Incorporation of Audio in Virtualized Reality

Saurabh Gupta, Sundar Vedula, and Takeo Kanade

CMU-RI-TR-00-22
September 2000

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Virtualized Reality™ is a visual medium that allows the user to view a scene from any viewpoint – independent of the physical limitations imposed on the cameras capturing the scene. However the 3D models of video do not provide the complete simulation of the real world. We present an approach for the addition of audio to the existing video capture and recreation mechanisms. Using a technique analogous to video, multiple audio streams are captured to provide the spatial distribution of sound. A 3D model of audio is then generated to find the approximate sound as a function of the world coordinates. Synchronized with the video, this enables the user to both 'see' and 'hear' from any dynamically defined view-coordinates.

Table of Contents

1 Introduction

1.1 Virtualized Reality™

Visual media like paintings, photographs, moving pictures, television and video recordings depict the audio/visual aspect of a scene from only one direction that is decided at the time of recording, and is independent of the viewer. Virtualized Reality is a new visual medium that delays the selection of the viewing angle till view time, using techniques from computer vision and computer graphics.

Virtualized Reality differs from virtual reality in the sense that virtual reality worlds are typically constructed using simplistic, artificially created CAD models. Virtualized reality starts with the real world scene and virtualizes it. Virtualization of the real world implies creation of models that depict the world as a 3D structure. Information can then be obtained about any point in this 3D model. This allows the users to look at the world from whatever viewpoint they want, and not just the pre-recorded viewpoints.

There are many applications of such Virtualized Reality models. If implemented in real time, this can allow the user to view sporting events from any viewpoint. Using the available 3D model, the viewer can control the view from which he wants to witness the event. Also, the viewpoint is not constrained by physical limitations, and can be as novel as the view from a football, or from the center of a basketball court.

Other applications are in the field of training. Training can become safer and more effective by enabling the trainee to move about freely in a virtualized environment. Medical students could revisit a surgery, recorded in a virtualized reality studio, repeatedly, viewing it from positions of their choice. Telerobotics maneuvers can be rehearsed in a virtualized environment that feels every bit as real as the real world. True telepresence could be achieved by performing transcription and view generation in real time. An entirely new generation of entertainment media can be developed: Viewers of a sporting event could be given the feeling of watching the event from their preferred seat, or from a seat that changes with the action.

1.2 Virtualized Reality Video

Virtualized Reality™ is a visual medium developed at the Robotics Institute of Carnegie Mellon University. It involves capturing video streams and delaying the selection of the viewing angle till *view time*. Generating data for such a medium involves recording the events using many cameras, positioned so as to cover the event from all sides. The time-varying 3D structure of the event, described in terms of the depth of each point and aligned with the pixels of the image, is computed for a few of the camera angles --- called the *transcription angles* --- using a stereo method. This combination of depth and aligned intensity images is called the *scene description*.

