HARD-DISK RECORDING AND EDITING OF DIGITAL AUDIO

James A. Moorer
Sonic Solutions

Hard-disk recording of digital audio has moved from the domain of university research projects into the mainstream of commercial professional audio. This paper attempts to define a vocabulary for discussing the differences between different approaches to hard-disk recording, and to categorize these approaches. In the process of so doing, a number of related topics will be discussed. These include different techniques for applying amplitude envelopes, various disk scheduling algorithms, an analysis technique for disk scheduling algorithms, RAM buffering techniques and constraints, and the subject of sound file systems as different from computer file systems, the problem of hard-disk load and unload time, the possibility of background processing, optical disks, and others.

WHITHER HARD-DISK EDITING?

In the days of manual editing, tape-based editing was essentially random-access, since new material could be inserted at any position in the tape. In film and sound-effects editing, this is still the case. The "trim-bin" becomes the random-access storage and retrieval device: every different sound (or picture) has a different hook in the trim bin (or around the room) and the editor can swiftly retrieve any piece of sound and splice it into place.

With editing of all-digital tapes, generally we require the material to be copied linearly from one tape to the other. This is especially critical for DAT and other tapes that cannot tolerate splicing. The editor must proceed from the beginning to the end in a strictly linear fashion. Since digital audio tape can tolerate copying without generation loss, it is possible to assemble the data in pieces, then assemble the pieces into the finished product. The problem with this two-step approach is that it doubles the time required to complete a project.

With background load and unload (defined and discussed below), the hard-disk editor combines the efficiency and random-access of trim-bin editing with the advantages of digital media. It need not take any more of the editor's time than that required to make the edits. Thus we see that the point of the hard-disk editor is to allow the editor the flexibility and efficiency of razor-blade editing with the advantages of digital audio. If the hard-disk editor does not accomplish this, it is hard to imagine what the point of it would be.

TYPES OF HARD-DISKS

Hard-disks can be categorized by their size and their external electrical interface. Sizes are 14", 8", 5.25", 3.5" and 2.5" at this time. Interfaces include SMD, ESDI, SCSI, and others. These are distinguished by a number of features, but the one we will mention here is the location of logical-to-physical sector number mapping. With SMD and ESDI, the location of the sector is given in physical terms: head number, cylinder number, and sector within the track. This triplet (cylinder, head, sector) is called the physical *sector* address. This means that the driver or the interface that is *external* to the disk must prepare these numbers. This generally involves a translation from a *logical* sector number to a *physical* sector address. Logical sector numbers are generally a contiguous set of integers from zero to the last used sector of the drive. With SCSI drives, there is a built-in controller that includes logical-to-physical sector number translation and some amount of buffering.

The manufacture of the recording surfaces of the disks is not perfect: each disk comes from the manufacturer with a list of surface defects on that disk. This means that it is not possible to write on every sector on the disk: some sectors are not usable and must be avoided. The responsibility for doing this is generally delegated to the same process that converts logical sector numbers to physical sector addresses, since it is simple to skip a sector here and there in this process. The point is, however, that contiguity in the domain of logical addresses does not necessarily mean contiguity in the physical addresses. In fact, with many SCSI drives, there is no way to find out what the exact mapping is.

The point of the above discussion is to show that the designer of an audio hard-disk recording system must first make a decision about what kind of disk to use, and must then deal with the consequence of this decision. By chosing a format like SCSI, the designer relinquishes the opportunity to do fine disk scheduling based on knowledge of the characteristics of the disk, such as was done in [1], since one never knows when the SCSI controller will decide to skip a sector.

<u>ABOUT EDITING</u>

For the purpose of discussion, we may say that editing consists of sequencing and overlapping segments of sound. Using the current industry parlance, we will call the silence between sounds *black* (by analogy to the world of video) or alternately, *digital zero* . This is not a sound that you can get from a microphone: it must be produced artificially by setting all the samples in a region to zero. It is an important concept, since it is customary to separate cuts on a compact disk with black. Even the best vinyl records had some amount of irreducible surface noise between the cuts.

In general, a sound is either placed next to another sound, or next to black. In either case, some amount of gain control is necessary to prevent clicks and pops at splice points. For the purpose of this discussion, we will say that *every* edit must have a *fade* of some kind associated with it. This may be a fade-in or a fade-out, when the sound is surrounded by black, or may be a cross-fade when two sounds are adjacent and overlapping. Note that a fade of zero duration can be termed a *butt-cut*, by analogy to a butt-joint in carpentry.

