Stream-based Versus Structured Video Objects: Issues, Solutions, and Challenges[*]

Shahram Ghandeharizadeh

Abstract

An emerging area of database system research is to investigate techniques that ensure a continuous display of video objects. As compared to the traditional data types, e.g., text, a video object must be retrieved at a prespecified rate. If it is retrieved at a lower rate then its display may suffer from frequent disruptions and delays, termed *hiccups*. This paper describes two alternative approaches to representing video objects (stream-based and structured) and the issues involved in supporting their hiccup-free display. For each approach, we describe the existing solutions and the future research directions from a database systems perspective.

[*]I want to thank Martha Escobar-Molano, Seon Ho Kim, Cyrus Shahabi, and Roger Zimmermann for contributing to the presented material. This paper is to appear as a book chapter in *Multimedia Database Systems: Issues and Research Directions*, Editors: S. Jajodia and V.S. Subrahmanian, Springer Verlag, 1995. This research was supported in part by the National Science Foundation under grants IRI-9222926, IRI-9203389, IRI-9258362 (NYI award), and CDA-9216321, and a Hewlett-Packard unrestricted cash/equipment gift.

1 Introduction

Video in a variety of formats has been available since late 1800's: In the 1870's Eadweard Muybridge created a series of motion photographs to display a horse in motion. Thomas Edison patented a motion picture camera in 1887. In essence, video has enjoyed more than a century of research and development to evolve to its present format. During the 1980s, digital video started to become of interest to computer scientists. Repositories containing digital video clips started to emerge. The "National Information Infrastructure" initiative has added to this excitement by envisioning massive archives that contain digital video in addition to other types of information, e.g., textual, record-based data. Database management systems (DBMSs) supporting this data type are expected to play a major role in many applications including library information systems, entertainment industry, educational applications, etc.

In this study, we focus on video objects and its physical requirements from the perspectives of the storage manager of a database management system. A DBMS may employ two alternative approaches to represent a video clip:

1. **Stream-based**: A video clip consists of a sequence of pictures (commonly termed two dimensional frames) that are displayed at a pre-specified rate, e.g., 30 frames a second for TV shows, 24 frames a second for most movies shown in a theater due to the dim lighting. If an object is displayed at a rate lower than its prespecified bandwidth, its display will suffer from frequent disruptions and delays, termed *hiccups*.

2. **Structured**: A video clip consists of a sequence of scenes. Each scene consists of a collections of background objects, actors (e.g., 3 dimensional representation of Mickey Mouse, dinosaurs, lions), light sources that define shading, and the audience's view point. Spatial constructs are used to place object that constitute a scene in a rendering space while temporal constructs describe how the objects and their relationship evolve as a function of time. The rendering of a structured presentation is hiccup-free when it satisfies the temporal constraints imposed on the display of each object. "Reboot" [Ber94] is an animated Saturday morning children's show created using this approach.

Each approach has its own advantages and disadvantages. The stream-based approach benefits from more than a century of research and development on analog devices that generate high resolution frames. This is because the output of these devices is digitized to generate a stream-based video clip. However, it suffers from the following limitations. First, while humans are capable of reasoning about the contents of a stream-based presentation, it is difficult to design techniques to process the contents of a movie for query processing (e.g., select all scenes of a movie where one car chases another). Second, it is difficult to extract the contents of one stream-based presentation to be re-used in another. To illustrate, with animation, it is difficult to extract Mickey Mouse from one animated sequence to be incorporated in another; typically Mickey Mouse is re-drawn from scratch for the new animation. However, this is not to imply that this task is impossible. For example, the movie "Forrest Gump" incorporates Tom Hanks (the main actor) with different presidents (J. F. Kennedy, L. B. Johnson, and R. Nixon). This was a tedious, time consuming task that required the efforts of: 1) a creative director choosing from amongst the old news clips available on different presidents and selecting those that fit the movie's plot, 2) a skilled actor imagining the chosen scene and acting against a blue background[1], and 3) knowledgeable engineers who incorporated this footage with the old news clips on the different presidents.

A structured video clip eliminates the disadvantages of the stream-based approach because it provides adequate information to support query processing techniques, and re-usability of information. It enables the system to retrieve and manipulate the individual objects that constitute a scene. While structured video is directly usable in both animation and video games that employ animated characters, their use in video clips is limited. This is because there are no devices equivalent to a camcorder that can analyze a scene to compute either its individual objects or the temporal and spatial relationships that exist among these objects. Perhaps, another century of research and development is required before such devices become commercially available. However, it is important to note that once a repository of objects is constructed, the potential to re-use information to construct different scenarios and stories is almost limitless.

In this paper, we describe each of these two approaches in detail (Sections 2 and 3). For each approach, we describe some of the existing solutions, and challenges that remain to be investigated. From a systems perspective, the structured approach has received little attention and requires

[1]A blue background is used because it can easily be eliminated once overlaid with the old news clip.

further investigation. Brief conclusions are offered in Section 4.

2 Stream-based Presentation

Stream-based video clips exhibit two characteristics:

1. they require a continuous retrieval rate for a hiccup-free display: Stream-based objects should be retrieved at a pre-specified bandwidth. This bandwidth is defined by the object's media type. For example, the bandwidth required by NTSC[2] for "network-quality" video is approximately 45 megabits per second (mbps) [Has89]. Recommendation 601 of the International Radio Consultative Committee (CCIR) calls for a 216 mbps bandwidth for video objects [Fox91]. A video object based on HDTV requires a bandwidth of approximately 800 mbps.

2. they are large in size: A 30 minute uncompressed object based on NTSC is 10 gigabytes in size. With a compression technique that reduces the bandwidth requirement of this object to 1.5 mbps, this object is 337 megabytes in size. A repository (e.g., corresponding to an encyclopedia) that contains hundreds of such clips is potentially terabytes in size.

One may employ a lossy compression techniques (e.g., MPEG [Gal91]) in order to reduce both the size and the bandwidth requirement of an object. These techniques encode data into a form that consumes a relatively small amount of space; however, when the data is decoded, it yields a representation similar to the original (some loss of data).

