MPEG-4 3D Graphics: from specifications to the screen

Sasko Celakovski*, Marius Preda**, Slobodan Kalajdziski*, Danco Davcev*, Françoise Preteux**

*Faculty of Electrical Engineering, Karpos II, bb, 1000 Skopje, Macedonia
**ARTEMIS Project Unit, GET-INT, 9, rue Charles Fourier, 91000 Evry, FRANCE

etfdav@etf.ukim.edu.mk, marius.preda@int-evry.fr

Abstract: - This paper presents a novel implementation of a 3D rendering engine able to display 3D graphics MPEG-4 objects. By using the MPEG-4 SDK (Software Developer Kit), the 3D objects are first decoded and the MPEG-4 scene graph structure is filed. We introduce a scene manager able to address in an optimized manner the rendering requirements. It is developed as part of the rendering engine and it enables to create an appropriate form representation of the data resources. The novel concept implemented here is to consider the scene management with respect to the rendering constraints and not to the representation of the data as in a usual MPEG-4 approach. This paper describes the software communication procedures between the MPEG-4 SDK and the rendering scene management in the case of static and animated (skinned) object and some results dealing with the representation of an articulated model illustrate the performances of the developed approach.

Key words -3D Graphics, MPEG-4 Standard, Rendering Engine, Software Developer Kit

1 Introduction

The latest developments in the multimedia field as well as the need of exchanging data through a wide spread of networks, lead the scientific and industrial communities to build new and rich multimedia standards. Among existent or on-going multimedia standards, MPEG-4 [1] is one of the most complete in terms of media representation, compression, 2D and 3D graphics primitives, user interaction and programmatic environment. As a member of the MPEG family, the MPEG-4 standard inherits and improves all the features of its predecessors, offering the possibility of efficient transmitting and/or storing a huge amount of digital audio / video. Furthermore, the standard addresses state of the art techniques such as advanced audio coding, video compression-based on visual object, wavelet deployment and mesh-based representation. In addition to representing elementary media, MPEG-4 goes further and specifies mechanisms that allow to create complex multimedia scenes. It is now possible to combine several media, to define synthetic content and to add interaction and dynamic behavior of the scene. These features are supported by using a special description language called BInary Format for Scenes (BIFS). BIFS is based widely on Virtual Reality Modeling Language [2] (VRML) and represents a binary encoded version of an extended subset of VRML, which can represent roughly the same scene as with VRML in a much more efficient manner. In this paper we are interested in the BIFS functionalities for representing synthetic content, especially the 3D objects. The wide range of functionalities supported by MPEG-4 makes this standard one of the most complete and advanced solution, and companies are slowly deploying MPEG-4 technologies inside their applications. The complexity of the standard is a serious drawback and its wide acceptance as a common multimedia format has difficulties to take off. In a previous paper [3], we stated that the development of an MPEG-4 SDK that make transparent for the developer some MPEG-4 key techniques such as encoding, can improve the standard acceptance and speed up the development of applications based on MPEG-4. We demonstrate this statement by developing a 3DSMax plug-in and a Maya exporter. In this paper, we will demonstrate the same concept (facility of building applications based on the MPEG-4 SDK) but in the case of a new application: an MPEG-4 3D player.

Currently there is no significant commercial application of MPEG-4 3D capabilities. Some 3D mesh compression implementations of the MPEG-4 3DMC approach are reported in the literature (IBM, Samsung). In addition simple players for face animation provided by face2face inc. [4] or for body animation provided by VRLab [5] and ARTEMIS [6] are only restricted implementations of subparts of MPEG-4 specifications.
Any application that would visualize MPEG-4 encoded 3D objects will influence the significance of the whole standard.

Real-time rendering engines are used to visualize 3D content together with animations while instantly reacting to different user interactions. These engines are widely used in applications for entertainment, educations, GUI and so on. Open source examples of such engines are OGRE [7] and Irrilich [8]. They both offer extended set of rendering functionalities but differ in design approaches.

The rendering process has to be hardware independent. The technologies that enable high level of hardware abstraction layer (HAL) allow functioning of the rendering engines on a wide range of graphics hardware. Two most important HAL technologies are DirectX [9] and OpenGL [10].

