# Systèmes multimédia et qualité d'expérience

Gianluca Di Cagno

# SYSTEMES MULTIMEDIA ET QUALITE D'EXPERIENCE

Thèse

Gianluca Di Cagno

présentée pour obtenir le grade de
docteur de l'Ecole National
Supérieure de Télécommunications

Spécialité: Informatique et Réseaux

ENST

2004

Soutenue le 6 juillet 2004 devant le jury composé de:

| | |
|---|---|
| Françoise Preteux | Président |
| Benoit Macq | Rapporteurs |
| Daniel D. Giusto | |
| Leonardo Chiariglione | Examinateurs |
| Olivier Avaro | |
| Jean Claude Dufourd | Directeur de Thèse |

# ACKNOWLEDGMENTS

I would like to express my thanks to my supervisor Jean Claude Dufourd for having welcomed me in his group and for the support he provided during this work. His talent at creating an extremely liveable and pleasing working place has made my stay at ENST an excellent period. Special thanks are also due to Leonardo Chiariglione and Mauro Quaglia for having given me the possibility to start this thesis and for the enthusiasm they always showed for my work.

I would like to thank Prof. Daniel Giusto and Prof. Benoit Macq for having accepted to review this thesis. I am grateful also to all the members of the jury for having accepted to be part of the thesis committee.

I am indebted to my friends and colleagues Guido Franceschini, Andrea Varesio, Gianluca De Petris, Stefano Battista, Zvi Lifshitz for the work they did in the context of the development of the MPEG-4 standard and for the excellent training I received from them. My thanks also to all the ENST gang: Cyril Concolato, Jean Le Feuvre, Frédéric Bouilhaguet, Souhila Boughoufalah. They all contributed to this work with many valuable discussions and suggestions.

I would also like to thank the following people with whom I shared most of my free time in Paris: Riitta Vaananen, Simone Fontana, Luca Prati, Gianluca Allaria, Simon Berth Andersen ("Il conte"), Marco Combetto, Marco Ricci, Alessandro Tinti, Eddie Cooke, Nigel Savage, Giorgio Mula and the other Italian students at ENST. They all contributed to make my stay in Paris an happy period. A big thanks to Anne-Christine Muri, because when she showed up ("hello") nothing was really ready. I finally express gratitude to my family for their constant support and love.

# RESUME

Au cours des dernières années, des efforts de recherche considérables ont été effectués dans le domaine de l'adaptation de diverses entités dans la chaîne de distribution des informations numérique, depuis la génération du contenu jusqu'au terminaux des utilisateurs finaux. Dans ce contexte, ce travail prend en considération les adaptations effectuées au niveau des terminaux. La contribution principale en est l'établissement d'un cadre multimédia pour la gestion de la qualité des présentations qui fonctionnent sur les systèmes d'exploitation d'usage universel. Le fait d'accorder les ressources disponibles, très variables, avec le volume des calculs des systèmes de multimédia, dont la dynamique est aléatoire, pose un problème stimulant. En modelant un terminal comme un système centré sur l'utilisateur, nous postulons que l'utilisateur apporte sa notion de valeur en spécifiant la qualité de service pour tous les médias. La spécification identifie une série d'options qualitatives pour chaque dimension de la qualité d'expérience de médias et établit une hiérarchie entre elles. L'adaptation est par conséquent réalisée en sélectionnant une combinaison d'options qualitatives pour chaque dimension de la qualité d'expérience d'un flux multimédia, afin de maximiser l'utilité du système et la satisfaction de l'utilisateur. En empruntant des principes empruntés à la théorie du contrôle pour des systèmes à évènements discrets, nous abordons les questions ayant trait à la contrôlabilité, à la résistance et à la configuration réactive. D'abord, nous définissons le processus de contrôle qui assure que tous les components du système atteignent de manière cohérente le paramètre QoS demandé. Ensuite, nous établissons un schéma de gestion de qualité. Les résultats montrent comment le cadre proposé satisfait les besoins typiques des applications fonctionnant sur les logiciels d'exploitation d'usage universel: contrôle d'admission, compétition entre les applications et gestion de puissance. La deuxième contribution est l'intégration des graphiques synthétiques 2D dans le cadre de gestion de qualité. Afin de choisir des options de qualité selon les ressources informatiques disponibles,

une évaluation de la tâche de rendu est essentielle. Nous avons subdivisé le processus de rendu en un certain nombre de tâches conceptuellement indépendantes et nous avons identifié les paramètres qui affectent leur exécution. Le modèle peut être employé dans une approche réactive pour fournir des chiffres de complexité au cadre de gestion de qualité, afin d'allouer les ressources informatiques à la mise en train ou quand des violations de QoS sont détectées. Nous prouvons que le modèle peut également être employé dans une approche prédictive, pour programmer correctement les taches de rendu de la vidéo et le graphique 2D. Pour finir, nous montrons comment la gestion de qualité peut être effectuée le long de la dimension de la qualité visuelle perceptuelle quand la bibliothèque de rendu est capable de fournir différents niveaux de qualité. La troisième contribution est l'étude sur la façon de réaliser la commande de charge sur une classe des applications, les flux à forte contrainte temporelle qui exigent le décodage continu des mises à jour successives de description de scène. Un modèle du temps de décodage est donné, ainsi que des prédicteurs pour estimer le décodage. Nous prouvons qu'il est possible de mettre en application des prédicteurs bon marché et efficaces qui peuvent prévoir des temps de décodage à moins de 3% et moins de 5%, avec un taux de succès allant de 90 à 100 % des échantillons. En conclusion, nous fournissons des exemples de l'utilisation de ce modèle d'amélioration du playback des dessins animés 2D. Nous avons basé ce travail sur le standard MPEG-4 Systems et sa plateforme de logiciel de référence. MPEG-4 Systems fournit un cadre intégré et normalisé pour présenter les médias principaux considérés dans ce travail, les graphiques 2D et la vidéo. Nous avons contribué au développement de la norme, et nous avons conçu notre cadre comme une extension conforme de l'architecture de MPEG-4 Systems. Nous sommes persuadés que ce choix a renforcé l'homogénéité de ce travail, et a augmenté l'intérêt et la réutilisation des résultats.

# ABSTRACT

In the last years, considerable research efforts have been spent on the concept of adaptation at various points in the digital information distribution chain, from content/service generation to end-user terminals. As part of these efforts, this work considers adaptation in multimedia end-user terminals. The core contribution is the establishment of a multimedia framework for quality management of presentations running on general-purpose operating systems. Matching the typically varying available resources with the dynamically changing computational load of multimedia systems poses a challenging problem. Modelling a terminal as a *user-centric* system, we assume the user to provide his notion of value expressing a user-level quality of service specification for each media. The specification identifies a set of *quality options* for each *media quality dimension*, and sets up a ranking among them. The adaptation is hence performed selecting a combination of quality options for each quality dimension of a stream, in order to maximize system utility and user satisfaction. Borrowing principles investigated in control theory for discrete event systems, we addressed the issues of *controllability*, *robustness* and *reactive configuration* first defining control processes that ensure that all system components meet the required QoS parameters consistently, and second establishing a quality management scheme. Results show how the proposed framework fulfils the requirements typical of applications running on general-purpose operating systems: admission control, competition among applications and power management.

The second contribution is the integration of 2D synthetic graphics in the quality management framework. In order to select quality options according to the available computational resources, an *estimate* of the rendering task is essential. We subdivided the rendering process into a number of conceptually independent tasks and we identified the

parameters that affect their performance. The model can be used in a *reactive* approach to provide complexity figures for the graphic enhancement stream, in order to allocate computational resources at start-up or when QoS violations are detected. We show that the model can also be used in a *predictive* approach, to properly schedule video and 2D graphics rendering tasks. Lastly, we show how quality management can be carried out along the dimension of perceptual visual quality when the rendering library has different rendering quality levels.

The third contribution is the study on how to achieve load control on a class of applications, specifically time-constrained flows that require the continuous decoding of successive scene description updates. A model of the decoding time is given, together with predictors to estimate the decoding. We show that it is possible to implement efficient cheap predictors that can predict decoding times within 3 and 5 ms with a rate of success varying from 90 to 100 % of the samples. Finally, we provide examples of the use of this model to enhance the playback of 2D animated cartoons.

We based this work on the MPEG-4 Systems standard and its reference software platform. MPEG-4 Systems provides an integrated, standardized framework to support the main media considered in this work, 2D graphics and natural video. We contributed to the development of the standard, and we conceived our framework as a compliant extension of the MPEG-4 Systems architecture. We believe that this choice has provided homogeneity to this work, and increased the interest and the reusability of the results.

TABLE OF CONTENTS

# LIST OF FIGURES

*Chapter 1*

# INTRODUCTION

The term multimedia, introduced during the 70s to describe a theater-based film and slide show extravaganza, is nowadays widely utilized to express "human-computer interaction involving text, graphics, voice and video" [hyp2003]. This definition clarifies two basic concepts of multimedia applications. First, they are composed of different media, glued together providing a rich audio-visual experience; second, they provide mechanisms to offer user interaction. Fields of application include entertainment, education, and advertising. According to this definition, a Multimedia System is a terminal that plays different media simultaneously, according to some synchronization rules, with a certain level of user interaction. Even if multimedia often refers to computer technologies, it can be used to describe a number of dedicated media appliances, such as digital video recorders, set-top boxes, advanced wireless devices and public video displays. Indeed the last years have shown an important growth of the number and type of these devices.

Compared to traditional data applications, multimedia presentations have different requirements. They are comprised of continuous and non-continuous media, which have to be presented at precise instants in time and require extensive computational resources. Besides, in contrast with data applications, errors in the media streams are often tolerable or conceivable using appropriate media-specific techniques. It follows that traditional metrics to capture the quality of data applications are not easily applicable in the field of multimedia. Still, requiring heavy computational loads, multimedia applications are traditionally difficult to support (only few years ago a

multimedia PC was considered a high end terminal, with expensive video and audio boards, CPU etc.).

This thesis deals with the problem of designing multimedia systems capable of scaling down the computational needs whenever they have to adapt their running conditions. The background of this thesis is provided by work in this field we did in the context of International Standards (MPEG), European Research Projects (SOMMIT, OKAPI, MPEG-4 PC) and hi-tech industrial projects with the Japanese TDK R&D, ENST and TILAB. In the context of these works, we often faced the problem of providing complex multimedia presentations composed of several media streams on different terminals, with different capabilities. For instance, Figure 1 shows a multimedia application composed of 16 MPEG-4 videos running concurrently. When clicking on one running video, a different screen is proposed (Figure 2), composed of two videos at a better quality, plus text and still images.


Figure 1: Low detail multiple streams


Figure 2: Video, still images and text presentation

Looking at these applications, questions may arise: how is the quality of a presentation quantitavely expressed? How does the terminal monitor and control the quality of a multimedia presentation? This thesis aims at providing some answers to these questions, contributing to the field of research called quality-based multimedia, which deals with presentation systems capable of adapting to varying system requirements on the basis of a Quality of Service specification (QoS). The next paragraphs illustrate why this subject is worth studying, and provide a clear statement of the problem under investigation and the technical approach adopted.

## 1.1 Motivation of the work

We investigate the problem of supporting adaptive multimedia terminals on the basis of the following facts:

- The demand for high-quality multimedia applications is constantly increasing.

The incredible success of hyperlinked documents has quickly accelerated the demand for real-time multimedia applications. In the Internet world we are now familiarized with the concepts of navigation through "static" text documents. A natural evolution is the concept of multimedia document navigation where synthetic and natural content are mixed to provide a more sophisticated navigation experience. A similar concept applies to the digital television broadcast domain where there is a demand for an increased level of interactivity (i.e. broadcast of sport events: a set of live cameras providing different views, additional text information with statistics, synthetic graphics overlay etc.). The advent of 3G multimedia mobile phones is yet another example of the rapid growing multimedia applications.

- Multimedia devices have different available resources.

Multimedia devices range from high-resolution television displays and advanced graphics workstations, over PCs to handheld, wireless devices such as PDA and mobile phones. Since the associated available resource power (memory, processing power and network bandwidth) varies by orders of magnitudes, it becomes mandatory to adapt the media content *and* the decoding and rendering algorithms to the device-specific resources.

- The data networks are evolving.

Circuit-switching networks are leaving the ground to packet-switching technologies. The circuit-switching networks were designed to minimize

switching and transmission jitter. Packet switching networks were instead designed to maximize link utilization rather than minimize transmission jitter. The burden of providing an acceptable quality of service (i.e. coping with packet losses and network jitter) is hence pushed to terminals, at the application level, rather than protocols.

The interest in research in this area is also evident in the work of standardization bodies in the area of multimedia. The ISO/IEC/SC29/WG11 group, also known as MPEG, is in the process of finalizing a multimedia framework (MPEG-21) for digital items consumption where the issue of multimedia adaptation is considered both at the content and terminal and network level.

### 1.2 Statement of the problem

The problem of supporting multimedia applications is a challenging problem because of the following facts:

- Operating Systems: multimedia applications are requested to run on general-purpose operating systems (GPOS), where applications can't reserve resources. Applications instead compete for the use of computational and memory resources.

- Computational requirements: multimedia applications have varying computational requirements (i.e. the amount of data varies from frame to frame in a video application). In operating systems where applications compete for the use of resources, the available resources are also a varying quantity. Hence, matching the varying available resources with the requirement of dynamically changing computational load poses a challenging problem.

- Synchronization: synchronized presentation of different media is necessary at the user end.

- Compressed media: in order to be stored/transmitted efficiently, media have to be compressed. At the terminal side, this implies the need for a decoding task before the display of each media unit.

- Synthetic Graphic: multimedia applications require the composition of continuous media with synthetic content whose processing time may vary considerably over different display boards.

- User Interaction: in order to be user-friendly, these systems require fast response time and rapid switching between different viewing options.

Multimedia terminals running on general-purpose operating systems must hence be robust to unpredictability and to variations in the computational load of incoming data. They must constantly monitor application progress and react to losses of value caused by the natural unpredictability of the underlying environment. In performing this, a presentation system has to take into account the value of service provided to the user. A "quality-based" presentation system performs the adaptation process on the basis of a quality of service specification (QoS). This work is meant to provide contributions to this field of research.

### 1.3 Technical Approach

We model a multimedia terminal as a *user-centric* system. We assume that users provide the notion of value of the different media of a presentation to a terminal, suggesting how they expect it to work under unpredictable events. This implies the need for a user-level quality specification, which brings this information to a terminal, and also metrics to control its performance and

then react to loss of value of its behaviour. We first identified user and application quality specification for a multimedia terminal and then we proposed a multimedia framework based on this model.

Our contribution is based on the MPEG-4 Systems standard and its reference software platform. MPEG-4 Systems provides an integrated, standardized framework to support 2D graphics and natural video. The rationale behind this choice is that we believe the standard to be the perfect playground to verify the value of this work, since it has the semantic power to express real-time multimedia applications composed of several media streams, applications that may end up being highly resource consuming. Being then an International Standard, the specification is public, and our work is not tied to a particular, private design. Allowing a great flexibility in combining different types of streams, the standard introduces some interesting challenges in relation to the problem we stated. Actually the choice of the standard as reference has also influence on the novelty of our work. In fact, most of the work in the area under study (detailed in the next section) concentrates on terminals handling audio-video streams. Instead we consider multimedia scenes where the scene description streams are rather complex and as a result challenge the task of providing quality-based presentations. We refer here for example to 2D animations (i.e. cartoons), multiple camera applications (i.e. remote surveillance, sport events), e-learning (mixed video and 2D graphics applications). While studying the subject, we tried to concentrate on this kind of applications rather than on the traditional ones based on single audio video flows. These considerations motivated our interest in the use of MPEG-4 Systems.

Another view of the technical approach we followed can be provided if we model a multimedia terminal as composed of three different layers: user, application, and network. It is well agreed in literature that in multiple layer models, QoS must be specified and enforced at all the layers [CCG93]. This

work focuses on *user* and *application* level layers, and do not consider any particular network service scenario. The assumption is then that the data a terminal needs is always available when requested. Although this may imply that a direct usefulness of this work is only possible in local scenarios (i.e. incoming data is stored on a file), we believe that our work can be complemented with existing work carried out on network level QoS (the following section mentions some related work). Since different networks have different requirements this can be done focusing on particular service scenarios.

In relation to quality of service, we can mainly organize issues related to QoS into three categories [VKvBG95]:

1 Assessing QoS in terms of user's subjective wishes or satisfaction with the quality of application performance, synchronization, etc.

2 Mapping results of the assessment into QoS parameters for various systems components.

3 Specification of control processes to ensure that all system components meet the required QoS parameters values consistently.

According to this characterization, this work falls into the second and the third category. Specifically, we propose QoS parameters in terms of which we define the quality of a presentation (2), presentation algorithms and load control techniques driven by these QoS parameters (3).

### 1.4 Related work

In the past Quality of Service has been mainly addressed in literature at the network level, with contributions to measure the level of service in a established communication. The need for an end-to-end infrastructure for quality of service in distributed multimedia systems has been first reported in

the work of Campbell [CCG93], [CCH95], [CAH97]. The papers propose a layered architecture of an end-to-end system with quality of service mechanisms present at each layer. QoS principles are introduced together with QoS user-level specifications and QoS mechanisms that realize end-to-end behaviours. It is a reference for any work on QoS. Similarly, [SCDS97] proposes a QoS-driven resource management framework for the management of conflicts with system resources. It proposes a taxonomy of Quality of Service parameters composed of metrics (performance, security), and policies (level of service, management policies). This work is interesting because it somehow generalizes the problem and provides a broader view in which to fit the less general problem of finding metrics for multimedia systems. Most of the research in QoS has occurred in the context of individual architectural layers (network and end-system). QoS on current best effort Internet is a topic that attracted several researchers, either proposing enhancements to the Internet protocol to enable scalable service (diffserv), or specifying protocols to request specific resources (RSVP), or metrics to be used by an application to measure the current quality of a communication. In end systems, we found contributions in the areas of scheduling, flow synchronization, playout and transport support. Playout techniques to preserve temporal relations in presence of network jitter are reported in [SDF93], clock synchronization problems over the Internet are tackled in [HPH 2000]. These papers well handle problems arising when deploying multimedia applications over best effort networks. Media synchronization algorithms are proposed in [IT 95, ITI2000, IYYS1996]. They introduce the notion of virtual time to extend or shrink the duration of the playback in order to cope with network delays. Even if it is focused on synchronization algorithms for distributed networks, their work is interesting because they propose also application level metrics to measure the quality of the playback: smoothness of video playback is captured by coefficient of output variation and media unit output rate, synchronization is captured by the mean square error of clock skew. Quality-based presentations are dealt in [BKWSAKG96].

The paper defines some quality of presentation (QoP) metrics and proposes a set of protocols to preserve multimedia streams presentation over network. The metrics they use are quality degradation due to deadline miss and data dropped due to destination buffer overflow. In [RB 1993] a first tentative of dealing with the mapping of application level QoS parameters to network level is reported. In particular, the skew between streams, a metric defined at application level, is translated in a relaxation of delay constraints imposed to the network. This implies a less demanding resource request from the network, such as allocation of packet buffers and assignment packet scheduling policy. The characterization of multimedia flows is best expressed in [BS96], where the authors illustrate every possible relation between media streams and define the concepts of inter-stream, intra-stream synchronization. The paper is a reference in the field of theory of synchronization. The subjective quality of synchronization in a multimedia presentation has been exhaustively assessed by the work of Steinmetz [Stein96]. He defines media synchronization quality in terms of skew between streams. He proposed various thresholds of clock skew to capture synchronization between audio, video, text, and images as perceived by the user. He proposed also a method to extend the property of synchronization from two objects to a third one, knowing the relationship between each couple. This work is very important to us, since it translates human perception of synchronization to quantitative values that can be used by algorithms at the application level. In the field of MPEG literature, [RPS93] is one of the first papers on software video decoding and rendering. [BMP98], [BA2000], [MB96] provide models to estimate MPEG-2 and MPEG-4 Video decoding times. [RKR96] describes in a clear way how synchronization is achieved in MPEG-2 streams, it has been very useful in understanding the MPEG-4 reference software use of clocks at start-up. [PE2002] provides basically all the MPEG-4 notions related to this thesis work. [N4848] (the official ISO specification) is the reference for MPEG-4 Systems. [FS93] and [NRLD2002] describe 3D frameworks that tried to address the same problem of supporting adaptive multimedia, for 3D

scenes. We did not find any contribution on presentation algorithms for 2D MPEG-4 terminals enhanced by quality of service specifications, or 2D terminals capable of mixing synthetic and natural content with quality metrics to capture the relationships among video streams and synthetic graphic. This work is meant to provide a contribution to this field.

## 1.5 Main contributions of the thesis

The core contribution of this thesis is the establishment of a multimedia framework for quality management of terminals presenting multiple streams on general-purpose operating systems. We introduce means to reason about the loss of value of multimedia applications introducing a user-level quality specification, and we identify application level metrics for system components. We provide algorithms to enhance the performance of media composition, a model to estimate 2D composition tasks and methods to integrate this model in the proposed quality management framework as a second contribution. The third contribution is the study on how to achieve load control on a class of applications, specifically time-constrained flows that require the continuous decoding of successive scene updates. A model of the decoding time is given, together with predictors to estimate the decoding. The last contribution is the work we did participating at the development of the MPEG-4 standard, in the context of the reference software activity (MPEG-4 Part 5).

## 1.6 Outline

This document consists of 6 chapters. They are organized as follows. Chapter 1 is this introduction. Chapter 2 introduces the MPEG-4 standard and the reference software implementation. Chapter 3 introduces the quality-based framework. Chapter 4 deals with the integration of 2D synthetic content. Chapter 5 concentrates on time-constrained scene description streams. Chapter 6 draws the conclusions of this work.

*Chapter 2*

# MPEG-4 STANDARD: CONCEPTS AND IMPLEMENTATION

This chapter illustrates the multimedia standard and the software framework on which we based our work. When we started this thesis, the ISO/IEC working group SC29/WG11 (that produced the successful MPEG-1 and MPEG-2 standards) was finalizing its new standard, MPEG-4. MPEG-4 became the international standard for communicating multimedia applications in 1999. We contributed to the standardization activity, and we participated to the implementation of the reference software. As we said in the previous chapter, we decided to base our work on the MPEG-4 standard because of its features enabling advanced composition of different media, and because it is an open standard. In the following, a brief explanation of the objectives of the standard is given, followed by further explanation of some features of the standard we used in the context of this work: the Binary Format for Scenes (BIFS), the System Decoder Model, Profiles and the Reference Software. It must be noticed however that this chapter does not cover the whole standard. MPEG-4 is a very broad standard, and probably a complete book would be necessary to cover all the issues. This chapter instead focuses on some aspects that are relevant to the work done, and that are vital for a good understanding of the following chapters.

## 2.1 MPEG-4 Standard

The MPEG-4 standardization process started in November 1992 with the initial aim at targeting very low bit rate coding [N271]. MPEG-1 being the compression standard for Video CD and MPEG-2 the standard for digital television, the new project was supposed to cover the missing scenario. However, the original vision soon evolved in a broader one. MPEG-4 would

support new ways of communicating, accessing, and manipulating digital audiovisual data [PE2002]. Content based interactivity, compression efficiency and universal access were the key concepts of the new vision. With the usual excellence in the Video (renamed Visual to highlight the presence of not only video tools, but also still pictures and 2D/3D synthetic graphics tools) and Audio parts of the standard, the vision was definitely appealing.

The Systems part of this new project would specify a multimedia terminal, capable of decoding and synchronizing different compressed media ("elementary streams"), and composing them according to a scene description stream in a 2D/3D application. Compared to previous versions (that the standard was not meant to replace), the MPEG-4 Systems scope was hence wider: it did not provide only tools to synchronize and multiplex audio-video streams, but enabled spatial composition of very different media: video, audio but also 2D/3D synthetic graphics.

The standard included also a part called Delivery Multimedia Integration Framework(DMIF), that defined a session protocol for the management of streaming over generic delivery technologies, providing an abstraction from the storage or transport medium. A complex multimedia application could have been created using only MPEG-4 tools, and conveyed on different networks and medium.

The following Systems tools are of particular interest for our work:

- Binary format for scenes (BIFS): a 2D/3D binary scene description language based on the textual VRML 3D [VRML97] language. This is the tool to achieve media composition and user interaction.

- Object descriptor framework: describes the properties of the different elementary streams. This tool describes the media assets of the multimedia application.

- Systems Decoder Model: defines the buffering and synchronization models. It is an advanced version compared to previous versions of the standard, modified to take into account the added complexity of handling multiple streams.

The first version of the standard was completed in 1999. By that time, the standard was mainly focused on broadcast environments, and lacked several important features. A non-framed life-cycle file format (mp4) and new BIFS features were added to the specification in 2001 and specifications for the carriage of MPEG-4 scenes over MPEG-2 Transport and over IP were added later.

MPEG-4 is still evolving. Two new media codecs have been added recently in March 2003. Advanced Video Coding (AVC) improves the natural video coding efficiency of the MPEG-4 Visual standard. AAC-HE (high efficiency AAC) is the most advanced coding technique for generic audio signals.

As of today, the MPEG-4 standard is still the only complete standard solution for the delivery of synchronized, compression and network optimal, rich media services on any media and any network [SIG99].

## 2.2 BIFS

The MPEG-4 scene description language is based on the Virtual Reality Modeling Language (VRML) [VRML97]. In VRML, a scene consists of three different tools: the scene graph, an event routing mechanism, and prototypes. The scene graph is constituted of nodes, grouped in a hierarchical structure (scene tree), which describe objects on screen and their properties. Nodes can be roughly divided in *grouping* nodes, that are used to logically combine objects and compose them spatially, and *leaf* nodes that provide graphic primitives (circle, rectangle, line, cube, cylinder etc.), text, video, audio and sensors to interact with objects on the screen (i.e. double-click, drag-n-drop

functionalities). The following is an example of a scene containing a video object.

```
OrderedGroup {
 children [
  DEF TR Transform2D {
   translation 100 100
   children [
        DEF TS TouchSensor{}
        Shape {
           appearance Appearance {
                        texture DEF Movie MovieTexture {
                              starttime 0
                              stoptime -1
                              url ["1"]
                        }
           }
           geometry BitMap {}
        }
     ]
  }
 ]
}
```

This simple scene contains two grouping nodes (OrderedGroup, that is the start of a 2D scene and Transform2D, used to spatially compose the subsequent nodes) and a shape node that includes a video node (MovieTexture). Nodes contain *fields* that store values and express features of the object that can be read/written. The scene contains also a TouchSensor node that may allow some form of interaction. The Transform, MovieTexture and TouchSensor nodes are preceded by a DEF construct that allows assigning a name to the nodes, name that can then be used to access their fields from other tools. No interaction is present in this scene. It just displays a video, centered at the x=100, y=100 position of the display. In order to add some form of interactivity, we need more tools.

Event routing gives authors a mechanism through which events generated by nodes can be propagated to other nodes. This processing can change the state of the node, generate additional events, or change the structure of the scene graph. Prototypes allow the extension of the pre-build set of nodes. If we add the route mechanism at the end of the previous scene, we may achieve simple interaction:

```
ROUTE TS.touchTime TO Movie.stopTime
```

In this way, clicking the mouse button when the pointer is on the video will cause the value of the TouchSensor's touchTime field to be copied to the Movie's stopTime field. This will cause the movie to stop, as the name of the fields suggests. Note that we used the DEF names to refer to the nodes in the ROUTE.

BIFS has added several new concepts to the VRML standard. BIFS is a compressed binary format, and as a consequence scenes are optimized in size and can be streamed. BIFS elementary streams are composed of access units that include BIFS commands to add new nodes, replace field values, and replace the whole scene tree. These commands enable single changes to the scene. BIFS-Anims streams enable structured changes to a scene. Together, these constructs add dynamic scenes. In other words, scenes are not static but can change over time. For instance, adding the following BIFS update to the previous scene would change the position of the video object at the time specified:

```
AT 10000 {REPLACE TR.translation BY 200 200}
```

Using this tool, an MPEG-4 terminal connected to a network can receive BIFS updates that modify completely the scene running on the terminal.

## 2.3 System Decoder Model

In MPEG-1 and 2 Systems, in order to provide precise definition for allowable synchronization and buffer states that can be produced by a legal bitstream (one that does not violate any syntactical or semantic rule of MPEG), MPEG defines the *system target decoder*, that is a theoretical target decoder used to model the behavior of a terminal. In analogy, MPEG-4 defines a *system decoder model*, which models the ideal behaviour of an MPEG-4 terminal. The bitstream is assumed to be de-multiplexed in elementary streams destined for various audio and video decoders, as depicted in Figure 3, taken from [N4848]. Elementary streams are composed of *access units*, the smallest entity to which time information can be attached.

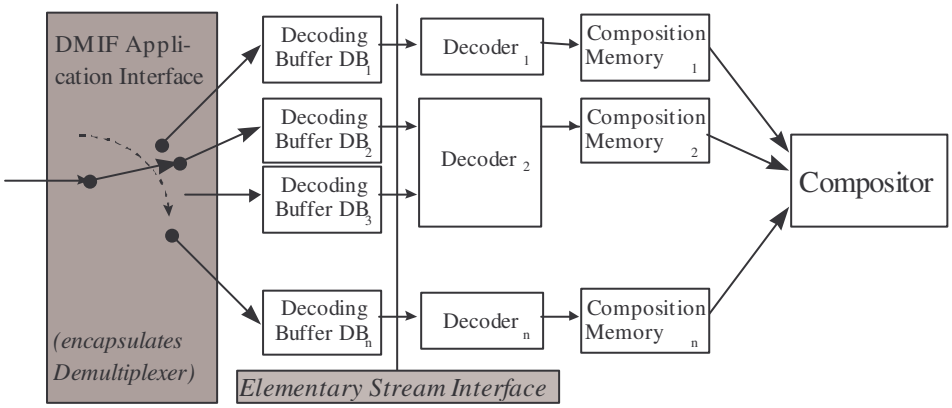


Figure 3: Systems Decoder Model

Access units are placed in decoding buffers $DB_1$,.. $DB_n$. The decoders start decoding the bits that compose every video or audio element at the time specified in the bitstream (decoding time stamp). The model is ideal because it assumes the decoding to be instantaneous, and fixed end-to-end delay (the time for the transmission of data from the stream encoder output to the decoder buffer is constant). The decoded frames are then placed in a composition memory. In the picture we see an element called "compositor". It fetches units from composition buffers and presents them on the display.

The exact size of buffers $DB_1$ ...$DB_n$ is specified in the bitstream. The size of the composition memory instead is not managed explicitly, although it can be estimated from profile and levels. Composition units stay available until the composition time of the subsequent unit is reached. During this period the compositor may access the composition as often as necessary. The model prescribes that in a legal bitstream if the elements are decoded and rendered using the times given in the bitstream, the buffers $DB_1$ …$DB_n$ never overflow nor underflow. Clearly no real decoder can instantaneously decode the bitstream elements. So it is up to the decoder system designer to insure that his system will be able to decode any bitstream that the system target decoder could decode.

## 2.4 Profiles

Profiles and levels are one of the most important MPEG tools to ensure interoperability and conformance testing. In general, profiles restrict the algorithmic features available; levels set a rough limit on processing power based on content complexity. A profile and level combination defines a conformance point. It ensures that content encoded for a particular combination can be exchanged and will work on any decoder implementation that conforms to that combination. For decoding hardware, the profile@level combination gives minimum performance constraints to be observed at design and manufacture time. For decoding software, the combination also may imply resource availability to be monitored at run time.

Applying these concepts to an MPEG-4 terminal, every elementary stream should comply with a profile and level. Audio and Video streams comply with Audio and Visual profiles, while BIFS elementary streams comply with scene graph and graphics profiles. Hence we can divide the set of profiles to which a terminal must comply in the following broad categories:

- Visual Profile

  Governs which visual object types can be present in the scene, thereby determining which coding tools can be used to code these objects and, hence, what their elementary streams (ESs) look like.

- Audio Profile

  Similarly to video, defines the audio object types present in the scene.

- Scene Graph

  Defines what types of scene description capabilities need to be supported by the terminal

- Graphics

  Defines which types of the graphics and textual elements can be used to build a scene

- OD profile

  Specifies the allowed configurations of the Object descriptor and Synch layer tools

One important difference with MPEG-2 to be noticed is that MPEG-4 is an object-based standard. Audiovisual scenes are hence composed of different objects. Levels do not define the maximum complexity per individual object but give bounds on the total of all objects in the scene. This is because the number of objects in the scene that need to be decoded simultaneously is important in determining the complexity of the decoder.

## 2.5 Reference Software Architecture

An MPEG-4 terminal implementation must be designed to process input from many sources in parallel, including various media streams and user interaction, and create a rich audio and video experience combining the input streams. Figure 4 [PE2002] illustrates the reference software architecture of such a terminal.

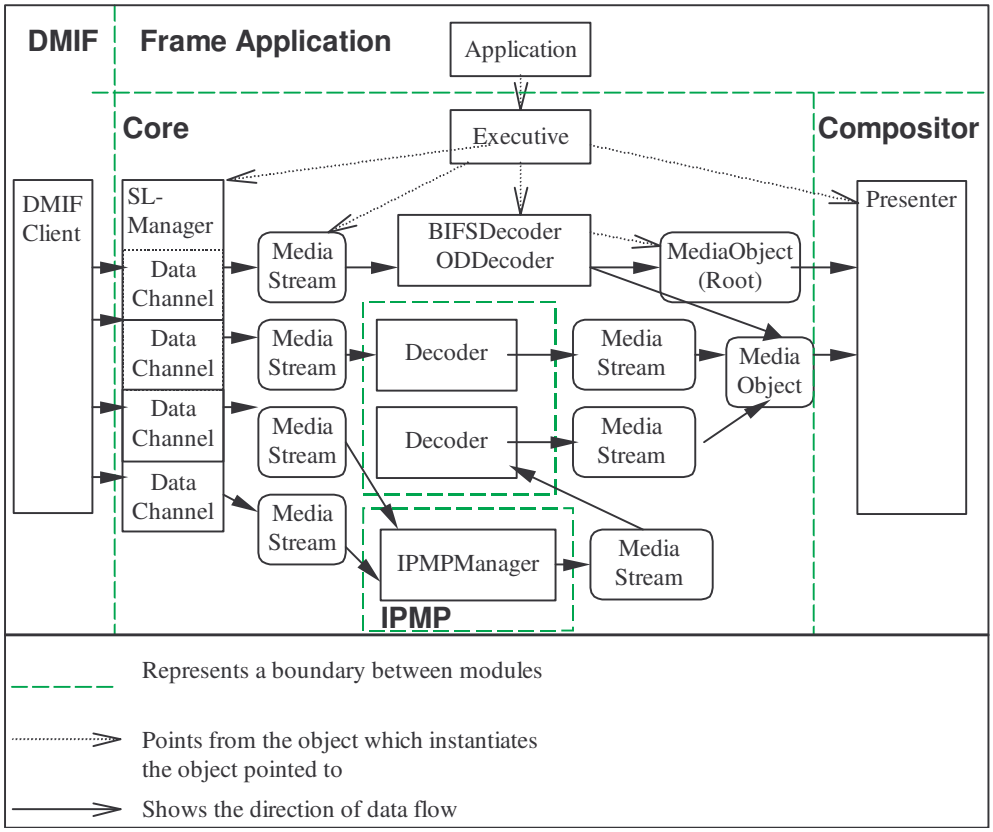


Figure 4: MPEG-4 Reference Platform

From left to right, DMIF is the implementation of the network part of the terminal; the Core part includes media decoders, media buffers and the scene tree; the compositor is responsible of the composition of the elementary streams. Binary scene description streams (BIFS) and compressed media data (elementary stream of encoded raw audio-visual information plus synchronization information) arrive through independent network channels

to their respective decoders. After the decoding step (done in parallel by independently running decoders), the uncompressed data units (composition units) are placed in playout buffers. A special decoder, called BIFS decoder, is in charge of decompressing the scene information stream and build a data structure called scene graph. The scene graph contains the description of the multimedia scene. It is built parsing the BIFS stream, and instantiating C++ objects for each node present in the description. A rendering engine cyclically traverses this linked list of nodes ("scene traversal"), takes composition units from each media buffer and composes the media following the specification of the scene description, adding synthetic graphics (text, 2D primitives) if necessary. This is the architecture of the MPEG-4 Systems reference software. The architecture is multi-threaded, each decoder runs in its own thread, and the composition is done by a different thread called "Presenter". A typical scene composed of audio-video and BIFS streams causes the launch of the following core and renderer threads:

- BIFS decoder: decodes the scene and possible following updates, creates the scene graph
- OD decoder: decodes objects descriptors and possible updates
- Audio Decoder: transforms compressed audio data in PCM samples
- Video Decoder: transforms compressed audio data in YUV/RGB samples
- Presenter/Visual Renderer: traverses the scene graph and performs composition and rendering at a given rate
- Audio Renderer: plays PCM samples, mixing samples from different sources if necessary

It is worth noting that MPEG-4 does not specify how to compose the different media. No composition algorithm is specified in the standard and included in the reference software.