Even with a lossy compression technique, the size of a stream-based video repository is typically very large. For example, the USC instructional TV program tapes approximately 1700 hours of video per semester. With MPEG-1 compression technique that reduces the bandwidth of each video object to 1.5 mbps (unacceptably low resolution), this center produces approximately 1.1 terabytes of data per semester. The large size of these databases motivates the use of hierarchical storage structures primarily by dollars and sense: Storing terabytes of data using DRAM would be very expensive. Moreover, it would be wasteful because only a small fraction of the data is referenced at

[2]The US standard established by the National Television System Committee.

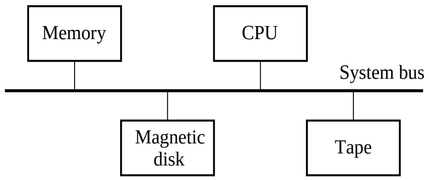

Figure 1: Architecture.

any given instant in time (i.e., some tapes corresponding to particular classes are more frequently accessed than the others, there exists locality of reference). A similar argument applies to other devices, i.e., magnetic disks. The most practical choice would be to employ a combination of fast and slow devices, where the system controls the placement of the data in order to hide the high latency of slow devices using fast devices.

Assume a hierarchical storage structure consisting of random access memory (DRAM), magnetic disk drives, and a tape library [CHL93]. As the different strata of the hierarchy are traversed starting with memory, both the density of the medium (the amount of data it can store) and its latency increases, while its cost per megabyte of storage decreases. At the time of this writing, these costs vary from \$40/megabyte of DRAM to \$0.6/megabyte of disk storage to less than \$0.05/megabyte of tape storage. An application referencing an object that is disk resident observes both the average latency time and the delivery rate of a magnetic disk drive (which is superior to that of the tape library). An application would observe the best performance when its working set becomes resident at the highest level of the hierarchy: memory. However, in our assumed environment, the magnetic disk drives are the more likely staging area for this working set due to the large size of objects. As described below, the memory is used to stage a small fraction of an object for immediate processing and display. We define the working set [Den68] of an application as a collection of objects that are repeatedly referenced. For example, in existing video stores, a few titles are expected to be accessed frequently and a store maintains several (sometimes many) copies of these titles to satisfy the expected demand. These movies constitute the working set of a database system whose application provides a video–on–demand service.

To simplify the discussion, we assume the architecture of Figure 1 for the rest of this section. Using this platform, we describe: 1) a technique to support a hiccup-free display of stream-based objects, and 2) a pipelining mechanism to minimize the latency time of the system.

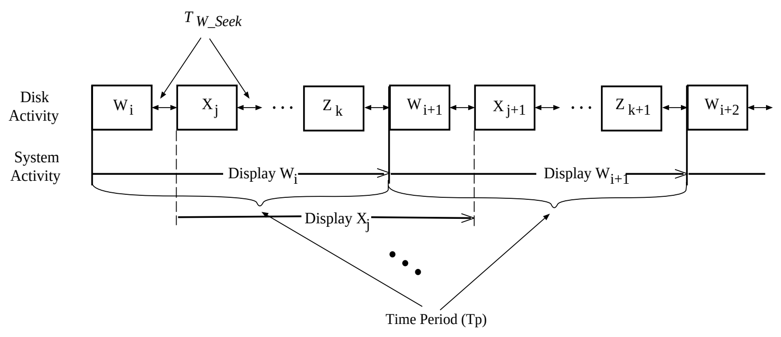

Figure 2: Time Period

2.1 Continuous Display

In this paper, we make the following simplifying assumptions;

1. The disk drive has a fixed transfer rate (R_D) and provides a large storage capacity (more than one gigabyte). An example disk drive from the commercial arena is Seagate Barracuda 2-2HP that provides a 2 Gigabyte storage capacity and a minimum transfer rate of 68.6 Megabits per second (Mbps) [Sea94].
2. A single media type with a fixed display bandwidth (R_C); $R_D > R_C$.
3. A multi-user environment requiring simultaneous display of objects to different users. Each display should be hiccup-free.

To support continuous display of an object X, it is partitioned into n equi-sized blocks: X_0, X_1, ..., X_{n-1}, where n is a function of the block size ($\mathcal{B}$) and the size of X. A *time period* (T_p) is defined as the time required to display a block:

$$T_p = \frac{\mathcal{B}}{R_C} \tag{1}$$

When an object X is referenced, the system stages X_0 in memory and initiates its display. Prior to completion of a time period, it initiates the retrieval of X_1 into memory in order to ensure a continuous display. This process is repeated until all blocks of an object have been displayed.

To support simultaneous display of several objects, a time period is partitioned into fixed-size slots, with each slot corresponding to the retrieval time of a block from the disk drive. The number of slots in a time period defines the number of simultaneous displays that can be supported by