OpenGL is the first standard which defined open interface to the graphics hardware. However at the moment it lacks of the support of the latest release of graphics hardware. DirectX is a set of low-level application programming interfaces for creating games and other high-performance multimedia applications. It includes support for 3D graphics. It represents the current trends in computer graphics development.

In this paper we report on a highly optimized 3D player able to decode and display MPEG-4 content. The rendering architecture is described in detail in Section 2. The content decoding is ensured by the use of the MPEG-4 SDK, previously reported and shortly introduced in Section 3. In Section 4 we show how the two applications are merged together in order to build the MPEG-4 player. In Section 5 we present some performances of the player in terms of speed and rendering quality.

2 Rendering Engine Architecture

The object oriented approach is used for the design of our rendering engine. In that way we manage to provide abstract representation of the 3D scene that enables handling of different input formats. At the same time we can encapsulate implementation of separate logical objects referring to different functionalities of the rendering engine. The rendering engine implements functionalities for representation, manipulation, maintenance and presentation of the 3D scene. Figure 1 shows the hierarchical structure of the objects in the rendering engine.

The representation of 3D objects in the scene is achieved by visualizing their corresponding abstraction named entities. Entities contain the geometry definition of 3D objects segmented in geometry subsets of meshes.

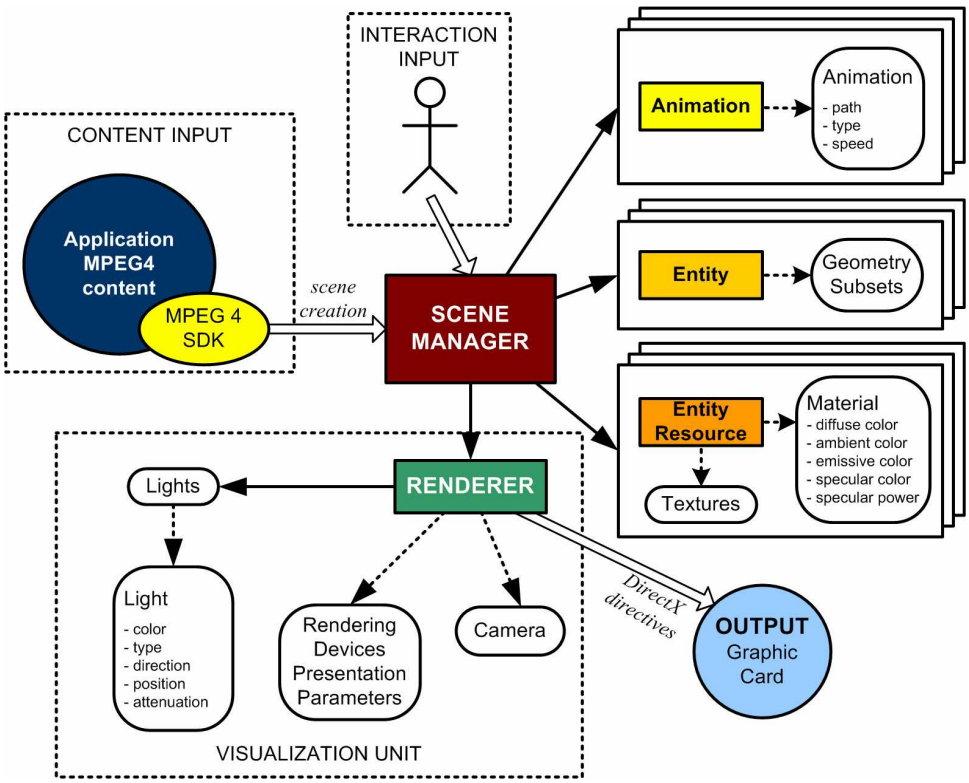

Figure 1 –Rendering engine architecture.