## 2.6 Player 2D software implementation

We contributed to the development of the MPEG-4 standard specification and we implemented a 2D player for Windows platforms when the specification was still an early draft, providing feedback to the members of the group and participating to the open discussions. This work has been done in the context of the MPEG-4 reference software activity, even though this part is not normative but only informative. It was anyway a fruitful activity, validating the concepts of the standard during its development. Since it was a 2D compositor, only a subset of the MPEG-4 BIFS nodes has been implemented (a list of nodes implemented can be found at [MPE99]). This is the multimedia player that we used in the context of this thesis. We had to solve the following problems:

- The reference software creates the scene graph, using an object oriented approach. Every BIFS node has a corresponding C++ class. We had to add rendering capabilities to each node, without modifying this normative part of the standard. This was solved by the use of the proxy objects design pattern (Section 2.6.1).

- The scene graph generated by the BIFS decoder may contain multiple references to other nodes. We used display lists to manage the duplication of objects to display (Section 2.6.2).

- Media composition, rendering, and synchronization algorithms had to be implemented (Sections 2.6.3 and 2.6.4).

### 2.6.1 Rendering Objects

The basic player architecture and the flow of information is the one depicted in Figure 4. As said before, the Core set of classes does not include the implementation of the rendering capability of an MPEG-4 Systems terminal. Implementing proxy objects has solved the problem of adding 2D rendering to the framework. The "proxy objects" are a design pattern useful for implementing a specific functionality on a framework of classes. The idea is to add to each BIFS node class a "proxy" class that encapsulates the rendering functionality of that node. In this way, a separation between the rendering part and the node semantics is achieved. This is particularly useful when implementing an MPEG player while the standard specification is evolving, because when cosmetic changes of the specification occur, these do not necessarily impact on the implementation of the proxy object. Also, this architecture mirrors the task allocation in the player implementation, where people working on the rendering were not necessarily the same working on the Core framework. The last important advantage of this approach is that it confines the platform dependent part of the Player implementation to the proxy objects, so that the porting of the Player to another OS or graphic library implies only changes in the proxy objects. There is however, one drawback to using this design pattern. The proxy nodes are traversed before scene rendering, at each tick of the simulation. BIFS nodes fields are cached in proxy objects data structures. This implies that at each scene traversal a proxy object has to test whether the associated node has changed or not, or, if an automatic update mechanism is present, new values have to be retrieved from the original nodes whenever there is a change.

### 2.6.2 Building the Display List

One important feature of the BIFS language lets the author define a portion of a scene as reusable in other parts of the same scene. This derives from the VRML specification [VRML97] and the reason is to improve performance by caching the geometric computation done for the first occurrence and merely

rendering for subsequent occurrences. Since scene traversal is done on the proxy objects, a proxy object has been designed to render more than one instance of a BIFS node during each simulation tick. This is achieved by creating a "drawable context", a data structure containing all the information (bounding rectangle, clipping rectangle, visible/not visible, pointers to its appearance, geometry, event handler nodes) that a renderer needs to render the node. A drawable context is taken from a pool of free contexts at each visit to a node. If a node is visited twice during a single tree traversal, this means that it is being "used", hence two contexts are used in order to render the media object in different positions. The Visual Renderer has a list of used drawable contexts and uses this list to draw the media objects. The list of drawable context is hence a display list, since it contains the list of objects that have to be displayed at each tick of the simulation. The next paragraph illustrates the method used to display the media objects.

### 2.6.3 Painter's algorithm

The rendering of Visual Objects is a process that basically implies two steps: object composition in a memory surface and copy of the memory surface into the frame buffer. Figure 5 shows a MPEG-4 scene composed of several Visual Objects.

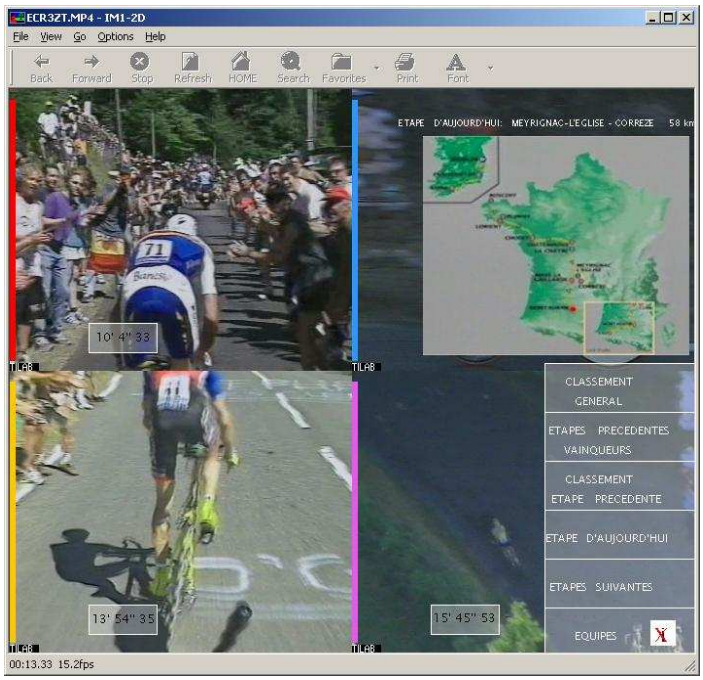


Figure 5: MPEG-4 Scene

It shows four MPEG-4 Visual Objects providing different views of a cycling event. BIFS text nodes are rendered on top of the videos displaying information about the event. The composition process starts when the BIFS and OD decoders have finished decoding the respective elementary streams. It is a cyclic process done at a specific rate (typically 25 or 30 times per second). First the BIFS tree is traversed to collect information about the nodes to draw. Media objects are drawn in the order they appear in the BIFS scene, unless an OrderedGroup node is in the scene. If this is the case, the drawing priority field indicates which media object should be drawn first. A list of objects to be drawn is created. Objects are then drawn using an algorithm that minimizes the number of objects to draw. Drawing all the media objects at each tick is not required and would be highly inefficient. For instance, a rectangle that is painted at the beginning of the scene and does not change appearance during the rest of the simulation does not require a 25 times per second drawing.

At each tick of simulation, a visual object is drawn if:

1. Its appearance or geometry nodes have changed (or, in the case of a movie node, there is a new frame to display).

2. Its position in the scene has changed.

3. The intersection with a drawn node is not empty; in that case, only the part that intersects the media object is drawn.

After composition, the memory surface is copied to the frame buffer.

### 2.6.4 Synchronization

In the following, it is described how elementary streams synchronization is achieved. The 2D Compositor implementation uses the reference software mp4 file format implementation for the playback of mp4 streams. As a consequence, this section applies only to local file playback, and does not cover all the issues concerning the synchronization of elementary streams in a "push" scenario. In a "pull" scenario, such as local file playback, elementary streams synchronization may be achieved without taking into account OCRs, since there are no problems of clock skew between client and server terminals.

The Core framework provides a set of classes that associate a clock to each elementary stream. If the authors of the MPEG-4 scene require two or more streams to be synchronized, they must signal in the ES descriptor of each stream at authoring time whether that stream has its own clock or shares the clock with other streams. Synchronization between streams is hence achieved by sharing the same clock between elementary streams. In an mp4 file, all the streams (tracks) are by default synchronized. As a result, no specific synch information has to be inserted, unless we specifically want to deviate from the default behaviour.

Let's consider now the case when the terminal has to synchronize two elementary streams: one carrying audio, the other video. Two different threads are used to render audio and video. The audio thread fetches audio frames produced by audio decoder and copies it in the audio board buffer. The audio thread is set to a higher priority than the video one, because failure in copying the audio samples into the audio board buffer on time (for instance in the case the CPU is loaded) heavily affects the quality of the presentation. Video frames, on the other hand, may be skipped without seriously affecting the overall quality. The audio and video renderers access composition buffers to get composition units. Core media buffers expose a set of methods to access the composition units in sequence or to get the most mature units, i.e., units whose composition time matches the composition time of the simulation. Video frames are fetched from the composition buffer using a method that compares the object clock to the composition time and gives back the most appropriate latest valid frame. As a result, video frames may be skipped if the terminal is late during the presentation. Audio frames on the other hand, are fetched one after the other, since no skipping is obviously desirable when rendering audio. Audio-visual synchronization is achieved using the Adjust Clock method of the Core. When elementary streams share the same clock, this method sets the clock to the value indicated as an argument. After the initial audio buffering required to smoothly play audio, the audio rendering starts, then the composition time stamp of the audio sample that is being played is used as an argument in the Adjust Clock function, causing all the elementary streams sharing that same clock to be in sync.

### 2.6.5 2D player profile and levels

The player is designed to comply with the Core2D Scene graph and Graphics profiles (both @Level2), and Visual Advanced Simple profile. The Graphics Core2D profile includes support for relatively simple 2D graphics and text: logos, animated ads, text, lines, and curves. Two levels are designed for this

profile: L1 and L2. They differ mainly for the number of IndexedFaceSet2D nodes: 15 and 31, and the number of points for Coordinate: 4 and 255.

The Scene Graph Core2D profile includes basic 2D composition, local interaction, animation through interpolators, BIFS updates, quantization, access to web links and subscenes, in addition to audio and visual elements. Two levels are set: L1 and L2. In terms of resources, the levels differ mainly for the maximum number of nodes allowed: 8191 in L1, 32767 in L2, and the maximum number of children that grouping nodes can have: 31 for L1 and 127 for L2.

Visual Advanced Simple accepts objects types Simple and Advanced Simple. It is useful in Internet Streaming but scales to television size picture and quality. There are six levels specified, that differ in the max resolution and bit rate supported: 176*144 to 720*576 bit rate from 128 to 8000 kbit/s. The maximum number of visual objects is four * simple or advanced simple.

These profiles capture the features of multimedia terminals that we considered in this thesis. They have been proposed by contributions made by several actors in the field of multimedia, and well capture the basic functionalities needed by modern multimedia applications.

## 2.7 Conclusions

The aim of this chapter was first to briefly introduce the multimedia standard on which we based our work, second to highlight our contribution to the development of the standard. We saw how MPEG-4 is the complete standard solution for the representation of synchronized multimedia presentations. We introduced key MPEG-4 concepts and the tools of the standard that are most important for the rest of this work: BIFS, the System Decoder Model and Profiles. We illustrated then the architecture of the reference software, and discussed some design principles introduced in the 2D player.

*Chapter 3*

# QUALITY-BASED MULTIMEDIA FRAMEWORK

In this chapter we present a framework to support quality-based 2D presentations. The framework is meant to be utilized on a General Purpose Operating System (GPOS), or in any other environment where computational resources cannot be reserved. In this context, applications must be adaptive and react to changes in the availability of computational resources caused by changes in the environment. Besides, since multimedia scenes have very different characteristics and computational needs, it is desirable that terminals adapt their rendering algorithms when the demand of increased resources needed by increased complexity of the scene cannot be satisfied. The framework is presented as an extension to the MPEG-4 Systems standard, in the sense that we defined optional quality information that can be carried in elementary streams descriptors. In this sense, it is a "legal" extension, because advanced terminals can exploit that information whilst others just ignore the additional descriptors and go on decoding the bitstream. In the following, we first provide evidence of the need of a management framework, showing how it fits into the tools available in the MPEG-4 standard (Section 3.1). Then in Section 3.2 we identify means to reason about the loss of value of multimedia applications introducing user and application level quality specifications. Section 3.3 introduces the framework architecture showing how distributed control and quality management of a terminal are achieved. Section 3.4 gives examples of the usage of the framework, considering applications composed of several video streams. Next chapter will focus instead on 2D graphics, and will provide examples of the integrated natural and synthetic content.

## 3.1 Motivation of the work

In the following, we analyse what are the tools available in the multimedia standard we considered and their relationship to the problem of designing multimedia terminals capable of reacting to changes in the environment and in the complexity of their input.

### 3.1.1 Profile and levels

Profile and levels are a powerful tool to declare the portion of the standard syntax that a terminal should be capable to process and express computational bounds. However, for a given profile and level, there might be variations in the resource demand. For what concerns just video decoding, it is well known that MPEG video does not consume a constant amount of processing [BMP98]. Considering an MPEG-4 video bitstream for instance, we have variations up to 2.5 of the resource usage during the decoding [MB96]. If we consider 2D composition, the variations are much higher, depending on a series of factors, for example: the number of objects to draw at each iteration, the properties of the objects to draw (fill, transparency), the size in pixels of the objects to draw, the quality of the rendering (antialiasing, deblocking and deringing filters), the level of user interaction. A very simple example is provided by a MPEG-4 Core2D graphics and scene graph bitstream with a 400x400 pixels rectangle scrolling from left to right over a bitmap still image 640x480. On a Pentium 1.3 Ghz, it consumes from 1% up to 3 % of the CPU time if the scrolling rectangle is opaque low quality, from 30% to 40 % if it is transparent low quality, from 40% to 60% if transparent and high quality (antialiased rendering). A BIFS update command that changes the properties of the object from opaque to transparent would dramatically affect the resource consumption during the playback of a bitstream. Clearly, the scene belongs to a given profile and level in all the cases, but we notice heavy variations of the CPU load. And this considers only one moving object. In order to keep into account changes in the computational needs due to changes in the scene we need a framework where

quality of the presentation is traded with resource usage and performance is monitored all over the presentation through appropriate metrics.

### 3.1.2 Graceful degradation

Great amount of work has been done by the Implementation Study Group (ISG) to define methods to enable the graceful degradation of MPEG-4 Visual streams, specifically video and 3D synthetic models (SHNC). As a result, the MPEG-4 Video standard syntax allows the insertion of optional complexity measures (basically statistics about the coded sequence) in the video bitstream at the encoder side, which let the terminal figure out the complexity of a part of a bitstream before decoding it. It has been shown how a terminal can control the decoding power needed to decode a stream using this complexity information and degrade the visual quality of the decoded image [MB96]. The same approach has been used by the SHNC group that identified complexity measures (size of projected surface of a model at different viewpoints) in order to control the decoding and rendering complexity of a 3D model. These efforts were motivated by the observed large variation in decoding and rendering times (up to factor 2.5 for video and up to 24 for SHNC models) depending on the incoming data [MB96] [LB1998]. Graceful degradation covers some areas, namely 2D mesh and texture decoding and 3D SHNC objects rendering. 2D scenes were not tackled by the ISG group. As we reported in the previous sub section, important variations in the complexity of a presentation of 2D scenes are easy to induce. Methods to predict the complexity of 2D rendering tasks are desirable and we will provide a model in the next chapter. It must be noticed however that in the best effort systems we are considering prediction of tasks can only be useful in limiting the effects of missed deadlines, not in avoiding them. Besides, high unpredictability due to deeper caching hierarchies, increasing prevalence of multiprocessors, variable processor speeds for power and heat management, and finally due to user interaction do require the need for QoS monitoring models.

### 3.1.3 MPEG-4 QoS management model

The QoS management model of MPEG-4 Systems is mainly oriented to adapt the standard to different service scenarios. In order to achieve scenario independence, the MPEG-4 object descriptor framework enables the insertion of desirable channel characteristic in the description of each elementary stream, and the delivery layer includes support to request channels with different properties. We observe that the notion of QoS in MPEG-4 is mainly oriented to network level. Annex E seems to confirm this statement:

*"In ISO/IEC 14496-1 the information concerning the total QoS of a particular elementary stream is carried in a QoS Descriptor as part of its elementary stream descriptor (ES_Descriptor). The receiving terminal, upon reception of the ES_Descriptor, is therefore aware of the characteristics of the elementary stream and of the total QoS to be offered to the end-user. Moreover the receiving terminal knows about its own performance capabilities. It is therefore the only possible entity able to compute the Quality of Service to be requested to the delivery layer in order to fit the user requirements. Note that this computation could also ignore/override the total QoS parameters."*

We can extend the ISO/IEC 14496 QoS management model in order to take into account presentation level issues. Since every stream has its associated elementary stream descriptor, which in turn contains a configurable QoS descriptor, we can easily extend the architecture with the QoS specification defined in the following sections.

## 3.2 QoS specification

As a terminal needs to control the value of a presentation and its complexity, a specification of how the different components of a presentation contribute to the value perceived by the user is necessary. In the process of defining quality descriptors we borrowed some concepts developed in the field of operating system schedulers, where the use of value functions to maximise system utility has been investigated. For instance in [LRS98] and [JLDB

1995], the authors report on the use of discrete quality dimensions to maximise task allocation in an OS scheduler. The use of these concepts in a multimedia framework we think it is an interesting application of those concepts.

### 3.2.1 User-level QoS specification

'Quality' in technical literature is commonly defined as: 'The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs' (ISO 8402). Quality is hence by definition connected to the *needs* of an entity. A QoS specification, performed with metrics that capture properties of an entity, clarifies these needs. In this sense, a user-level quality specification consists of a list of quality options for the relevant properties of a given application. The assumption is that the user is able to identify a number of desirable quality dimensions and their associated quality options and, most importantly, rate them. Since a multimedia scene is potentially composed of several streams, a global quality specification would not best capture the needs of the application. Consider for instance the scene reported in Figure 6. It shows a kart racing application that provides different cameras to enjoy a sport event.

Figure 6: Kart racing application

Three small QCIF (176*144 pixels) videos (10 fps) provide different views of the race and one CIF (352*288 pixels) video (30 fps) provides a better quality

view. The author can certainly identify a set of qualities that may express the concepts of timeliness (frame rate), size, or perception (post-processing filters, antialiasing) for the presentation of this event. Clearly, if these options are declared per stream, the rendering terminal can allocate resources at a finer granularity, optimising the utility of the system. As an example, suppose that during the playback of the kart application the frame rate of the CIF video suddenly decreases, because the CPU is overloaded. The CIF video is probably not the best candidate for quality degradation, because it is the most important view for the user. Instead, the three preview QCIF could be degraded (reducing the frame rate or substituting the videos with some text), and the released resources could be used to display the CIF video at full quality. In order to achieve that, quality options should be declared per stream, and each stream should convey a notion of importance relative to the others. Since multiple quality dimensions may be associated to each stream (ex: frame rate options, frame size options, perceptual quality options), there is also a need to rate each dimension, in order to choose among possible options taking into account the value that each user assigns to each dimension. For instance the quality dimension "size" of a video stream is a very important quality parameter for an action movie, while for a TV news program it is not as important. Similarly, the size of the screen is very important for remote surveillance applications, which on the other hand might tolerate frame rate reduction.

Figure 7: Video catalogue

A similar concept applies to 2D graphics. Figure 7 shows a video catalogue application. Six small QCIF videos scroll from right to left in the lower part of the screen. When the user clicks on one video, a bigger view is displayed in the upper part of the screen. Small video scrolling is achieved via a BIFS construct called time sensor, which fires events at a regular rate. BIFS quality dimensions may include viewport (display) sizes and frame rates for the animation. If the author assigns a value to each dimension, when the presentation system is in the need of assigning resources to each stream it may trade off resource availability and author preferences. The above examples suggest the structure of a user-level QoS specification: each stream should convey its list of QoS dimensions, and their contribution to user satisfaction. The relative importance of each stream should also be evident by the description. According to this, a user-level specification of the application shown in Figure 7 might be:

Stream 1: (Video CIF)

- Perceptual Visual Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 3
- Frame Rate {25, 12, 5} Dimension Value: 2
- Frame Size {CIF, QCIF} Dimension Value: 1
- Stream Value {3}

Streams 2, 3, 4: (Video QCIF)

- Perceptual Visual Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 2
- Frame Rate {10, 5, 2} Dimension Value: 1
- Frame Size {QCIF} Dimension Value: 3
- Stream Value {2}

Stream 5: (Audio)

- Perceptual Audio Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 1
- Audio Sampling Rate {44100, 22050} Dimension Value: 2
- Stream Value {4}

Stream 6: (BIFS)

- Perceptual Visual Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 1
- Frame Rate {10, 5, 1} Dimension Value: 2
- Frame Size {100, 50, 10} Dimension Value: 3
- Stream Value {1}

In the above example, a user-level specification is associated to each stream of the application. It is composed of a list of *quality dimensions* declared for each stream. In the example the quality dimensions capture timeliness, size and perceptual qualities, but the list is not meant to be exhaustive. Each quality dimension contains a list of ordered *quality options*, in descending quality order. Each dimension has an associated *dimension value,* which establishes a ranking among the dimensions in the user satisfaction. For instance in stream 1 the user rates Perceptual Visual Quality as the highest quality to preserve, followed by frame rate and finally frame size. This means that the first action that a terminal would consider when degrading the quality of the scene is a reduction in frame size. Each stream has an associated *stream value*. This establishes a ranking among the streams of a presentation. Streams with smaller values are the first ones to be degraded. We assigned the highest value to audio, and then to the CIF video. This is because audio is the media that is more sensitive to degradations, and even minimal quality reductions have a strong impact on user satisfaction.

The assumption on which this specification is based is that the user is able to identify some qualities for each stream and rate them. We may wonder if it is always possible to identify a set of quality dimensions. We believe that for most scenes the answer is yes. The concepts of timeliness, quantity of data, perceptual quality, well fit with the nature of multimedia streams. What we can say is that it is not always possible to have all descriptors for each stream. For instance the concepts of optional frame rates for video cannot be applied

to all encoded video. As reported in [IFS03], sometimes there is not a fixed frame rate nor a constant group of pictures structure (I, P, B pattern) for the whole bitstream, that will enable, for instance, the decoder to change frame rate skipping some data. So we foresee that it is not possible for all streams to have a fixed set of descriptors. But this is then connected to the inner meaning of a user QoS specification: to provide hints to the terminal about which quality dimensions degradations can be planned and safely executed.

### 3.2.2 Application-level QoS specification

It is agreed in literature [CCG92] that QoS is a concept that spreads at least over User, Application and Network levels. In this work we consider User and Application levels and we assume that the input data is always available at the terminal when it needs it. This assumption can be relaxed considering a particular service scenario and suitable network quality metrics [D2199]. In order to define the application level metrics suitable for multimedia streams, we followed Campbell's taxonomy [CAH96]. He defined an end-to-end QoS framework for audio-video streams based on the notion of *flow*. A flow is defined to characterize the production, transmission, and consumption of the state associated with a single media. The framework includes the definition of QoS specification, which captures quality of service requirements, and QoS mechanisms that realize end-to-end QoS behavior. The specification is different at each system layer, and as a consequence at each layer there should be QoS mechanisms specified too. According to his work, a taxonomy for the QoS metrics at each layer of a multimedia terminal is comprised of:

- Flow synchronization: specifies how tight the relationship among different media streams is.

- Flow performance: provides performance metrics for each stream in the application.

- Level of service: indicates the kind of guarantees that are provided by the layer addressed by this specification: deterministic, predictive, best effort.

- QoS management policy: specifies the kind of adaptation that the flow can tolerate, providing suggested actions to be taken when there are QoS misses.

- Cost of Service: provides an indication of the price associated to different levels of service.

Campbell considered mainly single audio video flows, and as a consequence the presentation system metrics did not include potential problems arising when dealing with multiple (natural and synthetic) streams. Since MPEG-4 enables the playing of several streams concurrently, we are interested in finding a QoS specification and management policies that take into account audio-video and synthetic flows, and their relationships. In the following sections, we provide a characterization of MPEG-4 elementary streams with the aim of finding metrics to fill up the previous QoS taxonomy and provide the mapping between user-level specification and application level.

#### *3.2.2.1 Characterization of MPEG-4 elementary streams (core 2D perspective)*

MPEG-4 elementary streams are composed of *continuous* and *non-continuous* media. Continuous media (i.e. video, audio streams) are comprised of several media units. A basic characteristic of continuous media is their uninterrupted nature, which entails processing of large amounts of data with real time deadlines. Non-continuous media (i.e. still images) are composed of single media units and are somehow easier to handle for a terminal. Their quality can be specified by parameters like colour depth, resolution, sampling size. These parameters are independent of temporal relations, and are not strictly under the influence of the presentation system, being decided at authoring time. A particular kind of stream is the scene description stream (BIFS). Even

if this stream can be considered as a non-continuous stream, some kind of scenes requires regular updates, and as a consequence BIFS may have also the properties of continuous data.

*3.2.2.1.1 Intra-Stream Synchronization*

Continuous media can be represented as a sequence of units, e.g. audio samples, video frames, or animation frames. Rendering a continuous stream implies the display of a sequence of samples with a regular frequency, namely the sampling frequency, which should hence match the rendition rate. *Intra-Stream Synchronization* is the process aimed at preserving the temporal relation between units of continuous media. The quality of continuous media depends on static authoring issues (i.e. sample rate, compression level), but is also under the influence of the presentation terminal. Because of operating systems, networks, disks etc, general-purpose computing environments are inherently non deterministic in nature. Network delay jitter (non constant variation in the arrival times of media units), if not handled at the presentation terminal, may interfere with the regular display of presentation units, because a unit that is supposed to be presented at a specific instant may not be available at the terminal. CPU overload may cause a similar effect, since the terminal is unable to process and display on time the units of a media. The scheduler of the operating system may also interfere in the regular display of frames, if the thread that is in charge of performing the display is not woken up long enough before the deadline. Hence there can be delays and disruptions in the presentation of media frames. Audio and video flows are both continuous media, but with different properties. Since errors in audio playback are more easily detected by end-users in form of small glitches, playback techniques try to avoid any audio frame skipping, trying to keep audio data continuous, with insertion of silence or white noise or interpolation of previous packets when a media unit is missing (due to packet loss or late arrival). Video allows somehow more flexibility, since it is possible to present for a longer time, duplicate or drop a limited number of frames,

without heavily impacting the user-perceived quality. BIFS streams can be considered continuous media, when they contain scene updates at some specified rate. Application level QoS metrics for continuous media try to quantify the *continuity* of the flow of media streams. We identified:

- Average frame rate: it captures the rate at which the presentation units of a stream are displayed. A nominal frame rate is selected as QoS baseline according to the available resources and user preferences.

- Maximal frame rate variation: indicates the allowable rate change in the display of presentation units. It is a number comprised between 0 and 1. If $f_r$ is the nominal frame rate and $r_c$ is the rate change, then the stream frame rate can vary between $f_r$ and $f_r*(1-r_c)$. The nominal frame rate depends on the quality option that the stream is targeting. This metric depends mainly on the capabilities of the terminal and the service scenario. If the maximal frame variation is violated then the system has to run management policies and compute new QoS baselines.

- Max instantaneous drift: indicates the maximum delay accumulated by a decoding/rendering process. It is defined as the difference between the current time of the clock of the stream and the time stamp of the unit being processed. It is an indication of the *progress* of the stream. We hence assume that every media unit has an associated time stamp (or it can be implicitly available), and that the terminal maintains a clock object for each stream being processed. This is exactly what the MPEG-4 architecture defines: elementary streams are time-stamped information and each stream has its own time base.

As we said before, the rate of a stream can change due to lost and late packets, too high decoding or display time. All these factors can determine a decrease of the presentation rate of a stream. If the maximum rate change is

violated then the presentation of the stream might be aborted or the flow should change rate towards a different QoS baseline. It should be noted that performance metrics are dependent on the presentation algorithm and the service scenario. In networks where constant end-to-end delay is not guaranteed, other metrics like coefficient of output frame variation may be added [ITI2000], in order to have a measure of the smoothness of the output units. However, for the scope of this work the above metrics are sufficient to monitor the performance of the display of each stream.

These metrics are valid for video streams and continuous BIFS streams, and, considering frequency instead of frame rate, also for audio playback. However, as we said before, audio streams are less tolerant to errors, hence presentation level rendering techniques are somehow limited to masking audio errors.

*3.2.2.1.2 Inter-Stream Synchronization*

Inter-Stream synchronization is meant to preserve the temporal relation between media objects. A common example is the relation between audio and video in a movie with a speaker (lip-synch). The term 'skew' indicates the time difference between related presentation units from different streams. Since it defines the affordable synchronization boundaries, synchronization can be defined by specifying a maximal skew between two media streams. It has to be observed that since we are dealing with presentation level synchronization between streams, it is assessed by the end user, and it is qualitatively judged by human perception. As a consequence, it is not straightforward to derive quantitative measures of quality of synchronization. Besides, skew values depend also on the specific media streams. For instance two audio streams to be played simultaneously in each channel of a stereo reproduction system have tighter requirements in respect of audio video synchronization; requirements of video and annotated text are yet less stringent. Steinmetz [RS1996] reported a series of experiment about human media perception,

which may be used as QoS guidelines. The results of his work show that streams that have skew values greater than zero are still perceived as in synch until the skew reaches a precise value. This means that some misalignments can be tolerated without degrading the quality of the presentation. The results are reported in the following table.

| Media | | Mode, Application | QoS |
|---|---|---|---|
| Video | Animation | Correlated | +/- 120 ms |
| | Audio | Lip-synch | +/- 80 ms |
| | Image | Overlay | +/-240 ms |
| | | Non overlay | +/-500 ms |
| | Text | Overlay | +/-240 ms |
| | | Non overlay | +/-500 ms |
| Audio | Animation | Event correlation | +/- 80 ms |
| | Audio | Tightly coupled (stereo) | +/- 11 us |
| | | Loosely coupled (dialog with various participants) | +/-120 ms |
| | | Loosely coupled (background music) | +/- 500 ms |
| | Image | Tightly coupled (music with notes) | +/-5 ms |
| | | Loosely coupled (Slide Show) | +/-500 ms |
| | Text | Text annotation | +/- 240 ms |
| | Pointer | Audio relates to showed item | -500 ms, 750 ms |

Table 1: skew values associated to perceived synchronization

From the table we observe that audio-video streams are considered in synch if the absolute skew between them is less than 80 ms. If the skew value is 160 ms, the streams are then perceived out of synchronization. Using the results shown in the table, we have then quantitative values to assess quality of synchronization between two streams.

#### *3.2.2.1.3 QoS management policy*

QoS management policies specify actions ("recovery actions") to be taken when performance or synchronization QoS for a flow is not met. When a terminal detects a performance problem, it degrades the quality of the presentation through recovery actions. Recovery actions follow directly from user-level QoS specification. If stream importance and quality dimensions and quality options are declared and accessible by the terminal, a recovery action will then select first a quality dimension according to the value information and then select a lower quality option according to the available terminal resources. We will illustrate in detail quality management in Section 3.3.2.

## 3.3 Framework Architecture

The architecture we propose is an extension of the MPEG-4 architecture described in Chapter 2. The architecture extensions are meant to provide support for the control of the playback of a multimedia scene according to a user provided QoS specification. In the design phase we borrowed design principles investigated in control theory for discrete event systems [KBE99], where the issues of controllability, robustness and reactive configuration have been identified. Even if a multimedia terminal can be defined as a time-driven system (at every clock tick events occur that advance system behaviour), the interactions with a hosting environment are rather event-driven (events not known in advance and not necessarily coinciding with clock ticks interfere with the behaviour of the terminal). *Controllability* refers to the capability of a system to performing according to the specifications when input load changes dynamically (between specified bounds). This is particularly important for a multimedia terminal whose data load may change over time. *Robustness* deals with unanticipated variations in the environment of a system. As the environment departs from the expected one, the system degrades gradually in performance rather than showing a catastrophic failure. This concept is relevant for terminals running in an un-controlled environment (i.e. general

purpose operating systems) where applications compete for system resources. *Reactive configuration* indicates the capability of a system of reconfiguring its main algorithms scaling their computational needs, and it's triggered by an evaluation indicating that the system is not accomplishing its mission. Controllability and robustness have been addressed by a design pattern called distributed control, illustrated in Section 3.3.1. Reactive configuration has been addressed in the context of the quality management process (Section 3.3.2). In the following we illustrate the architecture we propose for distributed control and quality management.

### 3.3.1 Distributed Control

In order to address robustness and controllability, the load of a system must be constantly monitored, variations in the load of a terminal identified, and a reconfiguration of the terminal data processing algorithms can eventually take place in the context of a quality management process. A major problem in dealing with the load of a multimedia system is that it varies over multiple timescales. More precisely:

- Data changes in continuous media usually take place at a time scale of tens of milliseconds (i.e. time-varying complexity of MPEG decoding [BMP98])

- Scene changes usually take place at a time scale of seconds (i.e. BIFS scene updates possibly add or remove streams from the presentation, which may impact the terminal load)

- User changes have time scales of minutes (i.e. user interaction may trigger events that cause load variations)

- Environment changes have time scales of hundreds of milliseconds (i.e. load variations due to other applications running on the same OS)

In order to cope with these requirements, a hierarchical control architecture has been conceived. In the theory of hierarchical control systems [ZU97],

separate control units make independent observations and have their own a priori information and control variables. Control is achieved via collaboration among independent units. A control unit is constituted of an *observation process* that collects information about the controlled system and its environment and a *decision process* which uses this information and any a priori information to perform the control. Following these design principles, the decoding and rendering pipeline described in Chapter 2 has evolved in the one in Figure 8.

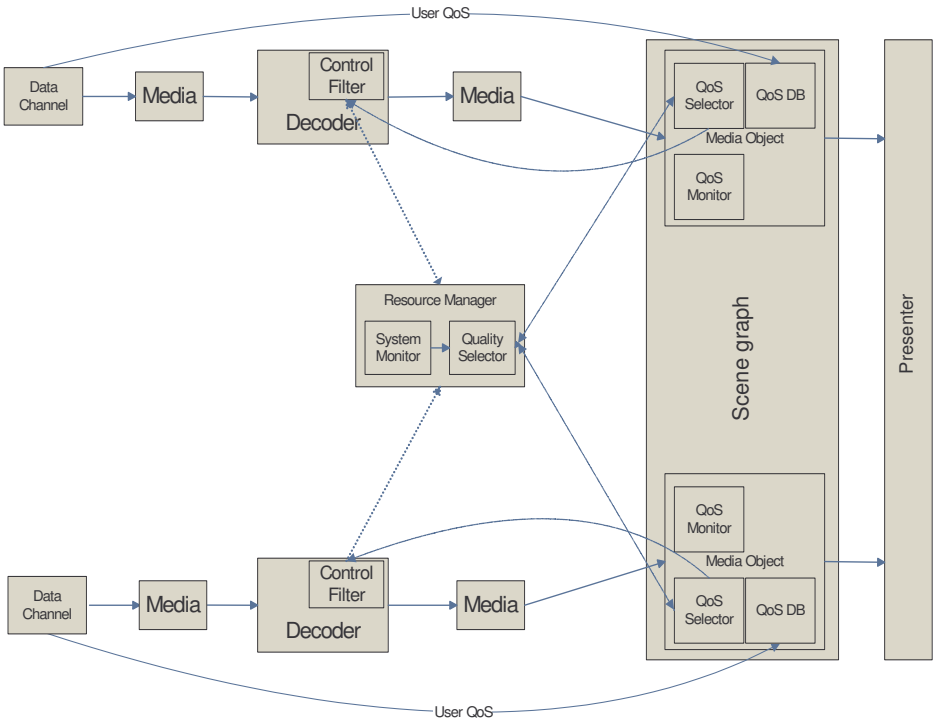


Figure 8: Extended decoding and rendering pipelines

The architecture is an extension of the MPEG reference software's one, since the main concepts of having separate pipelines running in separate threads, a shared Scene Graph with media objects that share decoder composition

buffers, and a presenter thread that performs the rendering of the nodes of scene graph, are also present in the proposed architecture. We addressed here the control at different time scales, achieved with the collaboration of the different control units located in the Decoders and the Media nodes, and a new component called Resource Manager ("RM"). Decoder's control units observe dynamic load variations of continuous media (i.e. video or BIFS scenes decoding), and the progress of the decoder process. Control units contained in media nodes ("QoS monitor" in Figure 8) monitor the continuity of the flow of the associated media. Control units encapsulated in the root of the scene graph monitor graph changes (caused by user interaction or scene updates). The System Monitor element of the Resource Manager observes instead environment changes, typically CPU usage of the terminal process and other OS processes. As we will see in the next subsections, observation performed by control units may lead to independent decision processes, or notifications to the Resource Manager that will eventually start a new quality assignment process (in this sense we designed a hierarchy in the control). The need for a separate entity is justified by the fact that we have to centralize the analysis of QoS misses and take actions on the basis of a global view rather than considering only local single flow violations. Each flow should signal the QoS misses to a unique entity since they have to be processed on the basis of a priori information (User QoS) and available resources. In the following we illustrate more in detail the control components of the framework for video and audio flows. BIFS streams media and decoding control units will be tackled in Chapters 4 and 5.