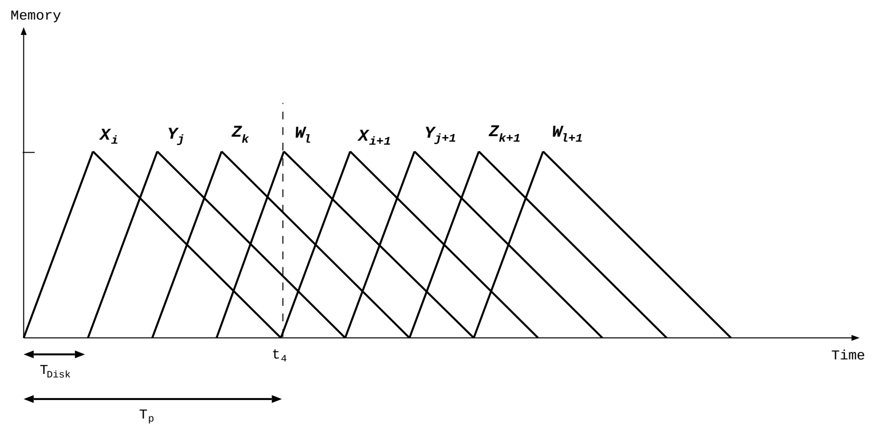

Figure 3: Memory requirement for four streams

the system. For example, a block size of 750 Kilobytes corresponding to a MPEG-1 compressed movie (R_C = 1.5 Mbps) has a 4 second display time ($T_p = 4$). Assuming a magnetic disk with a transfer rate of 24 Mbps (R_D = 24 Mbps) and maximum seek time of 35 milliseconds, 14 such blocks can be retrieved in 4 seconds. Hence, a single disk supports 14 simultaneous displays. Figure 2 demonstrates the concept of a time period and a time slot. Each box represents a time slot. Assuming that each block is stored contiguously on the surface of the disk, the disk incurs a seek every time it switches from one block of an object to another. We denote this as T_{W_Seek} and assume that it includes the maximum rotational latency time of the disk drive. We will not discuss rotational latency further because it is a constant added to every seek time.

To display $\mathcal{N}$ simultaneous blocks per time period, the system should provide sufficient memory for staging the blocks. As described in [NY94], the system requires $\frac{\mathcal{N}\mathcal{B}}{2}$ memory to support $\mathcal{N}$ simultaneous displays (with identical R_C). To observe this, Figure 3 shows the memory requirements of each display as a function of time for a system that supports four simultaneous displays. A time period is partitioned into 4 slots. The duration of each slot is denoted T_{Disk}. During each T_{Disk} for a given object (e.g., X), the disk is producing data while the display is consuming it. Thus, the amount of data staged in memory during this period is lower than $\mathcal{B}$ (it is $T_{Disk} \times R_D - T_{Disk} \times R_C$). Consider the memory requirement of each display for one instant in time, say t_4: X requires no memory, Y requires $\frac{\mathcal{B}}{3}$ memory, Z requires $\frac{2 \times \mathcal{B}}{3}$ memory, and W requires at most $\mathcal{B}$ memory. Hence the total memory requirement for these four displays is $2\mathcal{B}$ (i.e., $\frac{\mathcal{N}\mathcal{B}}{2}$); we refer the interested reader to [NY94] for the complete proof. Hence, if Mem denotes the amount of configured memory for a

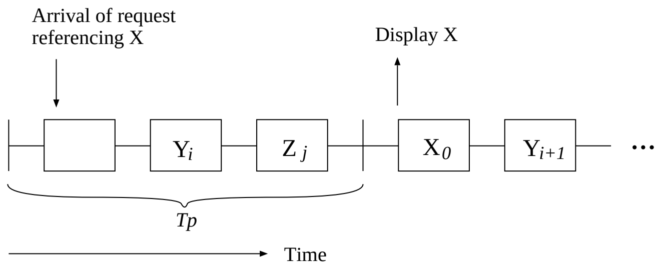

Figure 4: Maximum latency for a request referencing object X

system, then the following constraint must be satisfied:

$$\frac{\mathcal{N} \times \mathcal{B}}{2} \leq Mem \tag{2}$$

To compute the size of a block, from Figure 2, it is trivial that:

$$\mathcal{B} = (\frac{T_p}{\mathcal{N}} - T_{W\_Seek}) \times R_D \tag{3}$$

By substituting $\mathcal{B}$ from Equation 3 into Equation 1 we obtain:

$$T_p = \mathcal{N} \times T_{W\_Seek} \times \frac{R_D}{R_D - (\mathcal{N} \times R_C)} \tag{4}$$

The duration of a time period (T_p) defines the maximum latency incurred when the number of active displays is fewer than $\mathcal{N}$. To illustrate the maximum latency, consider the following example. Assume a system that supports three simultaneous displays ($\mathcal{N} = 3$). Two displays are active (Y and Z) and a new request referencing object X arrives, see Figure 4. This request arrives a little too late to consume the idle slot[3]. Thus, the display of X is delayed by one time period before it can be activated. Note that this maximum latency is applicable when the number of active displays is less than the total number of displays supported by the system ($\mathcal{N}$). Otherwise, the maximum latency should be computed based on appropriate queuing models.

Observe from Figure 2 that the disk incurs a T_{W_Seek} between the retrieval of each block. The disk performs wasteful work when it seeks (and useful work when it transfers data). T_{W_Seek} reduces the bandwidth of the disk drive. The effective bandwidth of the disk drive is a function of $\mathcal{B}$ and T_{W_Seek}; it is defined as:

$$B_{Disk} = R_D \times \frac{\mathcal{B}}{\mathcal{B} + (T_{W\_Seek} \times R_D)} \tag{5}$$

[3]Its display cannot start because it would interfere with the display of object Y, see Figure 4.

Block Size	No Users	Memory Required	Max Latency Seconds (Tp)	Wasted Disk Bandwidth (%)
8 Kilobytes	1	8 Kilobytes	0.042	96.526
16 Kilobytes	3	24 Kilobytes	0.083	93.285
32 Kilobytes	5	80 Kilobytes	0.167	87.416
64 Kilobytes	10	320 Kilobytes	0.333	77.645
128 Kilobytes	16	1 Megabytes	0.667	63.459
256 Kilobytes	24	3 Megabytes	1.333	46.476
512 Kilobytes	31	7.5 Megabytes	2.667	30.273
1 Megabytes	37	18.5 Megabytes	5.333	17.836
2 Megabytes	41	41 Megabytes	10.667	9.791
4 Megabytes	43	86 Megabytes	21.333	5.148
8 Megabytes	44	176 Megabytes	42.667	2.642

Table 1: An example

The percentage of wasted disk bandwidth is quantified as:

$$\frac{R_D - B_{Disk}}{R_D} \times 100 \tag{6}$$

Equations 2 to 6 establish the relationship between: 1) maximum throughput and latency time of a system, and 2) the available memory and disk bandwidth of a system. To illustrate, assume a system with a fixed amount of memory and a single disk drive. Given a desired throughput, one may compute the worst latency time using Equation 4. The theoretical upper bound on the throughput is determined by the transfer rate of the disk drive (R_D) and is defined as $\lfloor \frac{R_D}{R_C} \rfloor$ (the lower bound on this value is 0). Using Equation 3, the size of a block can be determined. The system can be configured with such a block size to support the desired throughput only if it is configured with sufficient amount of memory, i.e., the constraint imposed by Equation 2 is satisfied. Otherwise, the desired throughput should be reduced. This minimizes the amount of required memory, however, it results in a smaller block size that wastes a higher percentage of the disk bandwidth (Equations 5 and 6).