The entity abstraction is not considered for non-geometrical properties of 3D objects. These properties (texture and material) are described with a help of entity resource which are related to every instance of entity's subsets. The entity resource defines the appearance of an entity's subset by defining its material (diffuse, ambient, emissive and specular characteristics) and texture map. Texture and material properties have to be treated differently by the rendering engine because while the first one maps to the 3D surface the second defines the surface appearance parametrically.
The entities in the 3D scene can be static or animated. The animation controller stores the data needed to calculate the orientation and position of the animated entities in each frame of their animation.
The 3D scene management involves construction, manipulation and destruction of different entity, entity resource and animation controller instances. In our approach we delegate this complex task to the scene manager. It contains lists for all entities, entity resources and animation controllers in the scene and their mutual dependences or relations.
The scene manager provides external interfaces for creation of instances of different scene objects and user interaction. The content input feeds the scene manager with scene objects. In our approach we integrate MPEG-4 content by using decoder SDK. More details on this interface are given in the next section. The interaction input provides data about user actions on the screen to the scene manager.
The scene manager passes instructions to the renderer for visualization of the entities. The renderer uses this information together with the lightning, camera setup and presentation parameters to create output to the rendering device. We use DirectX technology in order to encapsulate hardware dependent 3D visualization functionality.

3 MPEG-4 SDK

The MPEG-4 SDK is designed to offer an access to the MPEG-4 fundamental elements - the BIFS scene and the elementary streams – transparently with respect to the compression layer. The developer using the SDK must know the interface of the nodes defined by the standard but no knowledge is required concerning how this information is compressed. Figure 2 presents the MPEG-4 SDK architecture.

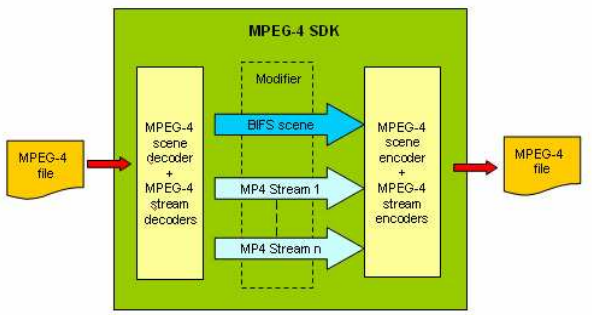

Figure 2 –Architecture of MPEG-4 SDK.

The MPEG-4 SDK main functionalities refer to (1) encoding/decoding MPEG-4 scene and elementary streams; (2) browsing and modifying the attributes of MPEG-4 scene nodes and the content of MPEG-4 elementary streams, and (3) creating/deleting MPEG-4 scene nodes and elementary streams.

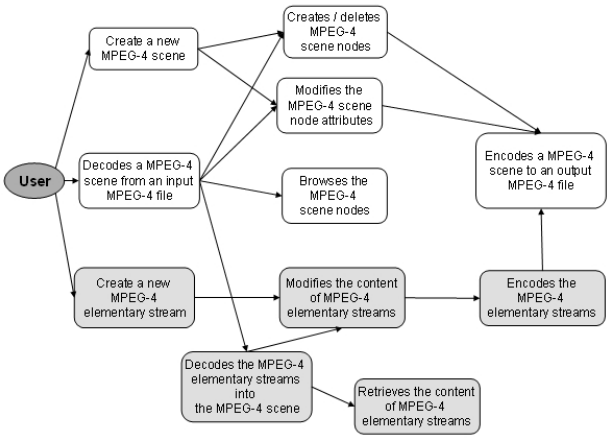

Figure 3 – Operations supported by the MPEG-4 SDK.

Here we are interested in the MPEG-4 capability for representing static and animated 3D objects. We previously showed that one needs only 11 scene graph nodes from the total number of more than 150 nodes that MPEG-4 specifies in order to represent static and articulated mesh objects.
The MP4SDK is a C++ library based on the MPEG-4 Reference Software [11] (IM1). In this experimentation we used the SDK generated according to the "Animated Character" profile. This profile, currently under consideration within the MPEG community, selects 11 nodes in the scene graph and 2 streams from the media layer. The components of this profile (the list of scene nodes and elementary streams) and the corresponding SDK classes are described in Figure 4.