#### *3.3.1.1 QoS Monitor control units*

QoS Monitor control units ("QoSM" hereafter) observe the consumption of media composition units, calculate media specific presentation QoS (using the metrics identified in 3.2), and notify the RM when violations are detected. QoSM units are media specific, and have been modeled as included in media nodes contained in the scene graph. Since each stream in a presentation has

an associated media node, the nodes are good candidates to include QoS monitoring functionality. In briefly describing video and audio units, we assume a model where the decoder is always producing data until either the composition buffer is full or the decoding buffer is empty. This model is particularly suited for applications that have to compose several different media combining media units from different buffers, and it is effective in service scenarios where frame arrivals suffer from jitter. We further assume that every stream has an associated clock that provides the notion of object time base of the stream.

#### *3.3.1.1.1 Video*

The rendering function of video media nodes regularly access composition buffers and display the frame whose timestamp is the closest to the value of the stream clock. The renderer has the possibility to skip some composition frames or repeat frames already displayed in order to cope with delays of the decoder or the renderer itself. This is a form of independent control and decision process governed by a clock. QoSM units observe the quality of this process using application level QoS metrics. Whenever a media node has a new frame to display, the current frame rate $f_c$ is calculated. If $f_n$ is the nominal frame rate selected by the current QoS baseline and $r$ the max rate variation, then $f_c$ has to be between $f_n$ and $f_n - r$. If this condition is missed for a number of consecutive times (given by application requirements), a notification is sent to the Resource Manager. This will eventually start management actions.

#### *3.3.1.1.2 Audio*

QoSM audio units observe the consumption of audio samples and monitor the progress of stream, comparing the time stamps of the samples being played with the stream time base clock. When a deviation is detected and is superior to a tolerable skew, then proper action is taken: samples are inserted or the clock is updated. This control is meant to compensate potential drift

between the audio board clock and the system clock. QoSM units hence check the progress metric. Control on the audio clock is also meant to provide synchronization between streams. Several algorithms to achieve synchronization are available in synchronization literature [IT2000] and MPEG literature (master/slave technique reported in [RKR96]). QoS synchronization specification is connected to the chosen algorithm. The QoS synchronization specification for MPEG-4 terminals may express the acceptable violations of the MPEG-4 Systems Decoder Model. As we explained in Chapter 2, the System Decoder Model provides an ideal definition of the behaviour of the terminal, ideal because it assumes instantaneous decoding time and constant end-to-end delay. This implies that composition units are available for presentation at the time expressed by their composition time stamp. It follows that, in the ideal model, assuming the same time base, then composition units from different streams with same time stamp have to be composed at the same instant, and any drift would be a violation of the model. However, some streams may tolerate little violations of the model. If two or more streams are supposed to be synchronized, they have to refer to the same time base, and so they share the same clock object. Intra and inter stream quality of synchronization then depends on how close the composition time stamps of the units being displayed are, in respect to the stream clock. QoS for synchronization does not require interactions between the resource manager and other control units, but has to be taken into account by the synchronization algorithm as proper thresholds to consider before taking corrective actions.

#### *3.3.1.2 Control Filter units*

These units observe the incoming data of media decoders, and perform media dependent control, independently or triggered by the Resource Manager. If the decoding process of a media is too high or the terminal load needs to be lowered, some action can be taken on media decoders. Control on media decoding is typically performed on video decoders, but in Chapter 5 we will

illustrate the need for a BIFS control unit and its design. The design of video decoding control units is strictly coupled with the design of the decoder they control. Two main strategies exist: quality reduction and frame skipping. Quality reduction is inspired by the imprecise computations model [JKWAJ91]. If a video decoder provides some form of downgraded decoding algorithm (i.e. complexity scalability via IDCT data pruning [PS2001]) control units may predict complexity of incoming data, using either complexity measures carried in the MPEG-4 Visual syntax [MB96] or performing an estimation extracting parameters from bitstream headers [BMP98][[LCZ2001]] and regulate the decoding process to produce frames of inferior quality if the complexity of a frame is superior to a threshold selected by the RM. A simpler strategy is frame skipping, where control units regulate the decoding process forcing the decoder to skip part of incoming data.

#### *3.3.1.3 Resource Manager: system monitor*

General-purpose operating systems are best effort operating systems, and hence they make no guarantees to applications. They do not reject an application during overload, but instead reduce the processor time available to other applications to "make room" for the new one [RBS2000]. Since this kind of situations cannot be predicted, applications should monitor the usage of the resources of the terminal and be able to dynamically adapt their needs. The System Monitor component of the architecture monitors mainly CPU usage, at regular intervals. The load of the multimedia terminal process and the total load of the system are quantified. It must be noticed that resource monitoring alone is not sufficient to determine if an application is successfully carrying out its tasks. A busy processor could mean that it is handling well a lot of work, or that is overloaded. Resource monitoring has hence to be coupled with application QoS monitoring. If QoS violations are detected and CPU is overloaded, then the quality management process will reduce the computational needs of the terminal, reassigning quality options to each stream.

### 3.3.2 Quality Management

Quality Management is the process of assigning the quality of the streams that compose a presentation on the basis of a user provided QoS specification and the amount of the resources of a terminal. The assumption we made in 3.2.1 is that the author is able to identify a set of quality dimensions for each stream, with their associated quality options, and rate them. The presentation system then selects the quality options for each quality dimension of a stream in order to maximize system utility and user satisfaction. The quality selection process may run several times during the execution of a presentation. In the proposed architecture, the quality management is performed via collaboration of the Resource Manager (Quality Allocator component) and the following components of the media nodes associated to each stream in a presentation: Quality DB and Quality Selector. The Quality Allocator runs a resource management algorithm in order to assign qualities to each stream. The Quality DB of each media node contains the stream User QoS specification conveyed to the terminal as specified in 3.3.3. The Quality Selector of each media node is responsible of setting the presentation of each stream to the combination of quality options identified by the Quality Allocator. In the following section we describe how quality options are assigned. Section 3.4 will provide examples of how the Quality DB and Quality Selector are used in the framework.

#### *3.3.2.1 Quality allocator*

The process of assigning quality options to streams of a presentation is connected to the available resources of a terminal. If the terminal had infinite resources, the best quality options would always be selected. Since this is obviously not the case, qualities have to be assigned so that the resource usage does not exceed the available resources. However, since the load of a terminal changes over time, quality options have to be negotiated and reassigned whenever changes in the load are predicted (if possible) or detected. We can formalize the quality allocation problem in the following way.

Let the following be given:

$S_1, S_2, ..., S_n$ Streams that compose a multimedia presentation

$Q_1, Q_2, ..., Q_n$ Quality specifications associated to each stream

$R_1, R_2, ..., R_n$ Resource consumption of each stream $S_i$

A quality specification $Q_i$ for a stream $S_i$ is composed of a finite set of *quality dimension* vectors (ex. PerceptualQuality, FrameSize, FrameRate etc.)

$$Q_i = \{Q_{i1}, Q_{i2}, ..., Q_{idQi}\}$$

Each $Q_{ij}$ is composed of a discrete number of quality options (ex. FrameSize{CIF, QCIF})

$$Q_{ij} = \{h_1, h_2, ..., h_{dQij}\}$$

in order to rate the quality options of quality dimension $Q_{ij}$, we define

$$u_{ij} : Q_{ij} \rightarrow \{1, 2, ...., |Qij|\}$$

$u_{ij}$ is a bijective function defined so that if $u_{ij}(h_1) > u_{ij}(h_2)$ then quality option $h_1$ is "better than" $h_2$.

If each quality dimension $Q_{ij}$ of a quality specification $Q_i$ has an associated dimension value $w_{ij}$, we can express the quality of a stream $S_i$ as a function of the chosen quality options:

$$u_i : Q_i \rightarrow \mathbb{N}$$

$$u_i(q_i) = \sum_{j=1}^{di} w_{ij} u_{ij}(q_{ij})$$

In a similar way, we can express the quality of the presentation in terms of the qualities of each single stream $S_i$ and its stream value $p_i$ :

$$u(q_1, q_2, ..., q_n) = \sum_{i=1}^{n} p_i u_i(q_i)$$

We aim at maximising $u(q_1, q_2, ..., q_n)$ subject to

$$\sum_{i=1}^{n} R_i < R_{\max}$$

where $R_{\max}$ is the available resource of the terminal.

As shown in [LRS99], this class of combinatorial problems (Single Resource Multiple QoS dimensions) are NP-hard, since they can be considered as an instance of the 0-1 Knapsack Problem. We use a simple greedy algorithm to choose qualities based on the stream value and the resource consumption associated to each quality level. We chose a greedy algorithm on the basis of the following considerations:

1. The number of streams and quality dimensions is not expected to be so high to justify a more complex algorithm. Note: the framework assigns qualities to streams. 3D frameworks using similar approaches assign qualities to each object in a 3D world; in this case more complex algorithms are necessary since the number of objects and the number of quality options per object can be quite high [NRLD2002].

2. The framework assigns the same quality options to streams with the same stream value. Streams with the same stream values are

considered to share some semantic meaning. Quality options are hence assigned so that streams with the same semantic value share the same quality options. This also simplifies the algorithm.

We observe that point 2 it is not a limitation, since if we need different quality options to be allocated to different streams it is sufficient to assign them different stream values. The greedy algorithm performs the following steps, assuming as input a value Rmax of available CPU:

- Order the streams in decreasing importance, grouping streams with the same stream value.

- Order quality levels in decreasing quality order. Ordered quality levels are determined combining the quality options of each quality dimension, starting from higher dimension values to lower ones.

- For each group of streams, starting with the group at highest stream value:

    - Start selecting quality level that corresponds to highest quality.

    - Estimate resources for this group at the selected quality level.

    - If the estimation is higher than Rmax, select a lower quality level and perform again estimation until the condition is satisfied.

    - If no combination of quality level is found, select lowest quality level and raise exception.

An example of how complexity estimation and quality levels are identified is reported in Section 3.4.

### 3.3.3 Carriage of QoS descriptors

All the streams of a MPEG-4 presentation are described by object descriptors, conveyed in the Object Descriptor (OD) stream. An object descriptor is defined as a set of elementary streams descriptors (ESD) that provide one or more encoding of the same media, plus descriptors with semantic information about an object and hooks for security. Each elementary stream descriptor contains information about its associated stream: location, type, encoding, bit rate, buffer requirements, and decoder set up information. The standard allows the insertion of QoS descriptors in a elementary stream descriptor. A QoS descriptor consists of an index of predefined QoS scenario, or a set of QoS qualifiers. The following syntax, taken from the ISO 14496-1 specification, describes the structure of a QoS descriptor:

```
class QoS_Descriptor extends BaseDescriptor : bit(8) tag=QoS_DescrTag
{
  bit(8) predefined;
  if (predefined==0) { QoS_Qualifier qualifiers[]; }
}
```

QoS qualifiers are hence defined as derived classes from the abstract QoS_Qualifier class. They are identified by means of their class tag. Unused tags values up to and including 0x7F are reserved for ISO use. Tag values from 0x80 up to and including 0xFE are user private. Tag values 0x00 and 0xFF are forbidden. A number of qualifiers have been hence predefined by ISO to express: maximum end-to-end delay for the stream in microseconds (MAX_DELAY), preferred end-to-end delay in microseconds (PREF_MAX_DELAY), allowable loss probability of any single access unit (AU) (0.0 1.0) (LOSS_PROB), maximum number of consecutively loss AU (MAX_GAP_LOSS), maximum size of AU in bytes (MAX_AU_SIZE), average size (AVG_AU_SIZE), max arrival rate of AU in AU/second

(MAX_AU_RATE), ratio of the decoding buffer to be filled in case of pre-buffering or rebuffering (REBUFFERING_RATIO). Since it is possible to add QoS descriptors via the QoS_DescrTag, we can extend the standard adding descriptors that carry user-level quality of service specification for each stream. New descriptors, conforming to the specification we introduced in Section 3.2, can be declared in the following way (taking as example the quality dimension "Frame Rates"):

```
class QoS_Qualifier_FRAME_RATES: public QoS_Qualifier
{
        public:
        SDLInt DimensionValue;
        SDLArray<8> FrameRates;
};
```

It should be noted that these qualifiers are vital information for the framework presented in this chapter. If the terminal detects a QoS specification for a stream then it includes the stream in the resource management process. If no descriptors are detected for a stream then the playback is performed in an un-controlled way.

## 3.4 Experimental Results

In the following, two examples of utilization of the framework are proposed. The examples are meant to provide evidence of how the different elements that we proposed so far cooperate in order to make a terminal adapt its running conditions.

### 3.4.1 Example 1

We consider here a scene constituted of ten video streams. Ten Movie Texture nodes are spatially composed in a BIFS scene as illustrated in Figure 9. The 704x576 screen is divided in four 352x288 areas, containing from top left:

1 video CIF MPEG-4 Advanced Simple Profile (ASP) 1 Mb/s, 25 fps (V1)

4 video QCIF Advanced Simple Profile 250 Kb/s, 25 fps (V3-V6)

4 video QCIF Advanced Simple Profile 250 Kb/s, 25 fps (V7-V10)

1 video CIF MPEG-4 Advanced Simple Profile 1 Mb/s, 25 fps (V2)


Figure 9: Example 1, two CIF streams and 9 QCIF BIFS scene

We assume, as we stated in Section 3.2, that the author/user is able to identify a set of qualities for each stream (dimensions), with associated options (quality options) and rate them. A User QoS specification for the scene in Figure 9 may assign different values to the CIF and the QCIF videos. Three levels of interest may be expressed (in descending order): V1 and V2 (the two CIF videos) have the same highest value, V3, V4, V5, V6 share the same middle value and V7, V8, V9, V10 have separate decreasing values. Besides, the user may declare that is willing to accept only frame rate

variations, and accept the QCIF nodes to be completely switched off, if it is necessary. We express this in the following way:

Streams CIF (V1 and V2)

QoS_Qualifier_FRAME_RATES{FRAME_RATES [25,8,2] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 6}

Streams QCIF (V3, V4, V5, V6)

QoS_Qualifier_FRAME_RATES{FRAME_RATES[25,8,2,0] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 5}

Streams QCIF (V7, V8, V9, V10)

QoS_Qualifier_FRAME_RATES{FRAME_RATES[25,8,2,0] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 1-4}

As we say in 3.3.3, this specification can be included in MPEG-4 systems elementary descriptors as additional information conveyed to the terminal. From this specification, the terminal is able to identify 3 levels of complexity for each CIF node, and 4 levels of complexity for each QCIF node. Various definition of complexity can be found in literature. We defined complexity as the percentage of CPU needed for the rendering and decoding of a media associated to a node at given level of quality over a second. If $C_{ij}$ is the number of CPU cycles per second for the display of a node $i$ at quality level $j$ and $F$ the CPU clock frequency then the complexity of node $i$ at quality option $j$ is defined as:

$$Complexity_{ij} = (C_{ij} * 100) / F$$

From the above then we can calculate complexity levels for each combination of quality levels of a stream, as follows:

CIF (stream V1)

$$Complexity_{10} = (((25 * a) + b_1) * 100) / F$$

$$Complexity_{11} = (((8 * a) + b_2) * 100) / F$$

$$Complexity_{12} = (((2 * a) + b_3) * 100) / F$$

$a$ = cycles for CIF Frame Rendering

$b_1$ = cycles for CIF Decoding ASP @ 1 Mb @ 25 fps

$b_2$ = cycles for CIF Decoding ASP @ 1 Mb @ 8 fps (skip B frames)

$b_3$ = cycles for CIF Decoding ASP @ 1 Mb @ 2 fps (skip P and B frames)

The number of cycles for video decoding is a quantity variable in time over bounds that depends on incoming data and software implementation on a particular architecture. We first assigned an average decoding complexity and then updated the value during playback. We could have chosen worst-case cycles, but, as reported in [MB97], worst-case analysis is not adequate to define a useful decoding complexity. The same paper reports that variances of complexity figures for fixed bitrates do not show large variations. We hence started with an average complexity and updated the estimate during playback. While this approach is clearly not suitable for scheduling on real time systems, it can be used for the purpose of this work to validate the systems concepts we introduced. Better complexity estimation figures, if available, can be retrieved from information inserted at the encoder side [MB96].

Cycles for frame rendering are instead constant, assuming the same display depth. We implemented simple Decoding Control filters that use the frame skipping strategy, assuming a fixed Group of VoP (GOV) structure for the whole bitstream. We are aware this is sometimes a false assumption, since

some encoders can change GOV structure in order to better encode changes in the content. However, if this is the case, then the author should not signal frame variations as a suitable quality dimension and provide other quality dimension (size, visual quality) descriptors.

In a similar way, QCIF Media Texture nodes have complexity over 4 levels, as follows:

QCIF (Stream V3)

$Complexity_{30} = (((25 * c) + d_1) * 100) / F$

$Complexity_{31} = (((8 * c) + d_2) * 100) / F$

$Complexity_{32} = (((2 * c) + d_3) * 100) / F$

$Complexity_{33} = 0$

$c$ = cycles for QCIF Frame Rendering

$d_1$ = cycles for QCIF ASP Decoding @ 250 Kb/s @ 25 fps

$d_2$ = cycles for QCIF ASP Decoding @ 250 Kb/s @ 8 fps (skip B frames)

$d_3$ = cycles for QCIF ASP Decoding @ 250 Kb/s @ 2 fps (skip P and B frames)

Note that $c$ and $d_1, d_2, d_3$ can be derived from $a$ and $b_1, b_2, b_3$.

QoS Application level metrics verify the continuity of media streams using frame rate and max rate variation. In our set up we chose 0.2 as value for max rate variation. This means that frame rate for stream V1 for example can vary from 25 to 20 (25*(1-0.2)), and still is acceptable. This value can be changed including in the scene the appropriate QoS_Qualifier_MAX_VARIATION descriptor (for instance in order to have different tolerable variations according to the value of each stream).

### *3.4.1.1 Admission control, low end terminal*

We first ran the scene on a low level CPU (600 MHz Pentium III), using a player ignoring QoS descriptors. Figures 10 and 11 show respectively CPU usage and Stream V1 (CIF) V3, V7, V9 (QCIF) frame rates over one minute of simulation.

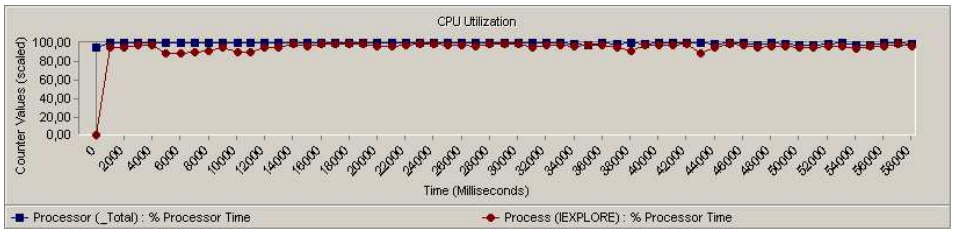


Figure 10: CPU load during one minute of simulation, QoS disabled

In Figure 10, the CPU utilization includes the percentage of processor time used by the MPEG-4 player (running as a plug-in of Internet Explorer), and the percentage of processor time used by the system. This is in order to distinguish load due to the player from load due to other processes.

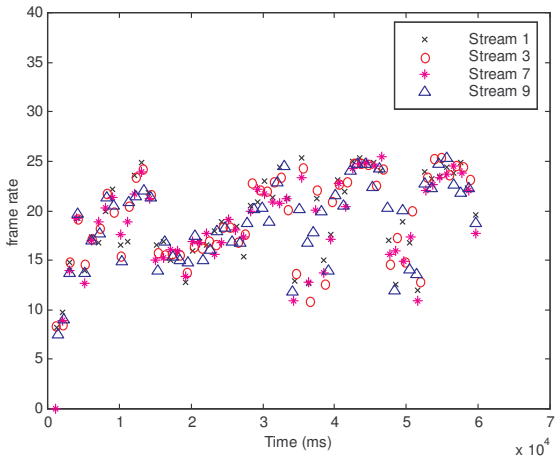


Figure 11: Frame rates of streams 1, 3, 7, 9 during one minute of simulation, QoS disabled

As we see from Figure 10, the CPU is at maximum utilization for all the duration of the simulation. In Figure 11 we observe that the system is indeed overloaded, since stream rates are not at their nominal value (25 frames per second), but instead they vary from 5 to 25. We observe that variations occur in the same way for all the streams in the presentation. So the presentation has a unique frame rate for all the streams. Figures 12 and 13 show the same sampling of CPU load and frame rates, in the case where QoS is enabled.

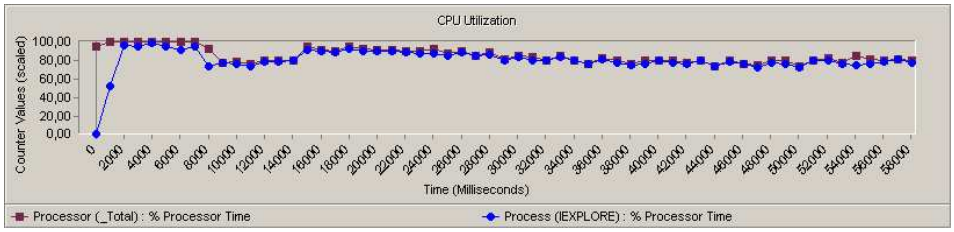


Figure 12: CPU load during one minute of simulation, QoS enabled

In Figure 12 we observe that the CPU load is at its maximum at start up, then it is always under 100% of utilization. Rates in Figure 13 are distributed according to the QoS parameter "Stream Value": stream 1 and 3 are distributed around their maximum quality (25 fps), stream 7 is around level 1 (8 fps), stream 9 is always at 0, that is it never runs. The QoS system performed admission control on the number of movie textures right at start up, and assigned qualities so that streams with higher value would run at maximum quality and the CPU would not be overloaded. The resource allocation algorithm described in 4.1.1 performs this. It greedily assigns resources starting from most important streams to less important ones, and from most important quality dimensions to less important ones (in this example, only one quality dimension is considered).

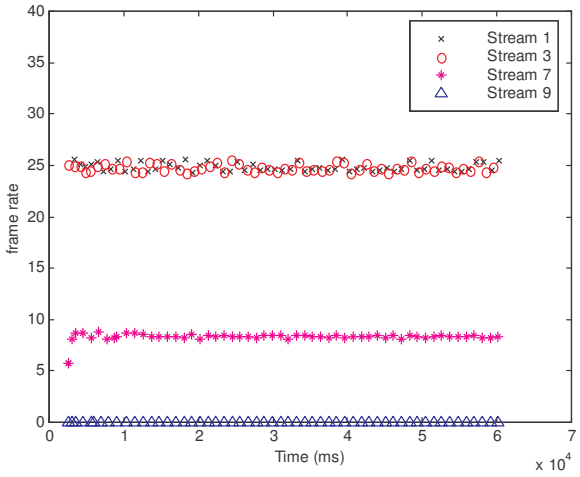


Figure 13: Frame rates of streams 1, 3, 7, 9
during one minute of simulation, QoS enabled

### *3.4.1.2 Competition among applications*

On a Pentium 1.3 GHz, the same example scene consumes from 50 to 60 per cent of the CPU (Figure 14). All the streams have frame rates of 25 fps for the whole duration of the simulation. In this configuration, we aimed at testing the behaviour of the QoS enabled terminal when other processes were running on the same terminal, inducing heavy load (in other words, we focused on changes in the environment of the terminal).

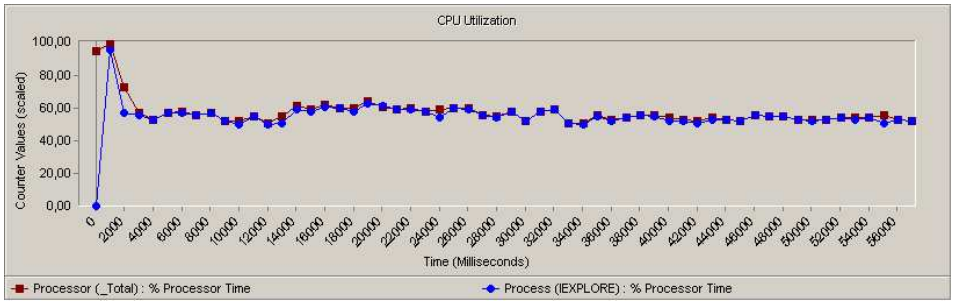


Figure 14: CPU utilization on Pentium 1.3
Ghz for Example 1

In order to produce CPU loads at different levels, we used a tool commonly used to simulate processor workload on Windows platforms, CPUStress.

Figure 15 shows CPU utilization when the CPU stress program is running, with three threads active, generating respectively busy medium and low activity. As we see from the picture, CPU utilization varies from 60 to 80 %.

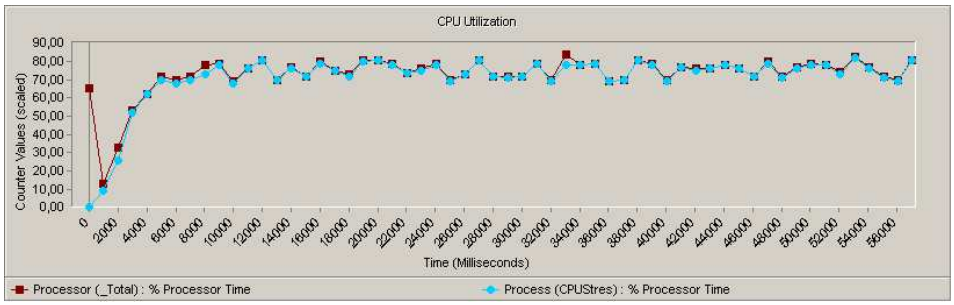


Figure 15: CPU Utilization for CPUStress program

From Figures 14 and 15 it is obvious that the player and CPUStress processes cannot run at full CPU utilization on the same system. Figure 16 shows the interaction between the CPU stress activity and the MPEG-4 player with no QoS handing. In performing the simulation, we took care in avoiding any difference in the priority of the two processes. This is because we are considering general-purpose operating systems with rate-monotonic programming model (i.e. Windows XP, Linux, Solaris) [RBS2000]. If one of the two processes had a major priority then the OS scheduler would have assigned more CPU time to the higher priority one. We also removed the foreground process advantage, that is the normal OS scheduler policy to assign higher priority and hence longer slices to the foreground process. We switched this feature off from the facilities offered by the OS, in order to better perform the simulation. In Figure 16, first the player is launched, then, after 12 seconds the CPUStress process is activated. As we see, the two applications compete for resources (overall processor activity is always at 100 per cent when both applications are running) and the OS scheduler tries to fairly share resources between the two applications.

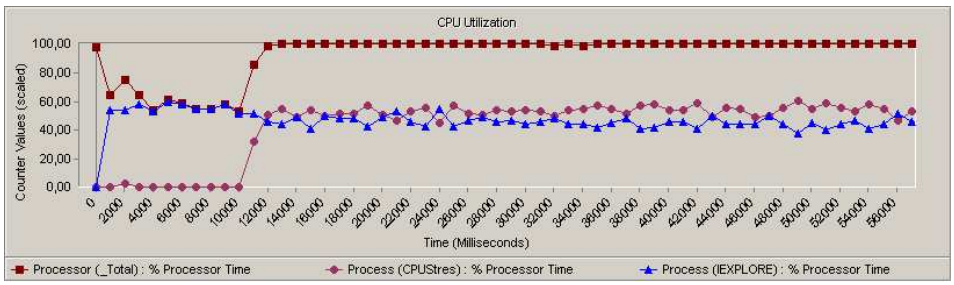


Figure 16: CPU utilization when MPEG-4 Player and CPUStress tool are active, QoS disabled

Figure 17 shows the interaction between the CPUStress tool and the MPEG –4 player with QoS enabled. Again first the player is launched, then the CPU stress tool is activated. As we see from the picture, the player is "adaptive", in the sense that it adapts its resource consumption to the environment situation. Frame rates for both simulations are reported in Figures 18 and 19. Frame rates are not at their nominal value in both simulations, but the QoS enabled player has selected a combination of rates so that the system is not overloaded (total CPU utilization is high, but not constantly at 100%). We observe that, when QoS is enabled, stream 1 and stream 3 show some variations in the frame rates. This is because the configuration of this example imposes a high stress to the system, and a high variation in the amount of computational resources available. Nevertheless, the variation in the frame rate occurs between the tolerable thresholds set by the QoS specification.

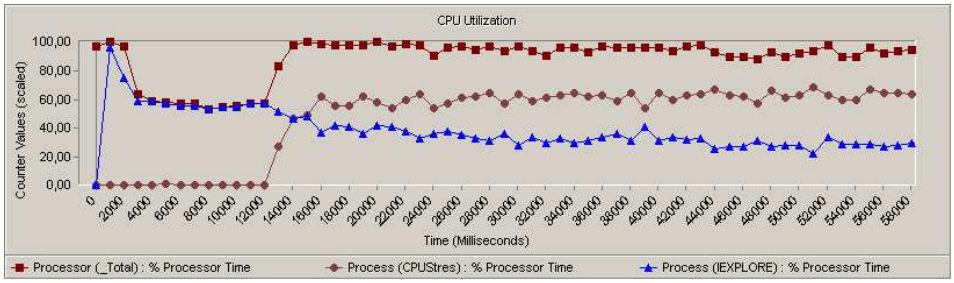


Figure 17: CPU utilization when MPEG-4 Player and CPUStress tool are active and QoS is enabled

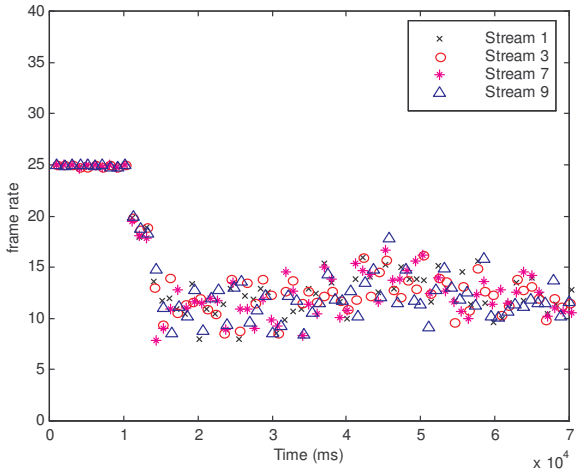


Figure 18: Frame rates when MPEG-4 Player and CPUStress tool are active and QoS is disabled

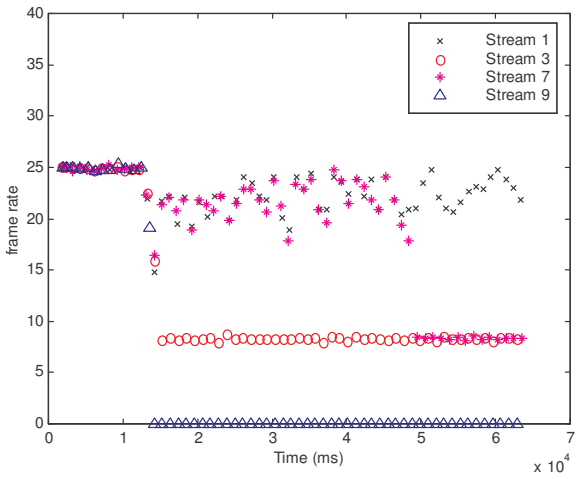


Figure 19: Frame rates when MPEG-4 Player and CPUStress tool are active and QoS is enabled

Using the same system stress tool, we observed V1 decoder lateness in the QoS enabled and disabled case. Figures 20 and 21 show respectively the sampling of decoder lateness when QoS is enabled and disabled. When the system is not under stress, the video decoder decodes frame in advance, up to the capacity of the composition buffer (set to 5 frames in our set up, i.e. 200

ms of video at 25 fps). Lateness is hence always negative when the system is not under stress. In the simulation of Figures 20 and 21, during a simulation of 40 seconds we created heavy stress after 10 seconds and we returned to normal load after 30 seconds. As we see from Figure 20 the QoS enabled player shows two peaks in the lateness when system stress is switched off and on and then stays always negative. On the other hand, the QoS disabled player shows an increasing delay (up to 7 seconds) from time 10 to time 30 and then slowly returns to normality. During system stress, the V1 stream in the QoS enabled player has frame rates between 20 and 25, while the V1 stream in the QoS disabled around 5 frames per second.

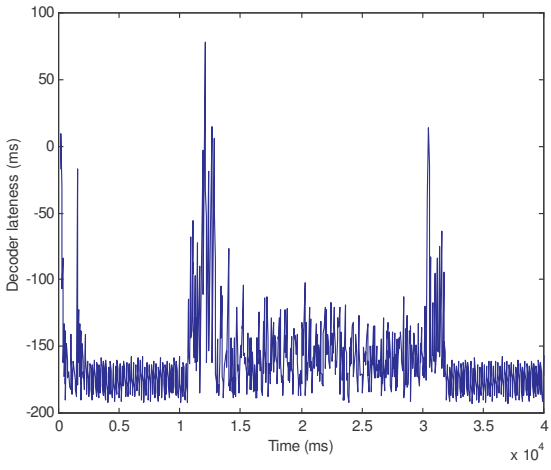


Figure 20: Decoder lateness stream V1, heavy stress, QoS enabled

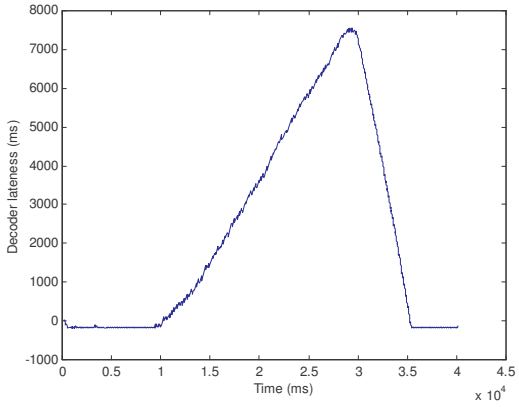


Figure 21: Decoder lateness stream V1, heavy stress, QoS disabled

#### *3.4.1.3 Power management*

An interesting scenario for the framework we propose is given by mobile computing. Modern mobile laptops can be configured to vary the CPU clock frequency when batteries are low or there is a switch in the power source (i.e. from AC adapter to battery). We configured an Intel Centrino laptop so that CPU clock speed would vary from 1.3 GHz to 600 MHz when switching from AC adapter to battery power source. Figures 22 and 23 show the CPU utilization during a run of 30 seconds. The player is started in AC power mode, then after about ten seconds the AC cable is disconnected. The figures show CPU utilization of the system, the MPEG-4 player, and clock frequency when QoS is enabled and disabled. As we see, when QoS is enabled the CPU utilization is high, but the system is not overloaded. The MPEG-4 player reallocates resources when it detects that the scene cannot run at full quality at a lower CPU frequency. Stream frame rates (Figures 24 and 25) are much different in the two cases. When QoS is enabled, as soon as clock speed is slow down, the player reallocates stream qualities accordingly. When QoS is disabled instead the frame rates of all the streams decrease uniformly.

