	10	10	1	[0, +I]	11	7
set_bind	SFBool		11					
isBound	SFBool			11				

Fig. 11. The Node coding table for the Fog node. The table shows the node name, its three NDTs, and the binary representation of the fog node for each of the NDTs. Following this is a listing of the node's field names, their type, and the binary ID for each of the four usage categories. Some of the fields also have a minimum and maximum value specified, as well as a quantization and animation category (discussed in the following sections).

siblings and children). Using a node to convey the quantization parameters makes use of the already existing BIFS framework to encode the parameters efficiently and takes advantage of specific reuse mechanisms (VRML DEF/USE) in order to reuse locally existing parameters. This mechanism also allows content creators to fine-tune the compression of their content when it has special redundant characteristics.

Quantizing BIFS data efficiently is complex because no clear statistics can be derived from the source data.[4] The problem is to find the best trade off between the declaration cost of the local quantization parameters using the QuantizationParameter node and the gain brought by this quantization. To overcome this difficulty, every numerical field in each node is classified into one of 14 quantization categories (see Table 2). The idea is that each category, for example "Position 3D", will contain data in a similar range of values, and that a minimal number of categories enable a minimal number of parameters addressing the maximum number of parameters. A value-range and number of bits for a linear quantizer for each quantization type is specified in the QuantizationParameter node. When a field is to be encoded, its quantization type is looked up in the NCT for the field's node. The values of the field are then coded using the number of bits and value-range specified for that quantization type in the QuantizationParameter node.

```
QuantizationParameter{
        position3DMax          0 -2 8
        position3DMin          0 -4 0
        position3DNbBits       4
        position3DQuant        TRUE
}

PositionInterpolator {
  key [ 0, .03125, .0625, .09375, 1]
  keyValue [ 0 0 0, 0 0 2, 0 -2 4, 0 -2 6, 0 -4 8]
}
```

Fig. 12. A scene portion containing a QuantizationParameter node.

Fig. 12 shows a scene portion in which a QuantizationParameter node affects the quantization of the node that follows it. In this example, the original VRML text has unlimited precision, but, because the keyValue field of the PositionInterpolator node has "position3D" quantization type in the NCT, the binary BIFS representation of this field's values would use only 4 bits per number. For example, the last coordinate of the last keyValue would be stored using binary value 1111, since it is the maximum value in the range specified in the QuantizationParameter node. The key values would be encoded using 8 bits, because this is the default behavior of the QuantizationParameter node for the quantization type of this field. Since the QuantizationParameter node affects all the nodes that follow it, if there were any other nodes with fields in the position3D category and

[4] Partly the problem is that because MPEG-4 is new, there is no significant quantity of source data.

Table 3
BIFS Quantization Categories

Category	Usage
None	NoQuantization
Position3D	Used for 3D positions of objects
Position2D	Used for 2D positions of objects
Color	Used for colors and color intensities
TextureCoordinate	Used for texture coordinates
Angle	Used for angles
Scale	Used for scales in transformations
Interpolator Keys	Used for interpolator keys, MFFloat values
Normals	Used for normal vectors
Rotations	Used for SFRotations
ObjectSize 3D	Used for 3D object sizes
ObjectSize 2D	Used for 2D object sizes
Linear Quantization	NCT holds max, min, and number of bits
Coord Quantization	Used for lists of coordinates of points, color, and texture coordinate that are indexed. E.g. in IndexedFaceSet

with values outside the range given in the QuantizationParameter node, those values would be clipped (Table 3).

The QuantizationParameter node has several other features. It has a useEfficientFloat field that holds a boolean value. When this value is true and when floating point values are not quantized in one of the quantization categories described above, they will be quantized using MPEG-4 specified "efficientFloat" coding that has less resolution than standard floating point representations and which requires fewer bits per number, especially for 0 values. The details of this coding are beyond the scope of this chapter, but the idea is that a small number of bits is used to specify how many bits are used for the exponent and mantissa of the floating point value. Values with small exponents and easily specifiable mantissas can then be stored using a smaller number of bits overall. However, the coding is limited to a 15-bit mantissa and a 7-bit exponent.

The QuantizationParameter node has another boolean field, isLocal, that specifies if the quantization parameters provided in the node should apply only to the next node in the scene graph. This design allows the declaration of quantization parameters to be factored and applied to the maximum amount of data.

4.3. BIFS Command

The BIFS-Command protocol represents its four top-level functionalities using 2 bits:

- Replace the whole scene with this new scene;
- Insertion command;
- Deletion command;
- Replacement command.