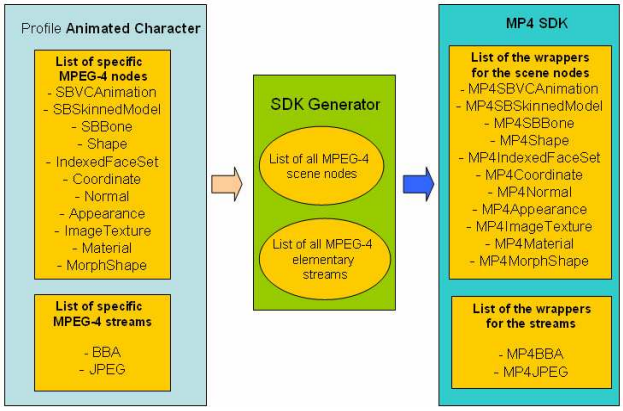

Figure 4 – The SDK Generator applied to the Animated Character profile.

4 Merging MPEG-4 SDK and the Rendering Engine

The integration of MPEG-4 SDK in the rendering engine will be explained through the example which will show how the 3D geometry objects given in MPEG-4 format are transferred to the scene manager.

The 3D static objects in the MPEG-4 scene are represented through instances of the Shape class and the 3D animated objects through instances of the SBSkinnedModel class. For each case we show the communication between the MPEG-4 SDK component and the scene manager.

4.1 MPEG-4 SDK and Rendering communication for static objects

The Shape class contains information for the 3D geometry and its appearance. The geometry is given in MP4 IndexedFaceset structure that contains an array of vertices and indexed faces. The Appearance class gives the description of the appearance of the 3D geometry object.

This information is used to generate the initial 3D scene as illustrated in Figure 5. Upon decoding of the MPEG-4 content the create scene process is initiated. This process populates the scene manager by triggering two scene manager interfaces: create entity and create entity resource. The create entity process uses the MP4IndexFaceset structures (MP4 coordinate and MP4 coordIndex), while the create entity resource process uses theMP4Appearance data (MP4 material and MP4 texture) from the MPEG-4 SDK. The actual mapping of MPEG-4 structures and rendering engine's structures is given in Figure 6.

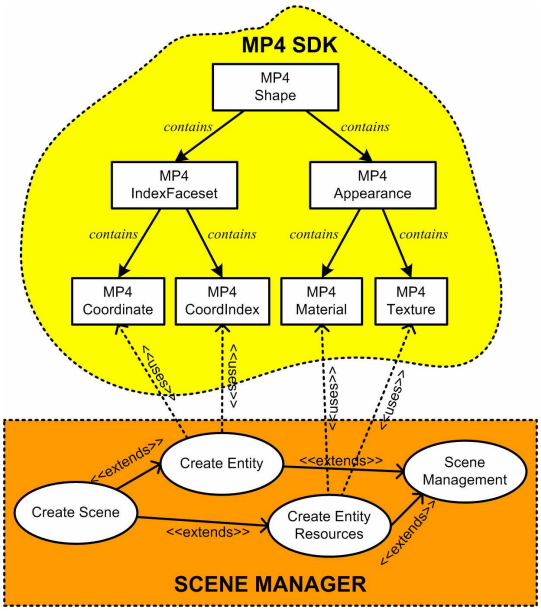

Figure 5 – The communication between the MP4 SDK and the Scene Manager for the Shape object.

As can be seen from the Figure 6 when MPEG-4 SDK finishes the parsing of the 3D content it stores the geometry in the MP4IndexedFaceset instances and the appearance attributes in MP4Appearance instances. After invoking the create scene process (presented on the Figure 5), it retrieves all of the MP4Shape instances in the 3D scene. By iterating through these instances, it fetches the geometry and appearance attributes necessary for converting the data needed for registering of corresponding Entity and EntityResource instances to the scene manager. The Entity class stores all necessary geometry of the object, while the EntityResource class gives the real look of the object by providing its materials and textures