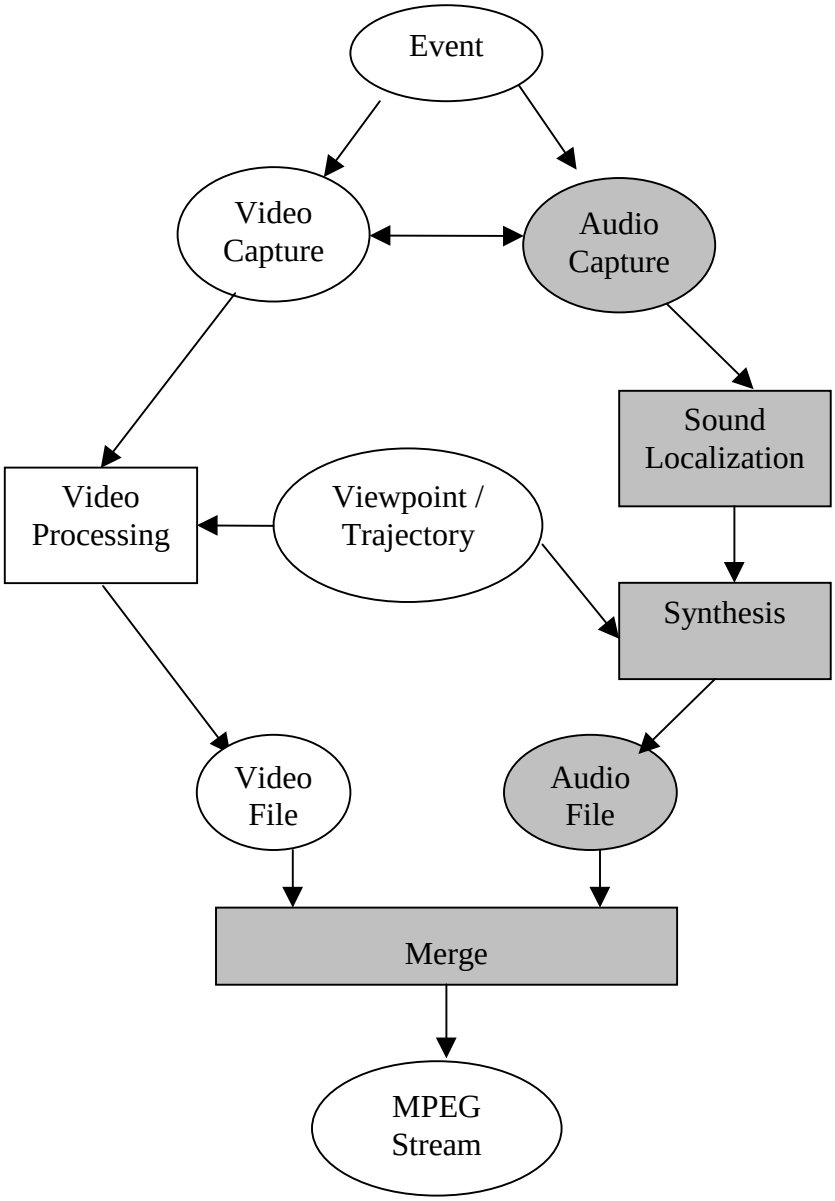

Figure 1: The audio/video processing pipeline

The collection of a number of scene descriptions, each from a different transcription angle is called the *virtualized* world. Once the real world has been virtualized, graphics techniques can render the event from any viewpoint. The scene description from the transcription angle closest to the viewer's position can be chosen dynamically for rendering by tracking the position and orientation of the viewer. The viewer, wearing a stereo-viewing system, can freely move about in the world and observe it from a viewpoint chosen dynamically at *view time*.

1.3 Incorporation of Audio in Virtualized Reality

Virtualized Reality™ deals with reconstruction of the visual aspect of the scene. The next enhancement is the ability to reconstruct the aural aspect to provide a more complete specification of the scene. This involves the addition of the facility to record multiple audio channels in synchronization with the video streams, and the generation of a spatial 3D model of sound. The basic architecture can be described as shown in Figure 1.

In the figure, the shaded boxes are the components of the architecture that are considered in this report. Video capture and processing are part of the existing Virtualized Reality process.

Addition of audio involves the following stages:

- Recording of a single channel of audio with the ability to set the desired sampling parameters, and also the ability to analyze and play the recorded stream.
- Recording of multiple audio channels in synchronization with each other – the recording for the various channels should be initiated in synchronization. There should be facility for saving the recorded streams onto the disk.
- Recording of the audio channels in synchronization with the video channels: the audio/video channels should be associated by specifying the time stamp using a common base clock.
- Processing of the audio streams for source localization and generation of a 3D sound model. Using this model, it should be possible to regenerate the sound at any point in 3D space.
- Generation of a movie stream combining the processed audio and video streams.

The organization of this report is as follows:

- Section 2 deals with the recording of multiple audio streams synchronized with each other and also the video streams.
- Section 3 describes the processing of individual audio samples – saving onto the disc, calculation of the Discrete Cosine Transform, and filtering.
- Section 4 describes the techniques used for source localization and view-dependent audio regeneration.

2 Recording of Audio channels

Generation of the virtualized reality models of sound requires recording and digitization of multiple audio channels. The channels have to be recorded in synchronization with each other and with the associated video streams.

2.1 Audio Recording and digitization

Analog recording of sound needs a device like a microphone. The basic principle of recording is based on the motion induced in the diaphragm by the sound waves. This

displacement is then converted into an electric signal, whose strength is proportional to the intensity of the sound wave.