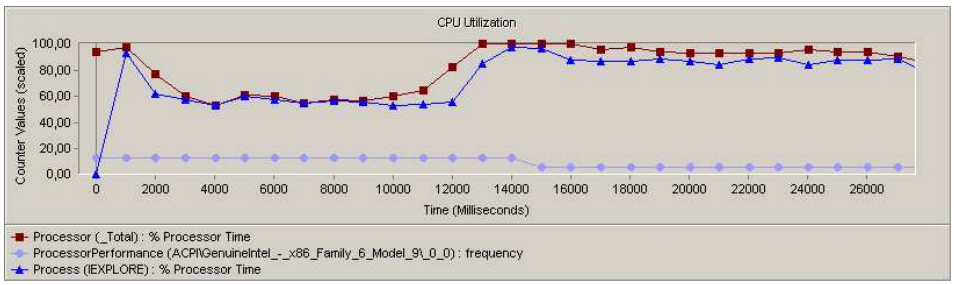


Figure 22: CPU utilization when clock frequency changes, QoS enabled

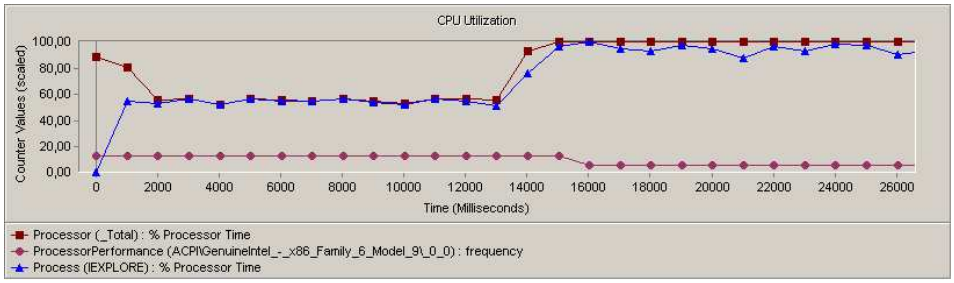


Figure 23: CPU utilization when clock frequency changes, QoS disabled

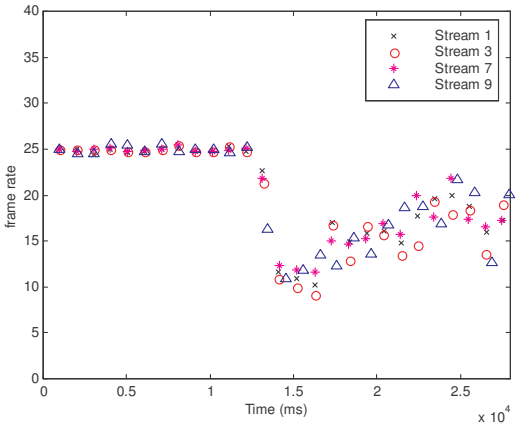


Figure 24: Frame rates when clock frequency changes, QoS disabled

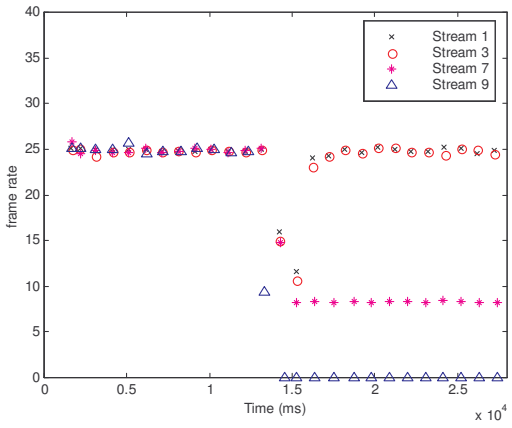


Figure 25: Frame rates when clock frequency changes, QoS enabled

### 3.4.2 Example 2

A more complex scenario of quality management is shown in Figure 26. We considered a BIFS scene with a CIF video scaled to 704x576, with height QCIF video superimposed on screen. The QoS specification is meant to assign more value to stream on the left of the display, and then streams on the right and finally the scaled CIF video. The quality dimensions of size and frame rate are included in the specification of each stream, as follows:

Streams QCIF (V1, V2, V3, V4)

QoS_Qualifier_FRAME_RATES{FRAME_RATES[25,8,2] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 3}

Streams QCIF (V5, V6, V7, V8)

QoS_Qualifier_FRAME_RATES{FRAME_RATES[25,8,2,0] dimensionValue 2}

QoS_Qualifier_FRAME_SIZE{FRAME_SIZE [100,50] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 2}

Stream CIF (V9)

QoS_Qualifier_FRAME_RATES{FRAME_RATES[25,8,2,0] dimensionValue 2}

QoS_Qualifier_FRAME_SIZE{FRAME_SIZE [100,50] dimensionValue 1 }

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 1}

Complexity levels for stream V9 are defined as follows:

CIF (stream V9)

$$Complexity_{90} = (((25 * a * s) + b_1) * 100) / F$$

$$Complexity_{91} = (((25 * a) + b_1) * 100) / F$$

$$Complexity_{92} = (((8 * a) + b_2) * 100) / F$$

$$Complexity_{93} = (((2 * a) + b_3) * 100) / F$$

$a$ = cycles for CIF Frame Rendering

$b_1$ = cycles for CIF Decoding ASP @ 1 Mb @ 25 fps

$b_2$ = cycles for CIF Decoding ASP @ 1 Mb @ 8 fps (skip B frames)

$b_3$ = cycles for CIF Decoding @ 1 Mb @ 2 fps (skip P and B frames)

s = factor of scale to take into account linear scaling complexity.

We hence build the complexity levels taking into account both Frame Size and Frame Rate Quality Dimensions, and we order the levels taking into account dimension values. In the case of stream V9, Frame Size dimensionValue is smaller than Frame Rate dimensionValue. This means that the author is willing to accept frame size reductions more "enthusiastically" than frame rate reductions. Levels of complexity are then built up in decreasing user satisfaction and stored in appropriate Movie Texture private data. As a result, the scene looks different whether it runs on a fast CPU or a slower one as shown in Figures 26 and 27.


Figure 26: Scene 2 running on fast CPU terminal


Figure 27: Scene 2 running on slow CPU terminal

## 3.5 Conclusions

In this chapter, we established a framework for quality-based multimedia presentations. The proposed terminal performs resource allocation and management, trading off user satisfaction and available system resources. User satisfaction is implied from a user-level quality specification, that identifies a list of quality options for each media quality dimension and sets up a ranking

among them. Monitoring of terminal performance is performed using proper application level metrics. The quality of a multimedia application is defined in terms of quantitative observations of its constituent media. Instead of suffering from lack of resources in an uncontrolled way, the proposed terminal regularly checks the performance of its streams against specified tolerances, and reacts to violations with precise actions selected trading performance and importance of each media. This is achieved adding monitoring capabilities to each media node, and providing an additional entity that centralizes the analysis of violations and initiates recovery actions. Results show how the proposed framework fulfils the requirements typical of applications running on general-purpose operating systems: admission control, competition among applications and power management. Next chapter will introduce a model to deal with 2D graphics and will show examples of the integration of this model in the proposed framework.

*Chapter 4*

# INTEGRATION OF 2D SYNTHETIC CONTENT

In this chapter we aim at introducing a model to estimate the resources for the 2D rendering tasks, and provide examples of how to use this model in the framework we proposed in Chapter 3. This work focuses on consumer hardware, namely a PC with a state of the art graphics card. It must be noticed that these are best-effort systems, and the graphic libraries provide no tools to give a hard time limit for the execution of a given set of rendering commands. What it can be achieved is a best-effort estimation of the resources needed, that can be used by a quality management algorithm. An important part of this chapter is constituted by an analysis of the rendering of 2D scenes described using the MPEG-4 BIFS standard, limited to the Core2D profile. Since the standard does not mandate algorithms to perform the rendering, part of this chapter proposes and discusses some techniques to perform the rendering. This has pertinence to the aims of this chapter, and also provides a contribution to the literature in the field of composition of MPEG-4 2D scenes. An assumption valid throughout this work is that the terminal uses a 2D rendering software graphic library. As a consequence, no hardware acceleration is used in the rendering of graphic primitives (but, if present, it can be used for bitmapped operations like copy transfers and scaling). Similar contributions, in the field of 3D based engines can be found in [FS93], [LB1998], and [WW2003]. These contributions are based on different assumptions because they imply that part of the rendering commands are executed by processing units located on hardware acceleration boards. As a consequence, their models cannot be used for systems based only on 2D rendering libraries. When we started this work, 2D rendering engines were commonly used for 2D tasks. Nowadays there is a shift versus

3D rendering engines, also for 2D scenes. However, for the kind of applications we target, basically videos with 2D synthetic graphics augmentations and 2D cartoons, 2D rendering libraries are still an interesting solution, independent from particular hardware board features.

## 4.1 Analysis

The methodology we followed is to subdivide the rendering process into a number of conceptually independent tasks that can be estimated separately. The subdivision we considered is the following:

- Scene graph traversal and construction of the display list ("Pre-render" in the following)
- Determination and Invalidation of the areas to draw ("Painter's Algorithm")
- Drawing of invalidated objects ("Rasterization")
- Copy of the drawn area to the screen ("Display update")

This subdivision is based on the assumption that the multimedia objects are stored in a graph data structure, and that a double buffering technique is used for the display. Both these assumptions are commonly true in most 2D rendering engines. The term "composition" is often used in MPEG-4 literature. In this subdivision, composition is comprised of the Painter's algorithm and Rasterization phases. In the following we consider each phase separately, deserving particular attention to the Painter's algorithm, since, as we will see, it plays an important role in the overall performance of the rendering task.

### 4.1.1 Scene graph traversal

MPEG-4 scenes are described using a tree graph technology similar to VRML and Open Inventor technologies [OP][VRML97]. A graph is constituted by nodes connected by directed links and a root node is used to define the start of the scene data. The scene graph is heterogeneous, since the nodes have different types. The main distinction is between interior and leaf nodes. Interior nodes structure the nodes in groups and define the visual layout of scene objects. Leaf nodes define the visual objects (basically geometry or bitmapped data) included in the scene. The target of the graph traversal is to build a list of visual objects that are contained in the scene, with their associated visual properties. Traversal begins at the root node. The structure of the scene graph suggests a depth-first method of traversal that has a asymptotic complexity of $O(n)$, where $n$ is the number of nodes of the tree. Object layout requires some geometric transformations involving seven 3x3 matrix multiplications for each Transform2D node [VRML97]. The asymptotic complexity will then be $O(t)$ where $t$ is the number of Transform2D nodes in the scene. For each shape node, we additionally perform a transformation of each vertex and we determine the smallest enclosing rectangle ("bounding box"). This step has hence complexity $O(v)$ where $v$ is the number of vertices of an object. From these observations a good estimation of the traversal time is given by:

$$T_{traversal}(n,s,t,v) = c_1 * n + c_2 * t + c_3 * (s * v)$$

where $n$ is the number of nodes of the scene graph, $s$ is the number of shapes, $t$ the number of Transform2D and $v$ the average number of vertices for each object. The parameters $c_1, c_2, c_3$ are constants that are determined experimentally.

Experimental results confirmed our analysis. We first built three scenes with a high number of nodes, but with no shapes. In this way, we gathered some

figures about the traversal and the number of nodes of a graph. The results are reported in Table 2. It shows an extract with the first 500 ms of a simulation that lasted 3 minutes.

| Time of simulation | Pre-Render 2000 Obj (ms) | Time of simulation | Pre-Render 8191 Obj (ms) | Time of simulation | Pre-Render 32767 Obj (ms) |
|---|---|---|---|---|---|
| 113 | 2 | 168 | 8 | 447 | 34 |
| 116 | 2 | 178 | 8 | 481 | 33 |
| 118 | 2 | 186 | 8 | 515 | 34 |
| 127 | 1 | 195 | 9 | 549 | 34 |
| 160 | 1 | 217 | 8 | 583 | 34 |
| 193 | 1 | 250 | 8 | 617 | 34 |
| 226 | 1 | 283 | 8 | 650 | 33 |
| 259 | 1 | 316 | 8 | 684 | 34 |
| 292 | 1 | 349 | 8 | 718 | 34 |
| 325 | 1 | 382 | 8 | 753 | 35 |
| 358 | 1 | 416 | 8 | 787 | 34 |
| 391 | 1 | 448 | 8 | 821 | 33 |
| 425 | 1 | 481 | 8 | 855 | 34 |
| 457 | 1 | 514 | 8 | 888 | 33 |
| 490 | 1 | 547 | 8 | 922 | 33 |

Table 2: timing of the tree traversal (15 samples)

The table reports the performance of tree traversal for three different tree sizes: 2000, 8191, 32767. We chose 2000 objects as the average tree size of a complex scene. 8191 is instead the maximum number of nodes allowed in Scene Graph Core2D@L1 profile, and 32767 is the maximum number of nodes in Scene Graph Core2D@L2 and Scene Graph Advanced2D@L1 profiles. As we can see from the table, the tree traversal has O (n) computational requirements, where "n" is the number of nodes. On the machine used for the experiment (a Pentium III 750 MHz CPU PC), the traversal time is almost negligible up to 2000 objects, but is quite high instead for 32767 objects. We noticed similar results on every MPEG-4 player publicly available as product or reference implementation. We see that the

traversal of a scene graph with 32767 nodes takes more than 30 ms. This means that for instance it is not possible to render the scene graph at 25 frames per second, since at that rate we have only 40 milliseconds to traverse and render. In this case, a terminal implementation can detect before the start of the presentation that the scene graph is too big to be rendered at full frame rate, and run a quality degradation algorithm right at the beginning. The following table reports tree size, pre-render time and display list size for a scene containing nearly 1000 shape objects. This scene has been built using an automatic tool, that translates a collection of animation frames (drawn in a concept similar to cartoons) in a BIFS scene. Cartoons will be extensively studied in Chapter 5. BIFS contains simple geometric nodes like rectangle, circle and more complex primitives like IndexedLineSet2D, IndexedFaceSet2D and Curve2D that can be used to build complex shapes.

| Time | Pre-Render (ms) | Tree size | Display list size |
|---|---|---|---|
| 1277 | 17 | 1971 | 973 |
| 1540 | 17 | 1963 | 969 |
| 1613 | 17 | 1963 | 969 |
| 1632 | 17 | 1963 | 969 |
| 1653 | 17 | 1963 | 969 |
| 1822 | 18 | 1921 | 948 |
| 1898 | 21 | 1921 | 948 |
| 1964 | 16 | 1921 | 948 |
| 1985 | 19 | 1921 | 948 |
| 2169 | 17 | 1842 | 906 |
| 2241 | 16 | 1842 | 906 |
| 2259 | 16 | 1842 | 906 |
| 2276 | 16 | 1842 | 906 |
| 2299 | 16 | 1842 | 906 |
| 2473 | 13 | 1384 | 679 |
| 2528 | 12 | 1384 | 679 |

Table 3: Pre-Render time for a scene constituted of 1000 shapes

Comparing Table 2 and Table 3, we notice that when tree sizes are comparable (2000 from Table 2 and 1970 from Table 3), pre-render times are much higher when the tree contains shapes. This is because during traversal of shape nodes (rectangle, circle, indexedlineset2D, indexedfaceset2D, and curve2D) we perform some additional operations (matrix multiplications and construction of smallest enclosing rectangle) as previously mentioned. We experimentally noticed that the size of the scene tree alone is not sufficient to capture the pre-render execution times. On the other hand, if the size of the tree is below a threshold C (2000 in our study); the pre-render time depends mainly on the number of shapes in a scene.

### 4.1.2 Painter's algorithms

In order to draw the objects of the display list ("drawables"), it is necessary to scan the display list and identify which parts of the drawables need to be redrawn, with the target of minimizing the number of pixels drawn. The commonly called "Painter's algorithm" performs this task. In Chapter 2 we reported the Painter's algorithm implemented in the MPEG-4 reference software (Section 2.6.3). A preliminary observation is that the algorithm does not perform well when the number of objects to draw ("drawables") increases. This is because it keeps track of the area to be redrawn using the union of the bounding boxes of the drawables. When objects that do not overlap and are distant from each other need to be redrawn, the area equal to the union of the two areas is invalidated and so marked for redrawing. All the objects intersecting that area are then redrawn, even if it would not be needed. For instance, suppose that at a given tick of simulation objects A and B in the following figure need to be updated.

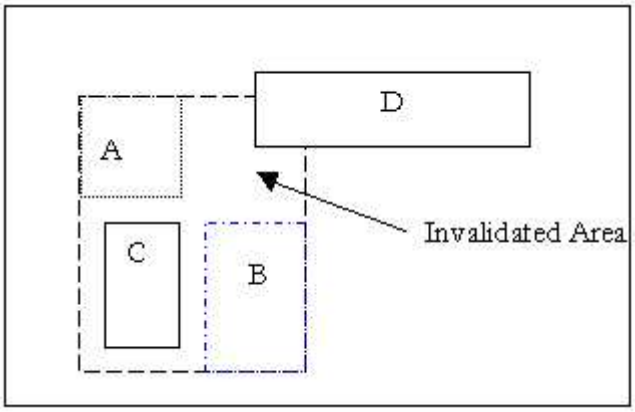


Figure 28: Invalidated area as union of bounding boxes in the reference software algorithm

The reference software algorithm considers the union of the bounding boxes of objects A and B, and invalidates as a consequence the area "Invalidated". Object C and part of object D are then redrawn in addition to objects A and B.

In this section we propose two variations of the painter's algorithm and then we compare the performances of the reference software algorithm with these two algorithms. The algorithms have been introduced to overcome the problems told above. They all follow a similar pattern quite common in videogame engines: at each frame, localize the areas of the screen that have to be updated, clear them if necessary, and draw all the objects that intersect those areas. To do so, first the display list is scanned and two data structures are built: the "NeedsDrawing", defining the area of the screen that needs to be updated because of changes in the appearance or in the shape of nodes, and the "Transparent", containing the area of the screen that must be cleared before starting to draw the new frame, i.e. the area of the screen where objects have been deleted, have changed position or are transparent and something behind them have changed. Then the Transparent area is deleted and objects in the display list are drawn if they need to be updated or they

intersect the Transparent area. This is the process that every algorithm proposed here performs. However, the algorithms are different in the data structures they use for the NeedsDrawing and Transparent areas: the first one ("Reference Software") stores the smallest rectangle that contains the union of all the areas, the second one (""Dirty Rects") uses a list of rectangles and the third one ("Sector") uses the concept of a grid superimposed on the screen, and stores in the two data structures the index of grid sectors that become invalidated. As we will see, the data structures have an impact on the final performance of the algorithm, in particular on the number of objects drawn at each frame display. The following sections illustrate each algorithm and compare their performances.

#### *4.1.2.1 MPEG-4 Systems reference software (RS)*

We use two rectangles to keep track of the changed areas: the "NeedsDrawingRect" and the "TransparentRect", both initially empty. While scanning the display list, if a drawable needs drawing or is transparent, we add the bounding box of the object to the appropriate rectangle (as described above), performing the union between the bounding box and the NeedsDrawingRect/TransparentRect. The data structures needed are just two bounding boxes. Inserting a bounding box means performing a union, and checking for overlapping implies performing the comparison between two bounding boxes. The algorithm is outlined in the following figure. The algorithm has low complexity (using asymptotic analysis, O(n) where n is the size of the display list), but, as we will see afterwards, storing the union of all the areas to be changed leads to high number of objects to be redrawn.

- For each drawable in the display list, if it needs to be drawn, add its bounding box to the NeedsDrawingRect (initially empty), performing a union between the two sets. If the drawable is also transparent, add its bounding box to the TransparentRect (initially empty), performing a union between the two sets.
- Add deleted nodes areas to TransparentRect, performing a union between the two sets.
- Clean TransparentRect.
- For each Drawable, draw it if:
    - It needs drawing (draw all the object)
    - It intersects the TransparentRect (re-draw the intersection)
    - It intersects the drawn area (re-draw the intersection)
- Add drawn area to the DrawnRect, performing a union between the two sets.

Figure 29: Reference Software algorithm

#### *4.1.2.2 Dirty Rects (DR)*

The idea of the algorithm is to achieve better precision in the selection of the areas that need to be drawn, using a list of rectangles to store the areas. The algorithm uses two lists of rectangles to keep track of the changed areas: the ”NeedsDrawingDirtyList” and the “TransparentDirtyList”, both initially empty. While scanning the display list, we add the bounding box of the objects to the appropriate list, adding their bounding boxes to the NeedsDrawingDirtyList and/or TransparentDirtyList. Particular care has to be taken when inserting a new rectangle in the list, since the rectangle may overlap another rectangle already present in the list. If this is the case and we

store the rectangle we will end up cleaning several times the same area of the screen, which is not optimal. We implemented a trade-off mechanism that, in case of overlapping:

- It does not insert an object if it is contained completely in another object already in the list.

- It stores the union between the two overlapping rectangles if the area of the union is smaller than the sum of the two areas; it stores the two rectangles otherwise.

Compared to the previous algorithm we achieve a better precision in the area to redraw, since we collect all the rectangles that need to be updated. The drawback is that when the scene contains several objects, the display list may grow considerably, and the algorithm can be costly (in fact it has a worst case of O(n^2) where n is the size of the display list). The algorithm is outlined in the following figure.

- For each drawable in the display list, if it needs to be drawn, add its bounding box to the NeedsDrawingDirtyList, taking into account possible overlaps. If the drawable is also transparent, add its bounding box to the TransparentDirtyList, taking into account possible overlaps.
- Add dirty areas to TransparentDirtyList, taking into account possible overlaps.
- Clean every bounding box in the TransparentDirtyList.
- For each Drawable, draw it if:
    - It needs drawing (we draw all the object).
    - It intersects a bounding box contained in the TransparentDirtyList (re-draw the intersection).
    - It intersects a bounding box contained in the DrawnRectDirtyList (re-draw the intersection).
- Add drawn area to the DrawnRectDirtyList, taking into account possible overlaps.

Figure 30: Dirty Rects algorithm

#### *4.1.2.3 Sector (SR)*

We divide the screen in a grid of rectangles ("sectors"). During the pre-render step, for each drawable we store in a list the sectors it is included in. During the composition step, if an object in the display list needs drawing or is transparent, we invalidate all the sectors it is included in, building as usual two lists (of sectors in this case), the NeedsDrawingSector and TransparentSector. We then clear all the invalidated sectors that are in the transparent list and draw all the objects contained in the invalidated sectors.

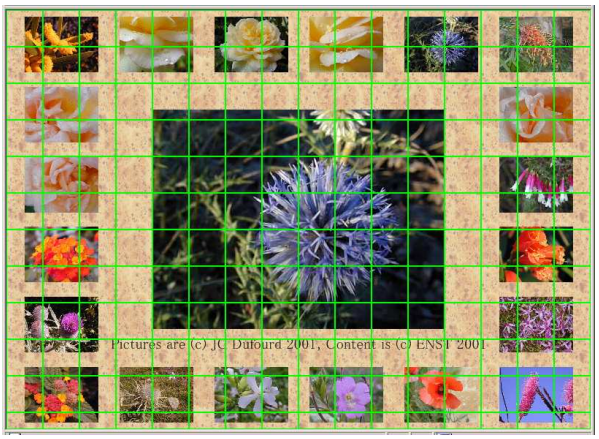


Figure 31: The Sector algorithm uses a grid to identify the area to redraw

The data structures needed are two lists of invalidated sectors (size equal to the number of sectors), a list containing the sectors bounding boxes (this list is created whenever the player window is resized), and a list of sectors for each drawable. The algorithm is outlined in the following figure.

- For each drawable in the display list, if it needs to be drawn, invalidate all the sectors it overlaps in the NeedsDrawingSector list. If the drawable is also transparent, invalidate all the sectors it overlaps in the TransparentSector list.
- For each deleted node, invalidate all the sectors it overlaps in the TransparentSector list.
- Clean every sector in the TransparentSector list.
- For each Drawable, draw it if:
    - It needs drawing (draw all the object)
    - It intersects an invalidated sector contained in the TransparentSector list (re-draw the intersection)
    - It intersects an invalidated sector contained in the DrawnSector list (re-draw the intersection)
- For each drawn rectangle, invalidate all the sectors it overlaps in the DrawnSector list

Figure 32: Sector Algorithm

The algorithm has been conceived to be an intermediate solution between the first two. It is not as good as the DR in determining the areas to redraw, but it does consider separate areas to redraw. Besides, it has complexity of O(n*k), where n is the size of the display list and k is the number of sectors. This becomes O(n) when the number of objects on the screen increases and the number of sectors k becomes negligible. Like the RS algorithm, the memory requirements for the data structures are known in advance, since we know the number of sectors we use. Clearly, the number of sectors that constitute the grid determines the performance of the algorithm. As the number of sectors

increases, the precision in determining the areas to draw increases as well, but the performance worsens.

#### *4.1.2.4 Evaluation of Painter's algorithms*

In order to compare the proposed painter's algorithms, we use the number of pixels that the algorithm invalidated as to be cleaned and drawn during the playback of a bitstream. The following table reports these measures for a number of bitstreams among the ones we performed the evaluation tests. The table contains streams taken from the systems conformance and others provided by IBM, ENST and Optibase. The first column reports the name of the bitstream, the second the algorithm used (RS = reference software, SR = SECTOR, DR = DIRTY RECTS), the third and the fourth columns the number of pixels cleaned and drawn respectively. Figure 33 provides, as a different view, a histogram that compares the algorithms. The table and the figure show that the DR algorithm produces the minimum number of cleaned and drawn pixels. SR is sometimes quite better than RS, but sometimes has results comparable to RS. This comes from the fact that we chose a fixed size for the sectors (16*16 blocks), and we tested bitstreams with different screen dimensions. Besides, conformance bitstreams very often contains a limited number of objects, because they are meant to test single BIFS functionalities. SR algorithm instead is designed to produce good results when the screen contains a large number of objects compared to the grid size.

| Bitstream | Algorithm | Pixels Cleaned | Pixels Drawn |
|---|---|---|---|
| IBMCoordinateInterpolator2Ds | RS | 15495534 | 23716707 |
| | SR | 9732608 | 18337635 |
| | DR | 8311027 | 16666031 |
| IBMCircleFading | RS | 11503887 | 22554867 |
| | SR | 4940032 | 16777173 |
| | DR | 4236795 | 15361107 |
| IBMCircleScaling | RS | 4871987 | 4791120 |
| | SR | 5030400 | 5478494 |
| | DR | 4462128 | 4789520 |
| IBMCurve2DMorph | RS | 7606800 | 15393600 |
| | SR | 7681200 | 15472200 |
| | DR | 7586400 | 15352800 |
| IBMCurve2DScaling | RS | 9092436 | 18508080 |
| | SR | 9204192 | 18637920 |
| | DR | 8937666 | 18198540 |
| IBMMpegLogo | RS | 121200 | 270600 |
| | SR | 159744 | 374104 |
| | DR | 121200 | 270600 |
| IBMTIles | RS | 9160000 | 16520000 |
| | SR | 9180160 | 16570160 |
| | DR | 7170000 | 14530000 |
| ENSTKangaroo | RS | 5263718 | 11259823 |
| | SR | 4650688 | 11706170 |
| | DR | 4087877 | 9771259 |
| ENSTCompfxMerged | RS | 1457280 | 3491547 |
| | SR | 1401888 | 3464010 |
| | DR | 1331775 | 3366042 |
| ENSTWipeIn | RS | 5765134 | 9127103 |
| | SR | 6111808 | 9277403 |
| | DR | 5741421 | 9103399 |
| ENSTTicker | RS | 1276416 | 4893096 |
| | SR | 2551296 | 4232664 |
| | DR | 1273224 | 2960832 |
| OptibaseMosaic31 | RS | 28554512 | 43597427 |
| | SR | 16018432 | 43442206 |
| | DR | 12571152 | 42119686 |
| TDKKaraoke | RS | 7019067 | 8802459 |
| | SR | 7231488 | 8785635 |
| | DR | 6838998 | 8491671 |

Table 4: Efficiency of Painter's Algorithms

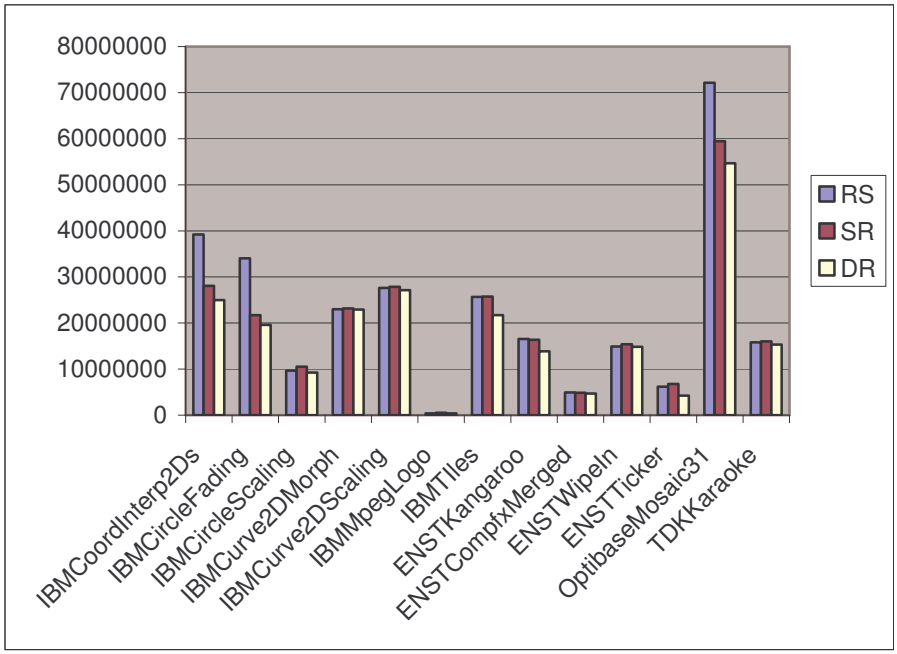


Figure 33: Sum of cleaned and drawn pixels
for some conformance bitstreams

The following figure shows the number of pixel drawn plus the number of pixels cleaned as a function of time for the three algorithms, for the "EnstTicker" bitstream. Visually, the bitstream is comprised of a static background, a video and a text scrolling under the video.

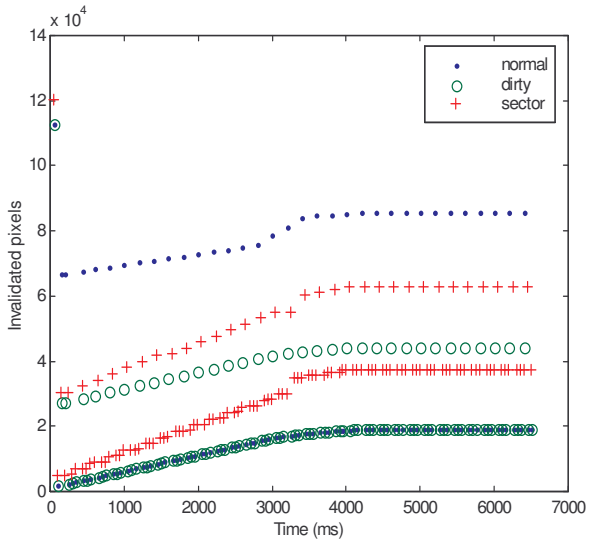


Figure 34: Number of invalidated pixels as a function of the time for the three proposed algorithms

We observe that the number of invalidated pixels is distributed around two different lines for each algorithm. The line with the highest number of pixels drawn corresponds to when both video and text have to be updated. The line with the lowest number of pixels drawn corresponds to when only text has to be updated. We notice that the DR produces always the minimal number of pixels invalidated. The SR higher line is lower than the RS higher line, but the SR lower line is higher than the RS lower line. The small number of objects penalizes the efficiency of the SR algorithm.

#### *4.1.2.5 Performance evaluation of composition*

The previous section contains a "static" evaluation of the three painter's algorithms, in terms of number of invalidated pixels during a simulation. In order to understand better the differences in the algorithms and the consequences that the adoption of one brings, we compared the execution performances timing the painter's algorithms time plus the drawing times (here after we call it "composition time"). This is because the immediate effect of a "bad" painter's algorithm is the excessive redrawn of display areas

that in turn produce a performance hit. As test case for this discussion, we utilize IBMConfetti.mp4, a conformance bitstream. It adds 5 shapes (rectangles and circles) every 100 ms, up to 5000 shapes. Then it displays a text message and then starts deleting the objects, 5 shapes every 100 ms as well. Figure 35 provides a snapshot of the screen after some seconds from the start.

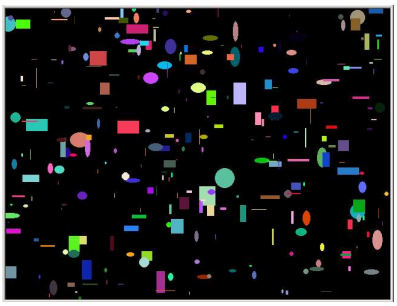

Figure 35: IBMConfetti.mp4 screen shot

This bitstream is an interesting test case because it requires the regular update of different areas of the screen, and requires the display of a high number of objects. Geometric objects are added and deleted at opposite sides of the display area. The following pictures (Figures 36, 37, 38) report the performances of the composition algorithms. They show the composition time in milliseconds as a function of the time of the scene. Looking at the graph of the RS algorithm (Figure 36), the composition time shows a big peak as soon as the player starts deleting objects and the text message has to be deleted. All the algorithms have a similar peak, because a high number of objects have to be deleted and redrawn. However, the RS algorithm continues having high composition times when deleting each group of five objects. This is because of the way we build the NeedsDrawing and Transparent rects in the RS algorithm. In fact we calculate the union of the bounding boxes of the five shapes to be deleted and we clear that area. If the objects are very distant among them, then the union contains a large number of objects. Cleaning that area means redrawing all the objects that overlap the area. In this particular

bitstream, after 500 ms that it started deleting, we removed 25 objects, and redrawn 15970, with an obvious peak in the composition time.

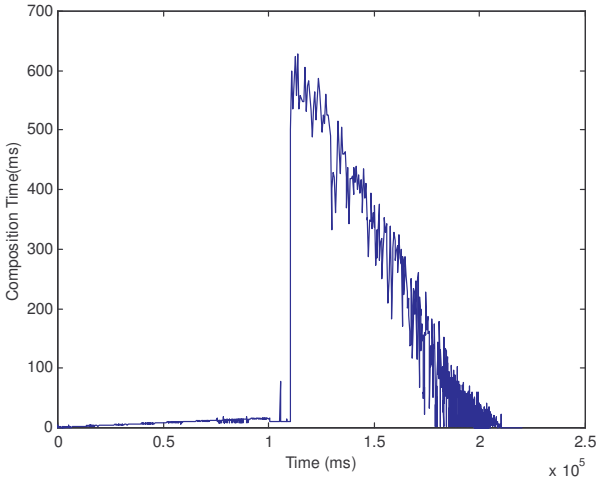


Figure 36: Performance of RS Algorithm

The SR (Figure 37) and DR (Figure 38) algorithms perform a lot better. We have a single peak around time 10000 (as before, when the player deletes the text message) and then the composition time decreases quickly (instead of slowly descending as in the RS case). The Dirty Rects algorithm shows the best performance.