To illustrate these concepts, consider a database that consists of MPEG-1 objects with a bandwidth requirement of 1.5 megabits per second. Assume a disk drive with a maximum seek time of 17 milliseconds, rotational latency of 8.33 milliseconds, and a transfer rate of 68.6 megabits per second. (T_{W_Seek} = 25.33 milliseconds.) Table 1 presents the number of users that can be supported as a function of the blocksize. A small block size (8 kilobytes) has a small transfer rate

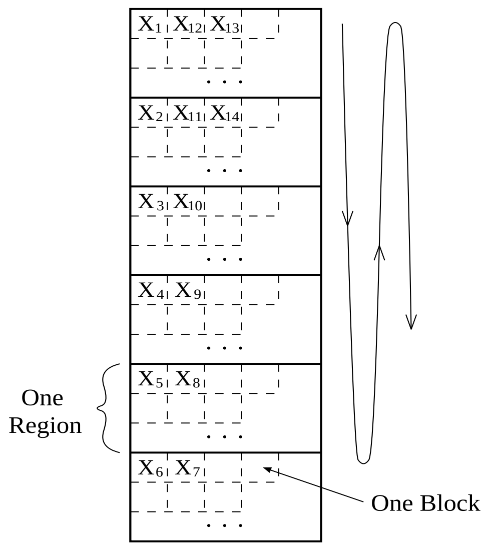

Figure 5: REBECA

and wastes a significant fraction of disk bandwidth. As one increases the block size, the percentage of wasted disk bandwidth drops. This enables the disk drive to support a higher number of users. However, note that this increases the maximum latency time that a user may observe and requires a larger amount of memory from the system.

Seek is a wasteful operation that minimizes the number of simultaneous displays supported by the system. (The disk performs useful work when it transfers data.) Moreover, the seek time is a function of the distance traveled by the disk arm [BG88, GHW90, RW94]. REBECA [GKS95] is a mechanism that minimizes the time attributed to a seek operation by minimizing the distance that the disk head travels when multiplexed among several requests. This is achieved as follows. First, REBECA partitions the disk space into $\mathcal{R}$ regions. Next, successive blocks of an object X are assigned to the regions in a zigzag manner as shown in Figure 5. The zigzag assignments of blocks to regions follows the efficient movement of disk head as in the elevator algorithm [Teo72]. To retrieve the blocks of an object, the disk head moves *inward* until it reaches the center of the disk and then it moves *outward*. This procedure repeats itself once the head reaches the out-most track on the disk. This minimizes the movement of the disk head required to simultaneously retrieve $\mathcal{N}$ objects. To achieve this minimized movement, the display of the objects should follow the following rules:

1. The disk head moves in one direction (either *inward* or *outward*) at a time.

2. During a time period, the disk services requests corresponding to a single region (termed *active region*, R_{active}). In the subsequent time period, the disk services requests corresponding to either $R_{active} + 1$ (*inward* direction) or $R_{active} - 1$ (*outward* direction). The only exception is when R_{active} is either the first or the last region. In these two cases, R_{active} is either incremented or decremented after two time periods because the consecutive blocks of an object reside in the same region. For example, in Figure 5, X_6 and X_7 are both allocated to the last region and R_{active} changes its value after two time periods. This scheduling paradigm does not waste disk space (an alternative assignment/schedule that enables R_{active} to change its value after every time period would waste 50% of the space managed by the first and the last region).

3. Upon the arrival of a request referencing object X, it is assigned to the region containing X_1 (say R_X). The display of X does not start until the active region reaches R_X ($R_{active} = R_X$) **and** its direction corresponds to that required by X. For example, X requires an *inward* direction if X_2 is assigned to $R_X + 1$ and *outward* X_2 is assigned to $R_X - 1$.

REBECA results in a higher utilization of the disk bandwidth, providing for a higher number of simultaneous displays (i.e., throughput). However, it increases the latency time incurred by a request (i.e., time elapsed from when the request arrives until the onset of its display). The configuration parameters of REBECA can be fine tuned to strike a compromise between a desired throughput and a tolerable latency time.

Trading latency time for a higher throughput is dependent on the requirements of the target application. As reported in [GKS95], the throughput of a single disk server (with four megabytes of memory) may vary from 23 to 30 simultaneous displays using REBECA when its number of regions varies from 1 to 21. This causes the maximum latency time to increase from a fraction of a second to 30 seconds. A *video-on-demand* server may expect to have 30 simultaneous displays as its maximum load with each display lasting two hours. Without REBECA, the disk drive supports a maximum of 23 simultaneous displays, each observing a fraction of a second latency. During peak system loads (30 active requests), several requests may wait in a queue until one of the active requests completes its display. These requests observe a latency time significantly longer than a fraction of second (potentially in the range of hours depending on the status of the active displays and the queue of pending requests). In this scenario, it might be reasonable to force each request to observe the potential worst case latency of 30 seconds in order to support 30 simultaneous displays.

Alternatively, with an application that provides a *news_on_demand* service with a typical news clip lasting approximately four minutes, a 30 second latency time might not be a reasonable tradeoff for a higher number of simultaneous displays. In this case, the system designer might decide to introduce additional resources (e.g., memory) into the environment to enable the system to support a higher number of simultaneous displays with each request incurring a fraction of a second latency time. [GKS95] describes a configuration planner to compute a value for the configuration parameters of a system in order to satisfy the performance objectives of an application. Hence, a service provider can configure its server based on both its expected number of active customers as well as their waiting tolerance.

