Low-Latency Interaction through Choice-Points, Buffering, and Cuts in Tactus

Dean Rubine, Roger B. Dannenberg, David B. Anderson, and Tom Neuendorffer
Information Technology Center
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
Email: {dandb+,rbd+,dba+,tpn+}@andrew.cmu.edu

Abstract

Multimedia streams usually require prefetching and buffering to ensure steady, glitch-free delivery to audio and video displays, but buffering causes undesirable latency. This latency may be manifested as startup delays, glitches, dropouts, and loss of synchronization. In interactive media presentations where there are a small number of choices, alternative streams can be prefetched to reduce latency. This technique is supported by the Tactus system, which manages the computation and synchronization of multimedia data. Tactus offers a systematic approach to prefetching, precomputation, choice points, and synchronous cuts. Tactus consists of an object-oriented client toolkit for media generation and a synchronization server for media presentation.

KEYWORDS: interface, toolkit, multimedia, synchronization, interactive, prefetch, real-time

1 Introduction

Multimedia systems that display continuous media such as audio and video are, for the most part, based on the stream paradigm in which data is continuously moved from source to sink. Along the way, data is buffered to prevent temporary disruptions in computation or communication from affecting the output. This is an appropriate model when media is "canned," that is, prepared in advance. However, one of the primary advantages of digital media is the possibility to control and compute media interactively.

Consider applications such as surrogate travel, help systems, hypermedia systems, and video editors. Stream-oriented media delivery systems work against interactivity in these applications because buffers impose delays. If a user's action should result in a change in the presentation, two responses are possible. In one, the system stops the current stream, creates a new one, prefetches data into the buffers, and starts the new stream. This causes an unwanted break or "glitch" in the presentation as well as a small delay. Another response is to splice the new data in at the source. Assuming the new stream is spliced cleanly, there will be no glitch, but there is still a long delay while the data moves through buffers from source to display.

To improve this situation, data can be prefetched before the user has an opportunity to make a choice. When the choice is made, the data will already be available, and the alternative presentation can be used without delay. This strategy works because the cut is made "close" to the presentation hardware where latency is low and buffering is minimal.

This arrangement imposes the burden of additional complexity on the application developer but results in lower latency and glitch-free presentations. In a system with infinite compute power and zero-latency storage devices, these techniques are unnecessary, but for now, the techniques are advantageous wherever computing, network, and disk latencies are significant considerations. Example sources of latency are CD-ROM-based video storage, network storage servers, operating system scheduling, graphics rendering, network transmission, database queries, path planning, search, garbage collection, and image processing.

We have designed the Tactus system to support prefetching, precomputing, low-latency cuts, and choice points. Our main contributions are:

1. Showing how precomputation and temporal media can be supported by an interactive graphical application framework;

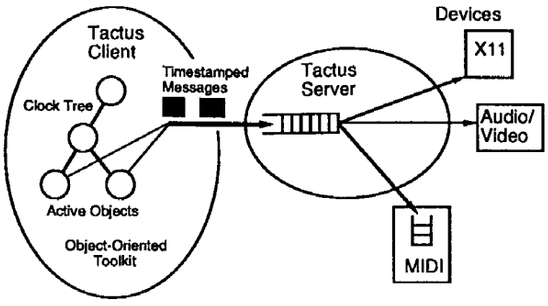

Figure 1: The Tactus System includes an object-oriented toolkit to manage the computation of streams of timestamped messages. The messages are delivered to a server where they are buffered for synchronous delivery to various devices.

2. Providing a generic synchronization server that handles both computed and "canned" media, eliminating complex synchronization code from application programs;

3. A study of "cuts" as a general mechanism for speculative precomputing and prefetching to reduce latency in interactive multimedia; and

4. An implementation of a general cut facility.

The first two contributions have been previously described [1], so we will concentrate on the last two.

Tactus (see Figure 1) includes an object-oriented toolkit to help client programs compute multimedia streams and a synchronization server to manage the synchronized presentation of multimedia data. As the client sends data to the server, the data is marked with timestamps for synchronization and choice points indicating potential cuts. The data is buffered in the Tactus server, which can make last-minute decisions to carry out synchronous cuts on behalf of the client. The "client/server" terminology here is analogous to that used with X11; the client is the application program, and the server delivers output to a display and other media.

The next section describes related work, and Section 3 presents the basic architecture of Tactus in greater detail. Then we discuss a taxonomy of cut types, outlining the parameters that clients use to specify alternatives to the Tactus server. Section 5 describes how prefetching, cuts, and choice points are supported in Tactus, and Section 6 is a step-by-step description of a cut. Section 7 describes more details and future work, then Section 8 describes the current status, and Section 9 presents our conclusions.