For processing the recorded sound, it has to be sampled and digitized. In a digital recording system, sound is stored and manipulated as a stream of discrete values, each value representing the air pressure at a particular time. These values are generated by a microphone connected to an Analog to Digital converter, or ADC. Each such value is called a sample, and the number of samples taken per second is the sample rate.

The number of bits in the sample has a direct bearing on the fidelity of the signal. The number of possible voltage levels at the output is simply the number of values that may be represented using these bits. If there were only one bit in the number, the ultimate output would be a pulse wave with a fixed amplitude and more or less the frequency of the input signal. If there are more bits in the number the waveform is more accurately traced, because each added bit doubles the number of possible values. The distortion is roughly the percentage that the least significant bit represents out of the average value. Distortion in digital systems increases as signal levels decrease, which is the opposite of the behavior of analog systems.

The rate at which the samples are generated is another important measure of the quality of sound. If the sampling rate is lower than the frequency we are trying to capture, entire cycles will be missed, and the decoded result would be too low in frequency (aliasing). If the sampling rate were exactly the frequency of the input, the result would be a straight line, because the same spot on the waveform would be measured each time. As per the Nyquist theorem, the sampling rate must be greater than twice the frequency measured for accurate results. Hence, ideally for sound, perfect regeneration can be obtained at a sampling rate of 40,000 Hz.

2.2 Audio Hardware

Microphones are required for the analog recording of audio. Since multiple audio channels are required for the regeneration of view-dependent audio, multiple microphones are used to capture the sound from different points in the 3D space. In our implementation, six BG1.1K Shure microphones are used.

Some of the features of the microphone used are:

- Dynamic: This is a dynamic microphone, i.e., it uses moving-coil technology. A dynamic microphone consists of a diaphragm of thin plastic being attached directly to a dense coil of wire. The coil has a magnet either surrounding it or at the center. As the diaphragm vibrates, the coil vibrates, and its changing position relative to the magnet causes a varying current to flow through the coil. This current is the measure of the analog sound input.

- Cardioid Pickup: The Shure 1.1BG has a cardioid pickup pattern. Cardioid microphones have a heart-shaped pickup pattern, as shown in the diagram. They reject sound coming from the back of a microphone and are progressively more sensitive to

sounds as the direction approaches the front of the microphone. Such a pattern of reduced pickup of surrounding noise leads to an improved SNR ratio.

Figure 2 shows the spatial variation of the pickup intensity as a function of distance. Sounds directly in front of the microphone are being recorded with zero dB attenuation. This attenuation gradually approaches to -20 dB. for sound sources located behind the microphone.

- Frequency range: Another important characteristic of a microphone is the frequency response. This denotes the relative variation in decibels for different values of frequencies of sound. The frequency range of this microphone is from 80 Hz to 14,000 Hz. This includes the frequency range of the audible sound spectrum. Within this range also, the pickup pattern is not uniform and is as shown in Figure 3.

The microphone is connected to an XLR connector that provides an unbalanced analog output, which is at microphone level. Before it can be digitized, the signal has to be pre-amplified to line-level. This is done through the use of a multi-channel mixer/pre-amplifier. For this, the Alesis Studio 12R mixer is used. This is an eight mono and 2 stereo channel mixer, which can be used to generate a main output signal from the various inputs. However, since only the pre-amplified versions of the input signals are required, the insert jacks are used.