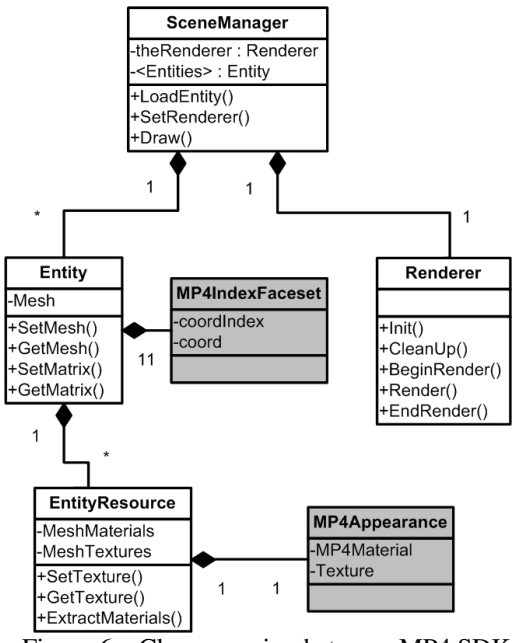

Figure 6 – Class mapping between MP4 SDK and rendering for the IndexedFaceSet geometry type.

4.2 MPEG-4 SDK and rendering communication for articulated objects

MPEG-4 defines the SBSkinnedModel node to describe articulated (skin-bone animated) 3D geometry. Communication between the MPEG-4 SDK and the rendering engine for providing support for skin-bone animated objects is presented in Figure 7.

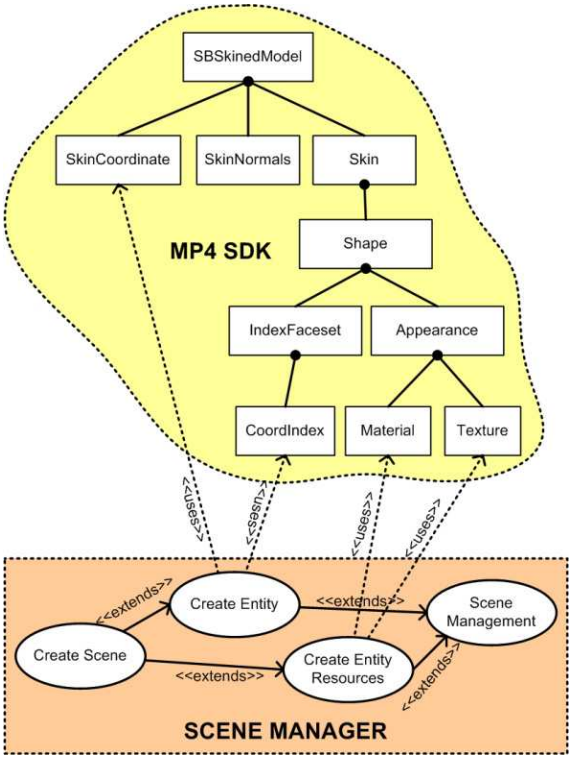

Figure 7 – The communication between the MP4 SDK and the Scene Manager for the SBSkinnedModel object.

Each instance of the SBSkinnedModel contains a set of vertices coordinates and 3D normals for the geometry object stored in the skinCoord and skinNormal members respectively. The vertices and normals in the SBSkinnedModel are shared by all of the different parts in the 3D object, so called skins. Different skins in the skinned model are described in an array of *Skin* field. Each of these *Skin* components stores appearance and geometry attributes. The geometry in each *Skin* component has an IndexedFaceSet which refer to the 3D vertices from the SBSkinnedModel and in addition it defines face indices in its coordIndex member. The appearance field contains the material and the texture for the skin. The entity creation process iterates through all of the SBSkinnedModel instances in the MPEG-4 scene and collects the necessary data to register appropriate entity and entity resource instances in the scene manager as it is illustrated in Figure 8. Similarly to static 3D objects in the scene, the Entity instances hold the 3D geometry data while the Entity resource instances provide realistic appearance of the 3D objects.