2 Related Work

A number of systems have proposed and implemented precomputation or prefetching with timestamping to solve latency problems [2, 3, 4, 5, 6]. Undoubtedly, these techniques are also used within commercial systems such as QuickTime [7] and MPC [8]. One of the features of our work is that it facilitates the *computation* of multimedia streams as opposed to simply playing canned media or scripts. Our system hides many programming details such as scheduling and interleaving computation, timestamping data, synchronizing media, and recovery from glitches, but *without also hiding* the computation steps that actually generate the media.

In speculative prefetching, data is fetched before it is known whether the data is actually needed. Speculative prefetching has been used in file systems, mail readers, and even CPUs. Speculative prefetching of data at choice points and branches to eliminate glitches has been used in several contexts. Multi-deck analog video editing consoles preroll and synchronize video decks to make clean electronic edits. We have heard of CD-Rom-based video games that pre-buffer data for a branching path in order to hide seek time. The DEMON project [5] supports prefetching of alternate paths through stored multimedia documents. To our knowledge, Tactus is the first system to provide and support a generalized flexible mechanism for precomputation and prefetching to reduce latency at discrete edit or choice points in interactive multimedia presentations.

3 Tactus

The Tactus system is intended to support interactive multimedia software. A central focus of Tactus is to hide the problems of latency and synchronization from the client, making it easier to develop multimedia software. The general idea is that clients compute multimedia data in time order, and client computations are scheduled slightly ahead of real time. Precomputed or prefetched data is sent from the client to the server with timestamps indicating the desired display time. Once data reaches the server, it is buffered until the indicated time. Then, it is delivered to output devices such as audio interfaces, video decompression devices, and graphics displays.

3.1 Clients

Client programs are supported by an object-oriented toolkit that includes special objects for temporal media. The programmer's basic model is that the system has no buffers or latency, and so computations are performed only at the instants output is required. In reality, computations are performed ahead of time and output is buffered. Computations are performed by various subclasses of class `Active`. An active object computes media for a given instant in time, then schedules itself to perform another computation in the future [9]. Active objects generate continuous media by running at fixed or variable time intervals.

The programming interface for active objects is simple. To indicate that the next output should occur at `LogicalTime`, the method `RequestKickAt(ActiveObj, LogicalTime)` is called (usually by `ActiveObj`). At the requested (logical) time, the method `Kick(ActiveObj, LogicalTime)` is called by Tactus. This method is overridden to create subclasses of active objects for various purposes. For example, we have a software video active object whose `Kick` method is roughly as follows:

```
Kick(self, time) {
  if(read(InputFile, Buffer, ImageSize)) {
    XPutImage(..., Buffer, ...);
    RequestKickAt(self,
      time + InterFrameTime);
  }
}
```

In this example, `XPutImage` is the standard X11 call to display a raster image, but it is implemented by a special library that appends a timestamp and sends the message to the Tactus Server. The message is buffered in the server, and at the proper time, the message will be forwarded to X11 for display. By fetching and displaying a new frame at regular intervals, a simple but effective software video system is realized.

Active objects are organized into a structure called the "clock tree" (see Figure 2). The time at which an active object asks to run is recursively mapped by clocks on the path from the active object (a leaf) to the real-time object at the root of the clock tree. Thus, a set of active objects can run in a convenient logical time system that is mapped into real time by clock objects. Using clocks, the programmer can vary the rate and offset of logical time with respect to real time.

On every path from a leaf to the root, there must be one and only one `Stream` object. Thus, each active object (leaf) is associated with a single stream

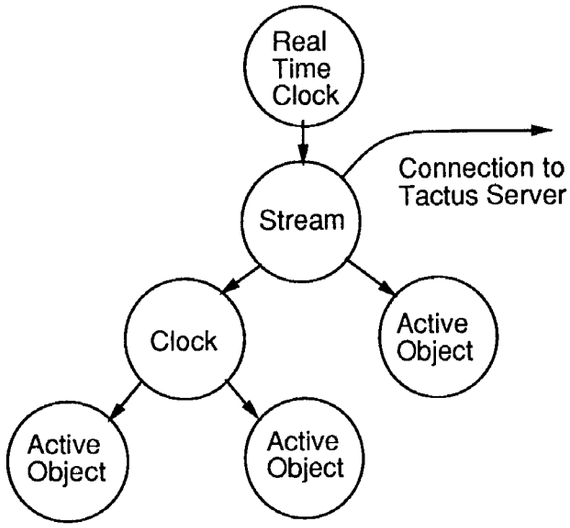

Figure 2: A clock tree: leaves are active objects that perform computation and output data to a multimedia stream. `Clock` nodes serve to map from logical to real time, `Stream` is a subclass of `Clock` which manages an interprocess connection to the Tactus Server, and the `Real Time Clock` object interfaces the clock tree to the operating system timer facility.