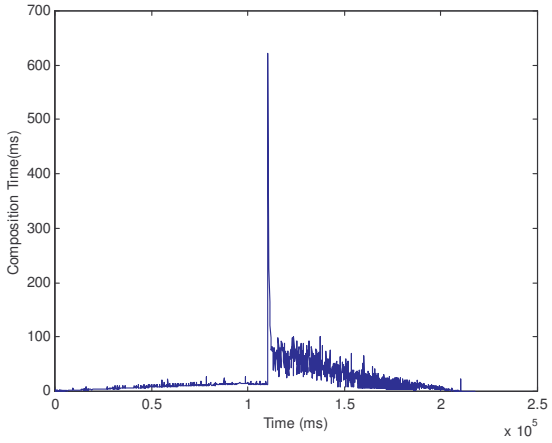


Figure 37: Performance of SR Algorithm

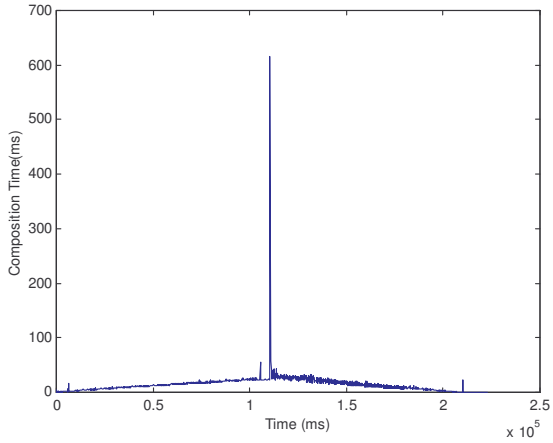


Figure 38: Performance of the DR algorithm

In order to better understand the differences between the SR and DR algorithm, we further analysed the composition time as a function of the time, restricting the y-axis (composition time) to the interval [0,100] (Figures 39 and 40). From the pictures we see that even if the DR algorithm performs better when deleting objects, the SR algorithm performs better when the number of object on screen is around 9000 and we keep adding new objects. The DR algorithm achieves the best finding of the areas to clean, because it stores the list of single drawable objects to clean. The SR algorithm on the other hand has a fixed number of areas to clean, and this explains its inferiority when cleaning areas. The DR algorithm suffers from the potential growth of its rectangles list (asymptotic analysis produced O(n^2) for the DR and O(n) for the SR). When the number of rectangles contained in the dirty rect list increases, the performance of the algorithm decreases because of the multitude of intersection tests that has to be done. The SR algorithm has a fixed number of sectors, and this explains why it is slightly better when adding objects to the screen in the case there are already lot of objects displayed.

We conclude saying that the DR algorithm is a good option for the composition of 2D scenes, provided that when the number of objects increases some load control technique can be used. In the following section

we introduce a load control method for composition algorithms based on the DR technique.

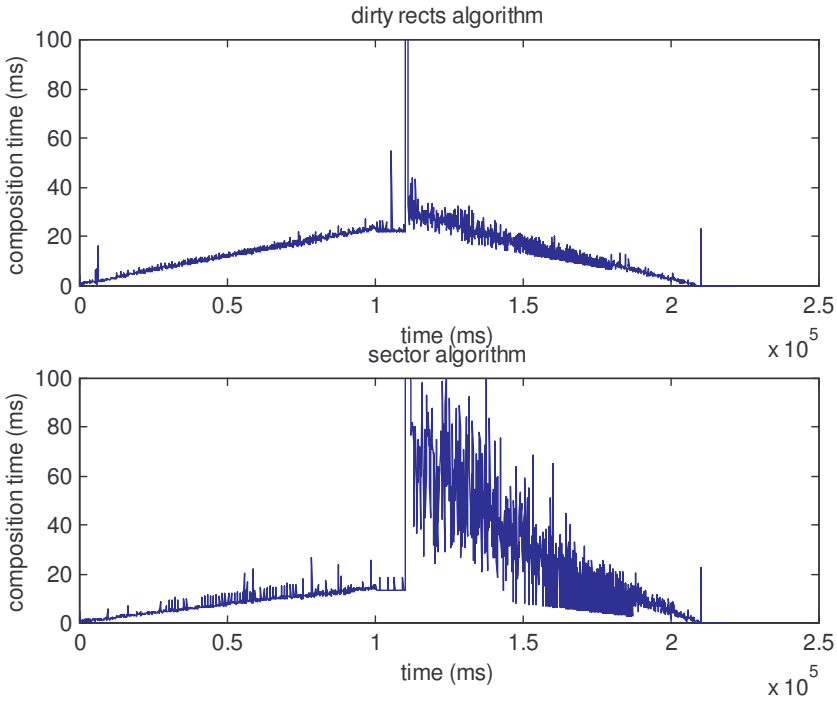


Figure 39: Comparison between DR and SR composition algorithms

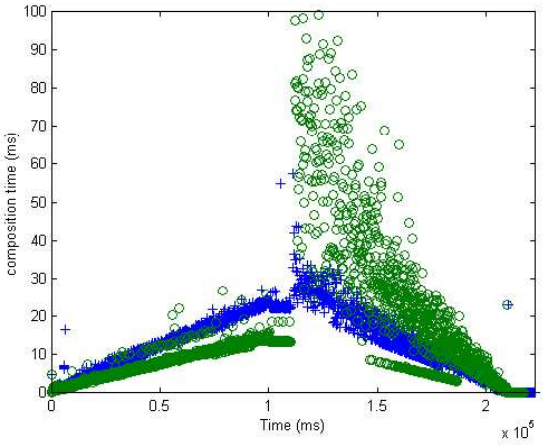


Figure 40: Comparison between DR and SR algorithms

#### *4.1.2.6 Load control*

In Section 4.1.1 we observed that when the number of shapes is high, scene graph traversal and construction of the display list become time consuming tasks that can preclude the rendering of a scene at a defined rate. This observation suggests as interesting load control option to skip the construction of the display list phase and substitute the painter's algorithm with an algorithm that simply performs the refresh of the entire screen ("Direct" algorithm). The following figure shows the comparison between the Dirty Rects and the Direct algorithms for a cartoon scene that contains an average of 1000 objects (ENSTCompfx). We plot the "Drawing Time" as a function of the time of the scene. "Drawing Time" is the sum of Pre-render and Dirty Rects painter's algorithm and Draw execution times in one case (DR) and Scene Graph traversal and Draw in the other case (Direct).

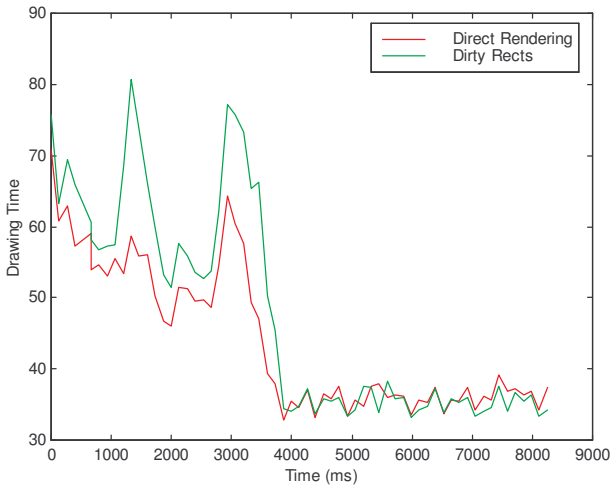


Figure 41: Comparison of Dirty Rects and Direct algorithms for ENSTCompfx bitstream

As we see from the picture, Direct rendering performs indeed better. We observed a similar behaviour for another cartoon, ENSTVectoFx, as reported in Figure 42.

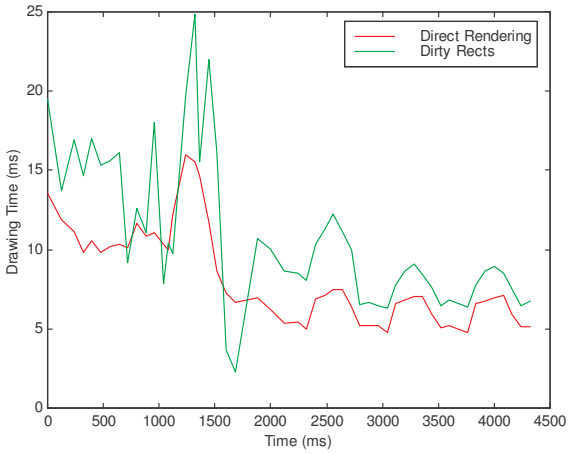


Figure 42: Comparison of DR and Direct algorithms for ENSTVectofx

On the other hand, the use of Direct rendering for IBMConfetti would instead produce a disastrous effect, as shown in Figure 43.

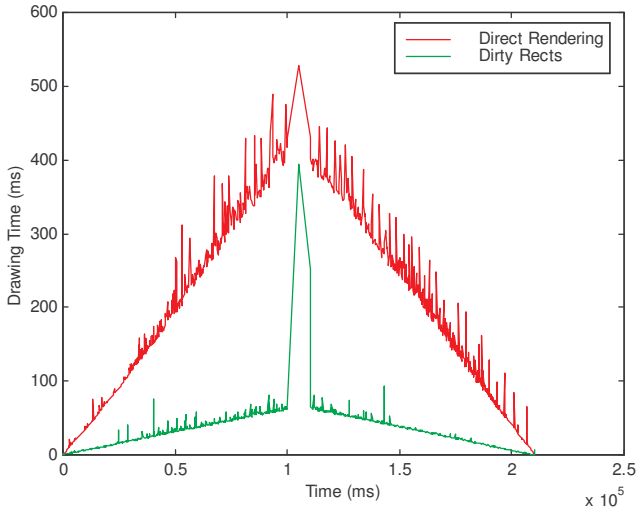


Figure 43: Comparison between Direct and Dirty composition of IbmConfetti

As we saw from the previous three examples, in order to decide which composition algorithm to use, the number of objects in the display list is not

enough, since for instance IBMConfetti displays up to 10000 objects compared to the average 1000 of ENSTCompfx, yet IBMConfetti is not suitable for direct rendering. What is most important instead is the number of new objects that are introduced or modified, at each frame display. The first two cartoons have a high number (on the order of magnitude 10^2) of objects to be updated, while IBMConfetti just 5 every 100 ms. The number of objects in the display list is superior (10000 versus 1000), but this does not penalize the performance of the DirtyRects algorithm. Ideally, we would like to find a way to switch between the two algorithms dynamically while rendering a scene. The idea is to keep track of the changes occurred to the tree, then decide which algorithm to use on the basis of the number of shapes modified or added. To achieve this, we count the modifications of the shapes and transform nodes of the scene graph at each tick of the simulation. We start composition using the Dirty Rects algorithm, then, at each rendering iteration, if the number of objects modified is bigger than a threshold T we use direct rendering. The following pictures show the result of this hybrid direct/dirty rendering technique, using a threshold T equal to 200 (that means that if more than 200 shapes are introduced or modified at each iteration we use Direct Rendering). We observe that hybrid rendering almost constantly produces lower drawing times.

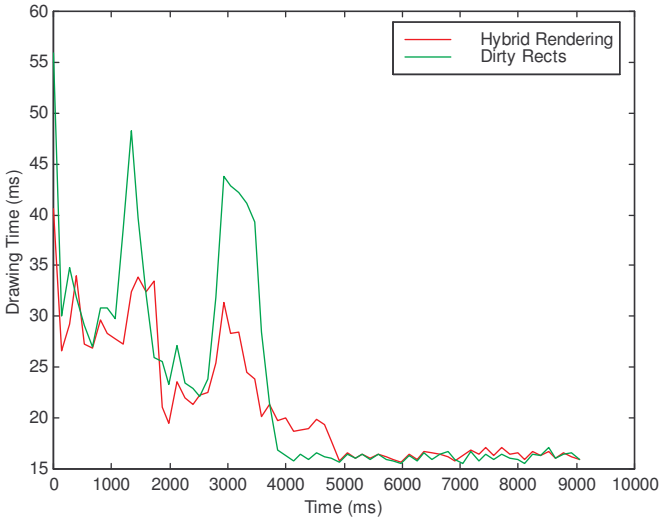


Figure 44: Comparison of Hybrid and Dirty composition for ENSTCompfx

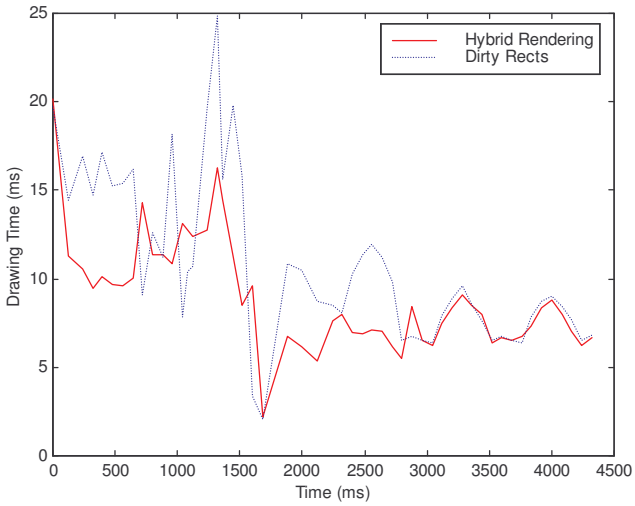


Figure 45: Comparison of Hybrid and Dirty composition for ENSTVectofx

To conclude, the Dirty Rects algorithm showed better performance compared to the other algorithms. Its complexity depends on the size of the display list. The proposed load control technique identifies Direct rendering as an

interesting alternative when, during the playback of a scene, the size of the display list grows and many modifications of the scene graph are detected.

### 4.1.3 Rasterization

Rasterization specifies which pixels on a display device each object in a scene covers. 2D scenes are basically constituted of bitmapped images and 2D geometry primitives (point, line, rectangle, circle, polygon, etc.). Rasterization of bitmapped images implies transforming the bitmaps in a format compatible with the screen resolution. Rasterization of geometric primitives specifies which pixels on a raster display are covered by the geometry. The basic algorithm is the Bresenham's algorithm for lines drawing [Bre77]. The performance of rasterization depends on both hardware (video board, CPU) and software (video board driver, rendering library) issues. If we consider a particular software and hardware configuration, we can determine execution time for the rasterization of graphic primitives using a linear regression model as a function of the distance between two pixels (line primitive) or multiple regression as a function of vertices and pixels covered by an object (rectangle, polygon). These models can be identified for a particular rendering library repeating several rasterizations at different object sizes and then running a regression on the results. Once the models are identified, we can calculate an estimation of the rasterization time for each frame simply summing the estimates for each object in the scene that needs to be drawn. We observe that the rasterization time for a frame depends on the objects we actually draw and it is bounded by the sum of the rasterization times of each object (since at each frame we draw a part or at maximum all the objects):

$$T_{Rasterization}(F) = T_{Rasterization}(o_1, o_2, ..o_n) \leq T_{Rasterization}(o_1) + T_{Rasterization}(o_2) + ...T_{Rasterization}(o_n)$$

where $o_1, o_2, ...o_n$ are the objects included in a frame $F$ .

Figures 46, 47, 48 show measured and predicted rasterization times for three BIFS scenes composed mainly of lines and polygon primitives.

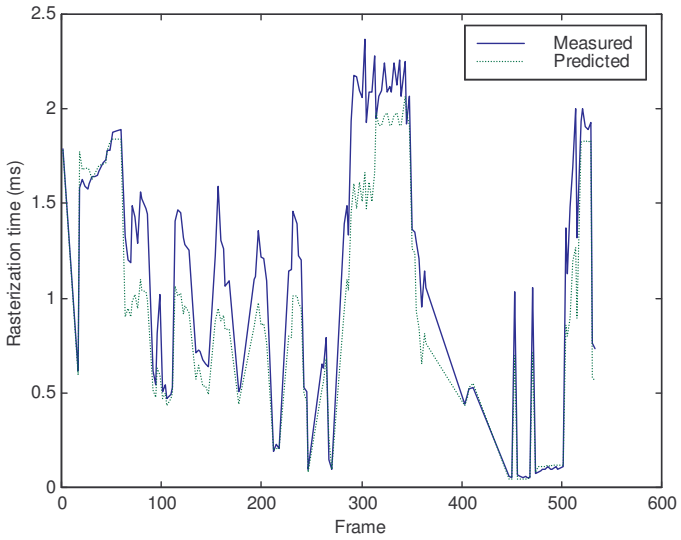


Figure 46: Measured and predicted rasterization times for ENSTKarate.mp4

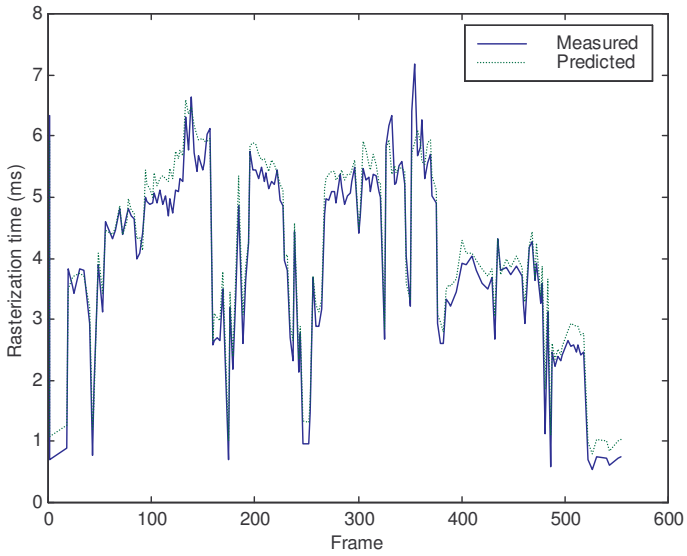


Figure 47: Measured and predicted rasterization times for ENSTAg001.mp4

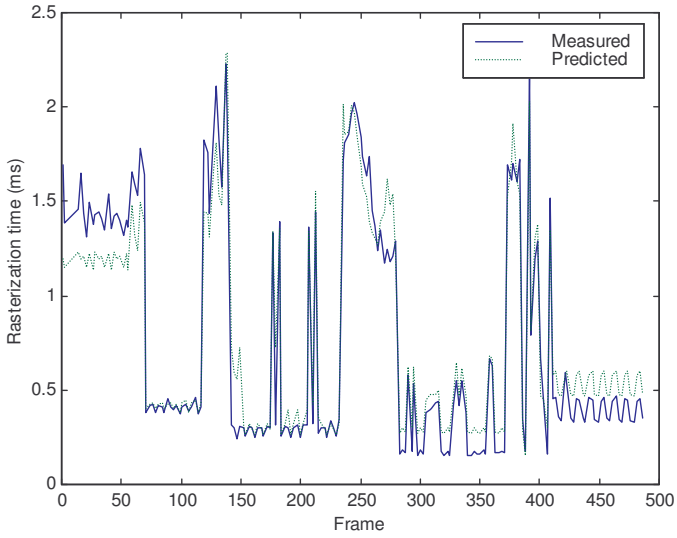


Figure 48: Measured and predicted
rasterization times for ENSTKangaroo.mp4

Basic rasterization techniques produce aliased lines, since they produce an approximation of the theoretical line. More sophisticated techniques improve the visual quality of the rasterization using partially transparent pixels together with opaque pixels (antialiasing). The assumption in the use of antialiased lines is that they are perceived as more smooth by the human eye. So, if the rasterization library enables antialiasing, we have two quality options for the rendering of geometric primitives. Even if antialiasing produces frames of higher visual quality, there is an additional cost due to the use of transparent pixels. Figure 49 shows a comparison between antialiased and normal rendering times for 180 animation frames mainly constituted of lines and polygons. As we see, the use of antialiasing produces constantly higher rendering times.

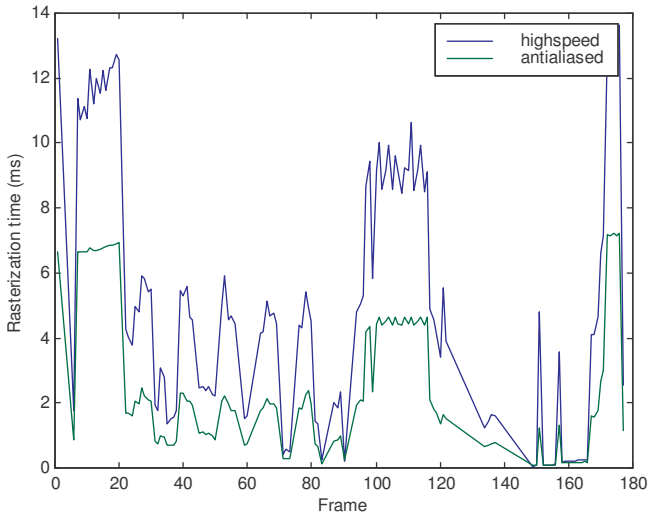


Figure 49: Comparison between high speed and antialiased rendering

We estimated the cost of antialiasing for lines and other basic geometric primitives. Antialiased rendering can be estimated from the models for normal rendering since it basically adds additional processing time that depends on the same linear models identified for the rasterization of lines and polygons. Section 4.2.3 will provide an example of how antialiasing and normal rendering can be used in the context of quality-based presentations.

### 4.1.4 Display update

The final step of the 2D rendering pipeline is the update of the display. Figure 50 shows a basic architecture of a multimedia PC.

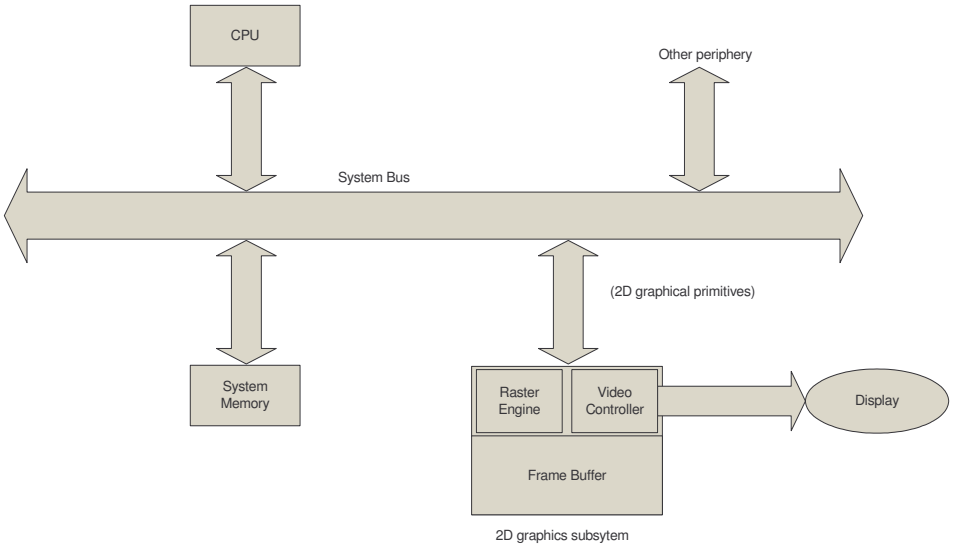


Figure 50: 2D Rendering Architecture

The video card consists of a frame buffer that contains special fast access memory, a raster engine and a video controller, which contains the digital to analog converter (DAC). From an application's point of view, the update of the display is usually performed using a method called double buffering. Instead of directly update the display it is more convenient to compose the scene in an off-screen surface, and then update the display. This has the advantage of eliminate flickering and provides also a gain in performance if the copy is performed asynchronously, because other computations can be performed in the meantime. The need for a double buffering technique rises however an important performance issue for what concerns the allocation of the off-screen surface. System memory or video memory are the possible options. We observe that in the latest years two diverging rendering methods have developed independently of each other. On one hand, video cards are offering hardware support for more and more rendering operations (i.e. scaling of 2D surfaces, alpha blending, 3D support). On the other hand, CPUs are getting faster and have specialized multimedia extensions (MMX, SIMD etc.). Using a mix of hardware and software methods to take advantage

of these independent technology trends raises some problems. If the off-screen rendering surface is located in video memory, the application can benefit of full graphics hardware acceleration. Write speed performed by the CPU is also quite fast; using AGP (accelerated graphics port) can in fact reach 200 MB/s. Read speed is instead quite slow (12 MB/s on latest AGP systems). This read performance strongly penalises MMX routines and other routines like alpha blending that require read operations. If on the other hand the off-screen surface is located in system memory, then CPU rendering is maximally efficient because of the CPU data cache. However, video hardware cannot be used, and parallel processing while copying off-screen to video memory cannot be performed. If the off-screen surface is located in AGP memory, video hardware acceleration is possible, but CPU read is again slow.

All these considerations conspire to make it difficult to use both software and hardware rendering to the same off-screen surface, as it would be necessary by the requirements of a complex multimedia presentation where text and 2D graphics have to be combined with video and bitmapped images. If a multimedia scene contains only video streams then a video memory off-screen surface would be the perfect candidate, since hardware acceleration maximise the performance. On the other hand, if a scene contains several 2D graphics objects, then a system off-screen surface is required, since most 2D graphics primitives require CPU processing. Since we are considering a multimedia terminal with heavy interaction between 2D graphics and video, an off-screen surface located in system memory is the choice for the update of the display using double buffering.

Experimental results confirmed this analysis. We performed a simple test using a PC with an ATI Radeon 7500 video board and AGP 4x. Ten geometric objects (circle and rectangles), with different line, fill and transparent properties are composed in a scene of 704 *576 pixels, 16 bits per pixel. In the first test, (first row of Table 5) objects are first drawn in an off-

screen surface located in video memory, and then the surface is copied on screen. In the second test (second row of Table 5), objects are drawn in a system memory surface and then copied to screen. Table 5 shows how drawing geometric objects in system memory is more efficient, and the overhead of copying into from system memory to video memory for display is negligible compared to the saving in the drawing phase.

| Off-screen Surface | Draw | Update Display |
|---|---|---|
| Video memory | 44 ms | 0.05 ms |
| System memory | 23 ms | 1.05 ms |

Table 5: Comparison between Video and System off-screen surfaces as target for rendering of geometric primitives

Table 6 shows that the copy of video frames to video or system memory require same time. This is because, as we said above, write operations from CPU are performed at high speed.

| Off-screen Surface | Draw | Update Display |
|---|---|---|
| Video memory | 10 ms | 0.05 ms |
| System memory | 10 ms | 1.05 ms |

Table 6: Comparison between Video and System off-screen surfaces as target for rendering of 640*480 video

Table 7 shows how the rendering of scaled video is instead more efficient when both surfaces are in video memory. This is because the video board has hardware acceleration. We repeated the same test using a Nvidia and a Matrox G400 video boards and AGP technology, and we noticed a behaviour similar to the one reported in the test.

| Off-screen Surface | Draw | Update Display |
|---|---|---|
| Video memory | 3.5 ms | 0.05 ms |
| System memory | 9.5 ms | 3.5 ms |

Table 7: Comparison between Video and System off-screen surfaces as target for rendering of 352*288 video stretched to 1024*768

From the simple tests we performed, we conclude that if 2D graphics and video have to be composed together, the current PC architecture suggests a double buffering technique that composes multimedia scenes in system memory and then copies it in video memory. The performance of this last step is then linear in the size of the surface (number of pixels * number of bits per pixel) and the write speed from system memory to video frame buffer. We measured values of about 0.25 ms for 352*288 (101376 pixels) copy and about 1.05 ms for 704*576 (405504 pixels) copy in 16 bits per pixel display mode. We conclude that the time $T_{Copy}$ of this last rendering step is linearly proportional to the size of display area $c$ and a constant $d_1$ that captures the transfer speed between off-screen and primary surfaces that depends on the memory and bus bandwidth of the hardware platform:

$$T_{copy}(c) = d_1 * c$$

If the off-screen surface is located in video memory the formula is still valid and would yield values close to 0 since both surfaces are located in video memory.

### 4.1.5 Preliminary conclusions

In these sections we discussed the rendering of MPEG-4 2D scenes, proposing and discussing different algorithms, and we provided models to estimate the complexity of each task. We observe that, given a scene graph, it is not possible to know a priori the required time for the rendering step, unless we assume that all the visual objects need to be drawn at every iteration

(direct rendering). This is because the painter's algorithm tries to minimize the number of objects drawn, and as a consequence we cannot know in advance how long the rasterization will take. However, after each traversal of the scene graph, running the painter's algorithm on the display list we know how many objects will be drawn or partially drawn, and we can estimate the rasterization time. In other words, we observe that the separate steps needed for the rendering can be estimated right before their execution, using the parameters we identified. More precisely:

- Scene graph traversal: number of shapes and size of the tree
- Painter's algorithm: size of display list
- Rasterization: type, dimension and properties of each object to draw
- Display Update: size of the window

In the next section we will see how to use the result of this analysis in the context of the quality management framework.

## 4.2 Quality management

We discuss here how to integrate the framework we proposed in Chapter 3 with the rendering estimation model that we introduced here. We first illustrate how basic graphic management is achieved and we provide a simple example. Then we consider the integration between video and heavy BIFS streams, clarifying the need for an improved presentation algorithm. Finally we consider animated cartoons and we give an example of utilization of the framework for this kind of applications.

### 4.2.1 Integration in the framework

Similarly to video streams, we envisage the use of descriptors for frame rate, size, and visual quality dimensions. The framework requires each stream in a

presentation to provide complexity information for each quality option declared by the author. Since a graphic stream may have complexity varying over time, complexity in the form of percentage of CPU cannot be calculated only once at the beginning. The scene may change during the playback and hence complexity estimated at the beginning can be completely wrong after only a few seconds. We can use the model we introduced in this chapter to build a running estimate during the whole length of the simulation. We calculate a first estimation based on the initial scene parameters like number of shapes, quality of the rendering, frame size, and desired frame rate. Then we refine the estimation, monitoring changes in the number of shapes in the scene graph and estimating the composition time at each frame after the display list is built. In this way we know how many objects will be displayed and we can produce an estimate. We use the estimation, rather than timing the effective drawing time for a frame, because timing errors on the considered platforms have high jitter, and so cannot be safely used in determining the resources needed to draw a frame. In order to fit into the framework, each stream should have an associated scene graph node that provides the monitoring and quality selector functionalities. For BIFS streams, the node associated to the root of the scene graph should provide the information. When a violation of the QoS specification of a scene is detected, the node associated to the root of the scene graph provides the estimation of the graphics complexity of the scene to the Resource Manager. This information will be used during the quality management process. Figure 51 provides an example of utilization of this basic management process. Five videos are laid out in the upper part of the screen, and two BIFS animations scroll from right to the left part of the screen, providing a commercial spot.


Figure 51: Video and BIFS augmentations

The following quality specification assigns high value to the CIF video, then the QCIF videos and finally the BIFS graphics. The BIFS frame rates include a 0 value, meaning that the animation can stop if the resources are not sufficient.

VIDEO 1 (CIF):

QoS_Qualifier_FRAME_RATES{FRAME_RATES [25,8,2] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 3}

VIDEO 2, 3, 4, 5 (QCIF):

QoS_Qualifier_FRAME_RATES{FRAME_RATES [25,8,2] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 2}

BIFS 6:

QoS_Qualifier_FRAME_RATES{FRAME_RATES [10,5,0] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 1}

Figures 52 and 53 provide frame rates of video1 and video2 and BIFS streams when QoS is disabled (Figure 52) or enabled (Figure 53) for 40 seconds of

simulation. We ran the scene on a low level machine and after about 5 seconds we additionally simulated heavy load using the tool CPU Stress described in previous chapter. We observe that the results are similar to the ones obtained in the previous chapter, and that the BIFS stream is taken into account when allocating quality options (in this example the stream is blocked when heavy load is induced).

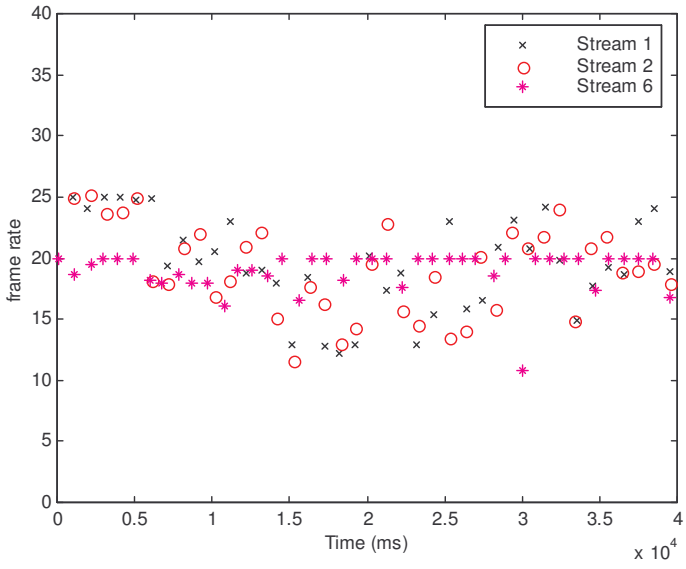


Figure 52: 40 seconds of simulation for the scene in Figure 51, QoS disabled

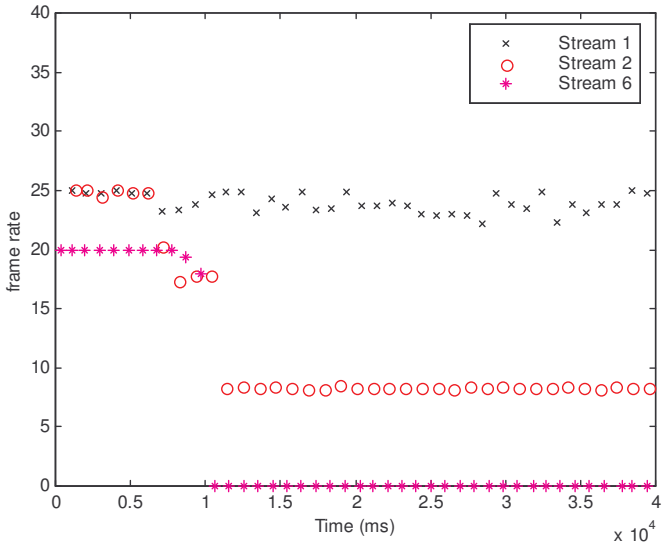


Figure 53: 40 seconds of simulation for the scene in Figure 51, QoS enabled

We observe that the use of the rendering estimation model in this example is *reactive*, since we use the model basically for providing an estimation of the resources needed for the BIFS stream, when violations are detected. The following two sections provide examples of how to use the model in a *predictive* way, that is in order to anticipate QoS violations.

### 4.2.2 Improved presentation algorithm

Integration between video and BIFS raises some problems. Videos are continuous streams, they are associated to nodes declared in the scene graph, and they need to be presented at high rates (i.e. 25 or 30 Hz). 2D graphics is declared in the scene graph, and does not usually require high rates. Scene graph traversal and object composition might be time consuming tasks, principally depending on the number of shapes. The systems reference software presentation algorithm performs the tasks of scene graph traversal and composition at a regular rate, equal to the rate needed by video streams. This solution does not well integrate video and 2D graphics, because scene traversals and composition are done at high rates and the difference in rates

of BIFS and video streams is not taken into account. A mechanism to decouple media rendering and BIFS objects rendering is needed. Separate rendering rates can be achieved storing the active video nodes in a separate list from the scene graph. The list can be updated at each full traversal of the scene tree. Full traversal is performed at the first simulation tick, then at the rate of the nominal BIFS rate, if rate descriptors are provided, or whenever the graph is invalidated if not rate descriptors are provided. In this way, we can render video nodes without traversing the scene graph. Composition is done on the basis of information collected at the latest full traversal, so that the player output can be consistent if videos have BIFS objects on top that need to be redrawn each time video is redrawn. User interaction is then taken into account allowing full traversals whenever a new event invalidates the scene graph.

This improved presentation algorithm, while optimizing the CPU utilization between the tasks needed by graphics and video, does not completely solve the problem of guaranteeing independent rates. Full graph traversals and composition, even if separate rendering rates are implemented, can influence regular processing of other continuous data. As an example, we created a simple test scene, with a BIFS stream and a video stream composed together, each one with different rates (Figure 54).

Figure 54: BIFS and Video composed together
at 10 fps and 25 fps

The QoS specification we associated to the scene is the following:

VIDEO:

QoS_Qualifier_FRAME_RATES{FRAME_RATES [25] dimensionValue 1}

QoS_Qualifier_MAX_RATE_VARIATION{MAX_RATE_VARIATION 0.1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 2}

BIFS:

QoS_Qualifier_FRAME_RATES{FRAME_RATES [10,5,1,0] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 1}

We notice that the rates are different, namely 25 fps for video and 10 fps for BIFS, and that the user assigns more value to the video stream. Besides, he is willing to accept variations in the rates of the BIFS stream, but he cannot accept variations in the rate of the video stream (a tolerance is provided, in the form of max rate variation descriptor). The BIFS scene is initially empty. Afterwards, every 100 ms 5 geometric objects are added. The initial complexity is low, since the scene graph is quite small. However, after 8 seconds, the scene graph contains 400 shapes, and about 1300 nodes. Each time a full traversal is done, about 40 ms are necessary. Figures 55 and 56 show CPU utilization and Video and BIFS frame rates for the first 9 seconds of the scene. We notice that CPU is never overloaded. This means that the resource manager and the BIFS and Video estimation components succeed in controlling CPU utilization. Still, Figure 56 shows that video QoS is not met, while the BIFS stream is always at its maximum quality, that is the opposite of what the quality specification of the scene mandates.