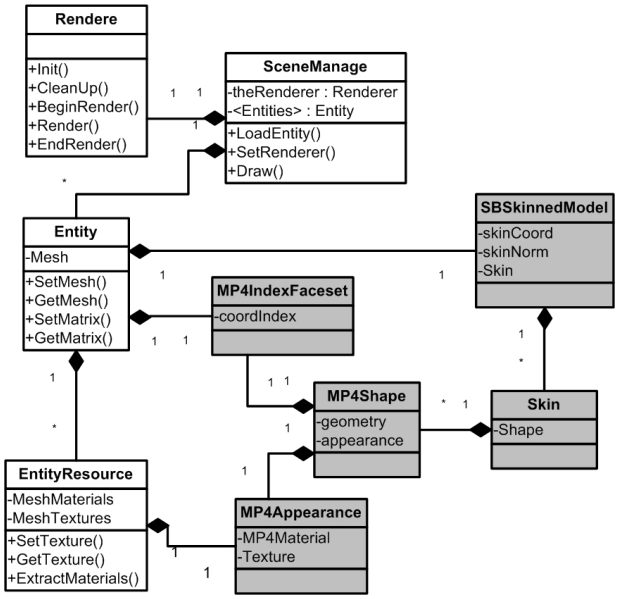

Figure 8 – Class mapping between MP4 SDK and rendering for the SBSkinnedModel node.

Figure 9 illustrates rendered MPEG-4 objects in wire frame, Goroud shaded form and textured model. The rendering engine enumerates all graphics devices on the local machine, and selects the most convenient device for the visualization of the 3D scene. If some required features are not supported by the rendering device, then the engine initializes the software emulation. This approach has two main advantages. The combination of hardware and software rendering enables usage of a wide range of graphics hardware. The support of hardware accelerated rendering enables more efficient visualization 3D scene with large number of polygons and rendering effects.

a) b) c)

Figure 9 – Visualization by the rendering engine: a) in wireframe, b) in Goroud shaded and c) textured model.

The current implementation of the renderer supports geometry, materials and textures. As soon

as other features such as bone-based animation, will be supported by the MPEG4 SDK, they will be included in the renderer as well.

5 Conclusion and future work

In this paper we presented a novel implementation of a 3D rendering engine able to display graphics objects represented by using the MPEG-4 standard. Based on the MPEG-4 SDK, the media entity (here the 3D objects) is first decoded and the MPEG-4 scene graph structure is filed. Then a scene manager, developed as part of the rendering engine, creates the data resources to be displayed in an optimized manner with respect to rendering requirements. The novel concept here is to consider the management of the scene close related to the rendering and not to the representation of the data as in a common MPEG-4 approach [12]. We described the software communication between the MPEG-4 SDK and the rendering scene management in the case of static and animated (skinned) object. Finally we illustrate some results dealing with representation of an articulated model. The future work will aim to integrating the elementary media (video, audio and animation stream) which requires synchronization and stream control.

Acknowledgment

This work is partially supported by the EGIDE and French Embassy in Skopje. The authors would like to thank Vladimir Trajkovik, Blerim Mustafa and Son Tran for their valuable contributions.

References

[1] ISO/IEC 14496-1:2001 Information technology -- Coding of audio-visual objects -- Part 1: Systems, International Organization for Standardization, Swiss, 2001.

[2] ISO/IEC 14772-1: 1998, Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding.

[3] O. Folea, M. Preda, F. Prêteux, MPEG-4 SDK: from specifications to real applications, 9th WSEAS International Conference on Communications, Athens, Greece, July 14-16 , 2005

[4] Facial animation solutions, face2face. Inc, http://www.f2f-inc.com/

[5] T.K. Capin, D. Thalmann, Controlling and Efficient coding of MPEG-4 Compliant Avatars, Proc. IWSNHC3DI'99, Santorini, Greece, September 1999.

[6] M. Preda, T. Zaharia, F. Prêteux, "3D body animation and coding within a MPEG-4 compliant framework", in *proceedings International Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging (IWSNHC3DI'99)*, Santorini, Greece, 15-17 September 1999, pp. 74-78.

[7] Object Oriented Graphics Rendering Engine - OGRE, http://www.ogre3d.org

[8] Open source high performance engine, http://irrlicht.sourceforge.net

[9] Microsoft DirectX, http://www.microsoft.com/DirectX

[10] OpenGL, http://www.opengl.org

[11] ISO/IEC 14496-5:2001 Information technology -- Coding of audio-visual objects -- Part 5: Reference Software, International Organization for Standardization, Swiss, 2001.

[12] F. Pereira, T Ebrahimi, *The MPEG-4 book*, IMSC Press Multimedia Series/Andrew Tescher, 2002.