object. When an active object outputs data, it is delivered to the Tactus server on a connection managed by this stream object. The data receives a timestamp based on the *requested* kick time, so all output generated by an activation of a `Kick` method receives the same timestamp. Data from many active objects may be interleaved in a single stream, and the clock tree schedules computations so that timestamps are monotonically increasing within a stream. The stream object (a subclass of `Clock`) also shifts logical time ahead of real time so its active objects are scheduled early by a fixed amount, *Latency*, set by the application.

The programmer is supported in several ways First, the clock tree allows the programmer to schedule many active objects. Computation is automatically scheduled in timestamp order. The clock tree also allows the programmer to express event times in any convenient logical time system and automatically maps this time to real time. The stream objects advance the logical time seen by active objects so that data will automatically be precomputed. Finally the data computed by active objects is automatically timestamped before being sent to the Tactus server.

3.2 The Tactus Server

The Tactus Server buffers media streams from one or more clients and presents them synchronously to various multimedia devices, including a window server (X11), an audio server, and a MIDI server (see Figure 1). The Tactus Server runs on the machine with the multimedia devices (this may be different from the machine running the client). The local buffering in the Tactus Server allows continuous media output in the face of momentary delays in computation and network communication. The server is well placed to make decisions supporting stream startup and synchronization. For example, the Tactus Server can be instructed to output multimedia data as soon as buffers reach a low-water mark. Synchronization failures can be handled entirely by the Tactus Server, eliminating complex recovery code in the client.

3.3 Tactus and Cuts

In this section, we present an overview of how cuts work in Tactus. First, note that clients are always computing ahead of real (presentation) time. In a surrogate travel application, for example, the client will reach an intersection before the intersection is displayed. When the client reaches the intersection, it is fairly easy to "fork" multiple streams representing "turn left," "go straight," and "turn right." As the new streams are being started, the user sees an approaching intersection and may select a right turn. The client processes this input and sends a request to the Tactus server to cut from the primary ("go straight") stream to the secondary ("turn right") stream. The Tactus Server makes a precise cut at the intersection so that the user sees a smooth uninterrupted display. Because the "turn right" data is prebuffered as the user approaches the intersection, the turn can be implemented with very low latency.

In general, a cut proceeds in three stages. First, the cut is created when a stream reaches a point where the presentation might branch. The stream becomes the primary stream of the cut, and additional secondary streams, representing alternate presentations, are created, added to the cut, and begin to generate media. Next, a decision to branch to a secondary stream is made, usually in response to user input. The client application requests that the secondary stream be *taken*. The branch to the secondary presentation may not happen immediately, because the secondary stream may not yet be ready, or the cut has been restricted to particular choice points. Finally, when conditions are right, the cut to the secondary stream occurs, and the server and client destroy the primary stream and the other secondary streams.

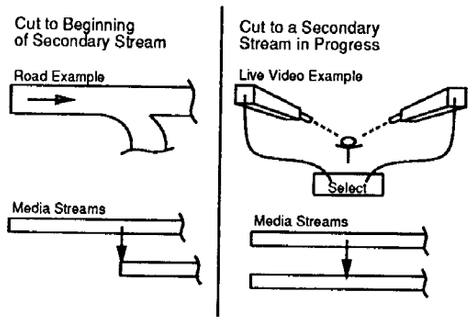

Figure 3: Cuts can be made to the beginning of a stream or to a stream already in progress.

4 A Cut Taxonomy

Cuts are a general mechanism for speculative prefetching and precomputing. *Cuts separate the act of buffering multimedia data from the decision to actually present the data to output devices.* Effectively, this decouples a number of mechanisms for fetching, buffering, and synchronization that are normally bundled into one. Cuts allow presentation decisions to be deferred, leading to a faster system response and thus greater interactivity.

Several attributes are used to specify details of the cut. When a secondary stream is created, an attribute indicates whether or not the cut is to the beginning of the new stream (see Figure 3). If the cut is to the beginning, then the secondary stream starts at the moment the cut occurs. In the surrogate travel example, a "turn" stream would be of this type. In contrast, if the cut is not to the beginning, then the secondary stream is started and synchronized with the primary stream. The secondary stream data is discarded until the cut occurs, at which time the secondary stream data is directed to output devices. Cuts to live video illustrate this type.