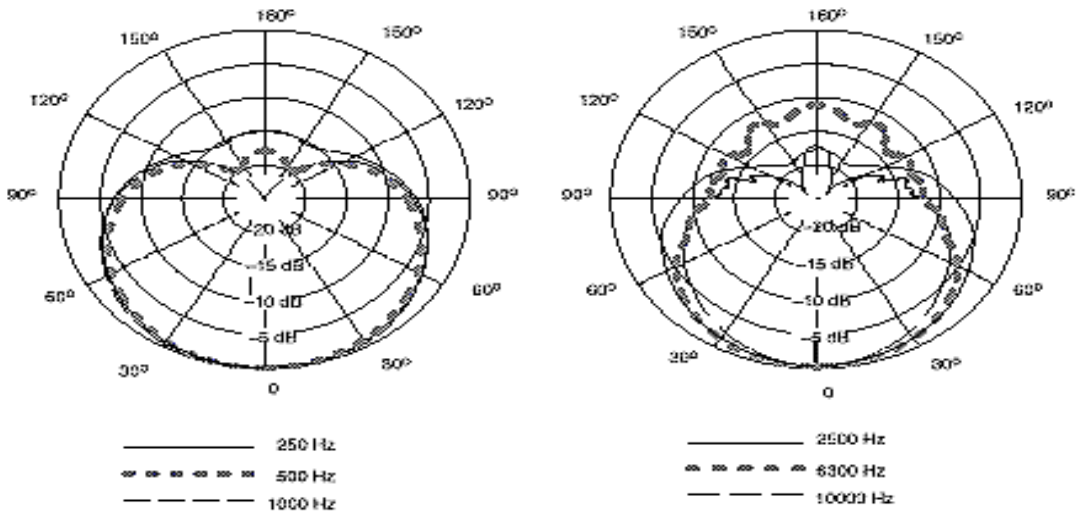

Figure 2: Cardioid Pattern for intensity pickup in Shure BG1.1 Microphone

Eight insert jacks are provided in the back panel of the mixer for each of the mono channels. These can be used to connect any external analog processing device inside the circuit of the mixer. To get the output of the pre-amplifiers, we can connect simple phono

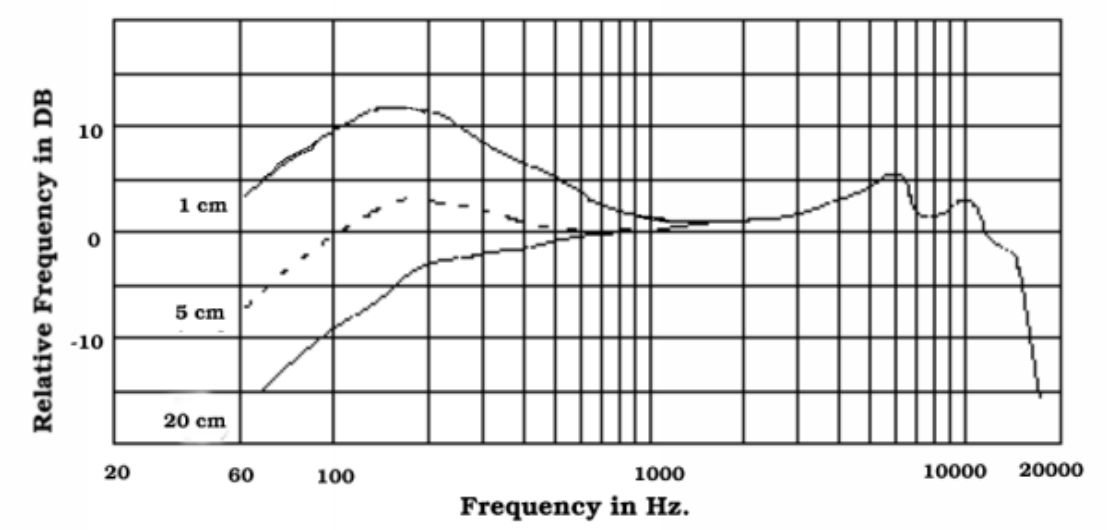

Figure 3: Frequency Response of the Shure 1.1G

connectors to the insert jacks. This output is dependent only on the knobs for amplification control of the 8 amplifiers, and is independent of the rest of the control circuitry of the mixer. This ensures that the audio signals remain independent of each other, and the minimum circuitry also reduces sound distortion.

The analog outputs from the insert jacks of the mixer now have to be digitized. For this, the Wave /824 sound card is used. This sound card supports simultaneous digitization of 8 mono audio channels. The mono inputs 1-6 were used for the six input channels. The sound card digitizes the channels separately, and makes them available for storage onto the disk. It also has 8 mono outputs to play up to 8 audio streams simultaneously.