The insertion, deletion and replacement commands can insert, delete and replace nodes, indexed values of MF fields, and ROUTEs. The replacement command can also replace a field value, bringing the total number of replacement comments to four, which makes that command completely efficient in its use of the 2 bits.

Fig. 13 shows a textual representation of a BIFS update command that can be applied to the scene in Fig. 4. After this command is sent to the MPEG-4 client, the scene would look almost identical, except with the new recent news headline replacing the previous field in the Text node.

The (simplified) binary representation of this update would consist of the following:

1. Two bits (with binary representation 10) indicating that the update is a replacement command.

```
Update {
      time 10
      action " Replace"
      what " Node"
      id " MyHeadline"
      data Text  {
            string " More recent news headline here"
      }
}
```

Fig. 13. Textual representation of a scene update.

2. Two bits (with binary representation 00) specifying that the update is replacing a Node (as opposed to an MFField element or a ROUTE).
3. A number of bits specifying the node ID of the node that is to be replaced. The number of bits is specified in a header that is sent at the top of the scene. In the example, the node ID is the string "MyHeadline", and this must be mapped uniquely to an integer in the BIFS representation.
4. A Binary representation of the Text Node that contains the new node. The specification of the node type is made by sending an index into the NDT table. In this case, the SFWorldNode NDT which contains all of the MPEG-4 nodes is used. We discuss this further.

The effect achieved in the example of Fig. 13 can also be produced by just replacing the value of the string field of the Text node. In this case the binary representation of the replacement command would consist of the following:

1. Two bits (with binary representation 10) indicating that the update is a replacement command.
2. Two bits (with binary representation 01) specifying that the update is replacing a field.
3. A number of bits specifying the node ID of the node containing the field to be replaced.
4. Two bits (with binary representation 00) specifying that the string field is to be replaced. There are four replaceable (IN) fields in the Text node, and string is the first. Note that in order to know this the decoder must first read the node id, look up the node type of the node with this id, and use the Node Coding Tables to determine how many IN fields this node has.
5. A Binary representation of string field. Since the decoder knows which field is to be replaced, it can look in the Node Coding Table to determine the field type and hence know what type of data to expect.

4.4. BIFS Animation

The BIFS Animation (or BIFS-Anim) protocol uses an arithmetic coder applied to a sequence of difference values that are computed from a sequence of animation parameter values. The binary animation data consists of an Animation Mask that contains a collection of Elementary Masks, one for each node that is animated. This is followed by a stream of Animation Frames that contain the animation data. Only updatable nodes (i.e., nodes with IDs) can be animated, because the Elementary Masks use these IDs to specify which nodes are animated. Each Elementary Mask also specifies which of the animated nodes' fields are animated and initial quantization parameters for the animation. Finally, when the animated fields are multiple value fields, the Elementary Mask specifies the indices of the elements that are to be animated.

Only fields that have a DYN index defined in the Node Coding Tables can be animated. These dynamic fields may be of the numerical data types:

- SFInt32/MFInt32
- SFFloat/MFFloat
- SFRotation/MFRotation
- SFColor/MFColor

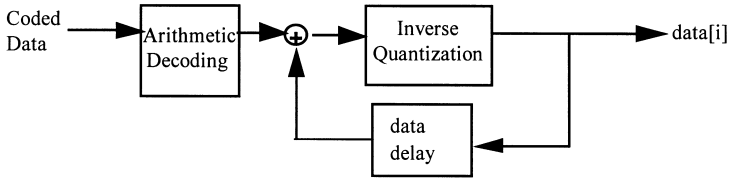

Fig. 14. The four steps of the BIFS-Anim predictive decoding scheme: arithmetic decoding, inverse quantization, delay, compensation.

- SFVec2f/MFVec2f
- SFVec3f/MFVec3f

The Animation Frames specify the values of the animated fields. An Animation Frame can contain new values for a subset of the animated fields at a specified time. That is, all the fields or a selection of fields can be modified at each specified time. The field values can be sent in Intra (the absolute value is sent) and Predictive modes (the difference between the current and previous values is sent).

In Intra-mode, the field values are quantized (without any entropy coding) using the quantization parameters defined for the animated field in the associated Elementary Mask. In Predictive-Mode, the difference with the previous sample is computed and then entropy coded. Fig. 14 summarizes the decoding steps.