The "start synched" cuts potentially have higher overhead, since until the cut is made two (or more) streams must run simultaneously, possibly for a long time. By contrast, a secondary stream in a cut "to beginning" runs until enough of the presentation has been buffered to ensure a smooth transition, and is then blocked. The stream is resumed only if the cut to it actually occurs.

When a request to take a secondary stream is is-

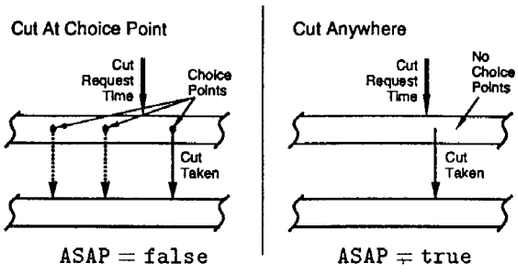

Figure 4: Cuts may be restricted to choice points or allowed anywhere. Either way, the cut may occur at some time later than the requested cut time.

sued, the `ASAP` attribute describes whether the cut can take place at any time or only at discrete choice points (see Figure 4). Choice points indicate where a cut *may* take place from the primary stream to a secondary stream. The client can insert any number of choice points into a stream, and choice points can indicate which secondary streams are allowable targets for the cut. Choice points can be used to place cuts on musical phrase boundaries and at intersections in surrogate travel. Another attribute indicates whether a cut should be delayed until the primary stream would have sent its next packet to a specified device. This allows cuts to take place on video frame boundaries, maintaining a constant video frame rate.

Another attribute (specified when the take is requested) indicates the urgency of the cut. Normally, cuts will not take place unless the secondary stream has been prebuffered up to a low-water mark. An urgent cut will cut to a secondary stream even if the low-water condition is not met. This risks an underflow, but underflow may be more desirable than missing the cut altogether.

5 Cuts in Tactus

Cuts involve new objects and code in both the client toolkit and the Tactus Server. On the client side, a `Cut` object serves as a placeholder in the clock tree. One child of the cut object is the primary stream, and the other children are secondary streams. Figure 5 illustrates a clock tree with a cut object.

When the cut object is created, it establishes a "shadow object" in the Tactus Server, and when children (secondary streams) are added to the cut object, analogous links are made in the Tactus Server. Thus, the portion of the clock tree structure relating streams and cut objects is replicated in the Tactus Server.

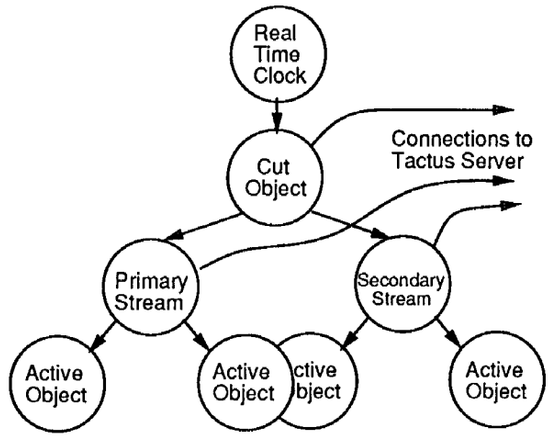

Figure 5: A clock tree with a cut object: only one secondary stream is shown, but multiple secondary streams are possible.

As implied by the tree structure, cuts are recursive in that descendants of the cut object may themselves be cut objects, with their own secondary streams. Consider again the surrogate travel example, and imagine that after the right turn there is a fork in the road (see Figure 6). As the (secondary) right turn stream is prebuffered, the fork will cause another cut object to be created. Figure 6 illustrates the clock tree for this situation. Note that if the user turns right, prebuffering the two choices at the fork will have already started. If the user does not turn right, the secondary stream (including both choices at the fork) can be deleted.

To request a cut to a secondary stream, the method `TakeStream(Cut, TargetStream, ID, Flags)` is sent to the cut object, which then sends a message to the Tactus Server. The server notes that a cut is now pending, and delays the cut if necessary until the following conditions are met:

- A choice point from the primary to secondary stream is reached OR the `ASAP` flag is true.
- The secondary stream has reached its low-water mark OR the `Urgent` flag is true OR the choice point's `GoForIt` flag is true (see Section 5.1).

As mentioned previously, the cut might be further delayed so that it occurs on a video frame or audio buffer boundary.

