11-1995

Quality of Service Specification for Multimedia Presentations

Richard Staehli
Oregon Graduate Institute of Science & Technology

Jonathan Walpole
Oregon Graduate Institute of Science & Technology

David Maier
Oregon Graduate Institute of Science & Technology

Quality of Service Specification for Multimedia Presentations*

Richard Staehli, Jonathan Walpole and David Maier
{staehli, walpole, maier}@cse.ogi.edu

Department of Computer Science & Engineering
Oregon Graduate Institute of Science & Technology
20000 N.W. Walker Rd., PO Box 91000
Portland, OR 97291-1000

ABSTRACT

The bandwidth limitations of multimedia systems force tradeoffs between presentation data fidelity and real-time performance. For example, digital video is commonly encoded with lossy compression to reduce bandwidth and frames may be skipped during playback to maintain synchronization. These tradeoffs depend on device performance and physical data representations that are hidden by a database system. If a multimedia database is to support digital video and other continuous media data types, we argue that the database should provide a Quality of Service (QOS) interface to allow application control of presentation timing and information loss tradeoffs.

This paper proposes a data model for continuous media that preserves device and physical data independence. We show how to define formal QOS constraints from a specification of ideal presentation outputs. Our definition enables meaningful requests for end-to-end service guarantees while leaving the database system free to optimize resource management. We propose one set of QOS parameters that constitute a complete model for presentation error and show how this error model extends the opportunities for resource optimization.

Keywords: Data Model, Synchronization, Resource Reservations.

1 Introduction

Multimedia database systems are being extended to support presentations of *continuous media* [8], such as video and audio, as well as synthetic compositions such as slide shows and computer-generated music. We call these presentations *time-based* because they communicate part of their information content through presentation timing. While applications with text and numeric data types expect correct results from database queries, the real-time constraints of time-based presentations commonly make it impossible to return complete and correct results. Some information loss is also inevitable in any conversion of continuous media between analog and digital representations.

Consider the reproduction of NTSC video in a digital multimedia system. The video stream is typically captured at 640x480 24-bit samples/frame and 30 frames/second, but it is rarely stored or played back at this bandwidth. Instead, lossy compression algorithms such as the MPEG encoding [7] are used to reduce the bandwidth requirements in exchange for some loss in quality. In addition, if the display window is smaller than 640x480 then the presentation will lose even more of the

*This research is supported by NSF Grants IRI-9223788 and IRI-9117008, and by funds from Tektronix, Inc. and the Oregon Advanced Computing Institute.

encoded data resolution. Contention for shared resources between applications also contributes to bandwidth restrictions and timing errors. Real-time MPEG players commonly drop late frames rather than delay the remainder of the presentation.

Since the usefulness of time-based presentations depends on the accuracy of both timing and data, computing the result of a query in a multimedia database is a question of *quality* rather than correctness. Where database design has traditionally been concerned with the delivery of correct results with acceptable delay, multimedia systems present a new challenge: to deliver results with *acceptable quality* in real-time. But how accurate must a presentation be to be acceptable, and how can we guarantee that a presentation achieves that accuracy? This paper helps to answer the first question by giving a formal definition of presentation *quality* that measures both accuracy of timing and the accuracy of output values. This definition of presentation quality is then used to specify presentation-level QOS requirements. The second question can be answered by a variety of the techniques found in existing systems [15]. However, we argue that current systems take an ad hoc approach to presentation quality. Without a specification of presentation QOS requirements, multimedia systems have no way of saying whether a presentation is acceptable or not. As a result, these systems tend either to be overly conservative, wasting resources in an attempt to guarantee maximal quality; or overly liberal, accommodating resource shortages by unconstrained degradation of quality.

The most common multimedia presentation tools use a *best-effort* approach, which aggressively consumes resources to present all data as promptly as possible. When resources are overloaded, a best-effort presentation will lose information. Many researchers have demonstrated best-effort systems that maintain approximate synchronization despite variable latencies and resource availability [6, 13, 2]. These systems show that a presentation can be acceptable even when quality degradation is noticeable, but we observe two problems with a best-effort approach. First, if perfect presentation is not necessary, why should a multimedia system expend extra "effort" for the best quality? Second, how much quality degradation can be allowed when many real-time presentations compete for scarce resources? If any application is to be guaranteed acceptable service, some information is needed about presentation QOS requirements.

Performance guarantees are an essential feature of real-time systems, including time-based presentations. While best-effort approaches offer only weak guarantees for synchronization, strong QOS guarantees for continuous media presentations have been demonstrated through reservation of processor, memory, network and storage system resources [11, 19, 1, 9, 12, 3]. The resource reservations are derived from low-level QOS parameters such as throughput, delay and jitter requirements for stream processing. We call this a *guaranteed-best* approach when reservations are based on requirements for a best-quality presentation. The primary problem with this approach is that it is often too expensive. For example, an instructional video with slow and deliberate motions may be digitized and stored at 30 frames/second, but playback at 15 frames/second is adequate for the purpose of instruction. The requirements for presentation quality depend not on the data type or view, but on the purpose of a presentation.

Others have recognized that best-quality presentations are often too expensive and unnecessary. The Capacity-Based-Session-Reservation-Protocol (CBSRP) [17] supports reservation of processor bandwidth from the specification of a range of acceptable spatial and temporal resolutions for video playback requests. The resolution parameters are intended only to provide a few classes of service based on resource requirements and not to completely capture presentation quality requirements. Hutchinson et al. [5] suggest a framework of categories for high-level QOS specifications that include *reliability, timeliness, volume, criticality, quality of perception* and even *cost*. They provide only a partial list of QOS parameters to show that current QOS support in OSI and CCITT standards is severely limited. The error model we describe in this paper extends their approach to provide a complete set of parameters to constrain presentation quality. We define presentation quality to include only factors that affect perception of the information content of a time-based presentation.

Database technology offers many benefits for multimedia applications, such as high-level query

languages, concurrency, and device and physical data independence. But current database systems do not adequately support time-based presentations. Relational data manipulation languages have demonstrated the value of letting the application specify *what* is wanted, and letting the database plan *how* to retrieve it. To support time-based presentations, a data manipulation language for a multimedia database should also allow the application to specify *when*, *where*, and *how precisely* the data should be delivered [10]. These constraints on delivery are an example of a QOS-based interface.

None of the proposed data models for time-based multimedia that we are aware of support queries for imprecise results. For example, Gibbs describes a data model that captures the structure and synchronization relationships of complex time-based multimedia presentations [4]. This model includes media descriptors that attach a *quality factor*, such as "VHS quality" or "CD quality", to each media object, but these labels describe the quality of the representation rather than the presentation. Without the notion of presentation quality in the data model, one would presume that all information would be preserved in the result of a query. In practice, information loss in a time-based presentation is inevitable and unconstrained by current data models.

This paper defines a methodology for presentation QOS specification. The definitions are intended to be general enough to apply to presentations in any multimedia system. In particular, our methodology supports the following goals:

- **Model user perception of quality.** Just as modern compression algorithms exploit knowledge of human perception [18], a multimedia system can better optimize playback resources if it knows which optimizations have the least affect on perceived quality.
- **Formal semantics.** We would like to be able to prove that multimedia system can satisfy a QOS specification.
- **Complex data model.** QOS specifications can be defined for a large class of complex multimedia presentations.

The next section defines our terminology in terms of an architectural model for multimedia presentations. Sections 3 and 4 describe a data model for the specification of *content* and *view* respectively for a presentation. We then define *quality* in Section 5 as a function of a presentation's fidelity to the *content* and *view* specification, in the context of an error model. We define one possible error model, and suggest in Section 6 how a formal QOS specification can be used to optimize resource usage in a presentation. Section 7 gives our conclusions.