The quantization is similar to the one used in the BIFS compression of field values. In BIFS Animation, as for BIFS scene compression, the notion of quantization category is used. The following animation categories are defined:

- Position 3D
- Positions 2D
- Color
- Angle
- Float
- BoundFloat
- Normals
- Rotation
- Size 3D
- Size 2D
- Integer

However, the animation categories correspond more to data types rather than a semantic grouping, since quantization parameters are not shared but declared individually for each field to be animated. Each category has a specific syntax for declaring its quantization parameters: Min and Max values, number of bits, in Intra and Predictive modes. Computing the difference of quantized values is direct except for normalized vectors such as rotations and normals. Rotations are coded using quaternions, and a specific prediction mechanism enables coding of rotations and normals in a predictive mode, just as with the other data types. The entropy coding uses an adaptive arithmetic encoder, which avoids the issue of unknown data statistics.

Fig. 15 contains an example of a 3D scene in which a Transform (translation and rotation) and some of the vertices of an IndexedFaceSet are continuously animated using a BIFS-Anim stream.

The object descriptor with id = 15 will be parsed. It contains the Animation Mask. The Animation Mask is stored in a field of the ObjectDescriptor, more precisely in the `specificInfo` field of the general `DecoderSpecificInfo` field. The field will contain the following information:

- 5 bits to specify the number of bits used to encode the Node ID;
- 5 bits to specify the number of bits used to encode the ROUTE ID;
- 1 bit to specify that the stream is an animation stream (versus a BIFS-Command stream);
- The AnimationMask itself containing a loop of Elementary Animation Mask (loop controlled by a 1 bit flag);
  - For each Elementary Animation Mask:
    - A node ID (here corresponding to TToAnimate and CToAnimate).
      - For each animated node, a DYN field mask of fields to be animated in the node (the TToAnimate node in our

```
DEF TToAnimate Transform {
      Translation …
      rotation …
      children [
            Shape {
                  Geometry IndexedFaceSet {
                        coord DEF CToAnimate Coordinate {
                                    point [ 0.1 0.2 03,
                                              0.3 0.4 0.75,
                                              ……
                                          ]
                        }
                  }
            }
            AnimationStream {
                  url "dmif://od=15"
            }
            ……..
      ]
}
```

Fig. 15. A sample scene with a transform and an IndexedFaceSet and coordinates to animate.

example would get a field mask of '01001' since rotation and translation are respectively the second and last DYN fields out of 5 DYN fields for the Transform node).

- For each field to animate (loop controlled by the field Mask):
  - If it is a multiple field (like in the case of the point field of the Coordinate node), an optional (controlled by a 1 bit flag) list of vertices to animate.
  - The quantization parameter for the field, containing the bounds and number of bits.

Finally, the BIFS-Anim elementary stream will contain animation frames, containing:

- The AnimFrameHeader, containing some specific information to enable the choice of nodes animated in the frame (the Transform or Coordinate node animation could be switched off at any frame), a number of skipped frames, timing information, etc.
- The AnimationFrameData, with:
  - The I or P value of the Transform rotation;
  - The I or P value of the Transform translation;
  - The I or P value of the Coordinate node point selected positions.

5. BIFS usage

The following sections list a few tricks and suggestions for using the BIFS tools.

5.1. Scene compression

The main complication in using BIFS well is the use of the QuantizationParameter node. This node can reduce the size of a scene significantly, even when used naively. This is because the native encoding of floating point and integer values typically has considerably more resolution than is needed for graphic scenes.

The most direct and naïve way to use a QuantizationParameter node is to compute the maximum and minimum values for each of the quantization types in the scene, decide on an acceptable error, and insert one QuantizationParameter node at the top of the scene graph with the corresponding quantization parameter values. Almost all scenes should at least use this technique, since the cost of the QuantizationParameter node is minimal. (The exceptions are very small scenes in

which the cost of specifying the QuantizationParameter node is greater than the bits saved in the scene.)

A more complicated technique for using the QuantizationParameter node involves grouping scene components that contain similar ranges of values and inserting a QuantizationParameter node at the top of each such grouping. For example, a scene that consists of two houses separated by some distance could benefit from a QuantizationParameter node at the top of the grouping nodes that hold each house. This technique essentially breaks the scene into smaller scenes and applies the naïve quantization approach to each sub-scene. The process of breaking up a scene and determining where to place QuantizationParameter nodes is something of an art, though it is possible to automate this function.

Finally, when possible, the QuantizationParameter node can be DEFd and USEd, so that the cost of specifying the quantization parameters is very low. In the above example of a scene containing two houses, if each house has about the same geometry, the first can be quantized normally and the second can be quantized using only a reference to the QuantizationParameter node used on the first house.