2.2 Pipelining to Minimize Latency Time

With a hierarchical storage organization, when a request references an object that is not disk resident, the system may service the request using the bandwidth of the tertiary storage device as long as: 1) the tertiary storage device is free, and 2) the bandwidth required to support a hiccup-free display of the referenced object ($\mathcal{R}_C$) is lower than the bandwidth of the tertiary storage device ($\mathcal{R}_T$). Indeed, one may envision multiplexing a tertiary storage device that provides a high transfer rate (e.g., Ampex DST [Joh93] with a 116 mbps sustained transfer rate) among several active devices using the paradigm of Section 2.1. However, this might be wasteful due to the significant seek time of these devices (in the order of seconds).

If $\mathcal{R}_C$ is higher than $\mathcal{R}_T$ (i.e., the tertiary cannot support a hiccup-free display of the referenced object) then the object must first be staged on the disk drive prior to its display. One approach might materialize the object on the disk drives in its entirety before initiating its display. In this case, the latency time of the system is determined by the bandwidth of the tertiary storage device and the size of the referenced object. Stream-based video objects require a sequential retrieval to support their display, hence, a better alternative is to use a pipelining mechanism that overlaps the display of an object with its materialization, in order to minimize the latency time.

With pipelining, a portion of the time required to materialize X can be overlapped with its

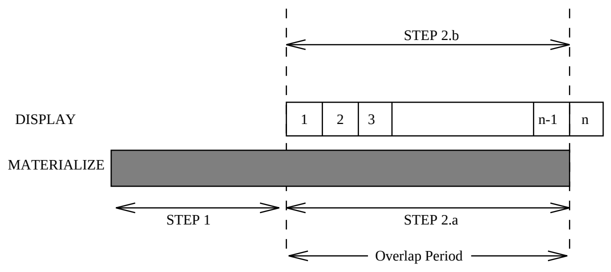

Figure 6: The pipelining mechanism

display. This is achieved by grouping the subobjects of X into s logical slices ($S_{X,1}$, $S_{X,2}$, $S_{X,3}$, ..., $S_{X,s}$), such that the display time of $S_{X,1}$, $T_{Display}(S_{X,1})$, overlaps the time required to materialize $S_{X,2}$; $T_{Display}(S_{X,2})$ overlaps $T_{Materialize}(S_{X,3})$, etc. Thus:

$$T_{Display}(S_{X,i}) \geq T_{Materialize}(S_{X,i+1}) \ for \ 1 \ \leq \ i \ < \ s \qquad (7)$$

Upon the retrieval of a tertiary resident object X, the pipelining mechanism is as follows:

```
1: Materialize the subobject(s) that constitute $S_{X,1}$ on the disk drive.
2: For $i = 2$ to $s$ do
    a. Initiate the materialization of $S_{X,i}$ from tertiary onto the disk.
    b. Initiate the display of $S_{X,i-1}$.
3: Display the last slice ($S_{X,s}$).
```

The duration of Step 1 determines the latency time of the system. Its duration is equivalent to $T_{Materialize}(X) - T_{Display}(X) +$ (one Time Period). Step 3 displays the last slice materialized on the disk drive. In order to minimize the latency time[4], $S_{X,s}$ should consist of a single subobject. To illustrate this, consider Figure 6. If the last slice consists of more than one subobject then the duration of the overlap is minimized, elongating duration of Step 1.

2.3 High Bandwidth Objects and Scalable Servers

There are applications that cannot tolerate the use of a lossy compression technique (e.g., video signals collected from space by NASA [Doz92]). Clearly, a technique that can support the display

[4]Maximize the length of the pipeline.

of an object for both application types is desirable. Assuming a multi-disk architecture, staggered striping [BGMJ94] is one such a technique. It is flexible enough to support those objects whose bandwidth requires either the aggregate bandwidth of multiple disks or fraction of the bandwidth of a single disk. Using the declustering technique of [GR93], it employs the aggregate bandwidth of multiple disk drives to support a hiccup-free display of those objects whose bandwidth exceeds the bandwidth of a single disk drive. Hence, it provides effective support for a database that consists of a mix of media types, each with a different bandwidth requirement. Moreover, its design enables the system to scale to thousands of disk drives because its overhead does not increase prohibitively as a function of additional resources.

In [GDS95], the authors describe extensions of the pipelining mechanism to a scalable server that employs the bandwidth of multiple disk drives. In [BGM95], the authors describe alternative techniques to support a hiccup-free display in the presence of disk failures.

2.4 Challenges

A server may process requests using either a *demand driven* or a *data driven* paradigm. With the demand driven paradigm, the system waits for the arrival of a request to reference an object prior to retrieving it. With the data driven paradigm, the system retrieves and displays data items periodically (similar to how a broadcasting company such as HBO transmits movies at a certain time). The clients referencing an object wait for the onset of a display, at which point, the system transmits the referenced stream to all waiting clients. Each paradigm has its own tradeoffs. With the demand driven paradigm, each request observes a relatively low latency time as long as the number of active displays is lower than the maximum number of displays supported by a system (its throughput). When the number of active displays is larger than the throughput of the system, the wait time for a queue depends on the status of the active requests, the average service time of a request, and the length of the queue of pending requests.

The data driven paradigm is appropriate when the number of active requests is expected to far exceed the throughput of the system (a technique based on this paradigm is described in [OBRS94]). However, with this paradigm, the system must decide: 1) what objects it should broadcast? 2) how frequently should each object be broadcast? 3) when is an object broadcast? and 4) what is the interval of time between two broadcasts of a single object? The answer to these questions are based on expectations. In the worst case, a stream might be broadcast with no client expressing interest in its display. A limitation of this paradigm is starvation: requests referencing unpopular video clips might wait for a long time before the referenced clip is broadcast. Moreover, with this paradigm, multiple clients that share a stream of data may fall out of synchronization with each other every time a user invokes a pause or fast-forward functionality. The system might either disallow such functionalities (as is done with the current broadcasting companies) or implement sophisticated techniques based on resource reservation to accommodate such operations.

