A Scene Representation Application implementing LASeR Using Object-Based Timing Model

Xiaocong Zhou[1], Jianping Chen[2], Tiejun Huang[3],

[1]Institute of Digital Media, Peking University, Beijing 100871, China,
[2] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[3]Key Laboratory of Machine Perception (Ministry of Education), Peking University, 100871, China

{xczhou, jpchen, tjhuang}@jdl.ac.cn

Abstract. In this paper, we present a multimedia scene representation application implementing MPEG4-Part 20 Lightweight Application Scene Representation standard (LASeR). In an interactive multimedia scene, we are usually concerned about two questions: (i) The spatial-temporal management of scene elements; (ii) How to manage the interaction between the user and the multimedia scene. In this paper we propose an object-based timing model to ensure all the elements are rendered in correct time, and we also propose corresponding strategies to deal with the user's actions. At the end of this paper, we show some instances of the rich-media system. The future of rich-media is also considered.

Keywords: LASeR, rich-media, scene description, interactive, object-based timing model.

1 Introduction

With the increasing demand of digital media and the broadband availability, the circulation of digital media is growing rapidly. When people consume digital contents, they do not be content with only audiovisual materials but also a feature-rich environment.

The term rich media can be used in contrast to media that only uses traditional forms of audiovisual material. Rich media, on the other hand, includes a combination of some or all of the following, audio, video, image, text and animation. Rich media can be considered synonymous with interactive media which allows users to control the playback of its elements, the sequence of content, and choose what to see or experience.

The multimedia scene is an example of rich media. So a scene description scheme is required to present a scene with plural number of media. The scene description scheme also enables system to place each specific medium on a scene with a suitable spatial and temporal organization and allow interactions between multimedia objects of this scene, so the users are able to consume digital contents consistently in a feature-rich environment. LASeR (Lightweight Application Scene Representation) is a MPEG International Standard which defines the scene description format. The LASeR specification has been designed to allow the efficient representation of 2D scenes describing rich-media services for resource-constrained devices [1]. A rich-media scene is a dynamic and interactive presentation comprising 2D vector graphics, images, text and audiovisual materials. The description of such a scene describes the spatial and temporal organization of its elements as well as its possible animations and interactions.

In this paper, we present a multimedia scene application using LASeR as its scene description format. We present the multimedia scene system model and analyze how to parse a multimedia scene description created according to the LASeR standard. Some strategies and methods are also proposed to ensure that the scene is represented correctly. We propose an object-based timing model which keeps the temporal module of the scene representation application running well.

The rest of this paper is organized as follows. After the introduction, section 2 describes system model in detail, including the strategies and methods used in this system. Section 3 shows the experimental results, and the conclusion of the work is given in section 4. Finally, section 5 is the acknowledgment [5].

2 System Analysis

In this section, we describe the framework, and analyze how to parse a multimedia scene, and then propose an object-based timing model. At the end of this section we present the interaction management.

2.1 System Model

In a scene representation system, a scene stream server is needed. A scene stream consists of scene units which describe the multimedia scenes, and it does not contain the real audio or video streams. Persistent LASeR scene stream should be created by this server so that end users can consume digital contents in a feature-rich environment. Some other media content servers are also needed in this system. Users will get real digital contents from these servers, such as images, video and audio streams. Imagine that you are watching a soccer match in a rich-media environment: you can get each player's images from a media content server which provides these images and you can also get match commentary from another media content server, and the match video stream is received from a live media content server. All these contents are downloaded and processed, and then represented in a scene consistently.

See Figure 1, the PCs and mobiles first receive LASeR scene stream from a scene stream server, then users could interact on the elements in the scene and choose the media contents they want. The media contents that users consume come from those media content servers.