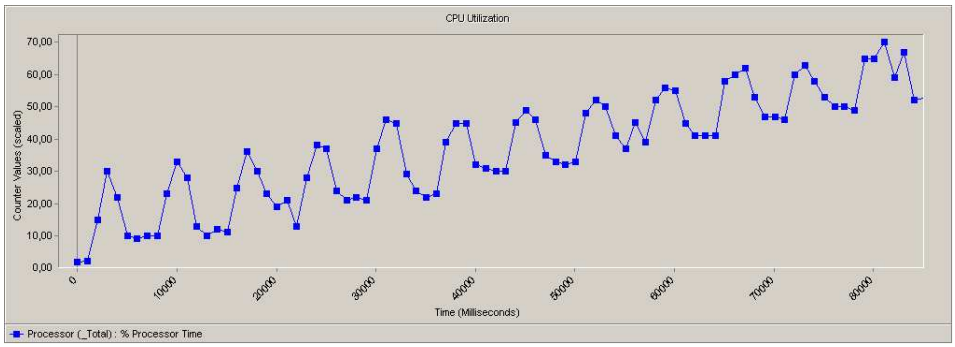


Figure 55: CPU utilization for the scene with BIFS and Video

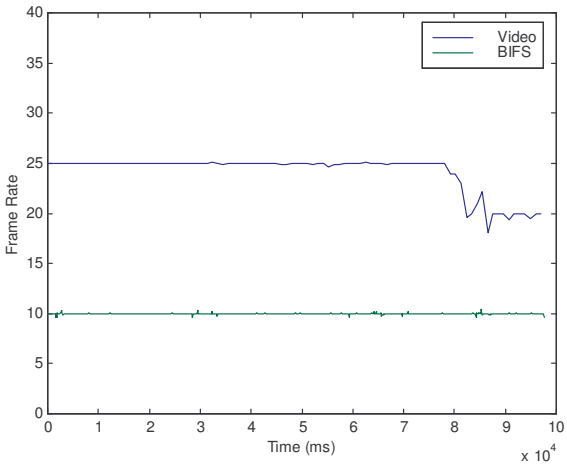


Figure 56: Video and BIFS frame rates

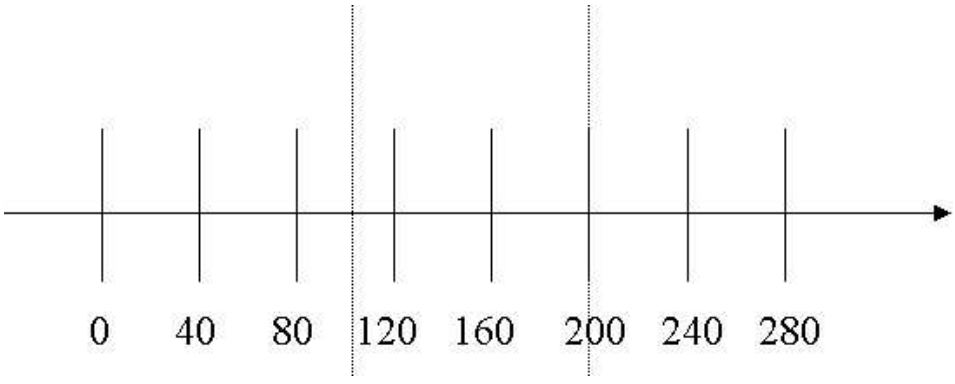


Figure 57: Deadlines of video (solid lines) and BIFS (dotted lines)

Figure 57 illustrates the different deadlines of presentation of video (solid line) and BIFS graphics (dashed line) for the first 280 milliseconds. Since video frame rate is 25 fps, every 40 ms a frame has to be presented. BIFS rate is instead 10 fps, so every 100 ms there is a deadline. Full traversals coincide with BIFS frames and are showed in dotted line. We notice that if full traversals take more than 20 ms, then some video frames will not be displayed (i.e. frame 120 in Figure 57). Video frame 240 instead will be regularly displayed, because last traversal is distant 40 ms. It follows that in order to assure video frame rates an estimation of the traversal has to be done, then compared to the next video deadline, and possibly postponed or skipped.

We observe that, when dealing with graphics and video, the quality management framework proposed in Chapter 3 cannot be only based on violations and reassignment of available CPU load. Other parameters have to be considered, as the proper scheduling of tasks like traversal and composition. The execution of these tasks can lead to quality degradations even if the CPU is not overloaded. The solution commonly used in video augmentation systems (MHEG-5, MHP) is to use overlay between video and graphics, in order to always assure video performance and achieve truly independent output of video and graphics. Since video overlay cannot be always assumed in a multimedia terminal or can be too strict (only two levels of overlapping are possible between graphics and video), it is interesting to see how to take into account this factor in our quality management model.

Using the estimation of traversal model, before actually traversing the scene graph we can foresee if the time to traverse would disturb a video deadline, and then postpone the traversal or skip it, with the goal of keeping video frame rates untouched. In the deadline graph of Figure 57, we can postpone the traversal scheduled at time 100 to time 120, adding 20 ms of jitter in the presentation of the BIFS frame scheduled at time 100. We observe that this will degrade the presentation of graphics elements, but this behaviour is

coherent with the quality specification of BIFS stream (the rate may even drop to 0). Figures 58 and 59 show CPU utilization and frame rates of the enhanced presentation algorithm. We observe that the BIFS rate drops to 0 even if the CPU is never overloaded. The video frame rate remains instead constant.

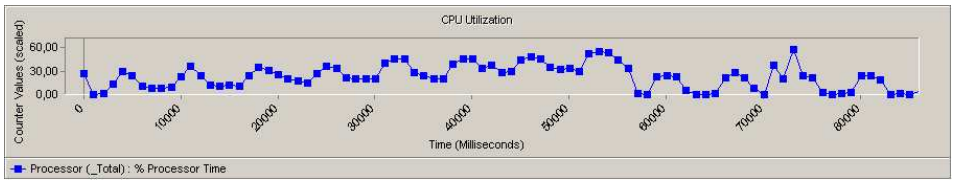


Figure 58: CPU utilization of example 1 with enhanced presentation

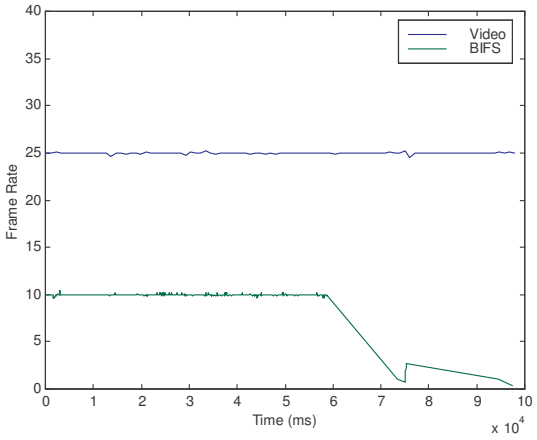


Figure 59: Video and BIFS frame rates for example 1 with enhanced presentation

When scheduling of traversal have to be delayed or composition of BIFS objects skipped (due to the number and dimension of graphics objects), the overall graphic enhancement quality can degrade considerably. This conclusion seems to be also shared in other contexts, like the MHEG-5, MHP standards we mentioned above. Reading the profiles for the implementation of terminals with 2D graphics augmentations, we found sentences like "*Rendering performance: authors should be aware that the graphics*

*drawing speed may decrease where visibles with an intermediate level of transparency are placed directly over objects other than MPEG video or MPEG I-frames. Also, visible objects may overlay RTGraphics or the TV services DVB Subtitles. However, the RTGraphic/Subtitle rendering speed may degrade and possibly even stop".* MPEG-4 profiles or industry committees have not (yet) covered these concepts in their developments. We believe that our management framework may provide a contribution to this topic.

### 4.2.3 Animated cartoons

We consider a scene comprised of a continuous BIFS stream and no other visual streams. The scene is conceived as a sequence of frames constituted of geometric objects. One frame of the scene is shown in Figure 60.

Figure 60: ENSTAg001b.mp4 cartoon

The QoS specification of the BIFS stream is the following:

QoS_Qualifier_FRAME_RATES{FRAME_RATES [12,8,2] dimensionValue 2}

QoS_Qualifier_VISUAL_QUALITY{QUALITY[HIGH,LOW] DimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 1}

We assume that the user is willing to accept changes in the perceptual quality of the scene instead of changes in the rates. The presentation algorithm can

then use the model we described to estimate the rendering for the rasterization of each frame, and switch from high quality to low quality (if the rendering library provides such options) when resources to complete a frame at a given quality level are insufficient. The information about desired BIFS frame rates (normally not contained in the MPEG-4 Systems standard) is useful in knowing how much time we have to complete a frame. A best effort system without quality management can render a scene whenever scene changes are detected, but in this way no assumptions about how much time is available to render a frame could be made.

A consequence of the use of this quality specification is that depending on the terminal, a scene can be displayed at a different quality. Quality changes can never occur for high-end (always high quality) or low-end terminals (always low quality), or they can occur during the presentation depending on the available resources. We used antialiased and normal rendering as high quality/low quality modes. As we saw in section 4.1.3, antialiasing provides a better visual quality but it requires more computational resources. We ran the scene first with QoS enabled. Then we ran the scene with QoS disabled and using high quality, antialiasing rasterization for the whole scene. Figure 61 compares the frame rates of the two experiments, for 250 animation frames. The figure shows also the quality mode used during the rendering, for the QoS enabled experiment. The quality level is set to 1 if the frame is rendered in low quality and 0 if a frame is rendered in high quality. We observe that, when QoS is enabled the terminal switches to low quality mode in order to keep the frame rate constant from time 2000 to 4000 and from time 10000 to 14000, while when QoS is disabled the terminal shows low frame rates in the same interval. Figure 62 compares instead the frame rates when QoS is enabled and when QoS is disabled but the terminal uses always low quality. We observe that minor frame variations occurs when QoS is enabled and the terminal uses high quality and low quality rendering, but the frame rate is similar to the one obtained using always low quality rendering.

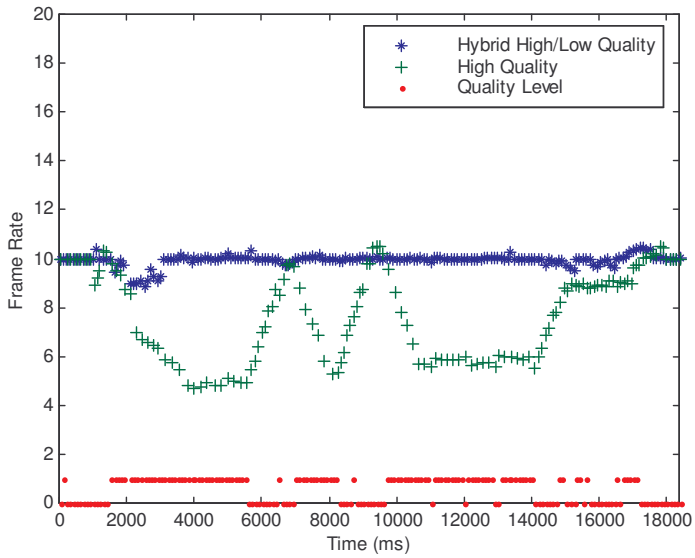


Figure 61: Comparison of frame rates for ENSTAg001b.mp4, hybrid and high quality

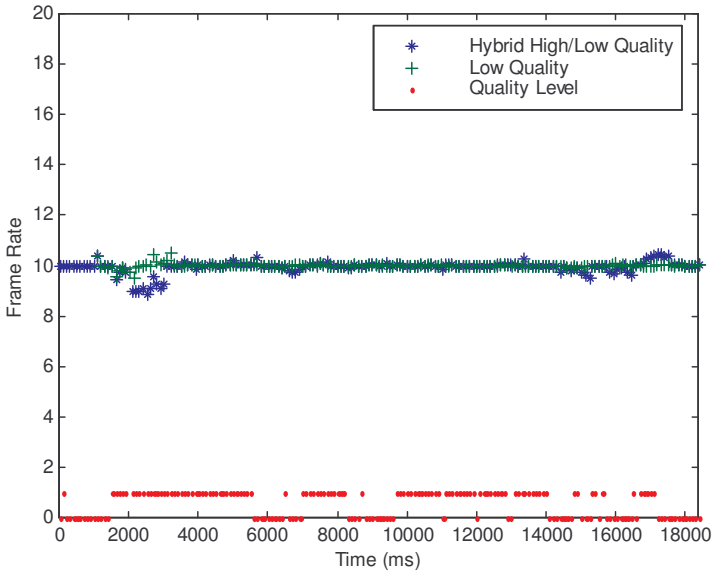


Figure 62: Comparison of frame rates for ENSTAg001b.mp4, hybrid and low quality

## 4.3 Conclusions

As part of the multimedia architecture proposed in this work, this chapter focused on the rendering components of an MPEG-4 Systems terminal. We subdivided the rendering process into a number of conceptually independent tasks and we identified the parameters that affect their performance. Since composition of visual objects is not specified by the standard specification, we also discussed algorithms for 2D composition, and identified load control processes. We observed that predicting a priori the time for the rendering of a frame is not possible when a painter's algorithm is used to minimize the number of object drawn. However, we showed how the estimation of the separate tasks could be used in the context of the proposed framework. The model can be used to provide complexity figures for the BIFS graphic enhancement stream, in order to better allocate computational resources at start-up or when QoS violations are detected. We also discussed the need for separate rendering rates for graphic augmentations and video streams, and we saw how to use the prediction of the separate rendering tasks in order to improve video rendering. Lastly, we showed how quality management could be carried out along the dimension of perceptual visual quality when the rendering library has different rendering quality levels.

*Chapter 5*

# CONTINUOUS SCENE DESCRIPTION STREAMS

In Chapter 3 we proposed a multimedia framework to provide quality-based presentations. In the proposed architecture, decoding control filters monitor the performance of the associated media decoders. In this chapter, we aim at providing evidence that a control mechanism is needed also for the decoding of scene description stream (BIFS), and discuss options to achieve that. Usually, decoding complexity of scene description streams is not a penalizing issue for the run-time performance of a terminal. In most multimedia applications the terminal is supposed to receive the scene, decode it and then start the simulation. Scene decoding is hence done only once, and when the simulation is not even started. However, some specific kind of multimedia applications requires the continuous decoding of scene updates *during* the simulation, and as a result the continuous update and render of the scene graph. This task is similar to the reconstruction of a time-constrained flow where presentation units are composed of set of nodes of the scene tree. In the following, we first provide evidence of the need of a control mechanism (Section 5.1), afterwards in Section 5.2 we provide a model to estimate decoding times and finally in Section 5.3 we discuss methods to use this model in the context of the proposed framework.

## 5.1 Motivation of the work

In 'animated cartoons', the illusion of character animation is achieved drawing consecutive frames one after the other in rapid sequence. Cartoons can be easily represented using the MPEG-4 BIFS language. Animation frames are represented using 2D primitives to draw lines, polygons and curves. A BIFS

cartoon scene is composed of a set of *layers*, and a *dictionary* of shapes [MCD2001].

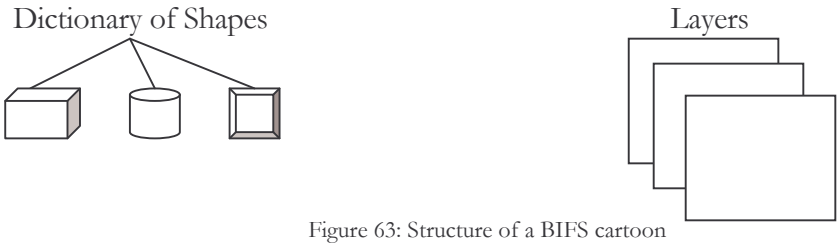


Figure 63: Structure of a BIFS cartoon

Layers are declared one after the other, at the root level of the scene. Each layer contains one or more reference to shapes contained in the dictionary. The initial scene contains the declaration of layers and the initial dictionary of shapes. Consecutive scene updates are declared using time-stamped access units with BIFS commands (introduced in Chapter 2, Section 2.2). Each BIFS command may contain an update of the shapes contained in the dictionary, either as a replacement of the dictionary shapes or a replacement of the properties of the existing ones, and an update of the layers. We observe that at each scene update, a MPEG-4 terminal has to decode the update's access unit, and render the scene graph. Obviously, the complexity of the decoding task affects the performance of the system. Fig. 64 shows three snapshots of an MPEG-4 based cartoon, "compfx". Fig. 65 shows the CPU load as a function of time during the execution of this scene. The graphs show clearly that the CPU is overloaded for the first part of the rendering. The visual effect during the play is definitely not pleasant: the animation goes slower and faster since the periodicity of the output is disturbed by the CPU load. As a result, many animation frames are not displayed (40 out of 70, ~57%).

Figure 64: Compfx snapshots

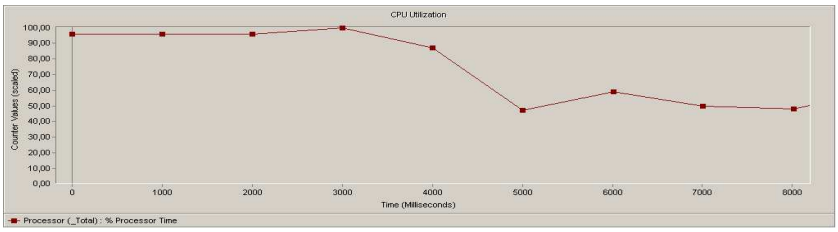


Figure 65: CPU load during Compfx

We profiled the reference software MPEG-4 player in order to get some insight on the time spent in each module during the execution of this cartoon. Figure 66 shows a broad repartition of the execution times between the MPEG-4 player and some dlls and drivers running on the terminal. We notice how the player uses 37% of the time while the graphic library uses the 25%.

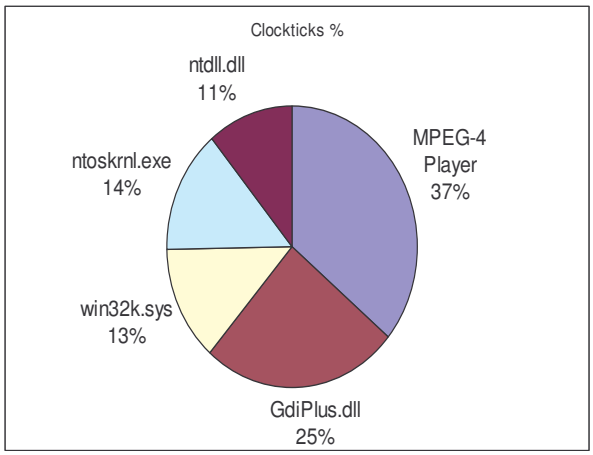


Figure 66: Distribution of execution time among the main system components during execution of the compfx bistream

Figure 67 instead shows the distribution of the execution time between the main modules of the player. We observe that the BIFS decoder runs for most of the time (41%), followed by the scene graph management (27%) and the visual renderer (17%). Figures 66 and 67 suggest the need for further profiling of the BIFS decoder, in order to get some insight about the functions that most influence its performance.

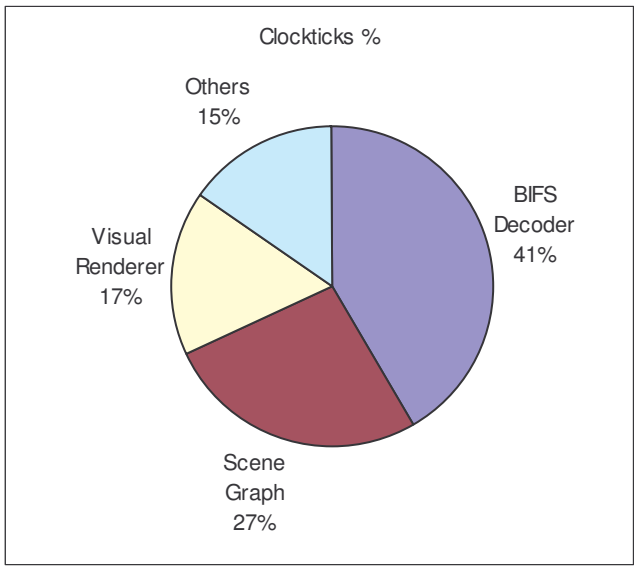


Figure 67: Distribution of the execution time between modules of the MPEG-4 player

Figure 68 shows the 4 functions that collected most clockticks in the BIFS decoder. We observe that most of the execution time is used by the GetBit and ParseFloat functions; indeed they collect nearly the totality of the clockticks samples. These functions are dependent on the size of the access units that constitute the bitstream. This suggests that the performance of the BIFS decoder is mainly affected by the size of the incoming access units.

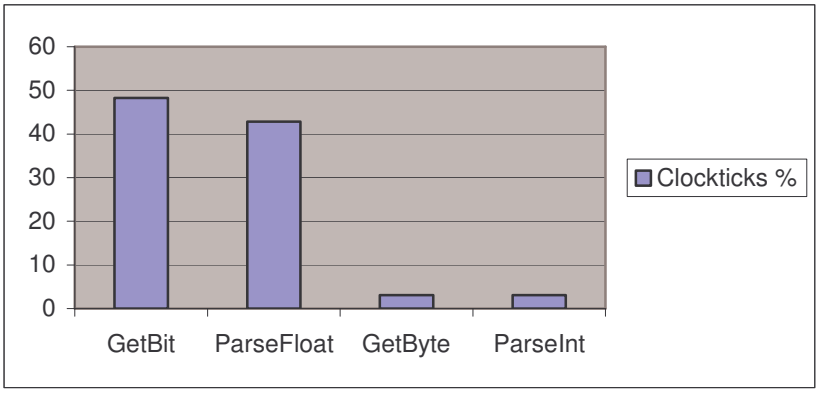


Figure 68: Distribution of the execution clockticks in the BIFS decoder module

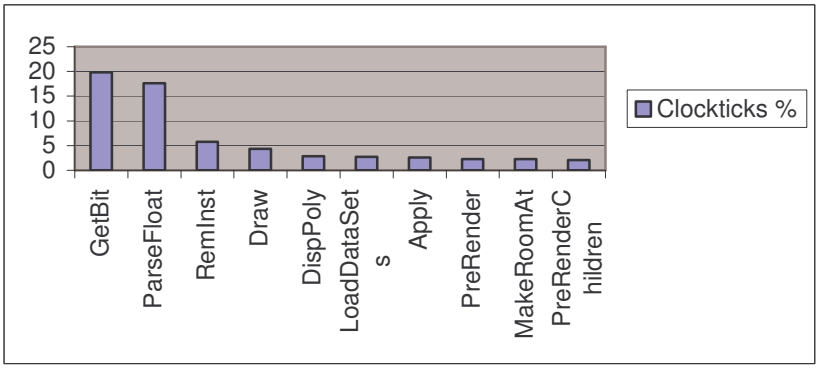


Figure 69: Distribution of clockticks over all the functions of the player.

In order to confirm our analysis, Figure 69 shows the 10 functions that have got most execution clockticks in the all MPEG-4 player. As expected, the GetBit and Parse Float functions received the most of CPU time.

We performed the same analysis on another MPEG-4 player implementation (GPAC source forge project). Figure 70 shows the 10 functions that received most CPU time for the GPAC player.

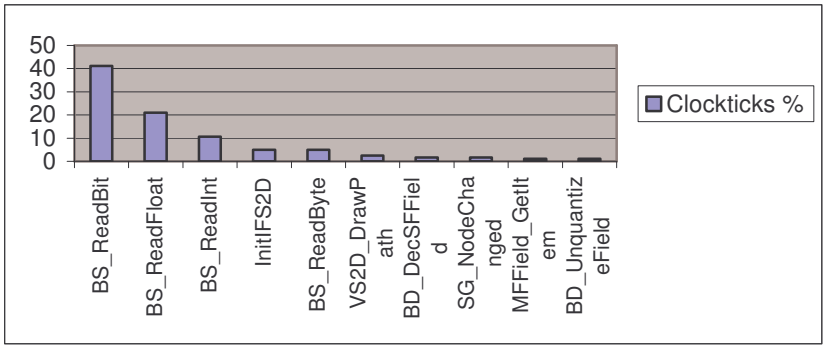


Figure 70: 10 functions with max execution clockticks (GPAC)

The profiling gives the very same results for what concerns the two functions most called, confirming our previous analysis. Finally we timed the BIFS decoding time (Figure 71).

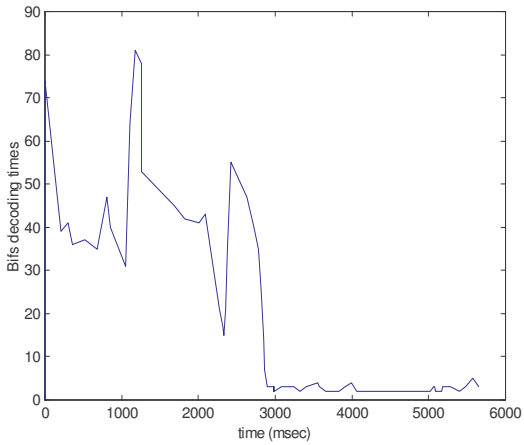


Figure 71: BIFS decoding time as a function of time during Compfx.mp4

We observe that, since the nominal frame rate of this application is 12.5 frames per second, the terminal has 80 milliseconds to parse an access unit, update the scene tree, and render the tree. As we can see from the picture, the average initial decoding times are around 60 milliseconds. Decoding and tree update times then vary a lot over the time, from ~10 to 90 ms for each access

unit. Decoding and rendering times in most cases do not fit in the interval between two access units. As a consequence, most frames (57 %) are decoded but not presented. It should be noted that since decoding times are very high, a control action only on the rendering process would not improve the terminal load. Even if we render the scene at a rate different from the nominal frame rate, there would not be much improvement. On the basis of this analysis, in the next section we investigate a model to predict the decoding times as a function of the size of the incoming data units.

## 5.2 Estimation Model

In this section we aim at finding a model to predict BIFS decoding times. First a brief overview of BIFS decoding is given. Then we illustrate the environment in which decoding time measurements were taken, followed by the mathematical analysis of the acquired samples. A set of predictors is given and finally the results are discussed.

### 5.2.1 BIFS Decoding

As pointed out in Chapter 2, BIFS is the MPEG-4 scene representation format, which can be seen as a binary encoding of VRML plain text standard, with some added features. Access units of BIFS elementary streams are called Command Frames. They contain a number of BIFS-Commands, that all share the same time stamp. These commands enable the insertion/deletion/replacing of nodes, fields, routes or the replacement of the whole scene. Nodes are encoded according to some coding tables and the fields containing floats or coordinates are further quantized to improve coding efficiency. BIFS decoding implies parsing access units of variable sizes and building the scene tree or updating an existing one. We don't go into the details of the encoding/decoding process here, since a thorough analysis of BIFS decoding is not in the scope of this work. BIFS decoding is described in [PE2002].

### 5.2.2 Regression Analysis

We considered a set of BIFS applications, all characterized by continuous scene updates, and we sampled the BIFS decoding times. The decoding of each access unit was timed using a high-resolution timer. We then ran a regression modelling the access units decoding time as a function of its size measured in bytes. The plot of the Compfx cartoon analysis is given in Fig. 72.

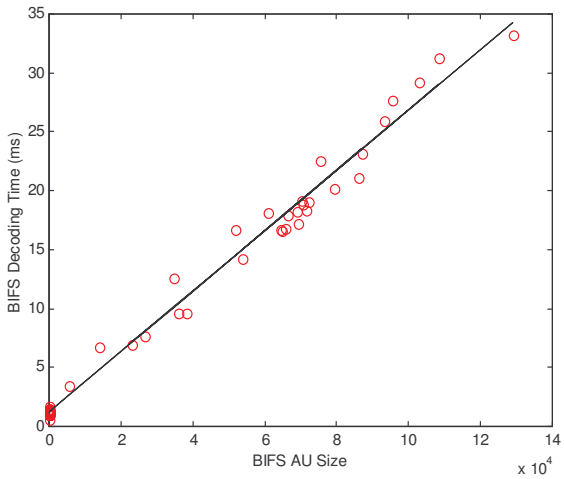


Figure 72: BIFS decoding time as a function of AU Size predicted by a linear model

The x-axis contains the AU size, the y-axis the actual decode time. The regression line is also reported. To further measure the goodness-of-fit of the linear regression, we calculated the $R^2$ correlation coefficient and we obtained 0.97, which confirms the visual appearance of the data. Figure 73 shows instead the plot of linear fitting for another cartoon, "gen002".

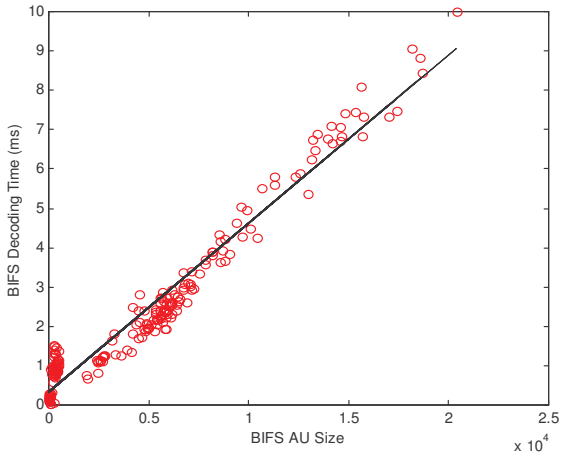


Figure 73: ENSTgen002.mp4

We found similar results for all the BIFS applications considered. Table 8 reports the $R^2$ coefficients for some bitstreams.

| Sample | N Pts | $R^2$ |
|---|---|---|
| Ad004c | 132 | 0.967034 |
| Ag001 | 191 | 0.993852 |
| Ag005 | 63 | 0.987426 |
| Ay001 | 266 | 0.986040 |
| Gen002 | 203 | 0.921692 |
| Vecto | 30 | 0.959497 |

Table 8: Regression analysis results

We also ran a regression on a set comprised of data from the cartoons in the table ("Union"). We obtained a $R^2$ value of 0.96. This study clearly shows that there is a strong correlation between the size of an access unit and its decoding time. In the following we compare predictors based on this observation.

### 5.2.3 Predictors

Dynamically computing a linear regression in real time is too expensive for a predictor that is meant to be used before every decoding. We implemented cheap predictors, which approximate a linear model. Our first predictor

(BIFS_P1) comes from a simple proportion. Let $Au_1$, $Au_n$ be n BIFS access units. For the generic $Au_k$ unit we apply the following proportion:

(1) DecodingTime ($Au_k$) = (DecodingTime ($Au_{k-1}$)* Size ($Au_k$))/ Size ($Au_{k-1}$)

The second predictor (BIFS_P2) uses a line with a slope computed on the basis of the last four access units decoding times and sizes. This linear predictor derives from the one used in [BMP98] to predict MPEG video decoding times, even if we applied it to another context. If we denote E ($X_k$) the mean of the variable X based on k observations, and $Slope_{k-1}$ the slope calculated at the k-1 access unit on the basis of the four previous one, we have:

(2) DecodingTime ($Au_k$) = E ($DecodingTime_{k-1}$) + (Size ($Au_k$) -E ($AuSize_{k-1}$))*$Slope_{k-1}$

The third predictor (BIFS_P3) is a little variant of the second one. Instead of using a slope based on the last four samples, we use a slope pre-calculated (once) by linear regression on a set of all the samples. In the following section we evaluate the predictors against seven chosen cartoons.

### 5.2.4 Evaluation

We tested the three predictors on the seven cartoons and the set composed of union of the samples, 955 access units. In the latter case, the data are only locally correlated, with a correlation window equal to the length of each cartoon. The following tables report the performance of the predictors for each cartoon. The performance is measured observing the sum of absolute error (ms), the maximum absolute error (ms) and the percentage of samples predicted within a threshold of one/three/five/ten milliseconds.

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 1569 | 30 | 82 | 92 | 94 | 96 |
| BIFS_P2 | 1790 | 28 | 49 | 90 | 95 | 98 |
| BIFS_P3 | 1615 | 28 | 55 | 92 | 96 | 98 |

Table 9: Performance of three BIFS predictors for Union Sample

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 325 | 28 | 73 | 93 | 95 | 97 |
| BIFS_P2 | 333 | 27 | 62 | 90 | 96 | 98 |
| BIFS_P3 | 375 | 27 | 60 | 90 | 96 | 98 |

Table 10: Performance of three BIFS predictors for Gen002 file

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 137 | 25 | 68 | 82 | 90 | 91 |
| BIFS_P2 | 109 | 24 | 64 | 78 | 91 | 97 |
| BIFS_P3 | 108 | 19 | 65 | 85 | 92 | 95 |

Table 11: Performance of three BIFS predictors for Compfx

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 24 | 5 | 65 | 96 | 100 | 100 |
| BIFS_P2 | 41 | 5 | 64 | 90 | 100 | 100 |
| BIFS_P3 | 48 | 5 | 66 | 90 | 100 | 100 |

Table 12: Performance of three BIFS predictors for Vecto

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 240 | 13 | 84 | 93 | 94 | 96 |
| BIFS_P2 | 341 | 12 | 58 | 90 | 92 | 97 |
| BIFS_P3 | 203 | 14 | 83 | 96 | 96 | 97 |

Table 13: Performance of three BIFS predictors for Ag001

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 109 | 10 | 95 | 97 | 98 | 98 |
| BIFS_P2 | 140 | 16 | 72 | 98 | 98 | 99 |
| BIFS_P3 | 138 | 10 | 95 | 98 | 99 | 99 |

Table 14: Performance of three BIFS predictors for Ay001

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 60 | 4 | 73 | 92 | 95 | 98 |
| BIFS_P2 | 80 | 3 | 53 | 90 | 96 | 98 |
| BIFS_P3 | 40 | 2 | 85 | 98 | 98 | 98 |

Table 15: Performance of three BIFS predictors for Ag005

| Predictor | Total Error (ms) | Max Error (ms) | % w/in 1ms | % w/in 3ms | % w/in 5ms | % w/in 10ms |
|---|---|---|---|---|---|---|
| BIFS_P1 | 370 | 62 | 71 | 83 | 84 | 91 |
| BIFS_P2 | 213 | 24 | 69 | 90 | 94 | 97 |
| BIFS_P3 | 237 | 15 | 59 | 90 | 94 | 96 |

Table 16: Performance of three BIFS predictors for Ad004c

As we can see from the tables, BIFS_P1 has always the best performance in predicting times within 1 ms, but has always the worst max error value. The following picture shows the histogram of error for BIFS_P1 predictor on the union sample. As we see, the error follows a normal distribution centred on zero.

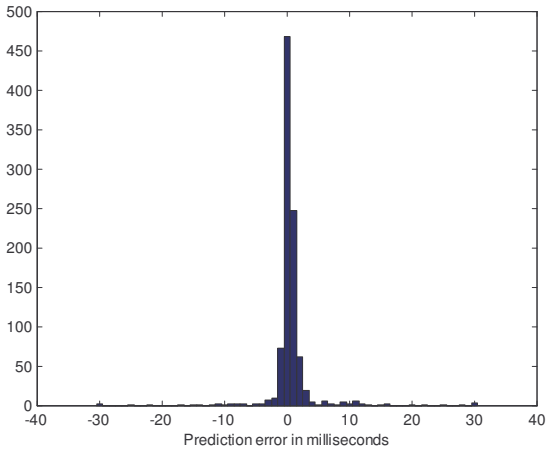


Figure 74: Histogram of errors for BIFS_P1

BIFS_P2 has low total and maximum errors. With the exception of compfx, all the cartoons decoding times can be predicted within a threshold of 3 ms in the 90-98% of the cases. If we look at the 5 ms threshold instead, BIFS_P2 works well on all the cartoons, since the 90-100% of all the decoding times can be predicted within a confidence of 5 ms. Figure 75 shows the histogram of error for this predictor.