A challenging task is to design a system that can switch between these two alternative paradigms (or support both simultaneously) depending on the number of requests requiring service, the throughput of the system, the pattern of reference to the objects, and the quality of service desired by the clients. The precise definition of quality of service is application dependent. For video servers, it may refer to either the functionality provided to a client (e.g., fast-forward, pause) or the resolution of the display[5].

3 Structured Presentation

(The material in this section appeared in [EM95].) As an alternative to a stream-based presentation, a video object can be represented as a collection of objects, spatial and temporal constructs, and rendering features (termed structured presentation). The spatial and temporal constructs define where in the rendering space and when in the temporal space the component objects are displayed. The rendering features define how the objects are displayed.

A *Rendering Space* is a coordinate system defined by n orthogonal vectors, where n is the number of dimensions (i.e., $n = 3$ for 3D, $n = 2$ for 2D). A spatial construct specifies the placement of a component in the rendering space. Analogously, different components are rendered within a time interval, termed *Temporal Space.* For example, if a movie has 30 scenes of 3 minutes each then the temporal space of the movie is $[0, 90]$. Moreover there is a temporal construct specifying

[5]The resolution of a display dictates the bandwidth required to support that display. A system may maintain multiple copies of a video object based on different resolutions and service the different clients using a different copy based on how much they pay.

Figure 7: Three levels of abstraction

the subinterval within the temporal space that should render each scene. For example, a temporal construct for the first scene will specify the subinterval $[0, 3]$.

To illustrate the use of both constructs simultaneously, consider the motion of a rolling ball. The motion is captured by a sequence of snapshots, represented by a sequence of triplets: (the object (i.e., the ball), its positioning, subinterval). Each triplet specifies a spatial and a temporal constraint. In this section, we partition the information associated with a structured video into three layers:

1. *Atomic* objects that define indivisible entities (e.g., the 3D representation of a ball).

2. *Composed objects* that consist of objects constrained using temporal and spatial constructs, e.g., a triplet: (the ball, a position, a subinterval).

3. The rendering features (e.g., viewpoint, light sources, etc.).

Figure 7 shows the different levels of abstraction of a media object and an example of the representation of a scene. Assume that the objective is to describe a character (e.g., Mickey Mouse) walking along a path in the scene. The atomic object layer contains the 3D representations of different postures of Mickey Mouse, denoted by p1, p2, etc. For example, his posture when he starts to walk, his posture one second later, etc. These postures might have been originals composed by an artist or generated using interpolation. We also include the 3D representation of the background (denoted by sc1) in the atomic object layer.

To represent the walking motion, the author specifies spatial and temporal constructs among the different postures of Mickey Mouse. The result is a composed object. The curve labeled **c1** specifies the path followed by Mickey Mouse (i.e., the different positions reached). Each of the coordinate systems describe the direction of a posture of the character. For each posture, a temporal construct specifies the time when the object appears in the temporal space. For example, the point labeled by (p1, 0, 1) indicates that posture p1 appears at time 0 and lasts for 1 second.

To associate the motion of Mickey Mouse to the background, we have the spatial and temporal constructs in the composed objects layer represented by **c2**. The spatial constructs define where in the rendering space the motion and the background are placed. The temporal constructs define the timing of the appearances of the background and the motion. In this example, the background sc1 appears at the beginning of the scene while Mickey Mouse starts to walk (c1) at the 5th second.

Finally, the rendering features are assigned by specifying the view point, the light sources, etc., for the time interval at which the scene is rendered. In the following two sections, we describe each of the atomic and composed objects layers in more detail.

3.1 Atomic Object Layer

This layer contains objects that are considered indivisible (i.e., they are rendered in their entirety). The exact representation of an atomic object is application dependent. In animation, as described in [TT90], the alternative representations include:

1. *wire-frame representation*: An object is represented by a set of segment lines.
2. *surface representation*: An object is represented by a set of primitive surfaces, typically: triangles, polygons, equations of algebraic surfaces or patches.
3. *solid representation*: An object is a set of primitive volumes.

From a conceptual perspective, these physical representations are considered as an unstructured unit, termed a BLOB. These objects can also be described as either:

1. A procedure that consumes a number of parameters to compute a BLOB that represents an object. For example, a geometric object can be represented by its dimensions (i.e., the radius, the length of a side of a square, etc.), a value for these dimensions, and a procedure that consumes these values to compute a bitmap representation of the object. This type of representation is termed *Parametric*.
2. An interpolation of two other atomic objects. For example in animation, the motion of a character can be represented as postures at selected times and the postures in between can be obtained by interpolation. In animation, this representation is termed *In-Between*.
3. A transformation applied to another atomic object. For example the representation of a posture of Mickey Mouse can be obtained by applying some transformation to a master representation. This representation is termed *Transform*.

Figure 8 presents the schema of the type atomic that describes these alternative representations. The conventions employed in this schema representation as well as others presented in this paper are as follows: The names of built-in types (i.e., strings, integers, etc.) are all in capital letters as opposed to defined types that use lower case letters. ANYTYPE refers to strings, integers, characters and complex data structures. A type is represented by its name surrounded by an oval. The attributes of a type are denoted by arrows with single line tails. The name of the attribute labels the arrow and the type is given at the head of the arrow. Multivalued attributes are denoted by arrows with two heads and single value attributes by arrows with a single head. For

Figure 8: Atomic object schema

multivalued attributes, an S overlapping the arrow is used to denote a sequence instead of a set. The type/subtype relationship is denoted by arrows with double line tail. The type at the tail is the subtype and the type at the head is the supertype.

For example, in Figure 8 Parametric is a subtype of Atomic, and it has two attributes: Parameters and Generator. Parameters is a set of elements of any type and Generator is a function that maps a set of elements of any type (i.e., Parameters) into a BLOB.