2 Architectural Model

In our architectural model, shown in Figure 1, multimedia data come from live sources or from storage. Digital audio and video data have default content specifications associated with them that specify the sample size and rate for normal playback. A time-based media *editor* may be used to create complex presentations from simple content. A *player* is used to browse and play-back content specified by the editor. A user may control a player's *view* parameters, such as window size and playback rate, as well as *quality* parameters such as spatial and temporal resolution. The combination of content, view, and quality specifications constitute a QOS specification. When a user chooses to begin a presentation, the player needs to verify that a *presentation plan* consisting of real-time tasks will satisfy the *QOS specification*. A presentation plan is feasible if guarantees can be obtained from a *Resource Manager* for the real-time presentation tasks that transport and transform the multimedia data from storage or other data sources to the system outputs.

This architecture is similar to other research systems that provide QOS guarantees based on an admission test [13]. However, our definition of QOS is novel in that we make strong distinctions between content, view, and quality specifications. A *content* specification defines a set of logical image and audio output values as a function of logical time. A *view* specification maps content onto a set of physical display regions and audio output devices over a real-time interval. *Quality* is

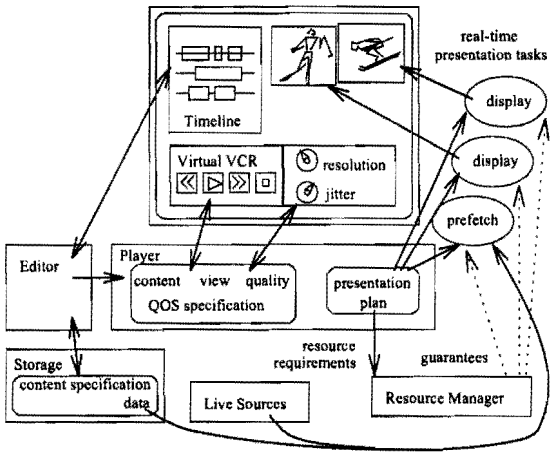

Figure 1: An architecture for editing and viewing multimedia presentations.

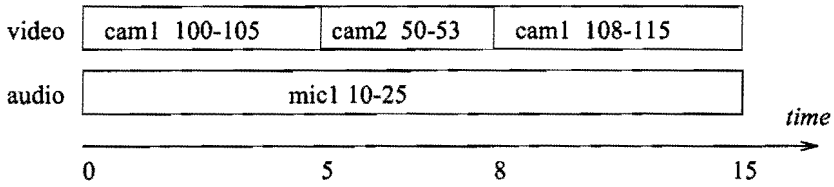

Figure 2: Timeline view of content specification for a presentation of bicycling video with audio.

a measure of how well an actual presentation matches the ideal presentation of content on a view and a *quality specification* defines a minimum acceptable quality measure. We will refer to quality when we mean the measure, and QOS when we mean the combination of content, view, and quality specifications.

By allowing independent control of content, view and quality, a multimedia system can offer a wider range of services that take advantage of the flexibility of computer platforms. To illustrate these services, consider the presentation of video and audio as described in Figure 2. The first video clip refers to 5 seconds of a digital video file. The video file is named *cam1* because it was captured with the first of two cameras recording the same bicycle racing event. The digital video for cam1 has a resolution of 320x240 pixels. A second video file named *cam2* shows another view of the bicycling event and has a higher resolution of 640x480 pixels. The video presentation cuts from *cam1* to *cam2* for 3 seconds, and then back to *cam1* for the last 7 seconds. The audio clip file *mic1* contains a digital audio sound-track recorded at the same time as the video clips. After selecting this content for presentation, a user should be able to choose view parameters and quality levels independently. For example, if the user chooses a view with a 640x480 pixel display window, but a quality specification that requires only 320x240 pixels of resolution, then the player may be able to avoid generating the full resolution images from cam2. The quality specification allows the user to indirectly control resource usage independent of the content and view selections. The player can optimize resource usage so long as the presentation exceeds the minimum quality specification. Users might also like to specify an upper bound on cost for resource usage, but since cost is independent of information loss, constraints on cost are beyond the scope of this paper.

3 Content Specification

To provide a concrete example of presentation QOS semantics, this section defines a data model for specifying the content of multimedia presentations. The model supports composition of audio and video data to create complex presentations. Other media such as text and still images may be included by modelling them as video stills with finite duration. Our data model does not support user interaction, but it could be extended to do so by making content specifications depend on the timed sequence of user inputs. Since user interaction and other extensions would not substantially change our definition of quality, we do not consider them further in this paper.

We describe the data model using the Z specification language [14] in order to focus attention on the mathematical properties rather than on details of syntax and implementation. See the accompanying sidebar for an overview of Z notation.

Z notation

We use a subset of the Z specification language as defined in The Z Reference Manual [14] and augmented with the standard arithmetic and calculus operators and relations defined over the set of real numbers. The example below shows a global declaration of a schema type S. A Z schema type consists of a signature of typed variables together with constraints on those variables. A declaration $v : S$ says that v has schema type S and the components $v.x$ and $v.y$ must obey the constraints for x and y respectively in schema S.

$$
\begin{array}{|l}
S \\
\hline
x, y : R \\
\hline
y \geq x \\
\hline
\end{array}
$$

Other global functions and constants can be declared with an axiomatic description. The following example declares that *length* is a function from schema type S to the set of real numbers and that, for all variables v of type S, *length* v is the difference between the y and x components of v.

$$
\begin{array}{|l}
\mathit{length} : S \longrightarrow R \\
\hline
\forall v : S \bullet \mathit{length}\ v = v.y - v.x \\
\end{array}
$$

Free type definitions declare type constructors and arguments that generate a new type. For example, a *Tree* is either a *leaf* or a *branch* with an integer value and two subtrees. The type constructor *branch* is a function from a 3-tuple of an integer and two *Trees* to a *Tree*.

$$\mathit{Tree} ::= \mathit{leaf} \mid \mathit{branch}\langle\!\langle \mathbb{Z} \times \mathit{Tree} \times \mathit{Tree} \rangle\!\rangle$$

The Z language includes common notation from set theory and first-order logic. We give brief definitions here for other notation that may be unfamiliar.

$S : \mathbb{P}\, X$	S is declared as a subset of X
$X \leftrightarrow Y$	binary relation
$X \longrightarrow Y$	function from X to Y
$\mathit{dom}\ f$	domain of the function f
$\mathit{ran}\ f$	range of the function f
$\mathit{min}(S)$	the minimum of a nonempty set of numbers S
$\mathit{max}(S)$	the maximum of a nonempty set of numbers S
$\mathit{seq}\ X$	a finite sequence with elements of type X
$\langle\rangle$	the empty sequence
$\mathit{head}\ s$	head of sequence
$\mathit{tail}\ s$	tail of sequence
$(\textbf{let}\ x == E1 \bullet E2)$	let x be an abbreviation for E1 in E2
$\{ x : T \mid P \bullet E \}$	the set of all E such that x has type T and P is true.

Our content specifications define a set of logical output channels and the acceptable real-number values for those outputs that may vary continuously with time. It is an important feature of this model that the audio and video specifications may have infinite resolution. For example, the visualization of a continuous function whose values can be computed rather than read from storage is limited by the computational resources and the display device, but not by the content specification.

We assume only two basic types: Real numbers and Integers. Digital inputs and outputs will be declared as Integers, but nearly all other quantities will be modeled as Real numbers. Real numbers are used for the specification of logical values to avoid placing an artificial limit on the content resolution. We begin with a declaration of these basic types:

$[R, \mathbb{Z}]$

The *Interval* schema gives a start position and an interval extent. We use this information to specify both clipping intervals and linear transformations. In a linear transformation, we use *start* as an offset and *extent* as a scale factor.

Interval

$start : R$
$extent : R$

To make it easier to treat all outputs uniformly, we define a single schema for describing output dimensions. This schema must contain the maximal set of dimensions for all output types. When used for audio specifications, we simply ignore the x and y intervals. The *Space* schema specifies intervals for t, x, and y coordinate dimension and a z interval for the output range. For example, we describe the dimensions of a sampled video source with a *Space* that stores the start index and number of frames in the t interval, the start index and number of samples across and vertically in the x and y intervals respectively, and the range of sample values in z.