Additionally, we may say that one *trims* an edit point, meaning that more or less of one of the sounds is used, or we can do a *slide edit* in which the relative synchronization of two adjacent sounds is maintained while more of one sound is taken, and the same amount less of the other sound is taken. Let us just add here that in our experience in the field, most of the time in editing is taken trimming edit points to produce the final results.

The record of a sequence of edits can be called an *edit decision list*, or more simply, an *edit list.* When we listen to part or all of our edited sound, we can be said to *audition* the edit list. When we audition the sound, the hard-disk system must *perform* the edit list. That is, it must synthesize, in one way or another, the sound specified by the editors instructions.

IMPLEMENTATION OF FADES

Once an editing decision is made, the designer of a hard-disk editor must decide how to perform the fade. There are several ways of doing this that are currently in use. We will attempt to describe some of them:

The simplest one might be called the *memoryless fade.* This is where the fade is done once, generally with a physical slider in real-time as the sound is played, and the resulting audio is recorded on another device. We call it "memoryless" because the only way to trim the edit or the fade is to do it over again.

The next kind we might term a *precomputed fade.* Here, the edit list is scanned, and all fades are noted. The sounds for each of the edits is read into memory (typically dynamic RAM), and the host computer applies the fade (that is, the gain control) to the sounds that are present during the fade. When the edit list is performed (*i.e.*, played), the audio stream is switched from a direct copy of the data on the hard disk to the audio data in memory that has the fade applied to it. After the fade is completed, one switches back to the direct disk data. This technique was reported in [2].

The most popular kind, as judged from its presence in the commercial marketplace, is the *prestored fade.* This is similar to the precomputed fade, but the computed fade is then written back onto the hard disk. To perform the edit then simply requires switching from the data on one part of the disk to another part of the disk, since the faded sounds are already present. This was used in [3].

The last kind we will discuss here might be termed a *reconstruction fade.* In this case, the fade is computed in real time, so the original, direct disk data is forwarded to some kind of digital signal processing device (DSP) that calculates the gain for each sample of the incomming signals and multiplies the gains times the signals and sums the resulting signals being faded in and faded out.

It is helpful to compare these different choices in terms of the system resources and the effects on the operator. The memoryless fade is useful for tape-based editors, since the operation is then applied in real-time as the material is copied from the source tape(s) to the destination tape. The problem is, of course, to make a single change in an edit at the beginning of the tape often involves copying the entire tape, which can be a time-consuming process.

With the precomputed fade, it is possible that only a certain number of fades can be stored in the memory, and thus you have hard, physical limits to the amount of fades that can be done this way before it is necessary to copy the sound off to secondary storage (digital audio tape of some kind). It also requires a sizeable memory buffer that may or may not be already included in the system.

The prestored fade places minimal requirements on system memory, and so can be done with quite small systems. It is not possible, however, to do a real-time edit with prestored fades, since the fade must be computed in advance and stored back on the disk. Furthermore, it takes roughly the same amount of time to compute and store the edit as it will take to play it. Thus a 90-second fade will take 90 seconds (or more) to precompute. If you trim the edit point by even one sample, you must wait another 90 seconds to hear the results. Additionally, each edit made occupies space on the audio disk. If it is necessary to keep several versions of a particular edit around for the client, each version occupies space.

The reconstruction fade overcomes the previous objections by being synthesized in real time. They are essentially instantanious, since the computation is done while the sound is being played and not before. It is also possible to make a real-time fade, since it just involves the manipulation of the gain that is being applied. It does, however, place greater demands on the system. For instance, for the duration of a cross fade, both the sounds being faded out and the sounds being faded in have to be transferred from the disk. This effectively requires twice the data rate from the disk for the duration of the fade. In certain configurations, this may not be available. It requires what might be seen as 100% excess capacity. Indeed, it might be a difficult decision as to whether a system designer should implement an N-channel system with reconstruction fades, or a system with twice the number of channels with one of the other fade types. The other requirement for reconstruction fades is some kind of DSP capability, either in hard-wired form or in programmable signal processors.

The other problem with reconstruction fades is that only one fade can be occuring at a time. Although it is a rare thing to make an edit in the middle of another edit, it still does occur. In a system with reconstruction fades, the sound must be re-recorded onto the disk before another edit can be made that overlaps a previous edit. One can argue that this simply makes explicit a process that was implicit in the case of a prestored fade.