2.3 Recording an Audio Stream

For recording audio samples, standard software interfaces such as Windows Sound Recorder are available. However, most of these packages are designed for mono or stereo channel recording. Also, these provide a very high level access to the recording process. Hence, they cannot be used to ensure synchronous multi-channel recording. So, for recording the audio streams, a device-level code is used so that the entire recording process is in the control of the user. The various parameters for recording can be chosen, and the initiation and termination of recording can be finely controlled to suit the application.

The sound recording is based on The Microsoft DirectSound® application programming interface (API) [4], which is the wave-audio component of the DirectX® API. DirectSound provides low-latency mixing, hardware acceleration, and direct access to the sound device. DirectSound enables wave sound capture and playback. It accesses

the sound hardware through the DirectSound hardware abstraction layer (HAL), an interface that is implemented by the audio-device driver.

The DirectSound HAL provides the following functionality:

- Acquires and releases control of the audio hardware.
- Describes the capabilities of the audio hardware.
- Performs the specified operation when hardware is available.
- Causes the operation request to report failure when hardware is unavailable.

The particular format of a sound can be described by a *WaveFormatex* structure. This structure as defined in the windows multimedia documentation is as shown:

```
typedef struct {
    WORD  wFormatTag;
    WORD  nChannels;
    DWORD nSamplesPerSec;
    DWORD nAvgBytesPerSec;
    WORD  nBlockAlign;
    WORD  wBitsPerSample;
    WORD  cbSize;
} WAVEFORMATEX;
```

The *wFormatTag* member contains a unique identifier assigned by Microsoft Corporation. The only tag valid with DirectSound is WAVE_FORMAT_PCM. This tag indicates Pulse Code Modulation (PCM), an uncompressed format in which each sample represents the amplitude of the signal at the time of sampling. DirectSoundCapture can capture data in other formats by using the Audio Compression Manager.

The *nChannels* member describes the number of channels, usually either one (mono) or two (stereo). For stereo data, the samples are interleaved. The *nSamplesPerSec* member describes the sampling rate, or frequency, in hertz. Typical values are 11,025, 22,050, and 44,100. The *wBitsPerSample* member gives the size of each sample, generally 8 or 16 bits. The value in *nBlockAlign* is the number of bytes required for each complete sample, and for PCM formats is equal to (wBitsPerSample * nChannels / 8). The value in *nAvgBytesPerSec* is the product of nBlockAlign and nSamplesPerSec. Finally, *cbSize* gives the size of any extra fields required to describe a specialized wave format. This member is always zero for PCM formats.

The recording and playback classes are based on encapsulations of the *WaveIn* and *WaveOut* classes defined for communication with wave devices in Windows. *WaveInCaps* and *WaveOutCaps* are used to get the input and output capabilities of the system. They query the wave device associated with the particular system, and obtain information regarding the name of the wave device, the manufacturer, the number of channels, and the various audio formats supported. These parameters are then checked against the list of available formats available for the particular device. If the format is supported, the wave input class (*WaveInDevice*) can be initialized. This requires the format information stored in a wave header object, and also the buffer in which to store the sound samples. The sequence of commands issued to record a sound sequence is:

- Initialize: This initializes of the wave device and provides the recording parameters.

- IsOpen: This checks that the chosen device is open. The device may be busy processing some other request, or may have been turned off. In this case, the recording sequence has to be terminated.
- Prepare: The prepare function prepares the device for recording. The device buffer is sent information regarding the recording parameters. However, the recording is not started at this point.
- SendBuffer: This is used to associate a memory buffer with the particular recording process. The sound samples are directly placed in this buffer.
- Start: This command starts the actual recording process. The sound samples available at the analog inputs of the sound card are now periodically sampled and placed in the buffer. This continues till the required number of samples is recorded, the buffer runs out, or the recording is stopped by the Stop command.
- UnPrepare: Once the buffer has been filled by the sound samples, it has to be processed (unprepared) to make it available for further analysis.
- Reset: After the recording is complete, the device is reset, so that it can handle a new request.
- Close: This closes the wave device. All the allocated parameters in the device buffer are reset.

2.4 Synchronization between multiple streams

For view-dependent regeneration of sound, multiple streams of audio have to be recorded. Also, these have to be synchronized with the video streams. Hence, the recording process has to ensure precise timing of the sound recording initialization and termination process. The recording procedure used facilitates the synchronization, as communication is directly with the device and there is no overhead of an intermediate software.