If a `TakeStream` message misses an intended choice point, it may be preferable not to leave the request pending. Consider Figure 7, which could represent

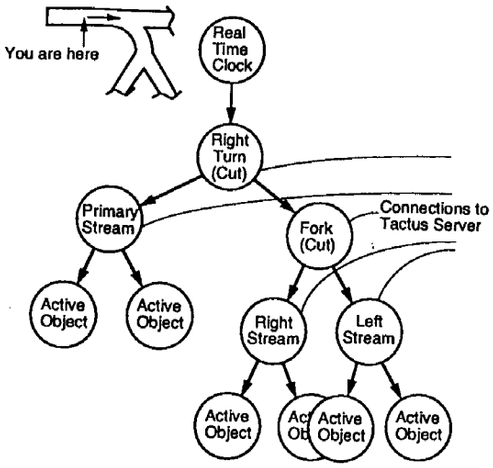

Figure 6: A recursive cut structure: in a surrogate travel application, the user is approaching a right turn, but the client has computed ahead. The right turn forked into two paths, resulting in a two-level cut structure. The objects labeled "Right Turn" and "Fork" are Cut objects. If the Right Turn is not taken, the entire Fork subtree can be deleted.

paths for surrogate travel or a musical vamp with several exit points. If a requested cut arrives too late to be acted upon, the cut remains in effect and will be taken on the next lap around the oval. This may be undesirable, so two mechanisms are provided to avoid this problem. First, there is an `UnTakeStream` message that cancels a previously issued `TakeStream`. Second, the `TakeStream` may specify a choice point identifier (`ID`). Identifiers are unique, so the cut can only apply to a particular choice point. A special `ID` value matches any choice point.

When all conditions are met, the cut is performed and a message is sent from the server to the client describing the cut. This message is handled by the corresponding cut object, which typically destroys the primary stream and other secondary streams and then removes itself from the clock tree.

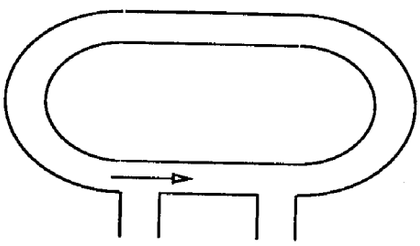

Figure 7: A looping path illustrating a single cut with two choice points. If a cut request arrives too late to be acted upon, the request may be canceled. Otherwise, the cut will be taken on the next lap.

5.1 Choice Points

Choice points are generated by active objects and sent as ordinary timestamped stream messages to the Tactus Server. A choice point specifies a cut object, a target stream (or `AnyStream`), a choice point identifier, and a `GoForIt` flag. The identifier gives the choice point a unique name that can be specified in the `TakeStream` request. `GoForIt` indicates that a requested cut should be made even if the secondary stream's low-water mark condition is not met. Essentially, this flag says it is better to risk a glitch than to miss the cut altogether. This has the same effect as the `Urgent` flag in the `TakeStream` message, but in this case the flag can be specified on a per-choice-point basis.

Whenever a choice point is encountered by the server, a message is returned to the client indicating that a cut was or was not taken. If taken, the target stream is indicated; otherwise, the reason for not taking the cut is indicated. This information allows the client to destroy secondary streams that are no longer useful.

A special case is the last choice point to a particular stream. If a cut is not taken at this choice point, a cut to the secondary stream will never be taken, so it can be destroyed. A special `LastChoicePoint` method is provided so that the application program can easily inform Tactus which is the last choice point. The default Tactus behavior is to destroy secondary streams when their last choice point is not taken, and to destroy cut objects when there are no more remaining secondary streams. Thus, stream cleanup is fairly automatic.

5.2 Time and Synchronization

A key to making clean predictable cuts is the idea that all media data is timestamped. Timestamps, sometimes combined with offsets, provide a specification of when data should be presented. Because of the need to precompute data and timestamps, Tactus must deal with a number of different time systems. There are two time offsets that are used:

- *Latency* is the (per-stream) amount of real time by which client data is precomputed. A typical value is 2 seconds.

- *Glitches* is the (per-stream) amount of time by which the presentation has fallen behind due to startup delays and buffer underflow.

There are three basic "types of time" or time systems used in Tactus:

- *StampTime* is used for the timestamps on messages sent to the Tactus Server. *StampTime* + *Glitches* is the real time at which data should be presented to the user. Note that timestamps start at the current time, not at zero.
- *RunTime* is the ideal real time at which data is computed, that is, the time at which a `Kick` message is sent to an active object. *RunTime* is typically earlier than *StampTime* to achieve precomputation.
- *KickTime* is the time an active object "thinks" it is. It is specified in `RequestKickAt` messages and passed in the corresponding `Kick` messages (ignoring the mapping to local time units). Above streams, the *KickTime* corresponds to *RunTime*, that is, the requested time specifies real time. Below streams, *KickTime* corresponds to *StampTime*, meaning that `Kick` messages are delivered early to support precomputation.

Using these definitions, we can now describe how the client and server use timestamps for synchronization and cuts. We will ignore the additional complexity of logical time systems created and managed by clock objects. In practice, logical times are simply the composition of linear mappings that can be easily inverted.