Using QuantizationParameter nodes, typical compression rates of 10 to 25 can be achieved over a textual scene representation, depending also on the content. In MPEG-4 Version 2, enhanced compression will be achieved by adding optimized multiple field (MF) and 3D mesh encoding. Compression rates over 20 can be expected by combining all these tools.

5.2. BIFS command usage

The capability of BIFS-Command to modify a scene can be used for real-time update of content, for example in news headlines. It can also be used to modify the user's experience, by interactively changing the user's position or view point, or by sending status or chat messages between different users on a server (or even directly between users).

Fig. 16 shows an example of using BIFS-update to progressively load a scene. In this example, a large scene is broken into parts. The parts that are viewable from the initial viewpoint are sent as part of the initial scene graph, but the other parts are sent using BIFS-Command at a later time: when the bandwidth allows, when they are required

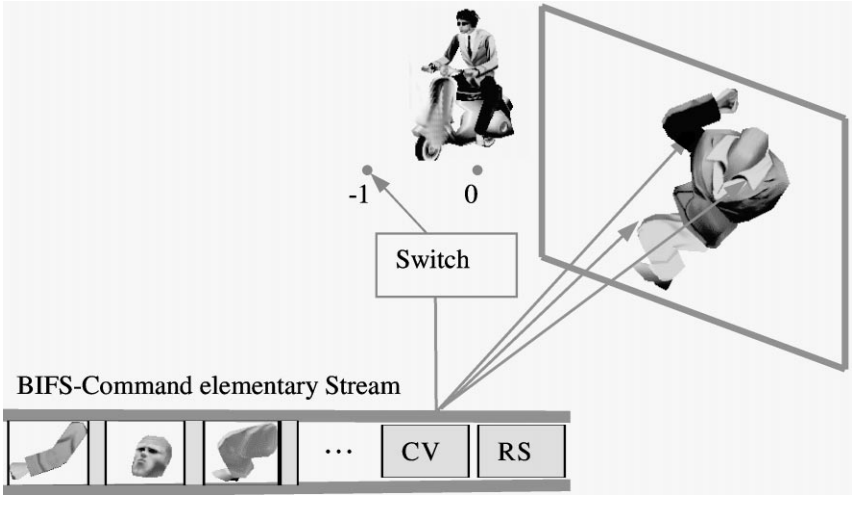

Fig. 16. The BIFS Command protocol. Building the scene graph progressively, with InsertObject BIFS command. The motorcyclist is added after the initial load and put off screen for future use, by changing the value of the Switch around it (CV). The Replace Scene command (RS) replaces the whole scene graph with the new scene.

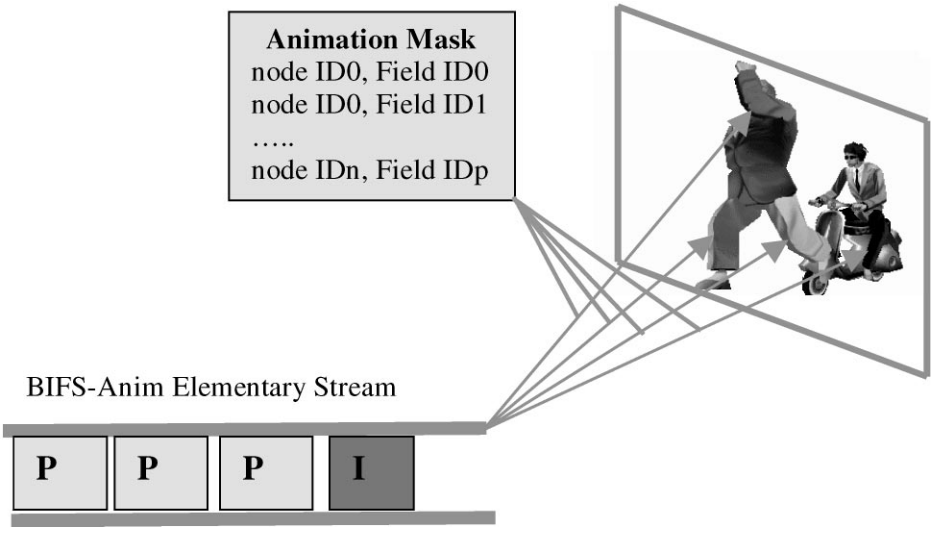

Fig. 17. The BIFS-Anim protocol: animating the position and the viewpoint. As with video, intra (I) frames are used for random access (tune-in) or error recovery. Predictive (P) frames are differentially encoded using a much smaller number of bits.

in the scene, etc. This mechanism can significantly reduce latency during initial scene loading.