3.2 Composed Object Layer

This layer contains the representation of temporal and spatial constructs. In addition to specifying positioning and timing of objects, these constructs define objects as composed by other objects. The composition might be recursive (i.e., a composed object may consist of a collection of other composed objects). For example, Mickey Mouse might be represented as a composed object consisting of 3D representation of: a head, two ears, a tail, two legs, etc. The spatial relationship between these atomic objects would define Mickey Mouse.

Spatial constructs place objects in the rendering space and implicitly define spatial relationships

Figure 9: Composed object schema

between objects. The placement of an object defines its position and direction in the rendering space. For example, consider a path from a house to a pond. The placement of a character on the path must include, in addition to its position, the direction of the character (e.g., heading towards the pond or heading towards the house).

A coordinate system defined by n orthogonal vectors defines unambiguously the position and direction of an object in the rendering space. For example, consider a 3D representation of a die. Figure 10 (a) shows three different placements of the die in the rendering space defined by the x-y-z axis. Notice that the position of the die in each placement is the same. But the direction of the die varies (e.g., the face at the top is different for each placement). However, the coordinate systems defined by the red, green and blue axis specifies unambiguously the position and the direction of the die.

Formally, a *Spatial Construct* of a component object o is a bijection that maps n orthogonal vectors in o into n orthogonal vectors in the rendering space, where n is the number of dimensions. Let *o's coordinate system* and *the mapped coordinate system* be defined by the n orthogonal vectors in o and the mapped vectors, respectively. The *placement* of a component object o in the rendering space is the translation of o from its coordinate system to the mapped coordinate system, such that its relative position with respect to both coordinate systems does not change. Note that there is a unique placement for a given spatial construct.

Temporal constructs define the rendering time of objects and implicitly establish temporal

Figure 10: (a) Three different directions for a die, (b) Two atomic objects, (c) A composed object constructed using spatial constructs and the atomic objects in (b).

relationships among the objects. They are always defined with respect to a temporal space. Given a temporal space $[0, t]$, a *Temporal Construct* of a component o of duration d, maps o to a subinterval $[i, j]$ such that $0 \leq i \leq j \leq t$ and $j - i = d$.

A *composed object* C is represented by the set:

$$\begin{array}{ll} \{(e_i, p_i, s_i, d_i) \mid & e_i \text{ is a component of } C, \\ & p_i \text{ is the mapped coordinate system in } C\text{'s rendering} \\ & \text{space defined by the spatial construct on } e_i, \text{ and} \\ & [s_i, d_i] \text{ is the subinterval defined by a temporal construct} \\ & \text{ on } e_i\} \end{array}$$

A composed object may have more than one occurrence of the same component. For example, a character may appear and disappear in a scene. Then, the description of the scene includes one 4-tuple for each appearance of the character. Each tuple specifies the character's position in the scene and a subinterval when the character appears.

The definition of composed objects establishes a hierarchy among the different components of an object. This hierarchy can be represented as a tree. Each node in the tree represents an object with spatial and temporal constructs (i.e., the 4-tuple in the composed object representation: *(component, position, starting time, duration)*), and each arch represents the relation *component of.*

3.3 Challenges

The presented data model is not necessarily complete and may need additional constructs. A target application (e.g., animation) and its users can evaluate a final data model and refine (or tailor) it to obtain the desired functionality. Assuming that a data model is defined, the following research topics require further investigation. First, a final system requires authoring packages to populate the database and tools to display the captured data. These tools should be as effective and friendly as their currently available stream-based siblings. An analogy is the archive of stream-based collections maintained by most owners of a camcorder. The camcorder is a friendly, and yet effective tool to capture the desired data. The VCR is another effective tool to display the captured data. A VCR can also record broadcast stream-based video objects.

Tools to author 3-D objects are starting to emerge from disciplines such as CAD/CAM, scientific visualization, and geometrical modeling (see [Kor94] for a list of available commercial packages). There are packages that can generate a 3-D object as a list of triangles. For example, one can draw 2-D objects using either AutoCAD or MacDraw. Subsequently, a user can interact with these tools to convert a 2-D object into a 3-D one. Finally, this 3-D object is saved in a file as a list of triangles. At the time of this writing, there are two other approaches to author 3-D objects. If the actual object is available, then it can be scanned using a Cyberware scanner that outputs a triangle list. The second method employs volume based point sample techniques to extract triangle lists. With this method, a point sample indicates whether the point is inside or outside of a surface or object (like a CT or MRI might).

Tools to display structured video are grouped into two categories: compilers, and interpreters. A *compiler* consumes a structured video clip to produce its corresponding stream-based video to be stored in the database and displayed at a later time. An *interpreter*, on the other hand, renders a structured video either statically or interactively. A static interpreter [EMGI95] displays a structure without accepting input. An interactive interpreter accepts input, allowing a user to navigate the environment described by a structured object (e.g., video games, virtual reality applications that either visualize a data set for a scientist or train an individual on a specific task). A challenging task when designing an interpreter is to ensure a hiccup-free display of the referenced scene. This task is guided by the structure of the complex object that describes a scenario.

For static interpreters, this structure dictates a schedule for what objects should be retrieved at what time. An intelligent scheduler should take advantage of this information to minimize the amount of resources required to support a display. At times, adequate resources (memory and disk bandwidth) may not be available to support a hiccup-free display. In this case, the interpreter might pursue two alternative paths. First, it may compute a *hybrid* representation by compiling the temporal constructs that exists among different objects to compute streams for these object. In essence, it would compute an intermediate representation of a structured video clip that consists of a collection of: 1) streams that must be displayed simultaneously, and 2) certain objects that should be interpreted and displayed with the streams. We speculate that this would minimize the number of constraints imposed on the display, simplifying the scheduling task. As an alternative, the interpreter may elect to prefetch certain objects (those with a high frequency of access) into memory in order to simplify the scheduling task.