The first problem is, given timestamped data, when should the data be displayed by the Tactus Server? Ideally, the display time exactly matches the timestamp. In practice, the stream will take some time to start, and underflows may occur, causing delays. The cumulative delay of a stream is called *Glitches*, and the presentation time is therefore *StampTime* + *Glitches*.

The second problem is: when an active object has requested a kick at *StampTime*, when should the `Kick` message be sent? The formula is:

$$RunTime = (StampTime + Glitches) - Latency$$

Since *StampTime* + *Glitches* is the presentation time, this formula precomputes the data by *Latency*.

Now consider a cut to the beginning of a secondary stream. By convention, the first timestamp of the secondary stream will be the time at which the stream is created; call it t_2. Now suppose a cut is taken from timestamp t_1 in the primary stream, whose *Glitches* value is g_1. We want to adjust *Glitches* of the secondary stream, g_2, such that its beginning will be presented at the time of the cut, i.e. $t_1 + g_1 = t_2 + g_2$. Solving for g_2, we obtain $g_2 = t_1 + g_1 - t_2$.

To perform a cut to a stream in progress, the two streams must have matching timestamps. This is achieved by setting the *Glitches* parameter of the secondary stream to *Glitches* of the primary stream. Whenever the primary stream glitches, both the primary and secondary streams are updated with the new value of *Glitches* so that they remain in lock-step.

6 An Example

Let us return to the surrogate travel example shown at the left of Figure 3. Initially, there would be one stream, perhaps with video and audio active objects. When the right turn is reached, the client creates a cut object as shown in Figure 5. A secondary stream representing the right turn is created and linked to the cut object as shown. A choice point is sent as part of the primary stream to indicate a possible cut to the secondary stream at the road intersection.

Tactus will now be buffering two independent streams. Suppose the user indicates a right turn. The client processes the user input and sends a `TakeStream` message to Tactus. If the message arrives before the choice point is reached, Tactus will continue outputting the primary stream until the choice point is encountered. Then, it will check that the secondary stream has filled to its low-water mark. If so, the cut is taken, and data is presented from the secondary stream buffer.

A message is returned to the client indicating that the secondary stream started and sets the *Glitches* value for that stream. By default, the client runtime system destroys the cut object and the primary stream, leaving the secondary stream immediately below the real time clock object, ready for the next choice.

If the cut is not taken, a "choice not taken" message is returned when the choice point is encountered. The client may destroy the secondary stream or leave it in place for use later.

7 Further Issues and Future Work

Some device drivers such as video decompression systems introduce additional latency, and drivers for

time-critical data such as MIDI may offer an additional layer of buffering of timestamped data. For these devices, it is desirable for the Tactus Server to dispatch data to the devices ahead of the display time. The Tactus Server maintains a separate time offset for each device.

Once data is dispatched to a device by the Tactus Server, it cannot be revoked. In the case of MIDI data, that data is timestamped and sent in advance, with the MIDI driver itself responsible for accurately-timed final dispatch. Thus, much like a Tactus client, the main loop of the Tactus server dispatches some packets in advance of real presentation time. An output thread in the server then dispatches packets early to those devices, like MIDI, that accept timestamped data. (The output thread, which runs at a higher priority than the rest of the Tactus Server, also dispatches data at its designated time to devices that respond immediately.) One noticeable effect of advance dispatch in the server is that there is a time interval, equal to the maximum amount of time that the server will dispatch data early, during which the server is committed to a presentation before it is displayed. Currently, input within 100 milliseconds of a choice point cannot cause a cut to be taken at that choice point.

Cuts from one aggregate stream to another require some correspondence between the two aggregate streams. For example, when cutting between stereo audio streams, the left and right channels must be preserved rather than swapped. In our current system, this problem is solved by sending each channel to a specific device, but systems with logical devices may require a more elaborate scheme.

It is up to the client (application) to initiate cuts based on user input. This causes some difficulty because when the user responds to a presentation at time T, the client is already computing the presentation for time $T + Latency$. Consider the problem of mapping mouse coordinates to moving targets. By the time mouse input arrives at the client, the objects will have moved and may no longer even exist.

In general, the client must keep a data structure that reflects history in order to deal with input. When the user selects "turn right," the client must know which intersection is being approached on the display. Graphical user interfaces commonly maintain a two-dimensional map from screen space to graphical objects. Buffered temporal media require the client to consider the time dimension as well. Tactus helps by timestamping incoming events. Although a complex "hot-spot temporal history" mechanism could be developed, we think that in practice most control buttons and icons will be static, and the mapping problem will not be difficult.