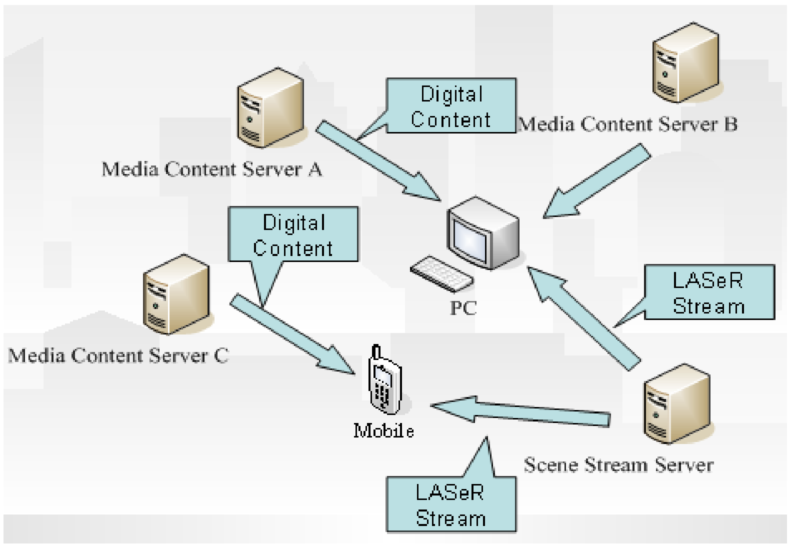

Figure 1: The system model

When the terminal receives a scene stream, it needs to process the stream and then represent the multimedia scene. This functionality is implemented by the scene representation application which works in a terminal such as a PC or a mobile phone. The scene representation application involves three main functions, scene parsing, object management and interaction management. First the scene parsing module parses the scene stream, and then gets all the elements information. The object management module will deal with the element information and render visual elements in correct time. When interaction happens the interaction management module needs to process user action and respond to the action. Figure 2 shows the scene representation application modules.

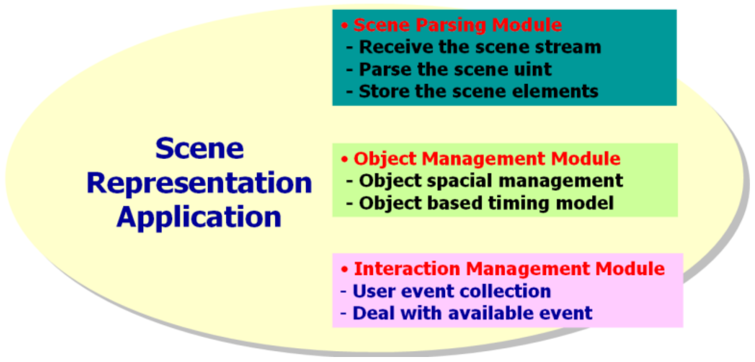
Scene Representation Application
• Scene Parsing Module
- Receive the scene stream
- Parse the scene uint
- Store the scene elements
• Object Management Module
- Object spacial management
- Object based timing model
• Interaction Management Module
- User event collection
- Deal with available event

Figure 2: scene representation application modules

2.2 Scene Parsing Module

The terminal device receives the LASeR scene stream from a scene stream server. One LASeR scene stream consists of scene access units, so a LASeR scene may have a set of access units. Terminal should parse these scene units and store their elements, and then represent the scene in proper time.

Every scene access unit has a time attribute to tell the terminal when to represent this scene unit so the terminal could parse these scene units in time order. When a scene unit needs to be represented, the terminal first constructs its scene tree whose nodes are the elements in this scene unit [1]. After that, the terminal cloud get all the nodes information of this scene tree, such as some graphs, remote images, video streams and so on. All the elements are kept in the object management module. In this module, elements in the scene unit are kept as scene objects. It is easy to understand that images and graphs are treated as scene objects, but some other LASeR elements such as LASeR "conditional", "animate" and event listener are also treated as scene objects [1]. For example, LASeR "conditional" is an important element which may contain several scene commands for later execution. For instance, insert an image or an animation into the scene. A "conditional" element may be activated by time or user event. After activated, the "conditional" element notifies a begin event and then the commands in this "conditional" element will be executed. The begin attribute of "conditional" element could specify the time when the "conditional" element will be activated. The default begin value is infinite. A fragment of a LASeR scene unit is shown below [3].

```
<sceneUnit ref="stream0" begin="1s">
  <lsr:NewScene>
    <svg id="sroot">
         ......
      <animate xlink:href="#im1" attributeName="x" from="-170"
      to="87" begin="1s" dur="3s"/>
         ......
      <lsr:conditional id="animRes" begin="5s">
          <lsr:Insert ref="sroot">
            <image x="180" y="50" xlink:href="stream:bg2" />
          </lsr:Insert>
      </lsr:conditional>
         ......
    </svg>
  </lsr:NewScene>
</sceneUnit>
```

In addition to the element itself, the spatial information of each element will also be stored. This information may be stored in the attribute transform of an element or inherited from its parent element. The spatial information contains the translation, scale and rotation of the element.