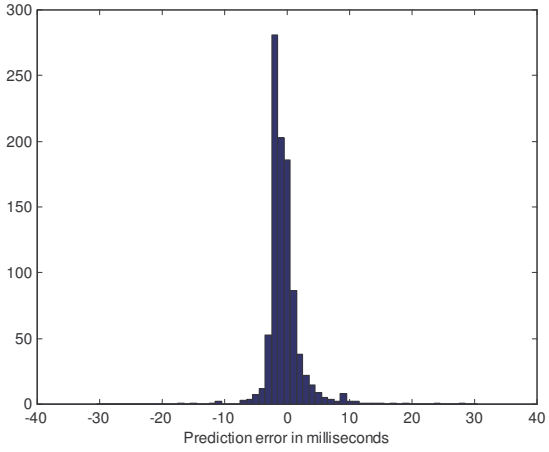


Figure 75: Histogram of errors for BIFS_P2

BIFS_P3 is similar to BIFS_P2, but it has always the minimum max error and almost always the minimum total error, and also shows the best performance in predicting times up to 3 ms seconds, with a percentage of 85% for compfx, and of 90% and more for all the other streams. Figure 76 shows the histogram of error.

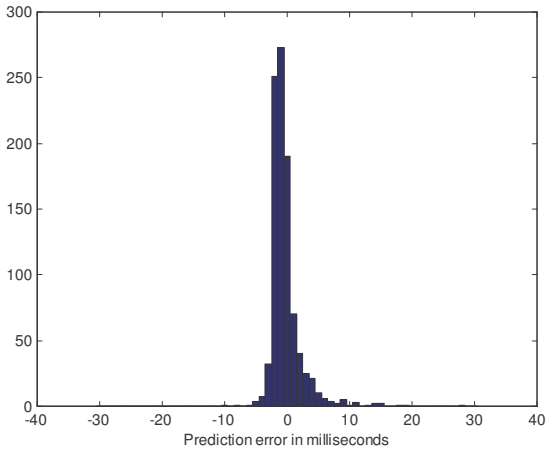


Figure 76: Histogram of errors for BIFS_P3

All the three predictors are very cheap to implement. BIFS_P3 is penalized because it requires a pre-calculated linear regression calculus, done on a large set of data. BIFS_P2 and BIFS_P1 are instead generic enough to be implemented on software architectures without changes. Since we are not interested in predictions within 1 ms, BIFS_P1 is not the most favourite candidate. BIFS_P2 has better total error values and good performances within 3 and 5 ms, that are good margins for any practical use of this predictor.

### 5.2.5 Preliminary Conclusions

This study shows that it is possible to predict BIFS decoding times, using a linear model, as function of the size of the BIFS access unit. It shows also that it is possible to implement efficient cheap predictors that can predict

decoding times within 3 and 5 ms with a rate of success varying from 90 to 100 % of the samples.

## 5.3 Use of predictor model

In the context of the subject of this thesis, we envisage the use of the BIFS prediction to intelligently skip some command frames or to modify the stream time base. In the following we illustrate these two options.

### 5.3.1 Frame skipping

As with video streams, a simple technique to reduce the load of a decoder is the skipping of some decoding units. However, all the decoding units may contain information that cannot be skipped. For instance, in the case of cartoons, BIFS update commands can contain shapes meant for the update of the dictionary, which might then be used in several other frames. Blindly skipping the decoding of these shapes would compromise the display of several animation frames. Clearly, a method to differentiate between access units is needed. Unfortunately, the header present in BIFS access units just define the kind of BIFS command, and does not provide any hint about the possibility to skip or not a given command. If we consider a particular service scenario, then some additional information can be used. For instance the mp4 file format provides a data structure, called "DegradationPriority" atom, which can contain degradation information for each sample of a presentation. The information is comprised of 15 freely usable bits. The author of a BIFS scene can specify which commands the decoder can skip without affecting the consistency of the scene. Using the BIFS decoding prediction model, the terminal can skip only the samples that are too costly to be decoded. This method preserves the temporal duration of the stream, while degrading the fluidity of the animation. However, the author has to add some degradation information to the scene, without which the presentation cannot be degraded.

### 5.3.2 Time base adjustment

A different approach is to work on the temporal scale of the presentation. The main idea is to slow down the presentation when the processing requirements cannot be satisfied. In order to achieve that, we can operate on the clock associated to the BIFS stream. For each BIFS access unit, we predict its computational requirements and we modify the clock by slowing it down or accelerating it on the basis of a prediction of the decoding and rendering time. We can express this in the following way.

Let the following be given:

$S=\{a_1, a_2, ...a_n\}$ stream of access units

$T=\{t_1, t_2, ...t_n\}$ time stamps associated to the stream S

The processing time (sum of decoding and rendering time) for an access unit $a_k$ can be expressed as a function:

$$p : S \rightarrow \mathbb{R}$$

The requirement to achieve the display of all the access units can be expressed as:

(1) $p(a_k) \leq t_{k+1} - t_k \qquad k : 1, ...n$

This means that the time to process access unit $a_k$ is bounded by the interval between two consecutive time stamps $t_{k+1}, t_k$. If this condition is not satisfied, delay will be introduced and subsequent frames will be skipped. In order to avoid any disruption, the stream clock speed should then be scaled of a factor given by:

(2) $c_k = p(a_k)/(t_{k+1} - t_k) \qquad k:1,...n$

In practice, if we assume a stream with nominal fixed frame rate $f_s$ , then the time interval between two consecutive access units is constant and given by $1000/f_s$ . The time to process an access unit can be approximated by the sum of the decoding and rendering models identified in this chapter and the previous one:

(3) $T_{rendering}(a_k) \leq (1000/f_s) - T_{decoding}(a_{k+1}) \quad k:1,...n$

In (3) we assume that we determine the clock speed factor at each cycle before starting the rendering. A more obvious solution would have been to modify it before the decoding of BIFS unit, but in that case the estimate of the rendering time would have been less precise because it would not have considered the variations in the scene graph produced by the BIFS unit itself.

Figure 77 shows the clock speed factor and the estimated decoding and rendering times as a function of the time for the compfx cartoon. The clock speed factor normally varies from 0 to 1 and in Figure 77 it is scaled by a factor of ten for clarity. The frame rate is 12.5 fps, so the terminal has 80 milliseconds to perform the decoding and rendering of each access unit. We observe that when the sum of decoding and rendering is higher than 80 milliseconds, the clock scale factor lowers accordingly. Considering the number of displayed frames, 64 out of 70 frames are correctly displayed. Some frames are nevertheless missed, because in order to improve the visual quality and the smoothness of the scene we implemented a system that gradually modifies the clock speed proportionally to the exact value of the clock speed factor. Running the player without time base adjustment, only 40 frames out of 70 would be displayed.

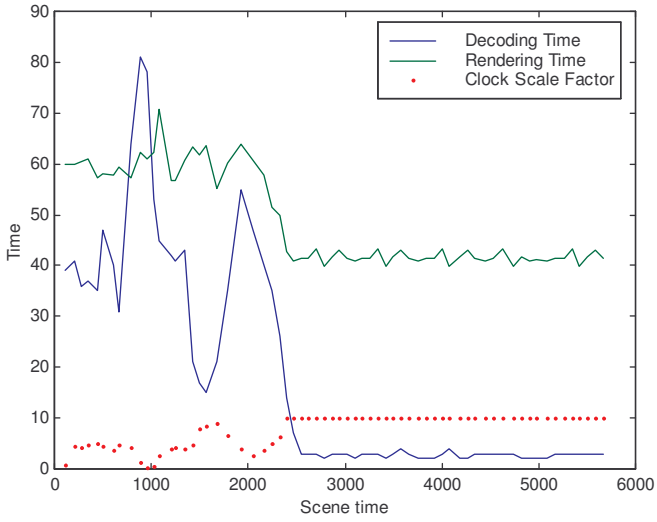


Figure 77: Compfx, estimated decoding and rendering times, and value of the clock scale factor (scaled)

Figure 78 shows the same plot for another cartoon, "aa001b", that in a way shows a more typical performance for the decoding and rendering tasks. We observe that BIFS decoding time is quite small compared to rendering time. Nevertheless, there are cases where the sum of decoding and rendering is higher than the 80 milliseconds time duration for each frame. Frame statistics show that 115 frames out of 130 are displayed, compared to 95 out of 130 when no time adjustment is used.

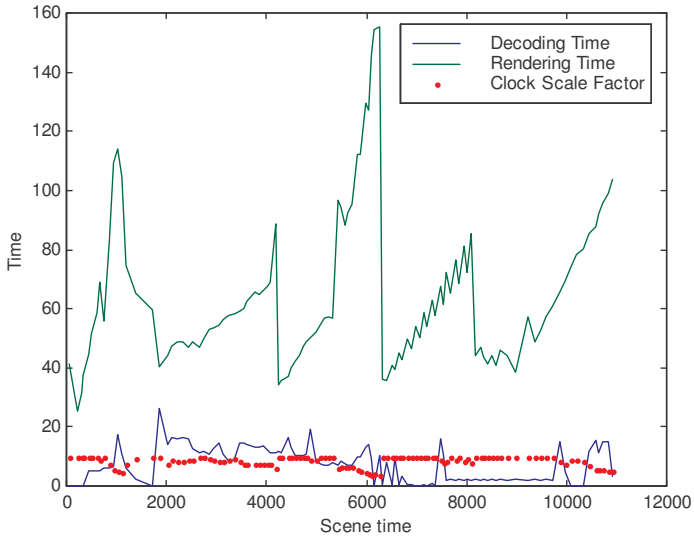


Figure 78: Aa001b, estimated decoding and rendering times, and value of the clock scale factor (scaled)

## 5.4 Conclusions

Even if usually BIFS decoding complexity is minor compared to other system tasks (i.e. rendering), there are cases where the complexity of the decoding task can affect the quality of the playback. In this chapter we introduced a model to predict BIFS decoding times, meant to be used in applications like BIFS cartoons that require the decoding of a continuous BIFS updates. The model can be used to predict decoding times within 3 and 5 ms with a rate of success varying from 90 to 100 % of the samples. We also showed how the model could be used in the context of the work of this thesis, namely for the adjustment of the time base of a stream, when the only other acceptable solution would be the abort of the presentation.

*Chapter 6*

# CONCLUSIONS

In this thesis we addressed the problem of supporting quality-based multimedia applications. The methodology adopted throughout this work consisted in modelling a multimedia terminal as a *user-centric* system, assuming end users to provide the notion of value to a terminal, suggesting how they expect it to work under unpredictable events. Consequently, we first identified QoS parameters to define the quality of a presentation, then proposed a multimedia framework based on this quality specification and subsequently identified control processes for the main system components of the framework. This contribution has been based on the MPEG-4 Systems standard and its reference software platform. However, we believe that most of the concepts expressed in this work can be used in other contexts as well. MPEG-4 Systems provides an integrated, standardized framework to support 2D graphics and natural video, and as such this contribution has progressed touching various aspects of the standard. Adopting and contributing to the development of the standard has provided homogeneity to this work, that otherwise would have suffered of multiple private systems choices that would have limited the effective interest and usefulness of this thesis. Section 6.1 briefly reports the main achievements of the thesis, while Section 6.2 discusses possible future developments.

## 6.1 Achievements

In the following the main achievements of the thesis are reported.

### 6.1.1 Quality-based multimedia framework

We established a framework for quality-based multimedia presentations. The framework has been designed to adapt to variations in the load of a terminal on the basis of a user quality of service specification. Borrowing design principles investigated in control theory for discrete event systems, we addressed the issues of controllability, robustness and reactive configuration defining an architecture together with control and quality management processes. The proposed terminal performs resource allocation and management, trading off user satisfaction and available system resources. User satisfaction is implied from a user-level quality specification, that identifies a list of quality options for each media quality dimension, and sets up a ranking among them. Monitoring of terminal performance is performed using proper application level metrics. We exercised the concepts on three scenarios: admission control, competition between applications and power management. The results show how the proposed architecture succeeds in controlling the terminal under variations in the load, trying to preserve the quality dimensions that the user valued the most.

### 6.1.2 Integration of 2D synthetic content

We identified the main tasks and the parameters that affect the performance of scene graph based 2D graphic rendering. The methodology we followed is to subdivide the rendering process into a number of conceptually independent tasks that can be estimated separately. We hence provided a model to estimate the resources of these main tasks. As a result we proposed ways to integrate 2D graphics in the quality management framework, using the rendering model to estimate the resources for the different quality options for 2D graphics augmentations and to anticipate variations in the load of a terminal due to changes in the performance of the graphic subsystem. We identified algorithms to achieve composition of 2D MPEG-4 scenes, and discussed their performances and their integration in the framework.

### 6.1.3 Continuous scene description streams

The decoding and rendering of continuous scene description streams has been analyzed in the context of the scope of this work. We showed how the estimation of the rendering task could be used for the playback of a stream at different levels of quality. We introduced a simple linear model to predict BIFS decoding times. The model can be used to predict decoding times within 3 and 5 ms with a rate of success varying from 90 to 100 % of the samples. We showed how the model could be used for the adjustment of the time base of a stream, when the resources are insufficient for the normal playback, and the only other acceptable solution would be the termination of the presentation.

### 6.1.4 MPEG-4 standardization and R&D projects

During this thesis we contributed to the development of the MPEG-4 standard. Part of the 2D player we developed was included in the reference software as informative annex in 1998 and since then has been providing advanced multimedia services in a number of European Projects (MPEG-4 PC, NEXTTV, SOMMIT, OCCAM) and Italian Research Projects (BROM, MAD). At the moment of writing this thesis, the player is a core part of a suite of advanced streaming media products shipped by TDK to Educational, Corporate and Residential markets in Japan (www.wonderstream.com).

### 6.1.5 Published work

Part of this thesis has been published in the following articles and book chapters.

J-C Dufourd, S. Boughoufalah, G. Di Cagno et alii, *MPEG-4 PC – Authoring and Playing of MPEG-4 Content*, EMMSEC 99 European Multimedia, Embedded Systems and Electronic Commerce Conference and Exhibition, Stockholm June 21-23 1999.

G. Di Cagno, F. Casalino, *MPEG-4 multicast over satellite*, IEEE MULTIMEDIA SYSTEMS 99 JUNE 7-11 1999, Florence, ITALY

L. Ronco, G. Di Cagno, F. Casalino *MPEG-4 Video decoder optimization*, IEEE MULTIMEDIA SYSTEMS 99 JUNE 7-11 1999, Florence, ITALY

Z. Lifshitz, G. Di Cagno, S. Battista, and G. Franceschini, *MPEG-4 Players Implementation*, Chapter 15, Advances in Multimedia: Systems, Standards and Networks, edited by Atul Puri and Tsuhan Chen Marcel & Dekker Inc, 2000

Z. Lifshitz, G. Di Cagno, M. Leditschke, *Implementing the Standard: the Reference Software*, Chapter 16, The MPEG-4 BOOK, edited by Fernando Pereira and Touradj Ebrahimi, Prentice Hall, IMSC Multimedia Series, 2002

## 6.2 Future developments

This thesis covered only a part of the problematic of supporting multimedia services, namely adaptation at end-user terminals caused by variations of the available computational resources. Adaptation is currently investigated at various entities in the digital information distribution chain, from content/service generation to end-user terminals. We believe our work can be leveraged in the context of achieving an *end-to-end* QoS for distributed multimedia. The first step towards this objective would then be to complement our framework with research on QoS on a particular networked service scenario (i.e. wireless or internet). Besides, since MPEG-21 is in the process of defining a framework for the management of terminal QoS, the integration of our framework into the MPEG-21 terminal architecture would be an interesting extension of our work.

# APPENDIX

## RESUME LONG EN FRANÇAIS

### Introduction

Le mot multimédia a été introduit pendant les années soixante-dix pour décrire la composition d'un film et d'une projection de diapositives. De nos jours, il est généralement utilisé pour exprimer l'interaction homme machine impliquant du texte, des graphiques, de la voix et de la vidéo [hyp2003]. Cette définition clarifie deux concepts de base des applications multimédia. D'abord, elles se composent de différents médias, aggregès pour fournir une expérience audiovisuelle riche; en second lieu, elles fournissent des mécanismes avec lesquels l'utilisateur peut interagir. Les champs d'application incluent le divertissement, l'éducation et la publicité. Selon cette définition, un système multimédia est un terminal qui joue différents médias simultanément, selon quelques règles de synchronisation, avec un certain niveau d'interaction utilisateur. Même si le multimédia se rapporte souvent à l'informatique, il peut être employé pour décrire un certain nombre d'appareils manipulant des médias, tels que les magnétoscopes numériques, les dispositifs sans fil et les affichages publics. En effet ces dernières années ont montré une croissance importante du nombre et du type de ces dispositifs. Comparé aux applications traditionnelles de données, les présentations multimédia ont des besoins différents. Elles sont composées de médias continus et non continus, qui doivent être présentés à des instants précis. En outre, contrairement aux applications de données, les erreurs dans les flux de média peuvent souvent être cachées ou tolérées. Il s'en suit que les mesures traditionnelles pour capturer la qualité des applications de données ne sont pas facilement applicables aux applications multimédia. Exigeant des volumes de calcul

lourds, il est traditionnellement difficile de soutenir des applications multimédia. Cette thèse traite du problème de concevoir des systèmes multimédia capables de réduire les besoins informatiques de telles applications. Les fondements de cette thèse sont fournis par le travail dans ce domaine que nous avons fait dans le contexte des normes internationales (MPEG), des projets de recherche européens (SOMMIT, OKAPI, MPEG-4 PC) et des projets industriels avec l'entreprise japonaise TDK. Dans le contexte de ces travaux, nous avons souvent fait face au problème de fournir des présentations multimédia complexes composées de plusieurs flux de médias sur différents terminaux. Par exemple, le Schéma 1 montre une application multimédia composée de 16 vidéos MPEG-4 fonctionnant en parallèle. En cliquant sur une vidéo, on propose un écran différent (le Schéma 2), composé de deux vidéos d'une meilleure qualité, plus le texte et les images.


Schéma 1: Flux multiples


Schéma 2: Vidéo, images immobiles et présentation de textes

En considérant ces applications, des questions peuvent se poser: comment la qualité d'une présentation est-elle quantitativement exprimée ? Comment est-ce que le terminal surveille et contrôle la qualité d'une présentation multimédia ? Cette thèse vise à fournir quelques réponses à ces questions, contribuant au champ de recherche qui traite des systèmes de présentation capables de s'adapter aux conditions variables de système sur la base d'une spécification de qualité de service (QoS). Les paragraphes suivants illustrent pourquoi ce sujet nécessite d'être étudié, et fournissent un rapport clair du problème et de l'approche technique adoptée.

**Motivations**

Nous étudions le problème du support des terminaux multimédia adaptatifs sur la base des faits suivants:

- La demande des applications multimédia de haute qualité augmente constamment.

Le succès incroyable des documents HTML a rapidement accéléré la demande des applications multimédia en temps réel. Dans le monde d'Internet nous sommes maintenant familiarisés avec les concepts de la

navigation de documents texte. Une évolution normale est le concept de la navigation de document multimédia où les contenus synthétiques et naturels sont mélangés pour fournir une expérience plus sophistiquée de navigation. Un concept similaire s'applique au domaine de télévision numérique où il y a une demande d'un plus grand niveau d'interactivité.

- ·Les dispositifs multimédia ont différentes ressources disponibles.

La puissance disponible (mémoire, capacité de traitement et largeur de bande de réseau) peut changer d'ordre de grandeur. Il devient obligatoire adapter le contenu des médias et les algorithmes de décodage et de rendu aux ressources spécifiques aux terminaux.

- ·Les réseaux informatiques évoluent.

Les réseaux à commutation de circuit laissent place aux technologies de commutation par paquets. Les réseaux à commutation de circuit ont été conçus pour réduire au minimum la fluctuation temporelle de commutation et de transmission. Des réseaux de commutation par paquets ont été différemment conçus pour maximiser l'utilisation de lien plutôt que pour réduire au minimum la fluctuation temporelle de transmission. La tâche importante de fournir une qualité acceptable de service est alors dévolue aux terminaux, au niveau applicatif, plutôt qu'au niveau protocolaire. L'intérêt pour la recherche dans ce secteur se retrouve dans le travail des groupes de standardisation dans le secteur du multimédia. Le groupe ISO/IEC/SC29/WG11, également connu sous le nom de MPEG, est en cours de normaliser un cadre d'étude (MPEG-21) pour la consommation de données multimédias où la question de l'adaptation de contenus multimédia est considérée à la fois au niveau du contenu, du terminal et du réseau.

**Problématique**

Le problème du support des applications multimédia est un problème stimulant en raison des faits suivants:

- Systèmes d'exploitation : les applications multimédia doivent fonctionner sur les système d'exploitation d'usage universel (GPOS). Les applications rentrent en concurrence pour l'usage des ressources informatiques et de mémoire.

- Besoins de calcul : les applications multimédia ont des besoins de calcul très variables. Dans les systèmes d'exploitation où les applications rentrent en concurrence pour l'usage des ressources, les ressources disponibles sont également variables. Par conséquent, faire s'accorder la variabilité des ressources disponibles avec la variabilité des besoins de calcul des applications multimédia est un problème intéressant.

- Synchronisation : la présentation synchronisée de différents médias est nécessaire.

- Médias compressés : afin d'être transmis efficacement, certains médias doivent être compressés. Du côté du terminal, ceci implique le besoin d'une tâche de décodage avant l'affichage de chaque unité de média.

- Graphique synthétique : les applications multimédia exigent la composition de médias continus avec du contenu synthétique.

- Interaction avec l'utilisateur : ces systèmes exigent des temps rapides de réponse et de commutation entre différentes options de visionnement.

Les terminaux multimédia fonctionnant sur les systèmes d'exploitation d'usage universel doivent par conséquent être robustes à l'imprévisibilité et aux variations du volume de calcul des données entrantes. Elles doivent surveiller constamment les progrès de l'application et réagir aux pertes de valeur provoquées par l'imprévisibilité fondamentale de l'environnement. En exécutant ceci, un système de présentation doit tenir compte de la valeur du service fourni à l'utilisateur. Le système de présentation exécute le processus d'adaptation sur la base d'une spécification de la qualité de service (QoS).

**Approche technique**

Nous formalisons un système multimédia comme un système centré sur l'utilisateur. Nous supposons que les utilisateurs fournissent la notion de valeur des différents médias d'une présentation à un terminal, suggérant comment ils prévoient le fonctionnement de la présentation lors d'événements imprévisibles. Ceci implique le besoin de spécifier la qualité au niveau de l'utilisateur. Nous avons proposé un cadre d'étude basé sur ce modèle. Notre contribution est basée sur les systèmes standard MPEG-4 et sa plateforme de logiciel de référence. Le raisonnement derrière ce choix est que nous pensons que la norme est un candidat parfait pour vérifier la valeur de ce travail, puisqu'elle a la puissance sémantique d'exprimer des applications multimédia composées de plusieurs flux de médias en temps réel. Étant une norme internationale, les spécifications sont publiques, et notre travail n'est alors pas attaché à une conception particulière et privée. Permettant une grande flexibilité en combinant différents types de médias, la norme présente quelques défis intéressants par rapport au problème que nous avons énoncé. En fait, le choix de la norme comme référence a également une influence sur la nouveauté de notre travail. La majeure partie des travaux existants dans le secteur étudié (détaillé dans la prochaine section) se concentre sur des terminaux manipulant des flux audio-visuels. Au lieu de cela, nous

considérons des scènes multimédia où les flux de description de scène sont plutôt complexes. Nous nous référons ici par exemple aux animations 2D (c.-à-d. dessins animés), applications vidéo multiples (c.-à-d. surveillance, événements sportifs à distance). Tout en étudiant le sujet, nous avons essayé de nous concentrer sur ce genre d'applications plutôt que sur les applications traditionnelles basées sur des flux vidéo et audio simples. Ces considérations ont motivé notre intérêt pour l'utilisation des systèmes MPEG-4. Une autre vue de l'approche technique que nous avons suivie peut être fournie si nous formulons un terminal multimédia comme composé de trois couches différentes : utilisateur, application, et réseau. On le conçoit bien dans la littérature où dans les modèles de couche multiples, la QoS doit être indiquée et imposée à toutes les couches [CCG93]. Ce travail se concentre sur l'utilisateur et les couches de niveau applicatif, et ne considère pas n'importe quel scénario particulier de service de réseau. Nous supposons que les données dont un terminal a besoin sont toujours disponibles une fois demandées. Bien que ceci puisse impliquer que ce travail est seulement directement utilisable pour les scénarios locaux (c.-à-d. lorsque les données entrantes sont stockées dans un ficher), nous croyons que notre travail peut être complété avec les travaux existants menés au niveau de la QoS du réseau (la section suivante mentionne des travaux sur ce sujet).

**Travaux associés**

Les études de la littérature sur la qualité de service concernent principalement le niveau réseau, avec des contributions pour mesurer le niveau de service dans une communication établie. Le besoin d'infrastructure de bout en bout pour la qualité du service dans les systèmes multimédia a d'abord été introduit dans les travaux de Campbell [CCG93], [CCH95], [CAH97]. Ces études proposent une architecture de système avec des mécanismes de qualité de service à chaque couche. Elles constituent une référence pour n'importe quel travail sur la QoS. De la même manière, [SCDS97] propose un cadre de

gestion de ressource pour la gérer les conflits avec des ressources système. Il propose une taxonomie de qualité des paramètres de service, composée de mesures (exécution, sécurité), et de politiques (de niveau de service, de gestion). Ce travail est intéressant parce qu'il généralise le problème et fournit une vision d'ensemble dans laquelle on pourra inscrire le problème moins général de trouver des mesures pour les systèmes multimédia. La majeure partie de la recherche dans le domaine de la QoS s'est produite dans le contexte de couches architecturales différentes (réseau et système). La QoS sur le réseau Internet est un domaine d'étude qui a attiré plusieurs chercheurs, proposant des perfectionnements au protocoles Internet afin de permettre un service scalable (diffserv), des protocoles pour demander des ressources spécifiques (RSVP), ou des mesures à employer par une application pour mesurer la qualité courante d'une communication. Dans le domaine des terminaux, nous avons trouvé des contributions dans les secteurs de la synchronisation, de la présentation et du transport. Des techniques de présentation qui préservent les relations temporelles sont rapportées dans [SDF93], et les problèmes de synchronisation d'horloge sont abordés dans [HPH 2000]. Des présentations basées sur la qualité sont présentées dans [BKWSAKG96]. L'article précèdent définit une mesure de qualité de présentation (QoP) et propose un ensemble de protocoles qui préservent la présentation de flux multimédia. La qualité subjective de la synchronisation dans une présentation multimédia a été évaluée exhaustivement par les travaux de Steinmetz [Stein96]. Il définit la qualité de synchronisation de médias en termes de biais entre les flux. Il a proposé divers seuils de biais d'horloge pour capturer la synchronisation entre l'audio, le visuel, le texte, et les images comme celle-ci serait perçue par l'utilisateur. Il a proposé également une méthode pour prolonger la propriété de synchronisation de deux objets à un troisième objet, connaissant les rapports entre chaque couple. Ce travail est très important pour nous, puisqu'il exprime la perception humaine de la synchronisation en valeurs quantitatives, qui peuvent être employées par des algorithmes au niveau applicatif. En ce qui concerne la littérature traitant des

standards MPEG, [RPS93] est un des premiers articles sur le décodage logiciel de la vidéo. [BMP98], [BA2000], et [MB96] fournissent des modèles pour évaluer les temps de décodage de la vidéo MPEG-2 et MPEG-4. [RKR96] décrit d'une manière claire comment la synchronisation est réalisée dans les flux MPEG-2 ; il a permis de comprendre l'utilisation des horloges dans le logiciel de référence MPEG-4. [PE2002] fournit fondamentalement toutes les notions sur MPEG-4 liées à ce travail de thèse. [N4848] est la référence pour les systèmes MPEG-4. [FS93] et [NRLD2002] décrivent des cadres d'étude 3D qui ont approché le même problème de soutenir l'adaptation multimédia mais pour les scènes 3D. Nous n'avons trouvé aucune contribution sur des algorithmes de présentation pour les terminaux 2D MPEG-4 qui améliorent la qualité des caractéristiques de service, ou les terminaux 2D capables de mélanger du contenu synthétique et naturel en considérant des mesures de qualité pour capturer les rapports entre les flux visuels et les graphiques synthétiques. Ce travail est censé fournir une contribution à ce domaine d'étude.

**Les contributions principales de cette thèse**

La contribution centrale de cette thèse est l'établissement d'un cadre d'étude multimédia pour la gestion de qualité des terminaux présentant des flux multiples sur des systèmes d'exploitation d'usage universel. Nous présentons des moyens de réfléchir au sujet de la perte de valeur des applications multimédia présentant des spécifications de qualité au niveau utilisateur, et nous identifions des mesures de niveau d'application pour les éléments d'un système multimédia. Nous fournissons des algorithmes pour améliorer l'exécution de composition de médias, un modèle pour estimer des tâches 2D et, comme deuxième contribution, des méthodes de composition pour intégrer ce modèle dans le cadre proposé de gestion de qualité. La troisième contribution est une étude sur la manière de réaliser le contrôle de charge sur une classe d'applications, les flux de données spécifiquement contraints au

temps qui exigent le décodage continu des mises à jour successives de la scène. Un modèle du temps de décodage est donné, ainsi que des prédicteurs pour estimer le décodage. La dernière contribution est le travail que nous avons fait dans le cadre de la participation au développement de la norme Mpeg-4, dans le contexte de l'activité du logiciel de référence (Mpeg-4 partie 5).

**Première contribution : cadre d'étude de QoS**

Parce qu'un terminal doit contrôler la valeur d'une présentation, il est nécessaire de spécifier la manière dont les différents composants d'une présentation contribuent à la valeur perçue par l'utilisateur. Lors de la définition des descripteurs de qualité, nous avons emprunté quelques concepts développés dans le domaine des programmateurs de système d'exploitation, où a été étudié l'utilisation des fonctions de valeur pour maximiser l'utilité du système. Par exemple dans [LRS98] et [JLDB 1995], les auteurs traitent de l'utilisation des dimensions discrètes de qualité pour maximiser l'attribution d'une tâche dans un système d'exploitation. Nous pensons que l'utilisation de ces concepts pour des applications multimédia est intéressante.

L'hypothèse est que l'utilisateur peut identifier et évaluer un certain nombre de dimensions souhaitables de qualité et leurs options de qualité associées. Puisqu'une scène multimédia se compose potentiellement de plusieurs flux, une spécification globale de qualité ne capturerait pas les besoins de l'application. Considérons par exemple la scène représentée sur le Schéma 3. Il montre une application de course de kart qui fournit différentes vues pour apprécier l'événement sportif.

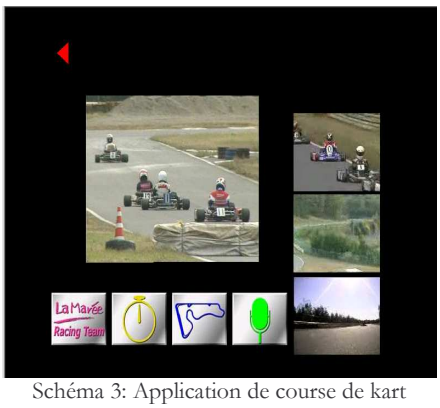

Schéma 3: Application de course de kart

Trois petites vidéos au format QCIF (176 * 144 pixels) (10 fps) fournissent différents points de vue de la course et une vidéo au format CIF (352*288 pixels) (30 fps) fournit une vue de meilleure qualité. L'auteur peut certainement identifier un ensemble de mesures de qualité qui peuvent exprimer les concepts de performance (fréquence d'affichage d'image), de taille, ou de perception (filtre de post-traitement) pour la présentation de cet événement. Clairement, si ces options sont déclarées pour chaque flux, le terminal de rendu peut allouer les ressources à une granularité plus fine, optimisant l'utilité du système. Par exemple, supposons que pendant le playback de l'application de course de kart, la fréquence d'affichage d'image de la vidéo CIF diminue soudainement, parce que l'unité centrale est surchargée. La vidéo CIF n'est probablement pas le meilleur candidat pour la dégradation de qualité, parce que c'est le point de vue le plus important pour l'utilisateur. Au lieu de cela, les trois vidéos QCIF pourraient être dégradées (réduisant la fréquence d'affichage d'image ou remplaçant les vidéos par un texte donné), et les ressources libérées pourraient être employées pour montrer la vidéo CIF à pleine qualité. Afin de réaliser cela, des options de qualité devraient être déclarées par flux, et chaque flux devrait donner une notion d'importance relativement aux autres flux. Puisque des dimensions multiples de qualité peuvent être associées à chaque flux, il y a également un besoin d'évaluer chaque dimension, afin de choisir parmi des options

possibles tenant compte de la valeur que chaque utilisateur assigne à chaque dimension.

Schéma 4: Catalogue vidéo

Un concept semblable s'applique aux graphiques 2D. Le Schéma 4 montre une application visuelle de catalogue. Six petites vidéos QCIF défilent de droite à gauche dans la partie la plus inférieure de l'écran. Lorsque l'utilisateur clique sur une vidéo, une plus grande vue est montrée dans la partie supérieure de l'écran. Le petit défilement visuel est réalisé par l'intermédiaire d'une construction BIFS appelée *time sensor*. Les dimensions de qualité de BIFS peuvent inclure des tailles d'affichage et des fréquences d'affichage d'image pour l'animation. Si l'auteur assigne une valeur à chaque dimension, lorsque le système de présentation est dans le besoin d'assigner des ressources à chaque flux, il peut alors considérer les besoins de l'auteur. Les exemples précédents suggèrent la structure des spécifications de QoS au niveau de l'utilisateur : chaque flux devrait donner une liste de dimensions de QoS, et leur contribution à la satisfaction de l'utilisateur. L'importance relative de chaque flux devrait également être évidente dans la description. Selon ce mécanisme, des spécifications au niveau utilisateur de l'application représentée sur le Schéma 4 pourraient être :

Flux 1: (Video CIF)

- Perceptual Visual Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 3

- Frame Rate {25, 12, 5} Dimension Value: 2
- Frame Size {CIF, QCIF} Dimension Value: 1
- Stream Value {3}

Flux 2, 3, 4: (Video QCIF)

- Perceptual Visual Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 2
- Frame Rate {10, 5, 2} Dimension Value: 1
- Frame Size {QCIF} Dimension Value: 3
- Stream Value {2}

Flux 5: (Audio)

- Perceptual Audio Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 1
- Audio Sampling Rate {44100, 22050} Dimension Value: 2
- Stream Value {4}

Flux 6: (BIFS)

- Perceptual Visual Quality {HIGH_QUALITY, LOW_QUALITY} Dimension Value: 1
- Frame Rate {10, 5, 1} Dimension Value: 2
- Frame Size {100, 50, 10} Dimension Value: 3
- Stream Value {1}

Dans l'exemple ci-dessus, une spécification au niveau utilisateur est associée à chaque flux de l'application. Elle se compose d'une liste de dimensions de qualité pour chaque flux. Dans l'exemple, les dimensions de qualité capturent l'opportunité, la taille et les qualités perceptuelles, mais la liste n'est pas censée être complétée. Chaque dimension de qualité contient une liste d'options de qualité, dans l'ordre descendant de qualité. Chaque dimension a une valeur associée, qui établit un rang parmi les dimensions dans la satisfaction de l'utilisateur. Par exemple dans le flux 1 l'utilisateur évalue la qualité visuelle perceptuelle comme plus haute qualité à préserver, suivie de la fréquence

d'affichage d'image et finalement de la taille d'image. Ceci signifie que la première action qu'un terminal considérerait pour dégrader la qualité de la scène est une réduction de la fréquence d'affichage d'image. Chaque flux a une valeur associée de flux. Ceci établit un rang parmi les flux d'une présentation. Les flux avec de plus petites valeurs sont les premiers à être dégradés. Nous avons assigné la valeur la plus élevée à l'audio, puis à la vidéo CIF. C'est parce que l'audio est le média qui est le plus sensible aux dégradations, où même des réductions minimes de qualité ont un impact fort sur la satisfaction de l'utilisateur. L'hypothèse sur laquelle ces spécifications sont basées est que l'utilisateur peut identifier quelques qualités pour chaque flux et les évaluer. Nous pouvons nous demander s'il est toujours possible d'identifier un ensemble de dimensions de qualité. Nous croyons que pour la plupart des scènes la réponse est oui. Les concepts d'opportunité, quantité de données, qualité perceptuelle, sont bien applicables à la nature des applications multimédia. Ce que nous pouvons dire est qu'il n'est pas toujours possible d'avoir tous les descripteurs pour chaque flux. Par exemple les concepts de fréquence d'affichage d'image facultative pour la vidéo ne peuvent pas être appliqués à toutes les vidéos encodées. Comme rapporté dans [IFS03], il n'y a parfois pas pour le flux de données entier de fréquence d'affichage d'image fixe, ni un groupe constant de structure d'images (modèle d'images I, P, B), qui permettra par exemple au décodeur de changer la fréquence d'affichage d'image en sautant quelques données. Ainsi nous prévoyons qu'il n'est pas possible que tous les flux aient un ensemble fixé de descripteurs. Mais ceci est alors relié à la signification interne des spécifications de QoS au niveau de l'utilisateur : il faut fournir des conseils au sujet du nombre de dégradations de dimensions de qualité qui peuvent être projetées et exécutées sans risque.