Recording of multiple channels takes place on the separate inputs of the Wave /824. The sound card has 4 stereo devices to handle the eight channels, and these are numbered 0,1,2 and 3 on the input device list. Hence, first the corresponding devices have to be identified. For all the channels, the initialization procedure is completed, and then the command to start recording is issued to all the devices simultaneously. Actually, since the command is issued serially one after the other, there is a slight delay equal to the processing cycle for the CPU. However, this time is almost negligible compared to the sampling rate of up to 44,100 kHz. Hence, almost perfect synchronization between the various audio channels can be achieved.

The audio recording also has to be synchronized with the video recording process. This synchronization is based on the use of Vertical Interval Time Clock (VITC). [5] VITC is a standard of time codes defined by the Society for Motion Pictures and Television Engineers (SMPTE). It assigns a number to each frame in the hours-seconds-minutes format, and stores it along with the video in two horizontal scan lines during the blanking period. A line of VITC consists of 90 bits, storing the sync data and the time code.

VITC is made part of the digitized image by configuring the digitizer to grab the scan lines of the blanking period. The video cameras are synchronized using a common sync signal so that they acquire image frames simultaneously. By recovering the unique time stamp embedded in every field of each video stream, the frames from multiple streams can be easily correlated.

For synchronization of the audio, the same sync signal has to used for the audio recording also. This enables starting the audio recording at the same instant as the video recording. Also, the initial VITC stamp is stored in each video stream. Using this initial time, and the frame rate, the time corresponding to each audio sample can be calculated.

For regenerating the scene, the processed audio and video samples have to be combined finally to generate the view-dependent scene content. Hence, the relation between the audio and the corresponding video files has to be preserved. This is done by including the information about the recording time stamp along with the audio file. Also, the location and names of the image files are stored along with the audio data, so that the audio-visual correspondence can be identified later.

3 Processing the digital audio samples

Once the audio streams have been sampled and digitized, these are available for further processing. The samples are now in the form of a uni-dimensional array, where each entry denotes the relative intensity of the recorded sound at a particular instant of time.

3.1 Saving as a wave file

The audio samples, which have been placed in a buffer, have to be saved onto the disc for future processing. The samples are saved in an uncompressed wave format, which makes it easy to playback the recorded samples. WAVE File Format is a file format for storing digital audio (waveform) data. It supports a variety of bit resolutions, sample rates, and channels of audio. This format is very popular upon IBM PC (clone) platforms, and is widely used in professional programs that process digital audio waveforms. It takes into account some peculiarities of the Intel CPU such as little endian byte order. This format uses Microsoft's version of the Electronic Arts Interchange File Format method for storing data in "chunks".

A WAVE file is a collection of a number of different types of such chunks. There is a required Format ("fmt ") chunk that contains important parameters describing the waveform, such as its sample rate. The Data chunk, which contains the actual waveform data, is also required. All other chunks are optional. Among the other optional chunks are ones that define cue points, list instrument parameters, store application-specific information, etc. In the project, only the format and data chunks are being used, and additional chunks are being defined to store the synchronization data.

All applications that use WAVE must be able to read the 2 required chunks and can choose to selectively ignore the optional chunks. There are no restrictions upon the order of the chunks within a WAVE file, with the exception that the Format chunk must precede the Data chunk. All data is stored in the in 8-bit bytes, arranged in Intel 80x86 (i.e., little endian) format. The bytes of multiple-byte values are stored with the low-order or least significant bytes first.

The Format (fmt) chunk describes fundamental parameters of the waveform data such as sample rate, bit resolution, and how many channels of digital audio are stored in the WAVE. It is defined as shown:

```
#define FormatID 'fmt '   /* chunkID for Format Chunk */
typedef struct {
  ID                 chunkID;
  long               chunkSize; //size of the chunk excluding the id and chunk size
  short              wFormatTag; //format used – normally 1
  unsigned short wChannels; //number of channels
  unsigned long  dwSamplesPerSec; //sampling rate
  unsigned long  dwAvgBytesPerSec; //bytes per second
  unsigned short wBlockAlign;
  unsigned short wBitsPerSample; //bit resolution of a sample
} FormatChunk;
```