2.3 Object-Based Timing Model

An object-based timing model is proposed in this section. When we have already parsed a scene unit, how can we manage the objects so that they will be rendered in correct time? Time is one of the most important information in a scene representation application. Firstly, we introduce three concepts: Media Time, Scene Time [1] and Rendering Time. Media Time is a common system time, but Scene Time is a relative time, and the two concepts come from LASeR standard. The Scene Time is set to zero when a new scene needs to be presented. For example, the scene time of the time point x in a new scene is calculated in Figure 3 [1].

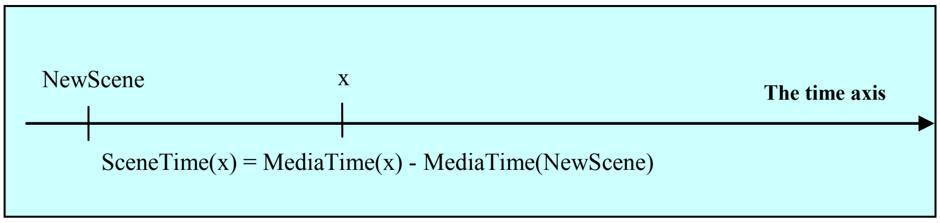

Figure 3: Media Time and Scene Time

Rendering Time is the time when an object is rendered in a scene, and it is relative time, too. Some elements of a scene unit may have time attribute which is Rendering Time. In the following scene unit, the value of attribute "time" equals to 14s, which indicates that the scene unit will be represented after the system has been running for 14 seconds; The element "lsr:conditionl" also has an attribute "begin" whose value is 5s which is the object's Rendering Time. It means that after this scene has been rendered for 5 seconds, the "lsr:conditional" element will be activated and then be rendered. In fact, the "lsr:conditional" element will be executed after the system has been running for 19 seconds.

```
<sceneUnit ref="stream1" time="14s">
      <lsr:NewScene>
           ……
           <lsr:conditional begin="5s">
           ……
           </lsr:conditional>
      </lsr:NewScene>
</sceneUnit>
```

After introducing the base concepts, we continue to describe the object-based timing model. All the elements in a scene unit are divided into two different types of objects. One is what we call time-static object, while the other is time-dynamic object. A time-static object will be rendered immediately when the scene it belongs to is rendered. It is consistently rendered and its spatial attribute and temporal attribute never change. Contrarily, a time-dynamic object needs to be activated by time or some events. A time-dynamic object can be activated by the events such as mouseclick, keyboard button down, an animation's ending and so on. For example, if a user clicks an image in a scene, another image may be activated and then will be rendered in the scene. The second image is that what we call a time-dynamic object because it is activated sometime by the user's click event. Time also can activate some objects in a scene. For example, the "conditional" element can be activated by time. The instance in previous section shows that the "conditional" element will be activated in 5 seconds after the scene is rendered.

In fact an object may have not time attribute inherently, but we can assign a time value to it as its rendering time. For a time-static object, we set its rendering time to zero that means it will be represented immediately when the scene it belongs to is rendered, and it is always rendered until deleted by some command or another new scene starts. For a time-dynamic object, there are two cases: (i) If it is activated by time, we set the trigger time value as its rendering time. We still use the above instance of a scene fragment. At scene time 5s, a conditional element is activated, and it inserts an image into current scene. In this instance, the rendering time value of the inserted image is set to 5s. (ii) If the time-dynamic object is activated by events, we do not know when it will be represented. Its rendering time is uncertain, so we set it infinity, which means that the object will not be rendered whenever the current scene time is. But if the corresponding event happens, the rendering time of the object should be changed to current scene time. As a result, this object could be represented now.

So in this system, we first check that if an object is time-static or not. If so, its rendering time should be set to zero. Then the object is stored and rendered immediately. If it is a time-dynamic object and will be activated by event, its rendering time should be set infinity. The object will not be represented until corresponding event happens. After that, this object can be represented. If the time-dynamic object is not activated by event but time, its rendering time equals to the time when it is activated in the scene(such as begin="5s"), and it will not be represented until current scene time value is not less than this object's rendering time value. Figure 4 is a flow chart.