For example, we implemented an application (described below) that allows the user to click on moving objects in a video clip. For each moving object, the path of an invisible moving button was specified as timestamped updates in a multimedia scripting language. A simple lookup operation on the script enabled the positions of the buttons at a given time in the past to be determined.

When a cut takes place, it may be desirable to output some (re)initialization messages to devices, especially when cutting to a stream in progress. Initialization messages could be treated as an independent stream.

Another issue is scheduling. When secondary streams are computed, we do not want the additional resource utilization to cause glitches in the primary stream. While the primary stream must keep up with real time, secondary streams do not have this requirement in the cut-to-beginning case. It is only necessary for the secondary stream to be sufficiently buffered by the time the cut takes place. A promising scheduling policy is to give highest priority to primary streams, although other policies might be desirable to avoid starving secondary streams. A global policy module could compute which `Kick` message to send next based on the stream and timestamp associated with the request.

A related topic concerns network requirements between client and server. While Tactus is perfectly capable of utilizing networks that give no guarantees on packet transfer time (at the cost of an occasional glitch when buffers underflow), it can certainly benefit from protocols that allow network resources to be reserved (*e.g.* [10]). However, traffic between Tactus client and server may be bursty. In particular, at stream startup time, as well as when sending secondary streams for a cut, clients become temporarily compute bound, sending packets as fast as possible. It would be unfortunate if a real-time network protocol was unable to give clients spare network capacity when available, given the benefits to the user (such as faster startup time) that result.

In our current model, client applications send to the Tactus server timestamped packets of uninterpreted data, which the server forwards at the correct time. We have found that this model makes it difficult to provide some kinds of immediate feedback in response to user input. For example, in the case of surrogate travel, it would be desirable to reflect the user's choice in the video as the intersection

approaches. This would be relatively easy to do if the system allowed graphics to be superimposed over video. Barring that, it is difficult to affect the video already cued for the primary stream. Assuming the secondary streams all begin at the intersection, the application cannot indicate the user's choice before the intersection is reached.

One solution is the use of "start synched" secondary streams, each illustrating a different choice as the intersection approaches. The various streams containing alternatives begin to run several seconds before the intersection, and the cut occurs immediately after the user chooses, before the intersection is reached. This gives the desired effect, although multiple streams of nearly identical video merely to given more rapid feedback seems like excessive overhead. In our solution, the primary stream contains a command to blit the contents of an off-screen window over the video after each frame. Normally, the off-screen window is clear and the blit has no effect. When user input is received, graphics are sent immediately (without buffering) to the offscreen window, which is copied to the video output to provide feedback. A more general model, which allowed the buffered device data to be parameterized, and allowed changes to the parameters to affect data already buffered, would make it easier to provide immediate feedback.

Tactus is a flexible system, but it is only useful to programmers. Further work is needed to build higher level tools such as scripts, interface builders, and new authoring tools for temporal interfaces.

8 Current Status

At present, the Tactus server runs under RT Mach, Mach 3.0, and AIX, and the Tactus Client Toolkit is an extension to the Andrew Toolkit [11]. Tactus supports interactive software video, graphical animation, audio, and MIDI. Client toolkit objects have been implemented, as have three prototype applications that demonstrate cuts.

One application (see Figure 8) shows a spreadsheet-like display in which each column represents a section (*e.g.* verse, chorus, bridge) of a piece of music. In the absence of user input, the currently playing section will keep repeating itself. For each section, the user can choose which instruments are used, what parts they play, accompanying animations, and other parameters. After specifying these, the user clicks to indicate the section that will be played next. A cut is cued for the new section, and occurs when the currently playing section reaches an appropriate choice

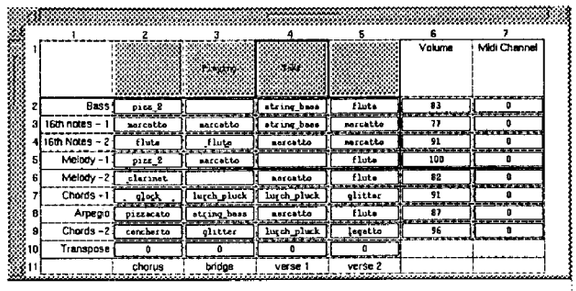

Figure 8: One window of an application that allows parameters of each of four musical sections to be specified, and cuts to each section cued.

point. Among other things, the choice point mechanism makes it simple to indicate that the places in the verse where a chorus is allowed to begin are different than the places where the bridge is allowed to begin.