Space

$t : Interval$
$x : Interval$
$y : Interval$
$z : Interval$

A *Content* specification is a recursive construct built from basic audio and video sources. Each *audio*, *video*, *sampledAudio*, and *sampledVideo* construct defines a single logical output channel. More complex content may be specified using *clip*, *transform*, *cat*, *synch*, and *select* constructs. The *LogicalOutput* type is used in the *select* construct to reference a particular logical output. The exact meaning of each of these constructs is described below.

$$LogicalOutput ::= lAudio\ \langle\langle \mathbb{Z} \rangle\rangle \mid lVideo\ \langle\langle \mathbb{Z} \rangle\rangle$$

$$\begin{aligned} Content \quad ::= {} & audio\langle\langle Space \times (R \longrightarrow R) \rangle\rangle \\ \mid {} & video\langle\langle Space \times (R \longrightarrow R \longrightarrow R \longrightarrow R) \rangle\rangle \\ \mid {} & sampledAudio\langle\langle Space \times (\mathbb{Z} \longrightarrow R) \rangle\rangle \\ \mid {} & sampledVideo\langle\langle Space \times (\mathbb{Z} \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \longrightarrow R) \rangle\rangle \\ \mid {} & clip\langle\langle Space \times Content \rangle\rangle \\ \mid {} & transform\langle\langle Space \times Content \rangle\rangle \\ \mid {} & cat\langle\langle \text{seq } Content \rangle\rangle \\ \mid {} & synch\langle\langle \text{seq } Content \rangle\rangle \\ \mid {} & select\langle\langle LogicalOutput \times Content \rangle\rangle \end{aligned}$$

The *audio* constructor takes a pair with a *Space* descriptor and a function from a real time coordinate to a real z value. For example, a sine function could be given as an audio source function with no limit on the resolution of the signal. As described in the following sections, the resolution of a presentation is limited only by an actual implementation on digital outputs. The *video* constructor also takes a pair with a *Space* descriptor and a function, but the video source function requires additional real coordinates for x and y. The domain and range for the functions are specified with the *Space* argument. The constructors for *sampledAudio* and *sampledVideo* require functions of integer coordinates. For simplicity, this definition supports only monochrome video, but the same approach can be generalized to specify a tuple of values at each point for color.

Figure 3 illustrates a content specification for the example presentation from Figure 2. We'll describe this specification from the bottom up, beginning with the two *sampledVideo* specifications

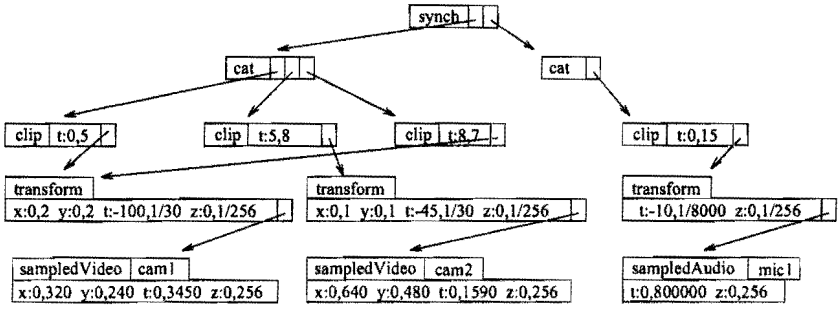

Figure 3: Content specification in normal-form for example presentation.

and the one *sampledAudio* specification that form the leaves of the tree. The first video specification declares that the source *cam1* contains 8-bit samples in 3450 frames and each frame has 320x240 pixels. The second video source named *cam2* has 1590 frames and each frame has 640x480 pixels. The audio source *mic1* has 100 seconds worth of samples at 8000 8-bit samples/second. Both videos are scaled in time to play at 30 frames/second and their z ranges are normalized by the *transform* specifications. The first video is scaled by a factor of 2 to match the dimensions of the second video and is offset by -100 seconds so that the clip can begin at logical time zero. The second video and the audio are both offset for synchronization with the first video. The audio is normalized and scaled in time to play at 8000 samples/second. The video presentation is assembled by concatenating a clip of seconds 0-5 from the first transformed video with seconds 5-8 from the second, followed by the clip of seconds 8-15 from the first again. The result is then synchronized with a clip of seconds 0-15 from the transformed audio.

The *transform*, *clip*, *cat*, *synch*, and *select* specifications support stretching and shrinking, cut, paste, and synchronization of logical outputs. Although other features are desirable, such as the ability to mix several logical outputs together, the constructs described are sufficient for editing useful time-based multimedia presentations and for illustrating the meaning of view and quality specifications in the next sections.

The meaning of a content specification is defined by a set of allowed logical output values for every point of the logical output space. Let the *Interval* function I return the set of real numbers in an interval. Then a point (x, y, t) is in the logical space s if $(x \in I\ s.x) \wedge (y \in I\ s.y) \wedge (t \in I\ s.t)$.

$$I : Interval \longrightarrow \mathbf{P}\ R$$

$$I\ v = \{\ r : R \mid (v.start \leq r) \wedge (r < v.start + v.extent)\ \}$$

We also use the *Interval* type to describe linear transformations. Given an *Interval* i, let the functions tr and utr respectively transform and untransform a real number r by a scale factor $i.extent$ and an offset $i.start$. For example, if $i.start = 3.0$ and $i.extent = 2.0$ then $tr\ 4.5\ i = 12.0$.

$$tr, utr : R \longrightarrow Interval \longrightarrow R$$

$$tr\ r\ i = r * i.extent + i.start$$

$$utr\ r\ i = (r - i.start)/i.extent$$

Content specifications constrain logical output values *only* during explicit time intervals. For example, the content specification in Figure 3 allows any output values before logical time 0 and after logical time 15. The functions *start*, *end*, and *duration* are used to reference the logical time interval over which output values are constrained by a content specification. The logical start of a content specification is the minimum time t at which *some* output value is *not* acceptable! The

logical end is the minimum time t such that no output value is constrained for times greater than or equal to t. The function *lAudios* returns the integral number of logical *audio* outputs that are constrained by a content specification and *lVideos* returns the number of logical *video* outputs.

$$
\begin{array}{|l}
start, end, duration : Content \longrightarrow R \\
lAudios, lVideos : Content \longrightarrow \mathbb{Z} \\
\hline
start\ c = min\ \{ t : R \mid \neg\ (\forall l : LogicalOutput; x, y, z : R \bullet (l, x, y, t, z) \in logical\ c) \} \\
end\ c = min\ \{ t : R \mid \\
\quad \forall t' : R \bullet (t \leq t') \Rightarrow (\forall l : LogicalOutput; x, y, z : R \bullet (l, x, y, t', z) \in logical\ c)\ \} \\
duration\ c = end\ c - start\ c \\
lAudios(c) = max\ \{ n : \mathbb{Z} \mid \neg\ (\forall x, y, t, z : R \bullet (lAudio(n), x, y, t, z) \in logical\ c) \} \\
lVideos(c) = max\ \{ n : \mathbb{Z} \mid \neg\ (\forall x, y, t, z : R \bullet (lVideo(n), x, y, t, z) \in logical\ c) \}
\end{array}
$$

The meaning of each of the content constructs is captured by the following definition of a function for logical content. For a given content specification, the *logical* function returns a relation between a point in the logical output space and the acceptable output values for that point. We read the expression $(l, x, y, t, z) \in logical\ c$ as: the content specification c specifies that logical output l, at point (x, y) and time t may have value z. Note that specifications reduce the set of acceptable values and where nothing is specified, all values are acceptable.