Probably the best method would be to combine reconstruction fades with prestored fades. We could introduce the concept of *freezing* a fade, which would mean playing it through and saving it back on the disk. Presumably, then, one could also *thaw* a fade for reediting. The only problem is that the edits can only be undone in the order that they were made. If you freeze an edit, then do another edit on top of it, it is meaningless to think of changing the first edit without entirely undoing the second edit. In general, once you start putting edits on top of edits (that is, putting a splice point in the middle of the fade portion of a previous edit), you lose the ability to trim the earlier edits independently.

We must also ask about what functions are useful for fades. The most popular fade in the commercially available hard-disk editors is the linear fade. When used as a cross-fade, this is what we will call a *gain-balanced* fade. This refers to the face that if the signals being cross faded are identical, then a gain-balanced fade results in no change in amplitude of the signal over the duration of the fade. The condition for a fade being gain-balanced is that the fade-in is simply one minus the fade-out.

Note that on some systems, one does not do cross-fades, but instead must juxtapose a signal being faded in on one audio stream with a signal being faded out on another audio stream with no automated way of assuring that they occur at the same time. This is rather unfortunate. Most editors consider a cross-fade to a fundamental editing entity, perhaps from an older way of doing things (*i.e.,* manually splicing tape together). There is no reason that computer programmers can't allow editors to work this way.

If the material being spliced together is different (or more precisely, if it is statistically uncorrelated), then the gain-balanced fade does not produce uniform loudness over the duration of the splice. In this case, an *energy-balanced* fade should be used. This kind maintains a constant energy across the fade. This is especially important when editing together applause. Editors tend to use long fades with applause, and any dip in the loudness during the fade becomes quite apparent.

Needless to say, in musical material, there are all degrees in between perfectly correlated (*i.e.,* identical) and uncorrelated sounds, so that all degrees between gain-balanced and energy-balanced need to be provided.

DISK SCHEDULING

Given an edit list that is to be performed, how should we transfer the sound into the buffer memory? The simplest method is just to transfer the sectors on demand. That is, each sector is transfered, in the order that it is to be performed, into the buffer memory somewhat before it is to sound. Since we are generally working in at least stereo, and possible in multi-channel audio, this is not strictly possible: the disk only transfers one data stream at a time, yet both left and right channels must be presented simultaneously. Many systems solve this problem by interleaving 2 or more channels on the disk. That is to say that in any given sector, the first sample goes to channel 1, the second sample to channel 2, and so on. This has the positive side effect of requiring the minimum disk bandwidth (and consequently the least expensive hardware). It does not allow, however, one channel to be slipped (shifted in time) with respect to the other easily. In general, it is more flexible to store each channel separately on the disk and then combine them before auditioning, but this requires enough more bandwidth that it may not be possible in some cases.

If we take a pure demand strategy, then we must face the possibility of an end-to-end seek on the disk between any two transfers. There is a tradeoff between how much data is read at each disk position and the ultimate efficiency attainable. For instance, in many modern SCSI drives, the rotation rate is 16.7 ms, and the end-to-end seek time is, say, 26 ms. This means that if you transfer the amount of data stored on 1.56 tracks each time you seek, you can achieve a 50% efficiency even with an end-to-end seek after *every* transfer. On the average, the performance will be much better than this.

The ultimate data rate attainable will be bounded by the amount of data on a single track, multiplied by the rotation rate. If we take the data on a track to be about 30,000 bytes, we get 1.8 MBytes/second. Note that this is somewhat lower than the published specifications (2.2 to 2.4 MBytes/second), but this is the actual maximum data rate sustainable. The higher numbers appear not to take sector overhead into account properly. At 48 kHz, 16-bit samples, that yields 18.75 channels of data as the theoretical upper bound for the transfer off a modern SCSI disk. Note that at this rate, even the 1.2 GByte disks would be completely emptied in 11 minutes. It would be reasonable, say, to set the transfer size at 2 tracks (a bit over 60k bytes). That would achieve somewhat more than a 50% efficiency, and would allow more than 8 channels of audio to be transferred off a standard SCSI drive in real time, with an end-to-end seek after each and every transfer, and with every channel stored on a different, discontiguous, file. At this rate, the 1.2 GByte disk would hold about 26 minutes of 8-track material. This is a somewhat more respectable amount for a single disk. One could imagine a system with 4 disks providing a reasonable amount of space for editing.