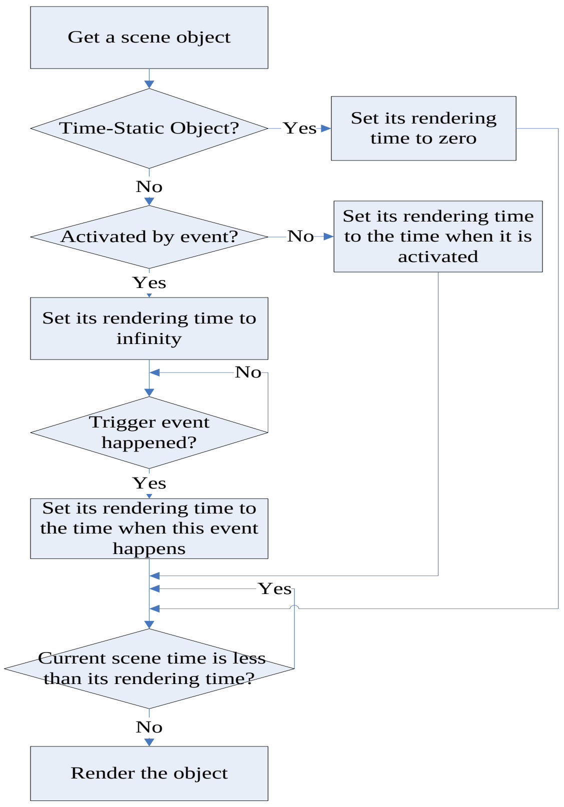
Get a scene object
Time-Static Object?
Yes
Set its rendering time to zero
No
Activated by event?
No
Set its rendering time to the time when it is activated
Yes
Set its rendering time to infinity
No
Trigger event happened?
Yes
Set its rendering time to the time when this event happens
Yes
Current scene time is less than its rendering time?
No
Render the object

Figure 4: Flow chart

2.5 Event Management

In a rich-media environment, users can interact with the objects in a scene. For instance, users can click on an image or a title in order to trigger some other events which will change the scene. So the application needs a robust event management to make sure the system is running correctly when user events happen.

The strategy is to build an event container. In current scene all the available events and its information are kept in this container. When the scene units are parsed, terminal could get all the available events information of this scene, such as a click on an image at some time. At the same time, if a user event is captured by the terminal, its information will be put into an event queue. When the queue is not empty, each event in the queue will be compared with the event in the pre-build event container. If the condition of the captured event matches that of some event in the container, some operations will be processed by the handler of this event. When the event has been processed, it will be removed from the queue and then the next captured event will be dealt with. Every time the scene makes some changes, it will be repainted. The algorithm shows below:

```
While(True)
{
   If( eventQueue is not empty )
   {
        Get the first event information from the eventQueue;
        For( every event in the event element container )
        {
             If( captured event matches some event condition in this container)
             {
                  Make corresponding changes to current scene;
                  Repaint current scene;
                  Break;
             }
        }
        Remove the first event information from the eventQueue;
   }
}
```

3 Demo of the Scene Representation

Below we show a video navigator scene in a mobile phone [10]. What we see first is some animations (Figure 5 and Figure 6) and then we enter a video navigator scene (Figure 7). There are some introductions to every video program (Figure 8), and if you click an image in the scene the corresponding video program will play (Figure 9 and Figure 10).