$$LogicalValue == LogicalOutput \times R \times R \times R \times R$$

$$logical : Content \longrightarrow \mathbb{P}\ LogicalValue$$

$$logical(audio(s,f)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l = lAudio\ 1) \land (t \in I\ s.t) \Rightarrow z = f\ t \bullet (l,x,y,t,z) \,\}$$

$$logical(video(s,f)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l = lVideo\ 1) \land (x \in I\ s.x) \land (y \in I\ s.y) \land (t \in I\ s.t) \Rightarrow z = f\ t\ y\ x \bullet (l,x,y,t,z) \,\}$$

$$logical(sampledAudio(s,f)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l = lAudio\ 1) \land (t \in I\ s.t) \land z = f\ \lfloor t \rfloor \bullet (l,x,y,t,z) \,\}$$

$$logical(sampledVideo(s,f)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l = lVideo\ 1) \land (x \in I\ s.x) \land (y \in I\ s.y) \land (t \in I\ s.t) \Rightarrow z = f\ \lfloor t \rfloor\ \lfloor x \rfloor\ \lfloor y \rfloor \bullet (l,x,y,t,z) \,\}$$

$$logical(clip(s,c)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (x \in I(s.x)) \land (y \in I(s.y)) \land (t \in I(s.t)) \Rightarrow (l,x,y,t,z) \in logical\ c \bullet (l,x,y,t,z) \,\}$$

$$logical(transform(s,c)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l,x,y,t,z) \in logical\ c \bullet (l, tr\ x\ s.x, tr\ y\ s.y, tr\ t\ s.t, tr\ z\ s.z) \,\}$$

$$logical(cat(\langle\rangle)) = \{\, l : LogicalOutput; x,y,t,z : R \bullet (l,x,y,t,z) \,\}$$

$$logical(cat(q)) = logical(head\ q) \cap \{\, l : LogicalOutput; x,y,t,z : R \mid (l,x,y,t,z) \in logical(cat(tail\ q)) \bullet (l,x,y,t + end(head\ q) - start(cat(tail\ q)),z) \,\}$$

$$logical(synch(\langle\rangle)) = \{\, l : LogicalOutput; x,y,t,z : R \bullet (l,x,y,t,z) \,\}$$

$$logical(synch\ q) = \{\, n : \mathbb{Z}; x,y,t,z : R \mid (lAudio\ n,x,y,t,z) \in logical(head\ q) \bullet (lAudio(n + (lAudios(synch(tail(q)))),x,y,t,z) \,\} \cap \{\, n : \mathbb{Z}; x,y,t,z : R \mid (lVideo\ n,x,y,t,z) \in logical(head\ q) \bullet (lVideo(n + (lVideos(synch(tail\ q)))),x,y,t,z) \,\} \cap logical(synch(tail\ q))$$

$$logical(select(lAudio\ n,c)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l = lAudio\ 1) \land ((lAudio\ n,x,y,t,z) \in logical\ c) \bullet (l,x,y,t,z) \,\}$$

$$logical(select(lVideo\ n,c)) = \{\, l : LogicalOutput; x,y,t,z : R \mid (l = lVideo\ 1) \land ((lVideo\ n,x,y,t,z) \in logical\ c) \bullet (l,x,y,t,z) \,\}$$

The first predicate for $logical(audio(s,f))$ says that if l is the logical output $lAudio\ 1$ and t is within the interval $s.t$ then the only acceptable value for z is the function $f(t)$. Otherwise, any values are acceptable for z. Note that the interval $s.z$ may indicate the intended range of source values, but there is no need to enforce this range when defining the logical content. The predicate for $logical(video(s,f))$ expresses a similar constraint for the logical output $lVideo\ 1$.

For sampled audio and video, the logical coordinates are rounded down to the nearest integer. Consequently, the number of samples (frames) is given by $\lfloor s.t.extent \rfloor$ and the pixel dimensions for video frames is $\lfloor s.x.extent \rfloor * \lfloor s.y.extent \rfloor$. This information about sample resolution is needed only for accessing the source functions and is not carried explicitly in the definition of logical content.

A $clip(s,c)$ construct specifies that for all logical outputs, points within the *Space* s are constrained to have the same values as specified by c. All points not in s are effectively "clipped" out and may have any value. Note that the z interval in s does not participate in the clipping.

A $transform(s,c)$ construct specifies a linear transformation of points in the content specified by c. For example, if $start\ c = 0$, $duration\ c = 60$, $s.t.start = 10$, and $s.t.extent = 2$, then $start(transform(s,c)) = 10$ and $duration(transform(s,c)) = 120$. The transformation construct

transform(s, c) with all start fields in s equal to zero and all extent fields in s equal to one is the identity transformation and has no effect.

A temporal sequence of content can be specified with a *cat*(q) construct. The content for a member of the sequence q is logically shifted in time to start just as the previous content in the sequence ends. The *synch*(q) construct specifies that a set of logical outputs all reference the same time scale. If the content specifications c_n and c_m specify n and m logical outputs respectively, then *synch*$(\langle c_n, c_m \rangle)$ specifies $m + n$ logical outputs.

The *select*(l, c) construct offers a way to reference only the content of a single logical output within a complex specification. Where the *synch* construct aggregates multiple logical outputs into a single content specification, *select*(l, c) specifies only a single logical output with the same content as c specifies for logical output l. For any n, the logical output defined by *select*(l, c) is $(lAudio\ 1)$ if $l = (lAudio\ n)$ and $(lVideo\ 1)$ if $l = (lVideo\ n)$. If a content construct does not specify the logical output l then *select*(l, c) is the null specification; that is, all values are permissible on all outputs.

It is worth noting that no matter how a content specification is composed, its logical content may be equivalently specified by a content specification with the normal-form shown in Figure 3. In normal-form, every specification is a tree with a *synch* construct at the root. The *synch* construct specifies a sequence of *cat* constructs. Each *cat* construct specifies a single logical output with a sequence of *clip* constructs. Each *clip* specifies a portion of a *transform* construct and each *transform* construct defines the logical dimensions of a basic media source. A basic media source must be either an *audio*, *video*, *sampledAudio*, or *sampledVideo* construct.

This definition of content satisfies our goal of a data model for complex presentations except that we have no way to relate the logical content to actual presentation outputs. The logical outputs of a content specification have both temporal and spatial proportions, but they have no physical size or real duration. To make the connection to actual presentation outputs, we need to specify a mapping between the content and physical devices.

4 View Specification

A *View* specification allocates physical devices for logical outputs and maps logical time to a real-time clock. While the physical devices may present an upper bound on spatial and temporal resolution, the view does not specify presentation quality. Figure 4 shows a view specification that allocates an unusually small 8x6 pixel window on a monochrome (black and white) display for the bicycling video presentation. Although the output device clearly limits the quality of the presentation, the view does not specify how the content is to be represented on the display. It is the presentation plan that must choose how to resample the source and how to represent gray scale information. The combination of content and view specifications serve as a device-independent specification of a perfect quality presentation. The idea of an ideal presentation is formally defined below. In the next section, we define less-than-perfect quality based on the difference between this ideal presentation and actual presentation outputs.

Since we are not interested in the details of the physical device I/O, we simply assume that there is a set of audio output devices *AudioDev* and video output devices *VideoDev*. A *Device* is either one of the audio devices or one of the video devices.