The Data chunk contains the actual sample frames (i.e., all channels of waveform data). It is defined as follows:

```
#define DataID 'data'  /* chunk ID for data Chunk */
typedef struct {
  ID            chunkID;
  long           chunkSize;
  unsigned char  waveformData[];
} DataChunk;
```

The ID is always data. chunkSize is the number of bytes in the chunk, not counting the 8 bytes used by ID and Size fields nor any possible pad byte needed to make the chunk an even size (i.e., chunkSize is the number of remaining bytes in the chunk after the chunkSize field, not counting any trailing pad byte). waveformData stores the actual sample values for the audio.

A large part of interpreting WAVE files revolves around the two concepts of sample points and sample frames. For waveforms with greater than 8-bit resolution, each sample point is stored as a linear, 2's-complement value which may be from 9 to 32 bits wide (as determined by the wBitsPerSample field in the Format Chunk, assuming PCM format -- an uncompressed format). For example, each sample point of a 16-bit waveform would be a 16-bit word where 32767 (0x7FFF) is the highest value and -32768 (0x8000) is the lowest value. For 8-bit waveforms, each sample point is a linear, unsigned byte where

255 is the highest value and 0 is the lowest value. This signed/unsigned sample point discrepancy between 8-bit and larger resolution waveforms has to be taken care of, both during the coding and the decoding process.

Because most CPUs read and write operations deal with 8-bit bytes, a sample point should be rounded up to a size that is a multiple of 8 when stored in a WAVE. This makes the WAVE easier to read into memory. If the ADC produces a sample point from 1 to 8 bits wide, a sample point is stored in a WAVE as an 8-bit byte (i.e., unsigned char). If the ADC produces a sample point from 9 to 16 bits wide, a sample point is stored in a WAVE as a 16-bit word (i.e., signed short). Furthermore, the data bits should be left justified, with any remaining pad bits zeroed. For example, consider the case of a 12-bit sample point. It has 12 bits, so the sample point must be saved as a 16-bit word. Those 12 bits are left justified so that they become bits 4 to 15 inclusive, and bits 0 to 3 are set to zero.

For a monophonic waveform, a sample frame is merely a single sample point (i.e., there's nothing to interleave). For multi channel waveforms, there are standard conventions to store channels within the sample frame. For stereo data, the left and right channel samples are interleaved, i.e., the odd bytes are for the left channel, and the even bytes are for the right channel.

Apart from these standard chunks, there are some user-defined chunks to add the synchronization data. The first chunk after the data stores the VITC clock value when the recording is initiated. Knowing this value and the sampling rate, the time corresponding to each individual sample can be obtained. The other chunks are to store the location and names of the associated video files. During the regeneration process, these are read to identify which frames to combine with the processed audio frames.

3.2 Calculation of the DCT

The array of samples contains information about the audio in the time domain. Another extremely useful representation is in the frequency domain. Here, the signal is represented as the relative strengths of the various frequency components. There are various frequency domain representations. Since the samples are real, the DCT (Discrete Cosine Transform) can be used to accurately encode the data.

The DCT [6] is a mathematical operation that transforms a set of data, which is sampled at a given sampling rate, to its frequency components. The number of samples should be finite. A one-dimensional DCT converts an array of numbers, which represent signal amplitudes at various points in time, into another array of numbers, each of which represents the amplitude of a certain frequency component from the original array. The resulting array of number contains the same number of values as the original array, and each represents a particular frequency component. This is shown in Figure 4.

The 1D discrete cosine transform is defined as

$$X_C[k] = \alpha[k] \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad k = 0,1, \quad ..., N-1 \qquad (1)$$