5.3. BIFS animation usage

BIFS-Anim can be used to stream efficiently animations over low bit-rate networks, or to "communicate" continuous changes in a scene (e.g. a networked game). Typically, animating approximately 80 parameters at 10 frame per second results in a bit-rate around 5 kbit/section. This means an average approximate bandwidth of 0.06 kbit/s per parameter (rotation or 3D translation). Fig. 17 illustrates the usage of BIFS-Anim for a 3D cartoon. BIFS-Anim is much more efficient than BIFS-Command for changing always the same field. As a comparison, changing one translation with a BIFS-Command costs 4 bits for the ReplaceField command, 10 bits for the Node ID, 3 bits for the field ID and 96 bits for the translation value, for a total of 113 bits, so roughly a kbit/s for 10 frames per second. For the 80 parameters, this means 80 kbit/s for animating with BIFS-Command versus 5 kbit/s for animating with BIFS-Anim. The gain comes from the fact that the NodeID and fieldID needs not be specified at each frame in the case of BIS-Anim and from the specific quantization, prediction and entropy encoding used.

6. Concluding remarks

MPEG-4 was designed to provide solutions to a slew of applications. Its tool set can be broken up into profiles to best fit the needs of specific applications, such as broadcast applications, which can be thought of as "TV mixed together with the interaction and presentations of the World Wide Web", to interactive consultation and communication applications, which can be thought of as "the telephone and TV cooked together with a dash of interactivity". (These days, everything is spiced with interactivity.)

Within MPEG-4, BIFS provides both compression needed for efficient transmission (and storage) on the current infrastructure, and the functionalities demanded by today's media consumers (who are becoming progressively more savvy). That MPEG-4 fills a technology space that is currently empty is clear; it remains to be seen, however, whether it will be accepted and widely used.

For further reading, see [1–3,5,6].

Acknowledgements

The authors would like to thank Shout Interactive Inc. (*www.shoutinteractive.com*) for providing the visual material that appears in Figs. 16 and 17.

References

[1] ISO/IEC JTC1/SC29/WG11 N2562, "MPEG-4 Requirements Document", December 1998.
[2] ISO/IEC 14496-1, Coding Of Audio-Visual Objects: Systems, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2501, October 1998.
[3] ISO/IEC JTC1/SC29/WG11 N2611, "MPEG-4 Systems Version 2 WD 5.0", December 1998.
[4] ISO/IEC 14772-1, The Virtual Reality Modeling Language, 1997, http://www.vrml.org/ Specifications/VRML97.
[5] ISO/IEC 14496-2, Coding Of Audio-Visual Objects: Visual, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2502, October 1998.
[6] ISO/IEC 14496-3, Coding Of Audio-Visual Objects: Audio, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2503, October 1998.
[7] ISO/IEC 14496-6, Coding Of Audio-Visual Objects: Delivery Multimedia Integration Framework, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2506, October 1998.

Julien Signès graduated from Ecole Polytechnique, Paris, 1992 and from Ecole Nationale Superieure des Telecommunications in 1994. For three years, he worked as a member of the research staff in the multimedia division of CNET, the France Telecom research center, in Rennes, France. During that period, he worked on fractal still image compression, and was then deeply involved in the MPEG-4 and VRML standardization process, as one of the main contributors to the MPEG-4 scene description language, BIFS. Currently, he works at France Telecom R&D, the San Francisco-based R&D unit of France Telecom and Deutsche Telekom, where he leads a project on mixed media client server services based on Java, VRML and MPEG-4.

Yuval Fisher has extracted various degrees from universities, most recently a Ph.D. in Mathematics from Cornell University. He has written and edited several books, and published a number of articles, mostly in the area of Fractal Image Encoding. Most recently, he has actively participated in the MPEG-4 standardization process. He is currently a visiting scholar at the Institute for Nonlinear Science at the University of California, San Diego.

Alexandros Eleftheriadis was born in Athens, Greece, in 1967, and received his Ph.D. in Electrical Engineering from Columbia University in 1995. Since 1995 he has been an Assistant Professor at Columbia, where he is leading a research team working on media representation with emphasis on multimedia software, video signal processing and compression, video communication systems and the mathematical fundamentals of compression. Dr. Eleftheriadis is a member of the ANSI NCITS L3.1 Committee and the ISO-IEC JTC1/SC29/WG11 (MPEG) Group, where he serves as the Editor of the MPEG-4 Systems specification. His awards include an NSF CAREER Award.