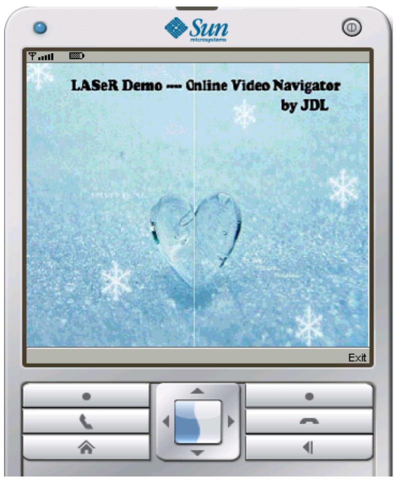

Figure 5: Scene A

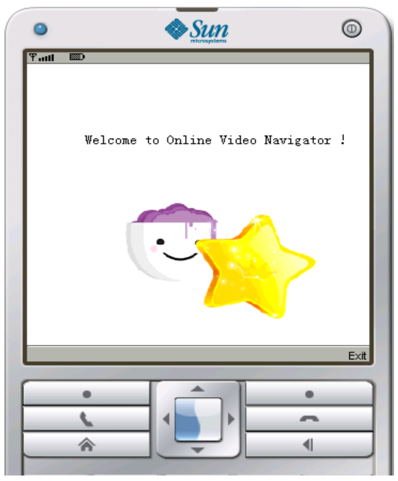

Figure 6: Scene B

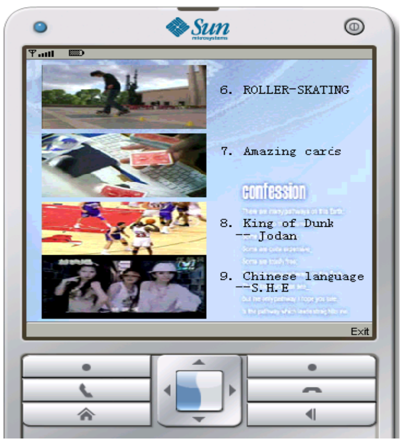

Figure7: Navigator scene

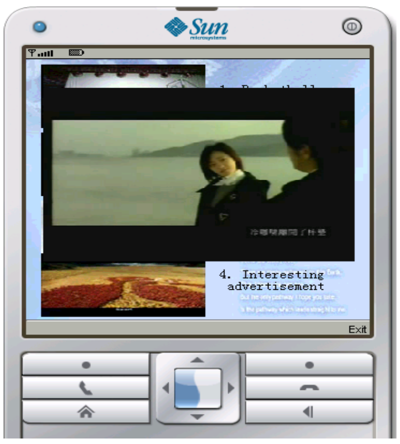

Figure 8: Preview of one video

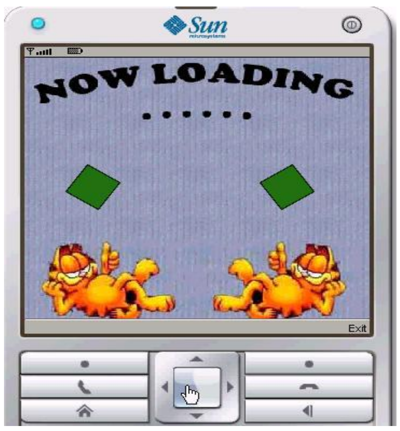

Figure 9: Loading one of the video

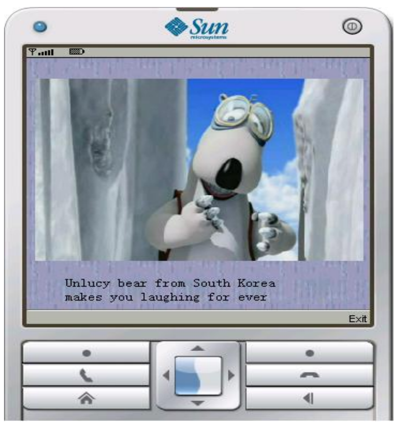

Figure 10: Video is playing

4 Conclusion

A scene representation application is described in this paper. We discuss scene parsing strategies, propose an object-based timing model, and design user interaction management. In this feature-rich environment, consumers could have good experiences. Besides, users also can get more information through a multimedia scene than traditional audiovisual materials. In the future scene representation application may be used in Web TV, mobile TV and IPTV, and consumers could watch TV program in an interactive environment. Although there are still many problems that need to be solved, it is no doubt that the scene representation application would have a bright future.

5 Acknowledgment

This work is based on LASeR reference software from MPEG [10]. It is a part of the WIM TV Beijing Olympics Trial of Digital Media Project and was supported by the National Key Technology R&D Program [2006BAH02A10 and 2006BAH02A13] of China .

References

1. ISO/IEC 14496-20, Information technology – Coding of audio-visual objects–Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF).
2. Filippo Chiariglione, Tiejun Huang, Hyon-Gon Choo, Streaming of governed content – Time for a standard.
3. W3C, Scalable Vector Graphics (SVG) 1.1 Specification.
4. W3C, Synchronized Multimedia Integration Language (SMIL 2.0)-[Second Edition].
5. WIM TV Beijing Olympics Trial: http://www.wimtv.net.
6. L. Chiariglione, Impact of MPEG standards on multimedia industry, Proc. IEEE, vol. 86, pp. 1222–1227, Jun. 1998.
7. Digital Media Project (DMP). http://www.dmpf.org/.
8. Information Technology–Coding of Audio-Visual Objects–Part 11: Scene Description and Application Engine, ISO/IEC 14496-11.
9. Information Technology–Multimedia Content Description Interface–Part 1: Systems, ISO/IEC 15938-1.
10. The MPEG-LASeR reference software.
11. Apache Software Foundation, Apache Tomcat, http://tomcat.apache.org/.