The above analysis is a worst-case analysis, using some relatively simple disk models. It is clear from the above discussion that there is a tradeoff between the size of a single transfer and the maximum efficiency attainable in the presence of end-to-end seeks. We will call the amount of data transferred at a time to be a *transfer unit*, or *TU*. Needless to say, the use of larger and larger TU size increases efficiency, but requires increasing amounts of buffer memory as well. Figure 1 shows how the maximum efficiency attainable is related to the total size of the buffer memory. *This figure is intended to be suggestive only, it is not intended to be exact.*

The problem with larger and larger TU size is that storing and transfering short bursts of sound becomes less and less efficient. With a TU of 2 tracks, each TU is about 2/3 of a second of monaural sound. This is already larger than some kinds of edits, such as some sound effects (pistol shots, for example), and single notes in musical sequences. The exact size of the TU is a tradeoff between the desire for more and more efficiency in storage and transfer, and the desire for dense editing of very short bits of sound. This tradeoff has to be made based on the expected usage of the system.

In hard-disk editing, the entire edit list is always known before the play is started. This is in contrast to the sampling-synthesizer model, where the sequence of keys that will be depressed by the performer is not known beforehand (unless the sequence has been pre-recorded). It then makes sense to ask whether it helps us to use this information to determine the order in which disk transfers should be initiated. As an example, let us say that we need to transfer track 1 first, then we will need data from track 1000, and then last we will need data from track 500. If we follow a strict demand-based strategy, we would transfer them in the order they will be used (auditioned). For this example, we would transfer track 0, followed by track 1000, followed by track 500. If sufficient buffer memory is available, however, it would clearly be more efficient to transfer track 0, followed by track 500, followed by track 1000. Thus by looking one transfer ahead, we can see that reordering two of the transfers results in an increase in efficiency. We might imagine, then, that looking through the entire edit list, it should be possible to produce the optimal order of doing the transfers. Indeed, this is possible. This is the approach taken in [1]. There is a tradeoff between the ultimate efficiency achievable and the amount of buffer memory that is available. It would not be possible to perform the example noted above without having memory available to hold the data from track 500 while the data from track 1000 was being auditioned.

In general, it is neither efficient nor desirable to look through the entire edit list. Often the calculation required to produce an exactly optimal schedule is prohibitive. We may then optimise a lesser problem, which would be to look N transfers ahead. We might call this the *optimisation horizon*. The point of doing this optimisation is either to transfer more channels of data, or to be able to use a lesser amount of buffer memory. Otherwise, there is no point at all. Straight demand schedule may be perfectly adequate for many applications.

To do any optimisation at all, we must have some idea of how much time it takes to do a seek. We will assume that the seek time depends only on the number of cylinders, and not on exactly which cylinder we start on. A typical seek curve is shown in as the middle curve in Figure 2. Physically, you can think of the head accelerating to some maximum speed, coasting at that speed, then decelerating. The straight line at the origin would be achieved if the acceleration were instantaneous, so that the head simply coasts to its destination and stops abruptly. The lower curve is a parabola, which represents constant acceleration followed by constant deceleration with no coasting. The actual curve lies above both of these. It should be clear from this discussion that it is more efficient to do a long seek than a short one, since the time it takes to accelerate is significant for a short seek and becomes less significant for a long seek.

The following discussion and observation is due to Mont-Reynaud [4]. The key point is that the curve is convex. From this property alone, and by use of the triangle inequality, we can derive a bound for a sequence of seeks, and we can also derive the worst case performance. To derive this bound, we will assume that there are a sequence of `N` seeks of length `dj`. Let `D` be the total distance spanned (`= sum dj`).

```
S(D) < N S(D/N)
```

This says that it is always faster to seek from one end of the disk to the other than it is to make `N` stops along the way. This may seem obvious, but it is only a consequence of the convexity of the curve and of the non-zero acceleration time. The point of this is that the worst-case pattern for a sequence of seeks that span the entire disk is attained when all the seeks are of equal length. We now have a basis for comparing strategies. Suppose, for instance, that we needed to decide between a strategy that did the transfers in order, then did a single seek back to cylinder 0. We might want to compare this to one that did half the transfers in order, then the other half in reverse order. We might call these the *rewind* strategy versus the *back-and-forth* strategy. Since the worst case is achieved for a series of `N` seeks of equal length, we can note that the back-and-forth strategy actually achieves the worst-case bound, and consequently the rewind strategy with its one long seek cannot achieve the worst-case bound, and must be faster. We can then bound any sequence of `N` transfers by the worst-case bound, which would be for `N` equal-length seeks across the disk.

Let us now repeat the above discussion of efficiency. We noted that 50% efficiency could be achieved by making the TU transfer time equal to the end-to-end seek time. If our event horizon is N transfers, then the equivalent statement would be that the 50% efficiency point could be reached by making the TU transfer time equal to the time required to seek (1/N) of the way across the disk. This is clearly a much smaller number (but not quite (1/N) smaller). Thus by the above argument, a TU consisting of, say, two tracks of data (somewhat over 60k Bytes) with an event horizon of only 4 transfers could achieve well over 8 channels of throughput. In fact, the number comes out to be about 14 channels of sustained audio transfer.