Unlike the interpreter, the compiler is not required to support a continuous display. However, this is not to imply a lack of research topics in this area. Below, we list several of them. First, the compiler must compress the final output in order to reduce both its size and the bandwidth requirements. Traditional compression techniques that manipulate a stream-based presentation (e.g., MPEG) cannot take advantage of the contents of the video clip because none is available. With a structured presentation, the compiler should employ new algorithms that take advantage of the available content information during compression. We speculate that a content-based compression technique can outperform the traditional heuristic based technique (e.g., MPEG) by providing a higher resolution, a lower size, and a lower average bandwidth to support a hiccup-free display. Second, the compiler should minimize the amount of time required to produce a stream-based video object. It may create the frames in a non-sequential manner in order to achieve this objective (by computing the different postures of an object only once and reusing it in all the frames that reference it).

If the term "object-oriented" was the buzz word of the 1980s, "content-based retrieval" is almost certainly emerging as the catch phrase of the 1990s. A structured video clip has the ability to support content-based queries. Its temporal and spatial primitives can be used to author more complex relationships that exists among objects (e.g., hugging, chasing, hitting). This raises a host of research topics: What are the specifications of a query language that interrogates these relationships? What techniques would be employed by a system that executes queries? What indexing techniques can be designed to speedup the retrieval time of a query? How is the data presented at a physical level? How should the system represent temporal and spatial constructs to enable a user to author more complex relationships? Each of these topics deserves further investigation. Hopefully, in contrast to "object-oriented", a host of agreed upon concepts will emerge from this activity.

Finally, the system will almost certainly be required to support multiple users. This is because its data (e.g., 3 dimensional postures of characters, structured scenes) is valuable and, similar to software engineering, several users might want to share and re-use each other's objects in different scenes. Minimizing the amount of resources required to support the above functionality in the presence of multiple users is an important topic. A challenging task is to support interpreted display of different objects to several users simultaneously. This is due to the hiccup-free requirement of an interpreted structured video.

4 Conclusion

Video is a new communication medium shaping the frontiers of the computer technology (both hardware and software). In this paper, we described two alternative approaches to represent this data type: stream-based and structured. For each approach, we described some of its solutions and challenges. We believe that in the near future, the structured approach will gain increased popularity because it provides for an effective user interaction with a repository. Its representation can support virtual worlds where a user becomes an active participants (instead of a passive recipient of information). It will almost certainly have a tremendous impact on educational, scientific, and entertainment applications. Virtual reality environments currently employ this approach for their target application. However, their primary focus has been on graphics (i.e., rendering aspects) and tools to interact with a user. In order for this paradigm to become useful on a day-to-day basis, a significant amount of research and development is required on: 1) its storage and retrieval of data to support user queries and a hiccup-free display, and 2) tools to populate and query a database.

References

[Ber94] S. Bernstein. Techno-Artists 'Tooning Up. *Los Angeles Times, Section F*, November 10 1994.

[BG88] D. Bitton and J. Gray. Disk shadowing. In *Proceedings of Very Large Databases*, pages 331–338, September 1988.

[BGM95] S. Berson, L. Golubchik, and R. R. Muntz. A Fault Tolerant Design of a Multimedia Server. In *Proceedings of ACM-SIGMOD*, 1995.

[BGMJ94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of ACM-SIGMOD*, pages 79–89, 1994.

[CHL93] M. Carey, L. Haas, and M. Livny. Tapes hold data, too: Challenges of tuples on tertiary storage. In *Proceedings of ACM-SIGMOD*, pages 413–417, 1993.

[Den68] P. J. Denning. The Working Set Model for Program Behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[Doz92] J. Dozier. Access to data in NASA's Earth observing system (Keynote Address). In *Proceedings of ACM-SIGMOD*, pages 1–1, June 1992.

[EM95] M. L. Escobar-Molano. Management of Resources to Support Continuous Display of Structured Video Objects. Technical Report 95-616, USC, 1995.

[EMGI95] M. L. Escobar-Molano, S. Ghandeharizadeh, and D. Ierardi. An Optimal Resource Scheduler for Continuous Display of Structured Video Objects. Technical Report 95-602, USC, 1995.

[Fox91] E. A. Fox. Advances in Interactive Digital Multimedia Sytems. *IEEE Computer*, pages 9–21, October 1991.

[Gal91] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, April 1991.

[GDS95] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Object Materialization with Staggered Striping. *Computer Communications*, March 1995.

[GHW90] J. Gray, B. Host, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proceedings of Very Large Databases*, pages 148–162, August 1990.

[GKS95] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Configuring a Single Disk Continuous Media Server. In *Proceedings of the ACM-SIGMETRICS Conference*, May 1995.

[GR93] S. Ghandeharizadeh and L. Ramos. Continuous Retrieval of Multimedia Data Using Parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–669, August 1993.

[Has89] B. Haskell. International standards activities in image data compression. In *Proceedings of Scientific Data Compression Workshop*, pages 439–449, 1989. NASA conference Pub 3025, NASA Office of Management, Scientific and technical information division.

[Joh93] C. Johnson. Architectural Constructs of AMPEX DST. *Third NASA GSFC Conference on Mass Storage Systems and Technologies*, pages 153–162, 1993.

[Kor94] K. Kornbluh. Active data analysis: Advanced software for the '90s. *IEEE Spectrum*, 31(11):57–83, November 1994.

[NY94] R. Ng and J. Yang. Maximizing Buffer and Disk Utilization for News On-Demand. In *Proceedings of Very Large Databases*, 1994.

[OBRS94] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A Low-Cost Storage Server for Movie on Demand Databases. In *Proceedings of Very Large Databases*, 1994.

[RW94] C. Ruemmler and J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, pages 17–28, March 1994.

[Sea94] Seagate. Barracuda family. *The Data Technology Company Product Overview*, pages 1–25, March 1994.

[Teo72] T.J. Teory. Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems. In *Proc. AFIPS Fall Joint Computer Conf.*, pages 1–11, 1972.

[TT90] Nadia Magnenat Thalmann and Daniel Thalmann, editors. *Computer Animation Theory and Practice.* Springer-Verlag, 1990.