$[AudioDev, VideoDev]$

$Device == AudioDev \cup VideoDev$

The logical dimensions in a content specification are generally not the same as the physical dimensions of the view. The *Output* schema declares a field *tr* that defines the transformation from logical to view output dimensions and a field *clip* that defines clipping bounds for view outputs. In Figure 4, the *Output* specification for *lVideo* 1 transforms the 640x480 logical image size to 8x6 and then offsets the image 2 pixels in x and 1 in y. The z values are transformed from the logical range of $[0, 1)$ to the view range of $[0, 256)$. The clipping bounds for both audio and video match the

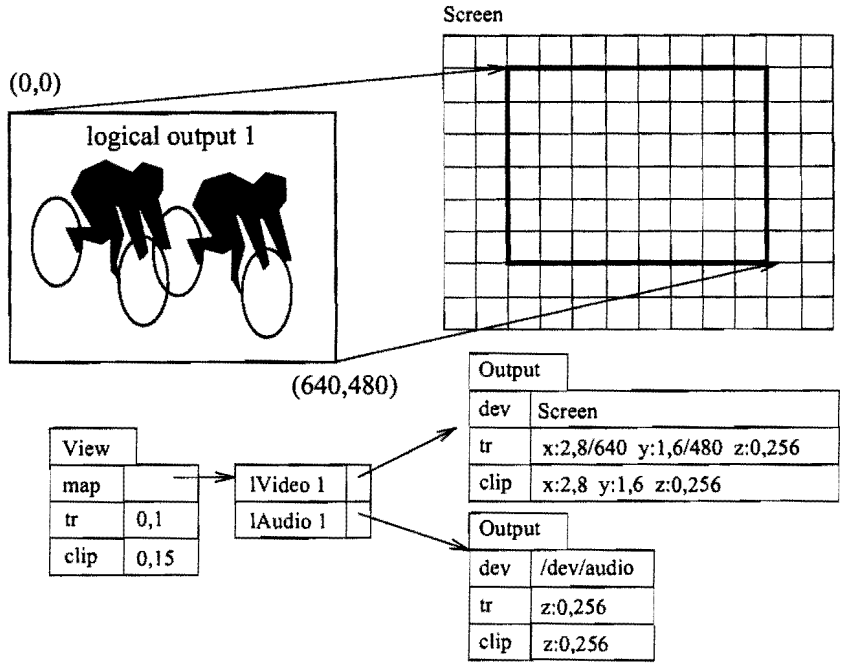

Figure 4: Example of a view that allocates an 8x6 pixel window on a display device for presentation of the bicycling video.

full range of the transformed content. Note that the time fields are ignored, because the temporal transformation and clipping for all outputs is given in the *View* specification. This asymmetry is necessary to preserve the content synchronization while allowing flexibility in the display of multiple logical outputs.

Output

$dev : Device$

$tr, clip : Space$

A *View* specifies a partial function *map* that assigns a subset of the logical outputs to physical output specifications. Logical content that is to be presented must be mapped to physical outputs of the appropriate type. Logical outputs that are not in the domain of the map function are ignored. The *tr* field is used to transform logical time in a content specification to a real-time clock. The *clip* field specifies the real-time start and duration of the presentation.

View

$map : LogicalOutput \pfun Output$

$tr : Interval$

$clip : Interval$

$(lAudio\ n \in \mathrm{dom}\ map) \Rightarrow (\exists\, d : AudioDev \bullet\ (map(lAudio\ n)).dev = d)$

$(lVideo\ n \in \mathrm{dom}\ map) \Rightarrow (\exists\, d : VideoDev \bullet\ (map(lVideo\ n)).dev = d)$

A view specification together with a content specification defines an *ideal* presentation, where the output devices are assumed to have infinite resolution. This assumption is necessary for a

device-independent definition of quality. We model a presentation as a set of *DeviceValue* tuples (d, x, y, t, z) that give the z value for a particular *Device* d and coordinates x, y, and t.

$$DeviceValue == Device \times R \times R \times R \times R$$

We define a function *ideal* c v that returns the relation between devices and the values specified by a *Content* specification c and a *View* specification v. The relation *ideal* c v contains all *DeviceValue* tuples (d, x, y, t, z), where the view maps a logical output l to a device d and x, y, and t are within the clipping bounds for d, only if the corresponding logical value is allowed by the content specification c. The corresponding logical point is computed by substituting l for p and "un-transforming" x, y, t, and z back to logical space.

$$\begin{array}{|l}
ideal : Content \longrightarrow View \longrightarrow \mathbb{P}\, DeviceValue \\
\hline
ideal\ c\ v = \{\, d : Device;\ x, y, t, z : R \bullet \\
\quad (\exists\, l : LogicalOutput;\ p : Output \bullet \\
\quad\quad ((l \in \operatorname{dom} v.map) \land (v.map\ l = p) \land (p.dev = d) \\
\quad\quad \land\ (d \in AudioDev) \land (t \in I\ v.clip)) \Rightarrow \\
\quad\quad\quad (\exists\, x', y' : R \bullet (l, x', y', utr\ t\ v.tr, utr\ z\ p.tr.z) \in logical\ c)) \\
\quad \land\ (\exists\, l : LogicalOutput;\ p : Output \bullet \\
\quad\quad ((l \in \operatorname{dom} v.map) \land (v.map\ l = p) \land (p.dev = d) \\
\quad\quad \land\ (d \in VideoDev) \land (t \in I\ v.clip) \land (x \in I\ p.clip.x) \land (y \in I\ p.clip.y)) \Rightarrow \\
\quad\quad\quad ((l, utr\ x\ p.tr.x, utr\ y\ p.tr.y, utr\ t\ v.tr, utr\ z\ p.tr.z) \in logical\ c)) \\
\quad \bullet\ (d, x, y, t, z)\,\}
\end{array}$$

The implementation of a presentation plan uniquely determines the value for every device at every point and time. The schema *Presentation* models the implementation with separate functions for audio and video. The *aVal* function takes an *AudioDev* and an integer clock value and return an integer z value. The *vVal* function takes a *VideoDev* and integer values for the clock, x, and y coordinates, and returns the integer value at that pixel.

$$\begin{array}{|l}
\textit{Presentation} \\
\hline
aVal : AudioDev \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \\
vVal : VideoDev \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \\
\hline
\end{array}$$

We define a function *actual* that takes a particular presentation and returns a relation representing these output values. The relation *actual* P contains a point (d, x, y, t, z) only if z is the value of the device d and pixel (x, y) while the clock value is t as defined by P. We are assuming that we can observe only one output value per clock tick and that the output value is constant over the duration of a clock cycle.

$$\begin{array}{|l}
actual : Presentation \longrightarrow \mathbb{P}\, DeviceValue \\
\hline
actual\ P = \\
\quad \{\, d : Device;\ x, y, t : R \mid \\
\quad\quad d \in AudioDev \bullet (d, x, y, t, P.aVal\ d\ \lfloor t \rfloor)\,\} \\
\quad \cup\, \{\, d : Device;\ x, y, t : R \mid \\
\quad\quad d \in VideoDev \bullet (d, x, y, t, P.vVal\ d\ \lfloor x \rfloor\ \lfloor y \rfloor\ \lfloor t \rfloor)\,\}
\end{array}$$

The relation *actual* P and the relation *ideal* c v have the same type. In the next section we use this fact to define a mapping between them.

5 Quality Specification

We define the *quality* of a presentation to be the ratio of the *worth* of the actual presentation to the worth of the ideal presentation. Although worth may be subjective, we believe the ratio can be usefully modelled with a few assumptions:

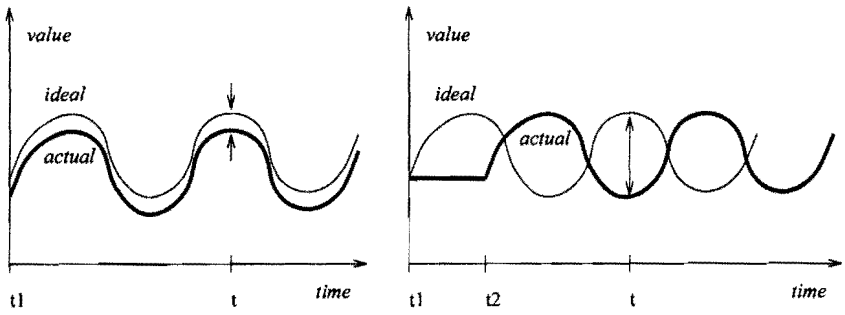

Figure 5: Presentation error may be attributed to value error alone, as shown on the left, or to some combination of timing and value errors.

1. User perception of presentation quality can be modelled by a continuous function of time and device coordinates.
2. The quality of a presentation that differs from the ideal is less than one. How much less depends only on user perception of the difference.
3. User perception of the difference between an actual presentation and the ideal is based on a mapping from points in the actual presentation to points in the ideal presentation.