*L'architecture du cadre d'étude*

L'architecture que nous proposons est une extension de l'architecture MPEG-4. Les extensions d'architecture sont nécessaires pour supporter le contrôle du

playback d'une scène multimédia selon les spécifications de QoS fournies par l'utilisateur. Dans la phase de conception, nous avons emprunté des principes de conception étudiés dans la théorie de contrôle pour les systèmes d'événement discrets [KBE99], où les problèmes de la *contrôlabilité*, *robustesse* et la *configuration réactive* ont été identifiés. La contrôlabilité est la capacité d'un système à être exécuté lorsque la charge d'entrée change dynamiquement (entre les limites indiquées). Ceci particulièrement important pour un terminal multimédia dont la charge de données peut varier. La robustesse traite des variations imprévues de l'environnement d'un système. Pendant que l'environnement sort du cadre prévu, le système dégrade graduellement son exécution plutôt que de montrer un échec catastrophique. Ce concept est approprié pour des terminaux fonctionnant dans un environnement non contrôlé (c.-à-d. système d'exploitation d'usage universel) où les applications entrent en concurrence pour des ressources système. La configuration réactive indique les possibilités d'un système de modifier ses algorithmes principaux en mesurant leurs besoins informatiques, et est déclenché par une évaluation indiquant que le système n'accomplit pas sa mission. La contrôlabilité et la robustesse ont été traitées par un modèle de conception appelé le contrôle distribué. La configuration réactive a été traitée dans le contexte du processus de gestion de qualité. Dans le paragraphe suivant nous illustrons l'architecture que nous proposons pour la gestion distribuée de contrôle de qualité.

Afin d'adresser la robustesse et la contrôlabilité, la charge d'un système doit être constamment surveillée, des variations de la charge doivent être identifiées, et une reconfiguration des algorithmes des terminaux peut par la suite avoir lieu dans le contexte d'un processus de gestion de qualité. Un problème important lorsque l'on traite de la charge d'un système multimédia est que celle-ci change sur plusieurs échelles. Plus précisément :

- les changements de données des médias continus interviennent habituellement à de échelles de l'ordre de la dizaine de millisecondes (c.-à-d. une complexité temps-variable de MPEG [BMP98])

- les changements de scène ont lieu habituellement à des échelles de l'ordre de la seconde

- les changements d'utilisateur ont des échelles de temps de l'ordre de la minute (c.-à-d. l'interaction de l'utilisateur peut déclencher des événements qui causent des variations de charge)

- les changements d'environnement ont des échelles de temps de l'ordre de la centaine de millisecondes (c.-à-d. des variations de charge dues à d'autres applications fonctionnant sur le même OS)

Afin de faire face à ces conditions, une architecture de contrôle hiérarchique a été conçue. Dans la théorie des systèmes de contrôle hiérarchiques [ZU97], les unités de contrôle séparées effectuent des observations indépendantes et ont leurs propres variables. Le contrôle est réalisé par l'intermédiaire de la collaboration entre les unités indépendantes. Une unité de contrôle est constituée d'un processus d'observation qui rassemble des informations sur le système commandé et son environnement, et un processus de décision qui emploie cette information et n'importe qu'elle information a priori pour effectuer la contrôle.

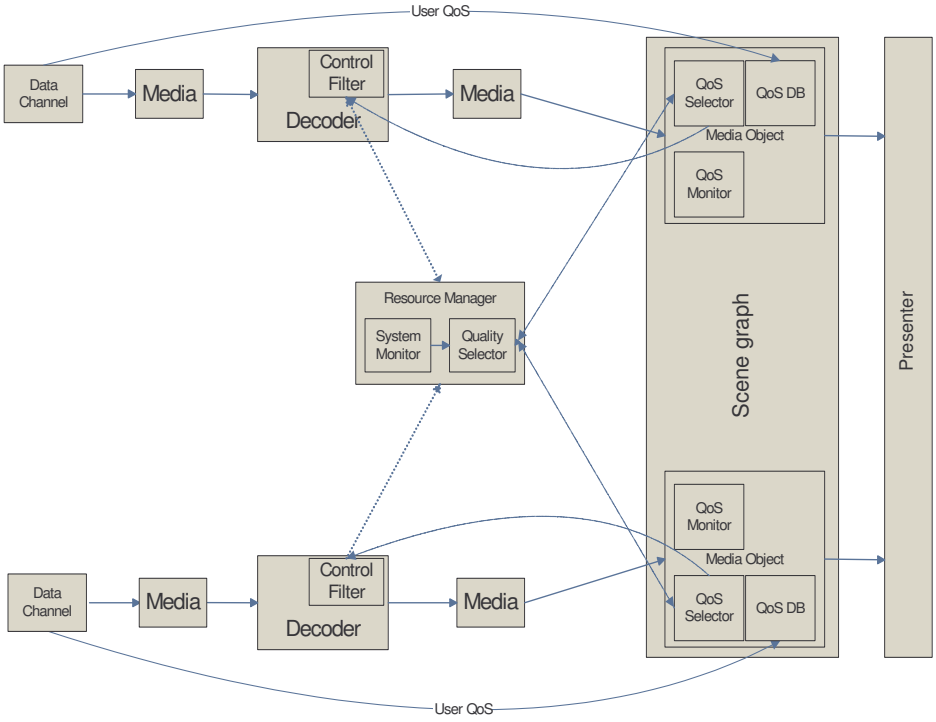


Schéma 5: Architecture

L'architecture est une extension de l'architecture de référence de MPEG-4, puisque les concepts principaux d'avoir des pipelines de décodage séparées qui fonctionnent en processus séparés, un graphe de la scène avec des objets médias qui sont partagés par la composition et le décodage, et un processus de présentation qui exécute le rendu des noeuds du graphe de la scène, sont également présents dans l'architecture proposée. Nous avons traité ici du contrôle à différentes échelles de temps, réalisé avec la collaboration des différentes unités de contrôle situées dans les décodeurs et les noeuds de médias, et un nouveau composant appelé le manageur de ressource (RM). Les unités de contrôle des décodeurs observent les variations dynamiques de charge des médias continus et le progrès du processus de décodage. Les unités

de contrôle contenues dans des noeuds de medias observent la continuité de l'affichage des médias associés. Les unités de contrôle encapsulées dans le graphe de la scène contrôlent les changements dans la scène (causés par des mises à jour ou par l'interaction de l'utilisateur). L'élément de contrôle de système du manageur de ressource observe des changements d'environnement. L'observation effectuée par des unités de contrôle peut mener à des procédés de décision indépendants, ou à des avis au manageur de ressource qui commencera par la suite un nouveau processus de tâche de qualité (dans ce sens nous avons conçu une hiérarchie dans le contrôle). Le besoin d'entité séparée est justifié par le fait que nous devons centraliser l'analyse du manque de QoS et agir sur la base d'une vue globale.

*Conclusions*

Nous avons établi un cadre d'étude pour des présentations multimédia basées sur la qualité. Le terminal proposé exécute l'attribution et la gestion de ressource, en fonction de la satisfaction de l'utilisateur et des ressources de système disponibles. La satisfaction de l'utilisateur est traitée en spécifiant une liste d'options de qualité pour chaque dimension de qualité de médias, et en installant un rang parmi ces dimensions. La surveillance de l'exécution est effectuée en utilisant les mesures appropriées au niveau de l'application. La qualité d'une application multimédia est définie en termes d'observations quantitatives de ses médias constitutifs. Au lieu de pâtir du manque de ressources d'une manière non contrôlée, le terminal proposé vérifie régulièrement l'exécution de ses flux en comparaison des tolérances indiquées, et réagit aux violations par des actions précises sur chacun des médias. Ceci est réalisé en surveillant des possibilités à chaque noeud de médias, et en fournissant une entité additionnelle qui centralise l'analyse des violations et lance des actions de rétablissement. Les résultats montrent comment le système proposé remplit les conditions typiques des applications fonctionnant sur les logiciels d'exploitation d'usage universel : contrôle d'admission, concurrence entre les applications et gestion de puissance.

**Deuxième contribution : intégration de contenu 2D**

Dans cette contribution nous présentons un modèle pour estimer les ressources des tâches de rendu 2D, et fournissons des exemples d'utilisation de ce modèle dans le cadre d'étude que nous avons proposé dans la première contribution. Ce travail se concentre sur le matériel du consommateur, à savoir un PC avec une carte graphique dernier cri. Notons que ce sont des systèmes dits « best effort », et que les logiciels graphiques ne fournissent aucun outil pour donner un délai strict d'exécution d'un ensemble donné de commandes. Ce qui peut être réalisé est une évaluation des ressources requises, qui peut être employée par un algorithme de gestion de qualité. Une partie importante de cette contribution est constituée par une analyse du rendu des scènes 2D décrites en utilisant la norme MPEG-4 BIFS, limité au profil Core2D. Puisque la norme n'exige pas d'algorithmes particuliers pour exécuter le rendu, une partie de cette contribution propose et discute quelques techniques pour exécuter le rendu. Ceci correspond aux objectifs de cette contribution, et fournit également une contribution à la littérature dans le domaine de la composition des scènes MPEG-4 2D. Une hypothèse valide dans tout ce travail est que le terminal emploie une bibliothèque de rendu 2D. Par conséquent, aucune accélération matérielle n'est employée dans le rendu des primitives graphiques. Des contributions semblables dans le domaine des moteurs graphiques 3D peuvent être trouvées dans [FS93], [LB1998], et [WW2003]. Ces contributions sont fondées sur différentes hypothèses parce qu'elles impliquent qu'une partie des contrôles de rendu soit exécutée par des GPU d'accélération matérielle. Par conséquent, leurs modèles ne peuvent pas être employés pour des systèmes basés seulement sur les bibliothèques de rendu 2D.

La méthodologie que nous avons suivie est de subdiviser le processus de rendu en un certain nombre de tâches conceptuellement indépendantes qui

peuvent être estimées séparément. La subdivision que nous avons considérée est la suivante:

- Parcours du graphe de scène et construction de la liste d'affichage
- Détermination et invalidation des objets à dessiner
- Peinture des objets
- Copie des objets dessinés à l'écran

Cette subdivision est fondée sur l'hypothèse que les objets multimédia sont stockés dans une structure de données (graphe de scène), et qu'une double technique de buffering est employée pour l'affichage. Ces deux hypothèses sont généralement valides dans la plupart des moteurs de rendu 2D. Le mot « composition » est souvent employé en littérature. Dans cette subdivision, la composition est composée de l'algorithme de Painter et de la phase de Rasterization. Nous observons que, étant donné un graphe de scène, il n'est pas possible de connaître a priori le temps exigé par l'étape de rendu, à moins que nous supposions que tous les objets visuels doivent être dessinés à chaque itération (rendu direct). C'est parce que l'algorithme de Painter essaye de réduire au minimum le nombre d'objets dessinés, et par conséquent nous ne pouvons pas savoir à l'avance combien de temps le « rasterization » prendra. Cependant, après chaque parcours du graphe de la scène, en exécutant l'algorithme de Painter sur la liste d'affichage nous connaissons combien d'objets seront dessinés ou partiellement dessinés, et nous pouvons estimer le temps de rasterization. En d'autres termes, nous observons que les étapes séparées peuvent être estimées bien avant leur exécution, en utilisant les paramètres que nous avons identifiés. Plus précisément :

- Parcours du graphe de la scène : nombre des objects et taille de l'arbre

- Algorithme de Painter : taille de la liste d'affichage

- Rasterization : type, dimension et propriétés de chaque objet à dessiner

- Mise à jour d'affichage : taille de la fenêtre

Dans le paragraphe suivant nous verrons comment employer le résultat de cette analyse dans le contexte du cadre de gestion de qualité.

L'intégration entre la vidéo et le BIFS soulève quelques problèmes. Les vidéos sont flux continus, ils sont associés aux noeuds dans le graphe de la scène, et ils doivent être présentés à des fréquences d'affichage d'images élevées (c.-à-d. 25 ou 30 hertz). Le graphique 2D est déclaré dans le graphe de la scène, et n'exige pas habituellement des fréquences d'affichage élevées. Le parcours du graphe de scène et la composition pourraient être des tâches lourdes, principalement suivant le nombre de objets. L'algorithme de présentation du logiciel de référence de MPEG-4 accomplit les tâches du parcours et de la composition à une fréquence d'affichage régulière, égale à la fréquence requise par les flux visuels. Cette solution n'intègre pas bien la vidéo et le graphique 2D, parce que le parcours et la composition de scène sont effectués à des fréquences d'affichage élevées et la différence de fréquences de BIFS et des flux visuels n'est pas prise en considération. Un mécanisme pour découpler la vidéo et le rendu d'objets BIFS est nécessaire. Des fréquences de rendu séparées peuvent être réalisées en stockant les noeuds visuels actifs dans une liste séparée du graphe de la scène. La liste peut être mise à jour à chaque parcours de l'arbre de scène. Un parcours complet est exécuté au début de la simulation, puis à la fréquence d'affichage nominale de BIFS, si des descripteurs de fréquence sont fournis, ou toutes les fois que le graphique doit être mis à jour si des descripteurs de fréquence ne sont pas fournis. De cette façon, nous pouvons assurer le rendu des noeuds visuels sans parcourir le graphe de la scène. L'interaction avec l'utilisateur est alors prise en

considération en permettant des parcours complets toutes les fois qu'un nouvel événement invalide le graphe de scène. Nous observons que cet algorithme amélioré de présentation ne résout pas complètement le problème de garantir des fréquences indépendantes. Même si les fréquences d'affichage séparées sont mises en application, les parcours complets et la composition de graphique peuvent influencer le traitement régulier d'autres données continues. Par exemple, nous avons créé une scène simple d'essai composant un flux de BIFS et un flux visuel, chacun avec des fréquences d'affichage différentes (voir le Schéma 6).

Schéma 6: BIFS et Vidéo à 10 fps et à 25 fps

VIDEO:

QoS_Qualifier_FRAME_RATES{FRAME_RATES [25] dimensionValue 1}

QoS_Qualifier_MAX_RATE_VARIATION{MAX_RATE_VARIATION 0.1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 2}

BIFS:

QoS_Qualifier_FRAME_RATES{FRAME_RATES [10,5,1,0] dimensionValue 1}

QoS_Qualifier_STREAM_VALUE {STREAM_VALUE 1}

Nous notons que les fréquences d'affichage sont différentes, à savoir 25 fps pour la vidéo et 10 fps pour le BIFS, et que l'utilisateur assigne plus de valeur au flux visuel. En outre, il est disposé à accepter des variations des fréquences d'affichage du flux BIFS, mais il ne peut pas accepter des variations des fréquences d'affichage du flux visuel (une tolérance est fournie sous forme de descripteur de variation de fréquence d'affichage maximum). Au début, la scène BIFS est vide. Ensuite, 5 objets géométriques sont ajoutés chaque 100 millisecondes. La complexité initiale est faible puisque le graphe de la scène est tout à fait petit. Cependant, après 8 secondes, le graphe de la scène contient 400 formes, et environ 1300 noeuds. A chaque fois qu'un parcours complet est effectué, environ 40 millisecondes sont nécessaires. Les Schémas 7 et 8 affichent l'utilisation du CPU et les fréquences d'affichage de la vidéo et du BIFS pendant les 9 premières secondes de la simulation.

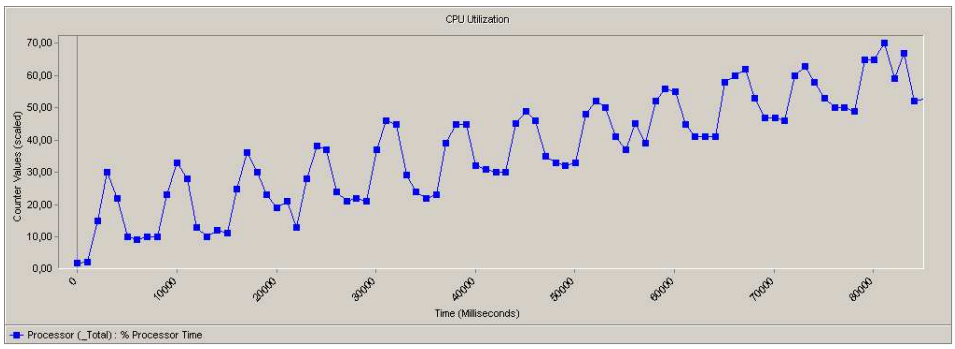


Schéma 7: utilisation CPU pour la scène avec BIFS et Vidéo

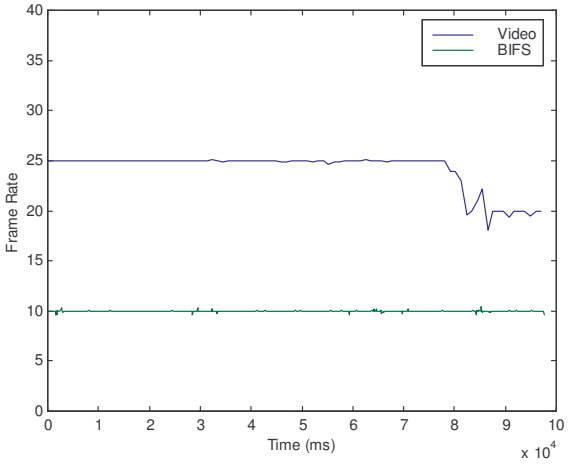


Schéma 8: fréquences d'affichage de la vidéo
et du BIFS

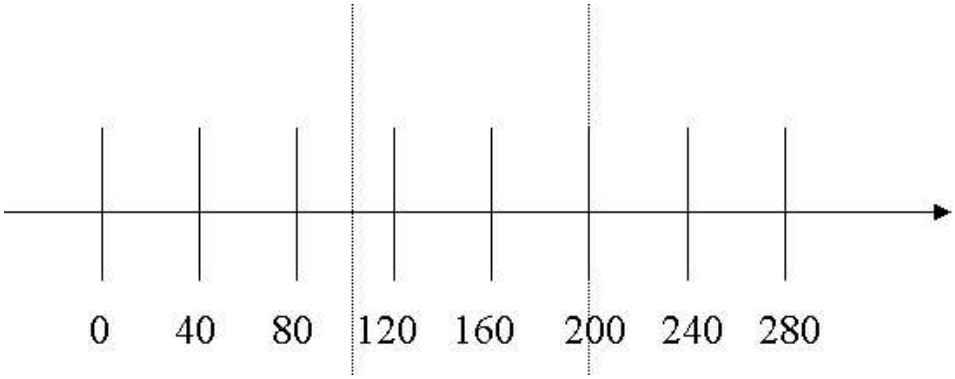


Schéma 9: Limite pour la vidéo (lignes solides)
et BIFS (lignes pointillées)

Nous notons que le CPU n'est jamais surchargée. Ceci signifie que le manageur de ressource (RM) et les composants d'estimation du BIFS et de la vidéo réussissent à contrôler l'utilisation du CPU. Le Schéma 8 prouve que la QoS visuelle peut-être affectée, alors que le flux BIFS est toujours à sa qualité maximale, ce qui est à l'opposé de ce que les spécifications de qualité de la scène exigent.

Nous observons qu'en traitant les graphiques et la vidéo, le cadre de gestion de qualité proposé ne peut pas être seulement basé sur des violations et sur la réattribution de la charge disponible d'unité centrale. D'autres paramètres doivent être considérés pour l'ordonnancement approprié des tâches comme le parcours et la composition. L'exécution de ces tâches peut mener à des dégradations de qualité même si le CPU n'est pas surchargé. En utilisant l'évaluation du parcours, avant de parcourir réellement le graphe de la scène nous pouvons prévoir si le parcours est susceptible de dépasser un temps limite, et remettre le parcours à plus tard ou bien le passer, avec pour but de maintenir la fréquence d'affichage visuelle intacte. Dans le graphique du Schéma 9, nous pouvons reprogrammer le parcours du temps 100 à plus tard au temps 120, ajoutant 20 millisecondes de gigue dans la présentation du flux BIFS programmé au temps 100. Nous observons que ceci dégradera la présentation des éléments graphiques, mais ce comportement est logique avec les spécifications de qualité du flux BIFS (la fréquence d'affichage peut même chuter à 0). Les Schémas 10 et 11 illustrent la fréquence d'affichage et le CPU de la présentation. Nous observons que la fréquence d'affichage du BIFS chute à 0 même si le CPU n'est jamais surchargé. La fréquence d'affichage d'image de la vidéo est quant à elle constante.

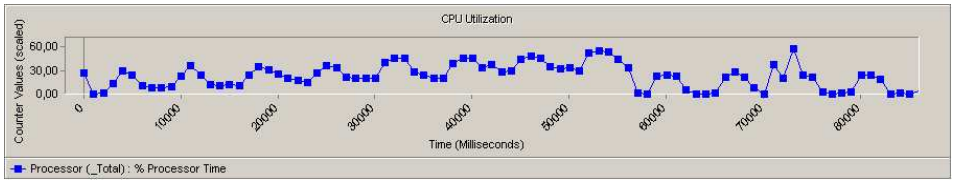


Schéma 10: utilisation CPU

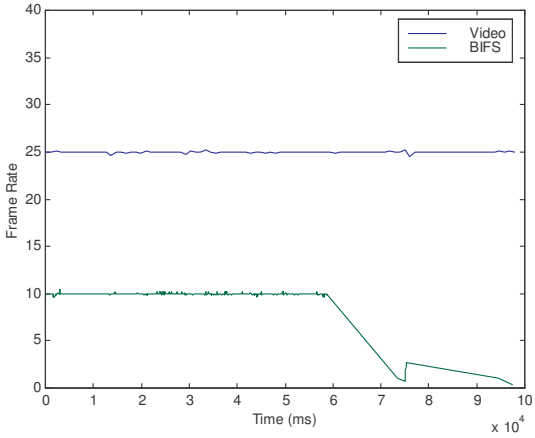


Schéma 11: fréquences d'affichage vidéo et BIFS

**Troisième contribution : prédiction de décodage du BIFS**

Dans l'architecture proposée dans la première contribution, une unité de filtrage contrôle l'exécution des décodeurs associés aux médias. Dans cette contribution, nous cherchons à fournir la preuve qu'un mécanisme de contrôle est nécessaire également pour le décodage du flux de description de scène (BIFS), et nous discutons de la manière pour réaliser cela. Habituellement, la complexité de décodage des flux de description de scène ne pénalise pas l'exécution. Dans la plupart des applications multimédia, le terminal reçoit la scène, la décode et puis commence la simulation. Le décodage de la scène est par conséquent fait seulement une fois, et alors que la simulation n'a même pas commencé. Cependant, un certain type d'applications multimédia exige le décodage continu des mises à jour de scène pendant la simulation, et en conséquence la mise à jour continue et le rendu du graphe de la scène. Nous avons considéré un ensemble d'applications BIFS caractérisées par des mises à jour continues de scène, et nous avons mesuré les temps de décodage du flux BIFS. Le temps décodage de chaque unité a été observé en utilisant un chronomètre à haute résolution. Nous avons alors effectué une régression modélisant le temps de décodage en fonction de la taille de l' AU mesurée en octets (Schéma 12).

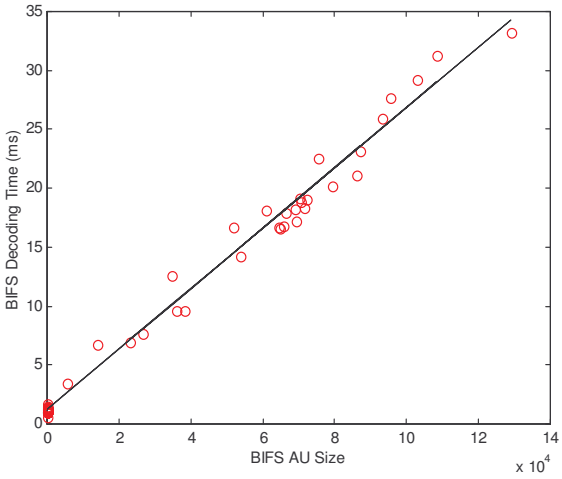


Schéma 12: temps de décodage BIFS en fonction de la taille de l'AU

Cette étude montre clairement qu'il y a une corrélation forte entre la taille d'une unité d'accès (AU) et son temps de décodage. Dans cette thèse nous comparons des prédicteurs basés sur cette observation. Cette étude prouve qu'il est possible de prévoir des temps de décodage du BIFS, en utilisant un modèle linéaire comme fonction de la taille de l'unité d'accès BIFS. Elle prouve également qu'il est possible de mettre en application des prédicteurs moins précis mais efficaces qui peuvent prévoir des temps de décodage de moins de 3 et 5 millisecondes avec un taux de succès changeant de 90 à 100 % des échantillons.

**Conclusions**

Dans cette thèse nous avons étudié le problème de supporter des applications multimédia basées sur la qualité. La méthodologie adoptée dans ce travail a consisté à modéliser un terminal multimédia comme un système centré sur l'utilisateur, les utilisateurs suggérant comment ils s'attendent à ce que l'application fonctionne sous des événements imprévisibles. Nous avons identifié des paramètres de QoS pour définir la qualité d'une présentation, puis avons proposé un cadre d'étude multimédia basé sur ces spécifications de

qualité. Ensuite, nous avons identifié des procédés de contrôle pour les composants du système principal de ce cadre d'étude. Cette contribution a été basée sur le système du standard MPEG-4 et sa plateforme de logiciel de référence. Cependant, nous pensons que la plupart des concepts exprimés dans ce travail peuvent être aussi bien employées dans d'autres contextes. Le système MPEG-4 fournit un cadre intégré et normalisé de support des graphiques 2D et de la vidéo, et notre contribution a progressé en considérant différents aspects de la norme. La contribution de ce travail au développement de la norme a donné une homogénéité à cette thèse, en améliorent la revalorisation des résultats.

# BIBLIOGRAPHY

[AD2002] Anthony Vetro and Sylvain Devillers, Position Paper: Delivery Context in MPEG-21, W3C Delivery Context Workshop, March 2002.

[BA2000] Lars-Olof Burchard, Peter Altenbernd, Estimating Decoding Times of MPEG-2 Video Streams, Image Processing, 2000. Proceedings. 2000 International Conference on, Volume: 3, 2000

[BKWSAKG96] Shabab Baqai, M. Farrukh Khan, Miae Woo, Seiichi Shinkai, Ashfaq A. Khokhar, Arif Ghafoor, Quality-Based Evaluation of Multimedia Synchronization Protocols for Distributed Multimedia Information Systems. IEEE Journal on Selected Areas in Communications 14(7): 1388-1403 (1996)

[BL2002] Gianluca Bontempi, Gauthier Lafruit, Enabling Multimedia QoS Control with Black-box Modelling. SoftWare 2002: Computing in an Imperfect World, Lecture Notes in Computer Science

[BMP98] Andy C Bavier, A. Brady Montz, Larry Peterson, Predicting MPEG Execution Times, SIGMETRICS '98. Proceedings of the joint international conference on measurement and modelling of computer systems, 1998

[BRE77] J.E. Bresenham, A linear algorithm for incremental digital display of circular arcs, Communications of the ACM, 20(2): 100-106, 1977

[BS96] Gerald Blakowski and Ralf Steinmetz, "A Media Synchronization Survey: Reference Model, Specification, and Case Studies", IEEE JSAC Vol 14, No. 1, Jan. 1996

[CAH96] Andrew Campbell, Cristina Aurrecoechea, Linda Hauw, A Review of QoS Architectures, Proceedings of the 4th International Workshop on Quality of Service, Paris, March 1996

[CCG92] A. Campbell, G. Coulson, F. Garcia, D. Hutchinson, and H. Leopold, Integrated Quality of Service for Multimedia Communications, In COMM '92, pages 99-110, 1992.

[CCH95] A. Campbell, G. Coulson and D. Hutchinson, Supporting Adaptive flows in a quality of service architecture, Multimedia Systems Journal, November 1995

[D21] Definition of Quality of Service, Deliverable D.21, Sequin Project.

[FS93] Thomas A. Funkhouser , Carlo H. Séquin, Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments, Proceedings of the 20th annual conference on Computer graphics and interactive techniques, p.247-254, September 1993

[HPH 2000] Orion Hodson, Colin Perkins and Vicky Hardman, Skew detection and compensation for Internet audio applications, Proceedings of the IEEE International Conference on Multimedia and Expo, New York, July 2000.

[HYP2003] www.hyperdictionary.com

[IT95] Y. Ishibashi, S. Tasaka, A synchronization mechanism for continuous media in multimedia communications. Fourteenth Annual Joint Conference of the IEEE Computer and Communication Societies, April 02 - 06, 1995, Boston, Massachusetts

[IFS03] Damir Isoviá, Gerhard Fohler, Liesbeth Steffens : Timing Constraints of MPEG-2 Decoding for High Quality Video: Misconceptions and Realistic Assumptions, 15th Euromicro Conference on Real-Time Systems (ECRTS'03) July 02 - 04, 2003 Porto, Portugal

[ITI2000] Y. Ishibashi, S. Tasaka, T. Iwama, Adaptive QoS Control for Video and Voice Traffic in Networked Virtual Environment, T. Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on, 2000

[IT2000] Y. Ishibashi, S. Tasaka, A comparative survey of synchronization algorithms for continuous media in network environments, 25th Annual IEEE Conference on Local Computer Networks, November 08-10, 2000, Tampa, Florida

[JKWAJ91] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C. Yu, J.Y.Chung, and W. Zhao, "Algorithms for scheduling imprecise computations," IEEE Computer, vol.24, no.5, pp.58-68, May 1991.

[JLDB 1995] M. Jones, P. Leach, R. Draves, and J. Barrera. Modular Real-Time Resource Management in the Rialto Operating System. In Proceedings of the Fifth Workshop on Hot Topics in Operating Systems, May 1995

[LB1998] Gauthier Lafruit, Jan Bormans, The need for Computational Graceful Degradation in SNHC. ISO/IEC JTC1/SC29/WG11/MPEG97/ M3009, San Jose, February 1998

[LCZ2001] C. J. Lan, Y. Chen, Z. Zhong. Mpeg2 decoding complexity regulation for a media processor. In Proceedings of the 4th IEEE Workshop on Multimedia Signal Processing (MMSP), Cannes, France, pp. 193 - 198, October 2001

[LRS99] Chen Lee, John Lehoczky, Ragunathan (Raj) Rajkumar, Dan Siewiorek, On Quality of Service Optimization with Discrete QoS Options

[LS98] C. Lee and D. Siewiorek. An Approach for Quality of Service Management. Technical Report CMU-CS-98-165, Computer Science Department, CMU, Oct. 1998

[KBE99] M.M. Kokar, K. Baclawski and Y.A. Eracar. Control Theory-Based Foundations of Self-Controlling Software. IEEE Intelligent Systems, pp. 37-45, Vol. 14, No. 3, May/June 1999

[Macro2003] www.macromedia.com

[MB96] M. Mattavelli, S. Brunetton, "Controlling decoding power by graceful degradation techniques," ISO/IEC JTC1/SC29/WG11/MPEG97/M1445, Maceio, November 1996

[MB97] M. Mattavelli, S. Brunetton, "Results of core experiment on CGD: methods to measure bitstream video decoding complexity for CGD implementation" ISO/IEC JTC1/SC29/WG11/MPEG97/M1445 Stockholm, July 1997

[MCD2001] J.C. Moissinac, C. Concolato, J.C. Dufourd, "Codage MPEG-4 de dessins animés", Journées Coresa, 12-13 novembre 2001

[MPE99] Contribution MPEG99/4460 IM-1 2D FDIS Player

[N271] MPEG Convener. New York Item Proposal (NP) for Very-Low-Bitrates Audiovisual Coding. Doc ISO/MPEG N271, London MPEG Meeting, November 1992

[N4848] ISO/IEC 14496-1: 2002 MPEG-4 3rd edition

[NRLD2002] N. Pham Ngoc, W. Van Raemdonck, G. Lafruit, G. Deconinck, and R. Lauwereins, A QoS Framework for Interactive 3D Applications, Proc. 10th International Conference on Computer Graphics and Visualization'2002, WSCG'2002, February 4-8, 2002.

[OP] Open Inventor (http://oss.sgi.com/projects/inventor)

[ITI2000] Yutaka Ishibashi, Shujii Tasaka, Tomohiro Iwana, Adaptive QoS control for voice and video traffic in networked virtual environments, Computer Communications and Networks, Proceedings. Ninth International Conference 2000

[PE2002] The MPEG-4 Book, edited by Fernando Pereira and Touradj Ebrahimi, Prentice Hall IMSC Multimedia Series 2002

[PS2001] S.Peng, Complexity Scalable Video Decoding Via IDCT Data Pruning, ICCE 2001

[R96] Thomas P. Ryan, Modern Regression Methods, Wiley Series in Probability and Statistics. Applied Probability and Statistics 1996

[RB 1993] K. Ravindran, V.Bansal, Delay Compensation Protocols for Synchronization of Multimedia Data Streams IEEE Transactions on Knowledge and Data Engineering, August 1993.

[RBS2000] John Regehr, Michael B. Jones, John A. Stankovic, Operating System Support for Multimedia: The Programming Model Matters, Technical Report MSR-TR-2000-89, September 2000

[RKR96] P. V. Rangan, S. S. Kumar and S. Rajan, Continuity and Synchronization in MPEG, IEEE Journal on Selected Areas in Communications, Special Issue on Multimedia Synchronization, 1996

[RPS93] Rowe, L.A., Patel, K., and Smith, B.C., Performance of a Software MPEG Video Decoder, Proceedings ACM Multimedia 93, pp. 75-82

[RS1996] Ralf Steinmetz, Human perception of jitter and media synchronization, IEEE Journal on Selected Areas in Communication Vol. 14 NO. 1, January 1996

[SDF93] H. Santoso, L. Dairaine, S. Fdida, E. Horlait Preserving temporal Signature: A Way to Convey Time Constrained Flows, Globecom 94, Houston, USA, November 29-December 2, 1993

[Stein96] Ralf Steinmetz: Human Perception of Jitter and Media Synchronization. IEEE Journal on Selected Areas in Communications 1996

[SCDS97] Sabata, B., Chatterjee, S., Davis, M., Sydir, J. J., and Lawrence, T., Taxonomy for QoS Specifications, In Proc. of the Third Annual Workshop on Object- Oriented Dependable Systems, Feb. 5 - 7, 1997

[SIG99] Signes, J., Binary Format for Scene (BIFS): Combining MPEG-4 media to build rich multimedia services Proc. SPIE Vol. 3653, p. 1506-1517, Visual Communications and Image Processing '99, 1999

[VFJF2001] Bobby Vandalore, Wu-chi Feng, Raj Jain, and Sonia Fahmy, A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia, Academic Press 2001.

[VKvBG95] Andreas Vogel, A Distributed Multimedia and QoS: A Survey. IEEE Multimedia Magazine, 1995

[VRML97] VRML 2.0 specification, ISO/IEC 14772-1:1997, http://www.vrml.org/Specifications

[WW2003] Michael Wimmer, Peter Wonka, Rendering Time Estimation for Real-Time Rendering, Rendering Techniques 2003 (Proceedings of the Eurographics Symposium on Rendering 2003), pages 118-129. June 2003

[WS1996] Wijesekera, Srivastava, Quality of Service Metrics for Continuous Media, 1996

[ZU97] Zekeriya Uykan, Hierarchical Control and Multimedia, Multimedia applications in industrial automation – Collected papers of the Spring 1997

postgraduate seminar. Helsinki University of Technology, Report 106, June 1997