Note that in the above discussion, we have not taken rotational latency into account at all. Since SCSI disks do not generally allow you to know the rotational position of the disk, we must assume that the disk is always in the worst position. This simply adds one rotation to each seek (in the worst case). This reduces the above number of 14 channels down to about 12 channels of continuous transfer.

If you have the freedom to spread the file system across two separate disks, then a wonderful optimisation occurs [5]. The effect of seek latency can be *totaly* eliminated. If you make the TU transfer time to be greater than or equal to the maximum end-to-end seek time, then you can always arrange to be doing a seek on one drive while transferring from the other. This effectively allows you to transfer at the maximum disk rate (1.8 MBytes/second) all the time. The only problem with this scheme is a maintenance problem: the reliability of the system is now the *product* of the reliabilities of the two disks separately, and that if one of the disks goes down, the entire system is down. In each disk has a separate file system, then only the material on that disk is lost in case of a failure.

Since we have shown a way to guarantee 12 channels of transfer off a commonly-available SCSI disk, one might wonder what we should do with all this *excess* channel capacity. Clearly we should use this capacity to address the single most difficult problem with hard-disk editing systems, and that is the *hard disk load/unload time*. The problem is that to edit, say, one hour of material, you must wait one hour (at least) to load up the hard disk, and when you are done, one more hour to unload the disk. Since digital audio tape recorders can only operate in real time and no faster, we are constrained to do the load and unload in real time. Of course, there are computer-type storage devices that can operate faster than real time, but these cannot be used as a medium of exchange between studios or for CD mastering, since not enough studios use these devices. They can be used within a single studio for backup and archival purposes. It is only a short leap of imagination to arrive at the conclusion that this excess channel capacity might be used for background load and unload of the hard disk. This allows one to start work on one project as soon as the first material for that project is on the disk. In fact, with two digital audio tape machines, one could simultaneously load material, dump material, and work on material all simultaneously.

As mentioned previously, it is a difficult design decision to say that the system will be used for N-channel editing with simultaneous background load and unload, rather than a 2N or 3N-channel editing system with no simultaneous load and unload. Each system designed takes this tradeoff differently.

There is a limitation of 8 devices on a single SCSI bus. Since the host itself takes up 1 slot, a maximum of 7 disks can be attached. We might say from the above arguments that the channel capacity of a single SCSI bus is about 12 channels of audio. For larger scale multi-channel systems, multiple SCSI busses must be used. This leads to an interesting problem of channel and disk allocation. If we have, say, a 16-channel system, we will probably use 2 SCSI busses. As one records new material, the data coming in on channels 1 to 8 will go onto the disks on one SCSI bus and the data on channels 9 to 16 will go onto the disks on the other SCSI bus. What happens, then, if the operator decides to edit material onto all 16 channels that come entirely from the disks one one particular SCSI bus? The channel capacity for a single SCSI bus will clearly be exceeded and it will not be possible to perform the edit list. The only solution for this is to duplicate some material on both SCSI busses. Since it is not possible to tell beforehand which sounds the editor will choose, this has to be done at the time the edit is requested. This is an annoying nuscience, but there is no particularly good solution for this.

Again, if reconstruction editing is used, there will be momentary bursts of additional channel capacity while overlapping material is fetched from the disks.

BUT WHAT ABOUT OPTICAL DISKS?

The most popular optical disk in the world today by huge factors is the compact disk. It is, however, not erasable. The most popular erasable optical disk right now is the magneto-optical (M-O) drive. These are made by a number of manufacturers. The problem with these devices is not so much their long seek times, since we showed above that this can be entirely offset by increasing the size of the buffer memory. The main problem is that the writing process must be done in 3 passes: the erase pass, the write pass, and the verify pass. Under certain circumstances, the erase pass can be largely eliminated by keeping the unused portions of the disk erased so they are always ready for use. This, of course, makes it impossible to implement a two-stage deletion policy (such as is used in the Macintosh "trash-can" metaphor), since any deleted file must be erased immediately so that it will be ready for recording. The verify pass can not be eliminated, but it can be absorbed into the write process. Several of the manufacturers are now working on one-pass M-O writing which should be available in the next few years. Without this, the M-O disk consumes virtually all the SCSI bandwidth just to write stereo audio. This being the case, it can not even be done as a background process without taking even longer, since it would interfere with foreground editing. Even as an archival medium, it is not clear why one would not choose a DAT (or some other helical-scan device) over an M-O disk, since it records and plays in real time, it is readily available on the marketplace, and the medium is quite cost-effective. In time, with one-pass writing, the M-O disk may well become the preferred medium of exchange between studios, but at present, it does not measure up to the requirements of modern digital audio recording.