With these assumptions, quality is independent of data representations and transport mechanisms. In particular, our definition of quality is not based on the data throughput required for a presentation, but instead can be used to determine throughput requirements as shown in the next section. In this section, we provide a model for computing quality and define quality specifications in terms of this model.

The declaration for an *ErrorInterpretation* below is the most important part of our QOS specification because it defines an *error model* for measuring presentation quality. An *error model* is a set of functions that describe the number of ways in which an actual presentation may be different from an ideal presentation. We refer to these functions as *error components*.

Consider the presentations illustrated in Figure 5. A minimal error model includes a function for error in the z dimension for every output, but assumes error in x, y, and t dimensions are zero. The error function returns the minimum difference between the actual output value z and an ideal value for each output, p, at the same point and time. This error model is the smallest model that can map from *actual* P to *ideal* c v for any P, c, and v. We say that an error model is *complete* if we can specify arbitrarily high quality (as judged by humans) by requiring that all error components in the error model are sufficiently close to zero. The minimal error model is complete because the presentation becomes indistinguishable from the ideal as the Z error component goes to zero everywhere.

The choice of error components in our error model is intended to provide a useful model of human perception. Ideally, a presentation QOS specification should accept all presentations that humans accept and reject only those that humans reject. A *conservative* specification is one that never accepts a presentation that humans would reject. If an error model is complete, we are assured that every error component has a non-zero threshold for which it becomes imperceptible. As a result, we can always find a conservative specification that constrains all errors to be less than their threshold values. But such a specification would undoubtedly reject many presentations that humans would accept.

We can show that any conservative QOS specification based on the minimal error model needlessly rejects presentations that we find acceptable. With the minimal error model, we can only

accept presentations with no timing error and small z error as illustrated on the left in Figure 5. The right side in Figure 5 shows that a simple startup delay produces large z error measurements under the minimal error model. But a person judging the presentation would be able to compensate for the small startup delay and perceive very little z error. By adding a constant time shift error component to the error model, we can accept the larger set of presentations where z error is small after compensating for some small startup delay.

Our error model adds many error components to achieve a better match between conservative QOS specifications and human perception. The *shift* error component allows a presentation to be behind (or ahead of) schedule. Rather than require the error component for time shift to be constant, we allow *shift* to vary through a presentation. The rate at which the time shift drifts is constrained by a *rate* error component. The *rate* error is zero while the *shift* error is constant, but increases in magnitude when the presentation speeds up or slows down. We added a *jitter* error component, which allows small "hiccups" in the timing that would otherwise be prohibited by constraints on *rate* error. For example, logical time for a video might be accurately perceived as stopping between frames and then advancing instantaneously as the next frame is presented. Rather than reflecting this rate fluctuation in the *rate* error component, the *jitter* error component accounts for these small timing errors. How much of the timing error is due to *jitter* and how much to *shift* is a matter of interpretation! We include a *synch* error component for each pair of outputs because shift errors are not very noticeable except as synchronization error between outputs.

The *shift*, *rate*, and *jitter* error components are defined similarly for x and y dimensions since spatial presentations can suffer from displacement, scaling and small distortions that are analogous to the temporal error components.

Even after accounting for temporal and spatial errors, the difference between an actual presentation value and the corresponding ideal value at an infinitesimal point is not particularly meaningful. The problem is that humans don't perceive independent values at infinitesimal points, but instead integrate over small display areas and time intervals. This fact is routinely exploited by graphics algorithms that use dithering. For example, a black and white display can represent a 50% gray tone by a pattern with every other pixel turned on. Dithering trades off spatial resolution for more accurate average values. The resolution of an image in the x dimension can be thought of as the width of the smallest resolvable vertical stripe. We define resolution in y and in time similarly. Then the interesting measure of z error is the difference in average value over the neighborhood of a point defined by x, y, and t resolution. This separates objective value errors into perceived resolution loss and perceived value errors. Our error model further decomposes value errors into z-dimension *shift*, *rate*, and *jitter* error components to model perceived offset, scale, and noise errors respectively.

Several type declarations and functions simplify the formal definition of our error model and quality constraints. An *xytFun* is just an abbreviation for a function that takes three real numbers for x, y and t coordinates and returns a real number.

$$xytFun == R \longrightarrow R \longrightarrow R \longrightarrow R$$

The *Error* data type provides names for the error components that our error model associates with each output.

$$Comp ::= err \mid shift \mid rate \mid jitter \mid res$$
$$Error ::= X\langle\langle Comp\rangle\rangle \mid Y\langle\langle Comp\rangle\rangle \mid T\langle\langle Comp\rangle\rangle \mid Z\langle\langle Comp\rangle\rangle$$

The *Error* type does not include a name for synchronization error because this error component is represented in the *ErrorInterpretation* by a separate function type. The *avg* function is needed to express the relation between error in z and resolution error in x, y, and t. The expression $avg\ (xres, yres, tres)\ f$ computes an *xytFun* that is the average value of the function f over the local region defined by *xres*, *yres*, and *tres*. Because audio outputs do not vary in x or y, $avg\ (X\ res, Y\ res, T\ res)\ f$ is independent of the values of $X\ res$ and $Y\ res$ in that case and is therefore well defined even when $X\ res$ and $Y\ res$ are not specified.

$$avg : (xytFun \times xytFun \times xytFun) \longrightarrow xytFun \longrightarrow xytFun$$

$$\begin{aligned}
&(avg\ (xres, yres, tres)\ f)\ x\ y\ t = \\
&\quad (\textbf{let}\ x_r == xres\ x\ y\ t;\ x_1 == x - (x_r/2);\ x_2 == x + (x_r/2); \\
&\qquad y_r == yres\ x\ y\ t;\ y_1 == y - (y_r/2);\ y_2 == y + (y_r/2); \\
&\qquad t_r == tres\ x\ y\ t;\ t_1 == t - (t_r/2);\ t_2 == t + (t_r/2) \bullet \\
&\qquad\quad \tfrac{1}{x_r * y_r * t_r} \int_{t_1}^{t_2} \int_{y_1}^{y_2} \int_{x_1}^{x_2} (f\ x'\ y'\ t')\ dx'\ dy'\ dt')
\end{aligned}$$

ErrorInterpretation

$$\begin{aligned}
&c : Content \\
&v : View \\
&P : Presentation \\
&error : Output \longrightarrow Error \longrightarrow xytFun \\
&synch : Output \longrightarrow Output \longrightarrow xytFun
\end{aligned}$$

$$\begin{aligned}
&\forall p : Output \bullet (\textbf{let}\ i == error\ p \bullet \\
&\quad \exists z_{ideal}, z_{actual} : xytFun \bullet \\
&\qquad (\forall x, y, t : R \bullet (p, x, y, t, z_{ideal}\ x\ y\ t) \in ideal\ c\ v) \\
&\qquad \wedge\ (\forall x, y, t : R \bullet (p, x, y, t, z_{actual}\ x\ y\ t) \in actual\ P) \\
&\qquad \wedge\ i(Z\ err) = (\lambda\ x, y, t : R \bullet \\
&\qquad\quad z_{actual}\ x\ y\ t - \\
&\qquad\quad z_{ideal}(x + i(X\ err)\ x\ y\ t)(y + i(Y\ err)\ x\ y\ t)(t + i(T\ err)\ x\ y\ t)) \\
&\qquad \wedge\ i(X\ err) = i(X\ shift) + i(X\ jitter) \\
&\qquad \wedge\ i(Y\ err) = i(Y\ shift) + i(Y\ jitter) \\
&\qquad \wedge\ i(T\ err) = i(T\ shift) + i(T\ jitter) \\
&\qquad \wedge\ i(X\ rate) = \partial i(X\ shift)/\partial x \\
&\qquad \wedge\ i(Y\ rate) = \partial i(Y\ shift)/\partial y \\
&\qquad \wedge\ i(T\ rate) = \partial i(T\ shift)/\partial t \\
&\qquad \wedge\ (\textbf{let}\ perceivedErr = i(Z\ shift) + ((1 + i(Z\ rate)) * z_{ideal}) + i(Z\ jitter) \bullet \\
&\qquad\quad avg\ (i(X\ res), i(Y\ res), i(T\ res))\ (i(Z\ err)) = \\
&\qquad\quad avg\ (i(X\ res), i(Y\ res), i(T\ res))\ perceivedErr)) \\
&\forall p, q : Output \bullet \\
&\quad synch\ p\ q = (error\ p\ (T\ shift)) - (error\ q\ (T\ shift))
\end{aligned}$$

The error model in this declaration defines a set of error components for each output through the *error* function as well as an error component for each pair of outputs defined by the function *synch*. The predicate for an *ErrorInterpretation* is like a differential equation in that it does not have a unique solution for the error component functions. Instead, we observe that error measurement is inherently subjective because the output signals do not carry meta-information about the intended relationship to the specification. An *ErrorInterpretation* merely defines one subjective mapping and a set of error components that are consistent with each other and the mapping. We say that an error model is *sound* if, for any trio of *Content*, *View*, and *Presentation*, a set of error component functions exist that satisfy the definition of the error model. We claim that the error model proposed above is both sound and complete.