Another application, mentioned above, allows the creation of interactive hypermedia, in which the user may click on moving objects in a video to cause branches in the presentation. Each stream (containing video, audio, and other media) is described in a simple scripting language. During authoring, the user may draw a rectangle over an object in the video displayed by a script (the primary stream), and move and resize the rectangle, representing an invisible button, to follow that object as it moves. Slow playback (implemented easily via Tactus clocks) allows easy object tracking. A secondary script is specified for each button created. The primary script is automatically modified to include the rectangle motion commands, and references to the secondary scripts. The Tactus cut mechanism ensures that when one of the invisible moving rectangles is clicked upon, the presentation of the appropriate secondary stream is fast and glitch free.

9 Summary and Conclusions

Our research is aimed at computation-oriented interactive multimedia systems. We have implemented an object-oriented framework that helps programmers schedule and create timestamped multimedia data streams. Data can be a mixture of computed and "canned," discrete and continuous information. The Tactus Server buffers and synchronizes multiple streams. Because computation and communication loads vary in practice, data is precomputed and buffered to avoid glitches in presentations.

Given that a certain amount of buffering (and therefore delay) is necessary, we have designed a mechanism to enhance interactivity and reduce latency. The mechanism is based on identifying discrete choice points, precomputing or prefetching alternative presentations (choices), and making synchronous cuts from one media stream to another. Our primary contributions are the use of cuts as a general mechanism for speculative precomputation and the specification and implementation of a cut facility within Tactus.

We believe the techniques described here are useful for an interesting class of multimedia systems. Applications include simulations, surrogate travel, help systems, music, games, and interactive fiction. These techniques are specifically useful for reducing the observed latency of local disks, remote video servers, and discrete computations.

The use of these techniques does not necessarily imply greater buffering or latency than that seen in more conventional systems. For example, even video teleconferencing systems, optimized for low latency, can exhibit appreciable delays, and similar timestamping and buffering techniques have been used [12]. The latency between computation and presentation in our system should be only as long as the worst-case computation or data access time that the system needs to hide. We expect this to be roughly in the range of 0.1 to 5 seconds. Below 0.1 seconds, precomputation is not likely to provide substantial benefits. As latency becomes very high, these techniques break down because more and more resources are needed for speculative computation.

Acknowledgments

This work was sponsored by the IBM Corporation. Joe Newcomer was a valuable contributor to the Tactus and cut mechanism design. We would also like to thank Jim Zelenka and Kevin Goldsmith for work on Tactus.

References

[1] R. B. Dannenberg, T. Neuendorffer, J. M. Newcomer, D. Rubine, and D. Anderson, "Tactus: Toolkit-level support for synchronized interactive multimedia," *Multimedia Systems Journal*, vol. 1, no. 2, pp. 77–86, 1993.

[2] D. P. Anderson and R. Kuivila, "Accurately timed generation of discrete musical events," *Computer Music Journal*, vol. 10, pp. 48–56, Fall 1986.

[3] D. P. Anderson and G. Homsy, "A continuous media i/o server and its synchronization mechanism," *Computer*, pp. 51–57, October 1991.

[4] T. D. C. Little and A. Ghafoor, "Spatio-temporal composition of distributed multimedia objects for value-added networks," *Computer*, pp. 42–50, October 1991.

[5] D. New, J. Rosenberg, G. Cruz, and T. Judd, "Requirements for network delivery of stored interactive media," in *Third International Workshop on Network and Operating System Support For Digital Audio And Video*, pp. 147–153, IEEE Computer and Communication Societies, November 1992.

[6] L. A. Rowe and B. C. Smith, "A continuous media player," in *Third International Workshop on Network and Operating System Support For Digital Audio And Video*, pp. 334–344, IEEE Computer and Communication Societies, November 1992.

[7] P. Wayner, "Inside QuickTime," *Byte*, vol. 16, p. 189, December 1991.

[8] T. Yager, "The multimedia PC: High-powered sight and sound on your desk," *Byte*, vol. 17, p. 217, Feb. 1992.

[9] K. Kahn, "Director guide," Tech. Rep. MIT AI Laboratory Memo 482B, MIT, December 1979.

[10] D. Ferrari, A. Banerjea, and H. Zhang, "Network support for multimedia: A discussion of the Tenet approach," Tech. Rep. TR-92-072, Computer Science Division, University of California at Berkely, November 1992.

[11] A. J. F. Palay, M. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neuendorffer, Z. Stern, M. Bader, and T. Peters, "The Andrew toolkit - an overview," in *Proceedings USENIX Technical Conference*, pp. 9–21, USENIX, Winter 1988.

[12] K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith, "Adaptive, best-effort delivery of digital audio and video across packet-switched networks," in *Third International Workshop on Network and Operating System Support For Digital Audio And Video*, pp. 1–12, IEEE Computer and Communication Societies, November 1992.