We should point out here that there is one significant difference between most optical disks and hard disks. That is that the optical disks often use dual-mode head positioning. For instance, one commercial M-O disk uses a galvanometer mirror for positioning below 100 tracks which is very fast, and then the head/mirror assembly is moves, which is relatively slow. This produces a non-convex seek curve. Thus other allocation techniques (such as clustering) may be more crucial for these devices.

FILE SYSTEMS FOR AUDIO

One of the large decisions the designer of a hard-disk editing system must make is what file system to use. The easiest choice is to use the file system of the host computer. This is very convenient, since sound files may be handled just like any other files on the disk, plus you get all the vendor's aids for disk backup and scavanging. The problem is, of course, that the host computer's file system is designed for normal computer activities, like word processing and spread sheets, and is not tuned for the specific requirements of audio.

What are the special requirements of audio? First and foremost is that transfers tend to be large (11.52 MBytes per minute for stereo at 48 kHz), and that they *must* be done in real time. Most operating systems are not set up for this kind of access. As was noted above, the efficiency of transfer can be greatly increased by making the TU size relatively large. There is no operating system on the market that directly gives you the flexibility to choose the TU size to be, say, two full tracks of data.

Unfortunately, it is even more complicated than that: not all files on the sound disk are sound files. There is still the need for relatively small files. For instance, it is not unreasonable to think of keeping edit lists on the sound disk itself. Thus there must be at least two classes of sounds: the first class is allocated in large units for real-time sound transfer, and the other class is allocated, say, one sector at a time.

If you do not have the freedom to choose the TU size, then you are at the mercy of the operating system to allocate the sound files. In this case, checkerboarding of the disk quickly becomes a significant problem. This sharply limits the number of channels that can be reliably played. Even with the argument above that the modern hard disk can easily support 12 or more channels of audio, this gets reduced to between 2 and 4 channels when a computer operating system does the allocation.

Some hard-disk editing systems do not allow the user to see the waveform. If it does allow the user to edit against the waveform, then some further difficult decisions must be made. For instance, to view 1 minute of stereo 48 kHz material would require reading 11.52 MBytes of data. It would take more than 5 seconds just to transfer this from the disk, much less format it for display. Consequently, it is important to compute and store certain reductions of the signal for display purposes. It is reasonable to store a series of reductions, starting, say, at 16x and going up to as high as 1 million times. The display routines can then simply choose the highest level of reduction that still fills the screen fully. At a reduction of 1 million times, one can view an entire CD (1.3 GBytes of data) in the same time that it takes to view a single cut, or to view 20 milliseconds of sound. The use of display reductions makes the display time of the waveform roughly constant, regardless of the amount of sound being viewed.

The reduction process can either be done during the recording, or out of real time after the sound is recorded. To do it during the recording requires some amount of DSP horsepower that may not be available in the less expensive systems. It does, however, reduce productivity quite a bit if we require a pass over the data, since this can usually not be done any faster than real-time. Note that this is another reason to have the file system capable of storing sector-allocated files as well, since some of the display reductions will be quite short.

There are other reasons to compute reductions of the signal as well. For instance, if one wants to play the sound at faster than real time (like 2x or 6x forwards or backwards), then you must store filtered reductions of the signal, since the bandwidth of the disk will be quickly exceeded trying to call all the data off the disk.

The point of the above discussion is to point out that a sound file is really a cluster of different files that carry different data. The operator, of course, always wants to deal with it as a single file, so that when it is deleted, all the reduced files and other information is deleted along with it. Likewise, when recorded, all the other information should flow into existence along with it. Generally, system designers have chosen to lump all this information into a single, large, inhomogeneous file so that it can be treated as a single entity. If the designer has the freedom to redesign the file system, then certain optimisations can be made. For instance, the various reductions can be interleaved on the disk so that they can all be written at record time in a single transfer, and so that as the user zooms in and out in the waveform, head motion is minimized. The file can still be presented to the editor as being a single entity, even though it is really multiple files.