We declare a *quality* specification to be a schema that gives the minimum acceptable level of quality and also provides values for calibrating the affect of each error component on presentation quality.

Quality

$$\begin{aligned}
&min : R \\
&calib : Output \longrightarrow Error \longrightarrow R \\
&calibSynch : Output \longrightarrow Output \longrightarrow R
\end{aligned}$$

$$(0 \leq min) \wedge (min < 1)$$

The meaning of the quality schema in conjunction with a content and view specification is given by the following schema for a *QOS* specification:

QOS

$c : Content$
$v : View$
$q : Quality$
$P : Presentation$

$$\exists i : ErrorInterpretation \bullet i.c = c \land i.v = v \land i.P = P \land$$
$$\forall p \in \text{ran } v.map; x, y, t : R \bullet$$
$$\textbf{(let}$$
$$error = (\lambda\ m : Error \bullet | \frac{i.error\ p\ m\ x\ y\ t}{q.calib\ p\ m} |);$$
$$synch = (\lambda\ p' : Output \bullet | \frac{i.synch\ p\ p'\ x\ y\ t}{q.calibSynch\ p\ p'} |) \bullet$$
$$q.min \leq (\prod_{m \in Error} e^{-error\ m}) * (\prod_{p' \in \text{ran } v.map} e^{-synch\ p'}))$$

This schema consists of *Content*, *View*, and *Quality* specifications that constrain a presentation P. The *QOS* specification is satisfied only if an *ErrorInterpretation* exists for c v and P such that, at all times and all points on every output, the quality of the presentation is greater than or equal to $q.min$. We compute quality with an exponential decay function that depends on the absolute value of error components. This model has the following properties:

- quality is one when all error components are zero.
- quality is monotonically decreasing with increase in the absolute value of any error component.
- quality approaches zero as all error components approach infinity.

This definition for QOS specifications is very strict in that quality must exceed the minimum *everywhere* during a presentation. It would be nice to extend the specification semantics to allow a presentation to occasionally drop below this minimum quality, but this extension is left for future work.

For a given presentation and its specification, our error model allows an infinite number of interpretations; each with a different affect on the calculation of presentation quality. What matters is that an interpretation exists that has acceptable errors. We assume that humans are good at recognizing the intended presentation content and that they will perceive the interpretation with errors that are the most acceptable.

To calibrate this quality function to approximate user perception, we can adjust the values returned by the *calib* and *synchCalib* functions in the *quality* specification. We call these values *critical error values*. For every error component in our error model, there is a corresponding critical error value in the *quality* specification. When an error component equals the corresponding critical error value the quality is at most e^{-1} or approximately 0.37. Consequently, we should choose these critical error values to correspond to poor quality.

Figure 6 gives an example of critical error values that are used in the next section. The units for temporal *shift*, *jitter*, *res*, and *synch* are in seconds. Measurements for x and y *shift*, *jitter*, and *res* components are relative to $v.clip.x.extent$ and $v.clip.y.extent$ respectively for a *View* v. Measurements for z *shift* and *jitter* are also made relative to $v.clip.z.extent$. All *rate* error components are pure numbers with no units. The numbers in the figure represent the principle author's determination of intolerable error in experiments with a five minute video clip from a basketball game. Comparable values have been reported by other researchers for synchronization error [16], but more experimentation is needed to determine how these values depend on the content, task, and the person who rates the quality.

	shift	rate	jitter	res	synch
VideoDev X	0.1 width	0.1	0.003 width	0.006 width	0.25
VideoDev Y	0.1 height	0.1	0.003 height	0.006 height	
VideoDev T	15	0.5	0.1	0.2	
VideoDev Z	0.2 range	0.8	0.1 range		
AudioDev T	15	0.2	0.0002	0.0003	
AudioDev Z	0.2 range	0.8	0.01 range		

Figure 6: Example critical error values.

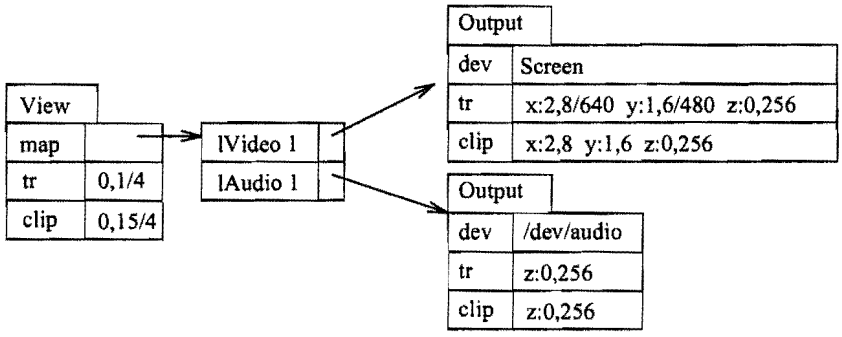

Figure 7: View specification for playback of bicycling video at four times normal rate.

A quality specification q can use the values from Figure 6 for its *calib* and *synchCalib* functions. For example, for any video output p, *q.calib p (T jitter)* would be 0.1 seconds.

6 Using Quality Specifications for Resource Reservation

A multimedia player can generally meet a QOS specification with fewer resources than are needed for a maximal quality presentation. Consider the content specification from Figure 3 and a new view specification shown in Figure 7. Let the quality specification q have the critical error values from Figure 6 and the value 0.75 for $q.min$. The view represents a user request to play the presentation at 4 times the normal rate. The resulting ideal specification then calls for 120 frames/second of video. However, the quality specification only requires that quality exceed 0.75. If all aspects of the presentation were perfect except for video jitter, the quality specification would admit a presentation with jitter less than or equal to 0.029 seconds, which allows the playback algorithm to drop more than five out of six frames. We can derive this result from the predicate in the *QOS* schema as follows.

Let p_v be the video output and i be an interpretation that finds all error components to be zero except for $i.error\ p_v\ (T\ jitter)$. We can be sure that such an interpretation exists if our presentation transports and displays the video data without loss on output devices that match the resolution of the source. Note that the interpretation i considers all timing error to be *jitter* as opposed to *shift* or *rate* errors. Since the exponential functions are equal to one when error is zero, we get:

$$\forall x, y, t : R \bullet 0.75 \leq exp(- \mid \frac{i.error\ p_v\ (T\ jitter)\ x\ y\ t}{q.calib\ p_v\ (T\ jitter)} \mid) \tag{1}$$

Assuming that jitter is always positive, we can substitute the critical value $q.calib\ p_v\ (T\ jitter)$ from Figure 6 and solve for the jitter:

$$\mid i.error\ p_v\ (T\ jitter)\ x\ y\ t \mid \leq -\ln(0.75) * 0.1 \doteq 0.029 \tag{2}$$

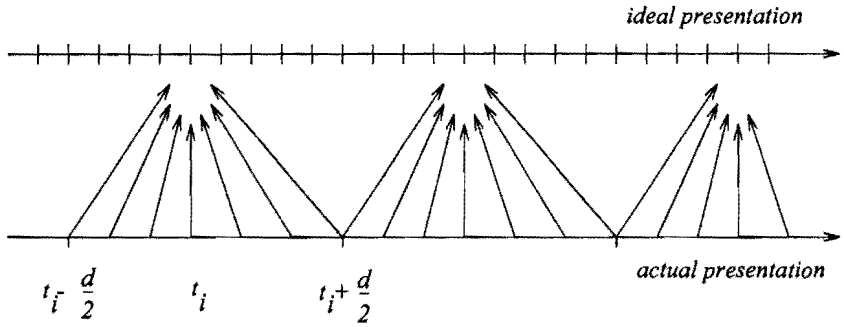

Figure 8: Example mapping from actual presentation times to ideal presentation times. When shift error in an interpretation is zero, all timing error must be attributed to jitter.