In audio editing, when a project is finished, one must archive the data for the project so as to reclaim the disk space. Likewise, if some further work must be done on the project, it is necessary to load it back on the disk. Note that this differs a little bit from computer disk backup and restore, since with the computer system, you generally back up either the entire disk, or maybe one or two files. In the studio, the concept of the "project" is a bit better defined, and can often even be associated with a specific work-order number. It is a bit of management help for the computer to take on some of the burdon of tracking a project through the system. If a list is maintained of which sounds and edit lists are associated with a specific project, then it is a simple matter to archive a project entirely, without affecting any of the other projects that are on the disk. Similarly, it becomes simple to restore a project entirely at some later date.

CONCLUSIONS

We have discussed most of the aspects of hard disk editing. Any such system must include (1) a storage strategy, whether it is on a computer file system or a special-purpose file system. The special-purpose system allows the designer a number of optimisations that are generally not available on standard computer file systems. (2) a transfer strategy, which orders the sectors as they come off the disk. This can vary from something as simple as taking them strictly in time order, to something as complex as examining the entire edit list before the first transfer is made. Generally, some kind of transfer horizon that looks N transfers into the future is adequate. (3) a fade, crossfade, and other gain control. This may be done in real time as the edits are performed, or they may be computed beforehand. Some combination of these two strategies is probably the best. (4) Ancillary services, such as project management, archiving, display reductions, and so on.

In general, there are ways of eliminating most of the disadvantages of the non-removable medium (such as background load and unload), and the gains in productivity availble by being able to try out many different combinations in a relatively short time lead to an increase in efficiency in the modern commercial music studio.

REFERENCES

[1] Abbott, Curtis W. "Efficient Editing of Digital Sound on Disk," JAES v32, #6, p394 (1984)

[2] Ingebretsen, Robert B., Stockham, Thomas G., Jr. "Random-Access Editing of Digital Audio," JAES v32, #3, p114 (1984)

[3] Freed, Adrian. "Recording, Mixing, and Signal Processing on a Personal Computer," Proceedings of the AES 5th International Conference *Music and Technology*, p158, 1987

[4] Bernard Mont-Reynaud, Private communication, 1990.

[5] Knuth, Donald C. "The Art of Computer Programming, Vol III", Addison-Wesley, 1973