Thus, the absolute value of the jitter can be as large as 0.029 seconds. Since all other errors are assumed to be zero, jitter is defined by the error model to be the difference $t_{ideal} - t$ where content displayed by the presentation at time t should have been displayed at time t_{ideal}. As Figure 8 illustrates, if the duration d of the ith frame in a presentation is centered on the ideal time for presentation of that frame t_i then the absolute value of the jitter is always less than $d/2$ seconds. Setting $d/2 \leq 0.029$ and solving for d gives us a maximum frame duration of 0.058 seconds and a minimum frame rate of approximately 18 frames/second.

Analysis of a QOS specification can identify a range of presentation plans that might satisfy the specification as illustrated above. A multimedia player can perform this analysis automatically in response to playback requests. To guarantee that a particular presentation plan will satisfy a QOS specification a player must reserve resources for storage access, decompression, mixing, and presentation processes. The attempt to reserve resources is called an admission test. The admission test may invoke resource reservation protocols for network and file system resources with resource-level QOS parameters derived from the process timing requirements. If the player can not find a presentation plan that both satisfies the QOS requirements and meets the admission test, then the QOS requirements must be renegotiated.

7 Conclusions

This paper has described a new methodology for QOS specification in multimedia systems. The primary contributions of this methodology are the clear distinction between *content*, *view* and *quality* specifications, and the formal definition of presentation quality. Because every component of our QOS specifications has an unambiguous meaning, it is in theory possible to prove the correctness of a presentation plan. These formal QOS specifications allow meaningful requests for end-to-end service guarantees and enable service providers to optimize resource usage. Section 6 gave an informal illustration of how a QOS specification can be used to derive the minimal frame rate for an acceptable presentation.

Our formal definition of presentation quality is based on an *error model* that maps from an actual presentation to an ideal specification. This mapping property ensures that our error model is complete because, as all error components in the model approach zero, the presentation necessarily becomes indistinguishable from the ideal specification. Previous definitions of QOS parameters do not satisfy this completeness criteria. We have proposed an extensible set of error components that are a superset of the presentation QOS parameters suggested by other researchers.

The proposed error model supports modelling of human perception. Each of the error components corresponds to observable artifacts that have a measurable effect on human perception. We have planned experiments to measure these effects. In the meantime, the quality function in Sec-

tion 5 offers a simple model of human perception. Because this function is monotonically decreasing with an increase in any error component, we are assured the existence of a specification that allows only imperceptible errors.

We defined a data model that supports complex audio and video compositions in order to show a concrete example of presentation QOS semantics. More expressive data models exist. The methodology we used to define presentation QOS semantics can be applied to any data model that provides a formal specification of continuous media outputs. Such a specification allows us to measure differences between the specified and actual output values. The error model we described is also only an example. Other, better, error models may exist, but our methodology will still apply.

Another important achievement of our methodology is the recognition that presentation quality should be specified in terms of a subjective error interpretation and not in terms of the presentation mechanism. By requiring only that an acceptable error interpretation exists, our QOS specifications allow a player to choose an optimal presentation mechanism according to current resource costs and availability. Knowledge of a presentation mechanism can be used to prove that an acceptable quality interpretation exists.

We have implemented simple playback systems that make use of this QOS specification method. More work is planned to investigate algorithms for translating QOS specifications into feasible presentation plans.

Acknowledgements.

We would like to thank the referees for helpful comments on our initial submission. Tom Little of Boston University and John Nicol of GTE Labs also gave comments on early versions of this work. Jim Hook of The Oregon Graduate Institute of Science & Technology aided our understanding of the formalisms required.

References

[1] David P. Anderson: Metascheduling for Continuous Media. ACM Transactions on Computer Systems, Vol. 11, No. 3, August 1993, pp. 226-252.

[2] Shanwei Cen, Calton Pu, Richard Staehli and Jonathan Walpole: A Distributed Real-Time MPEG Video Audio Player. To appear in Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video, April 1995.

[3] Geoff Coulson, Gordon S. Blair and Philippe Robin: Micro-kernel Support for Continuous Media in Distributed Systems. Computer Networks and ISDN Systems, Vol. 26, 1994, pp. 1323-1341.

[4] Simon Gibbs, Christian Breiteneder and Dennis Tsichritzis: Data Modeling of Time-Based Media. SIGMOD 94 Proceedings, May, 1994, pp. 91-102.

[5] David Hutchison, Geoff Coulson, Andrew Campbell and Gordon S. Blair: Quality of Service Management in Distributed Systems. Tech. Rep. MPG-94-02, Lancaster University, 1994.

[6] K. Jeffay, D.L. Stone, T. Talley, F.D. Smith: Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, November 1992, pp. 1-12.

[7] Didier Le Gall: MPEG: A Video Compression Standard for Multimedia Applications. CACM, Vol. 34, No. 4, April 1991, pp. 46-58.

[8] T.D.C. Little and A. Ghafoor: Network Considerations for Distributed Multimedia Object Composition and Communication. IEEE Network Magazine, November 1990, pp. 32-49.

[9] P. Lougher and D. Shepherd: The design of a storage server for continuous media. The Computer Journal, Vol. 36, No. 1, February 1993, pp. 32-42.

[10] David Maier, Jonathan Walpole and Richard Staehli: Storage System Architectures for Continuous Media Data. FODO '93 Proceedings, Lecture Notes in Computer Science, Vol. 730, 1993, Springer-Verlag, pp.1-18.

[11] C.W. Mercer, S. Savage and H. Tokuda: Processor Capacity Reserves: Operating System Support for Multimedia applications. Proceedings of the International Conference on Multimedia Computing and Systems, May 1994, pp. 90-99.

[12] K.K. Ramakrishnan, Lev Vaitzblit, Cary Gray, Uresh Vahalia, Dennis Ting, Percy Tzelnic, Steve Glaser and Wayne Duso: Operating System Support for a Video-On-Demand File Service. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, November 1993, pp. 225-236.

[13] Lawrence A. Rowe, Ketan D. Patel, Brian C. Smith, Kim Liu: MPEG Video in Software: Representation, Transmission, and Playback. High Speed Networking and Multimedia Computing, IS&T/SPIE, February 1994.

[14] J.M. Spivey: The Z Notation: A Reference Manual. Second edition, Prentice-Hall International, 1992.

[15] Richard Staehli and Jonathan Walpole: Constrained-Latency Storage Access. Computer, Vol. 26, No. 3, March 1993, pp. 44-53.

[16] Ralf Steinmetz and Clemens Engler: Human Perception of Media Synchronization. Tech. Rep. 43.9310, IBM European Networking Center, 1993.

[17] Hideyuki Tokuda, Yoshito Tobe, Stephen T.-C. Chou and José M.F. Moura: Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network. SIGCOMM '92, 1992.

[18] Gregory K. Wallace: The JPEG Still Picture Compression Standard. CACM, Vol. 34, No. 4, April 1991, pp. 30-44.

[19] Marek Wernik, Osama Aboul-Magd and Henry Gilbert: Traffic Management for B-ISDN Services. IEEE Network, September 1992, pp. 10-19.