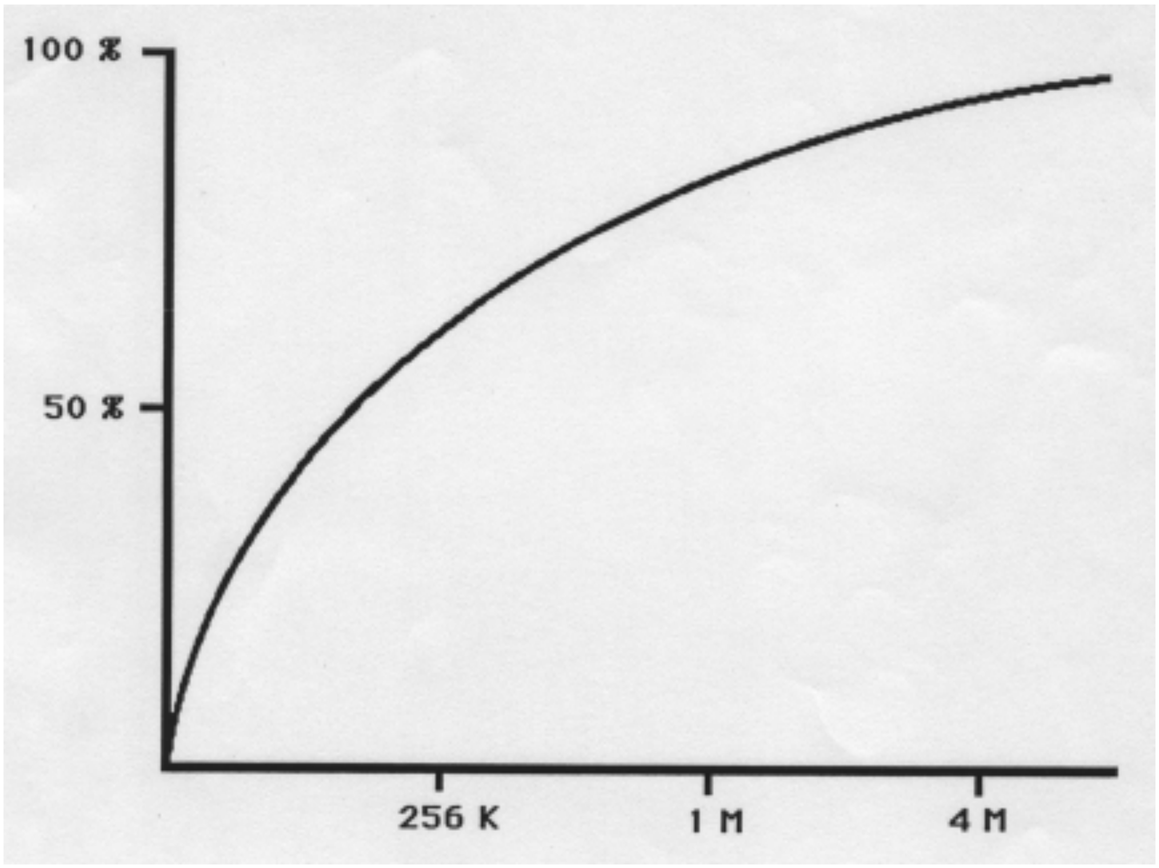

Figure 1 - This shows the increase in efficiency of transfer as the size of buffer memory is increased. We assume that the TU size is allowed to grow as well. This is meant to be suggestive, and that the exact numbers should not be taken literally.

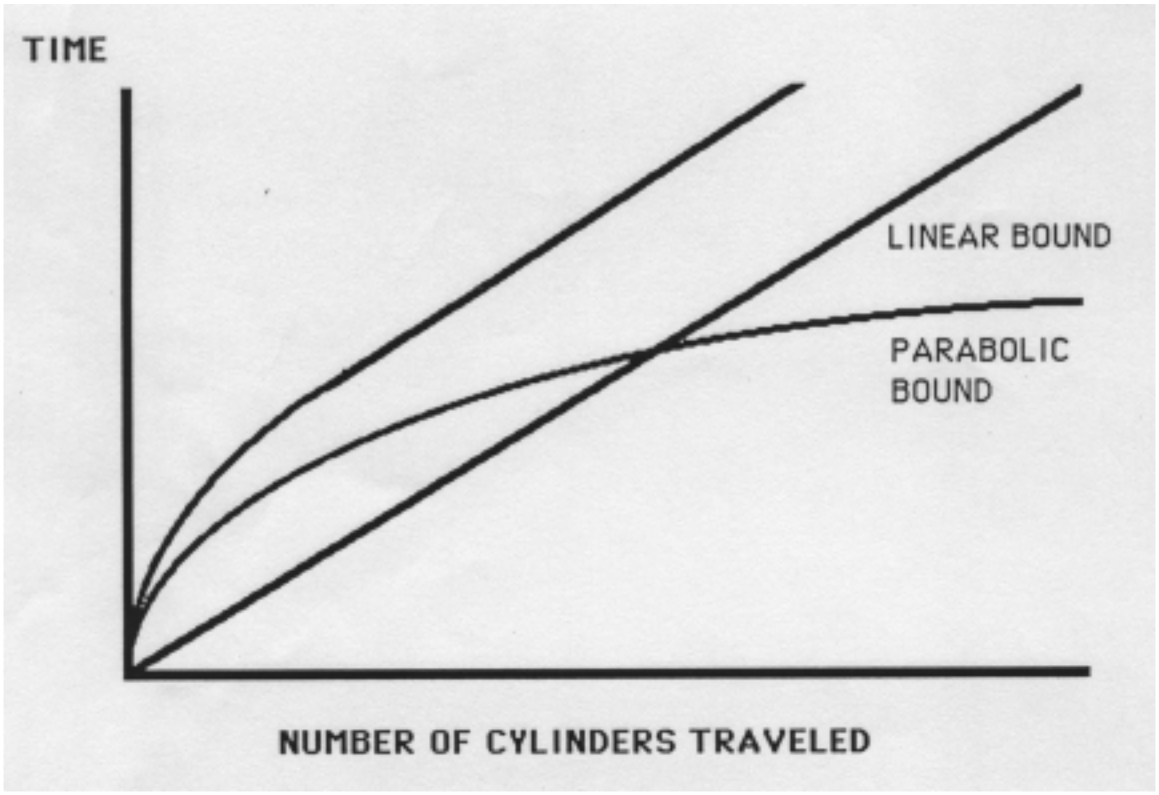

Figure 2 - A typical seek curve for a modern hard disk. It is bounded below by a straight line, which represents the head moving at a fixed rate with instant acceleration to that rate. It is also bounded below by a parabola, which represents constant acceleration followed by constant deceleration (*i.e.*, no coasting at all). The actual seek curve is slower than either of these bounds. The important fact here is that the seek curve is convex and is thus subject to the triangle inequality.