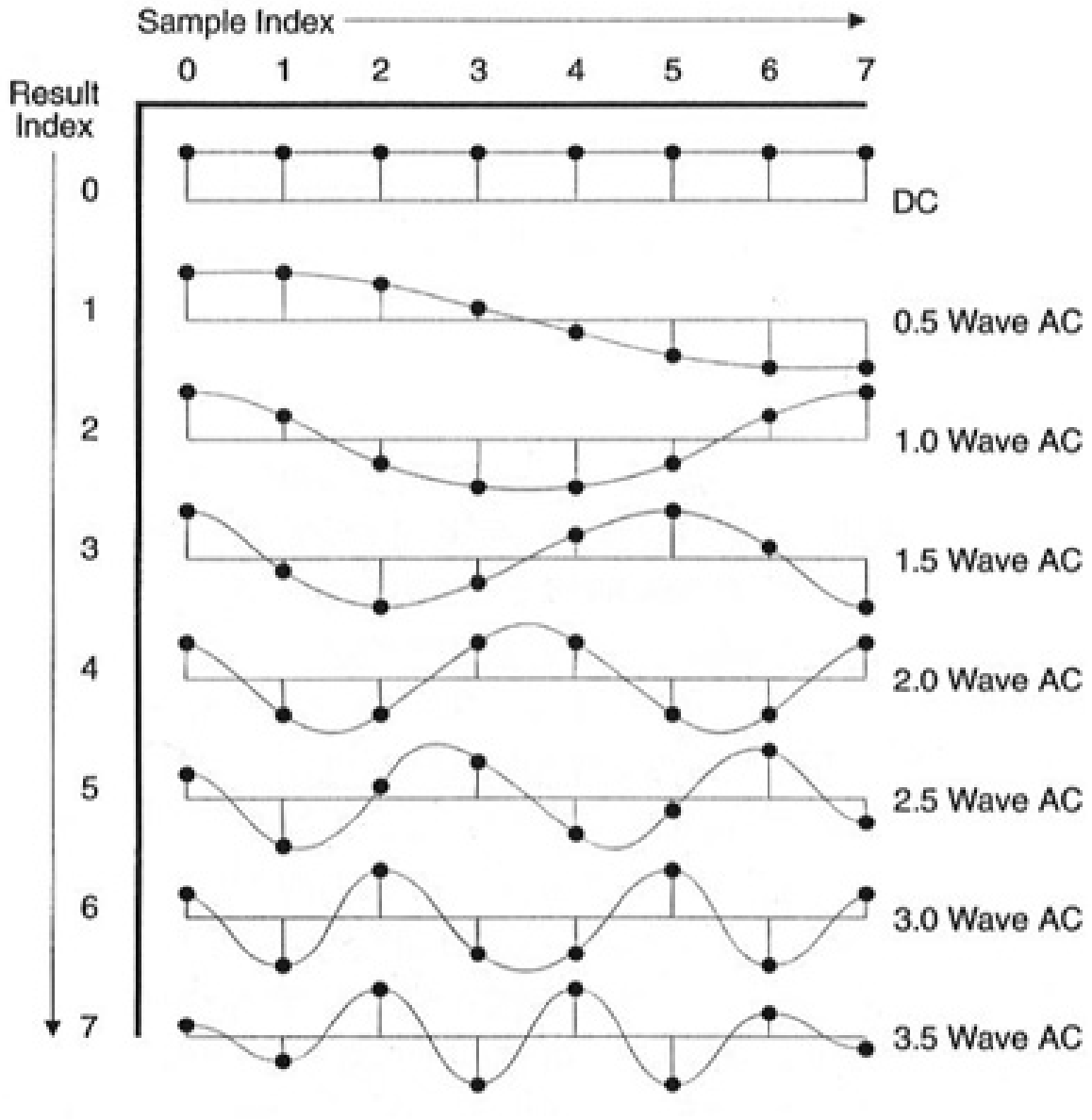

Figure 4: Representation of DCT as a series of frequency components

The inverse DCT is defined as

$$x[n] = \sum_{k=0}^{N-1} \alpha[k] X_C[k] \cos\left(\frac{\pi(2n+1)k}{2N}\right) \qquad k = 0,1,\ldots,N-1 \qquad (2)$$

where in both the equations,

$$\alpha[k] = \sqrt{\frac{1}{N}} \quad for \;\; k = 0$$
$$= \sqrt{\frac{2}{N}} \quad for \;\; k = 1,2,...,N-1 \qquad (3)$$

The Discrete Cosine Transform was implemented using these basic equations. For all the waveforms, the DCT can be calculated and displayed. This gives an analytical description of the signal. For example, a high DCT coefficient denotes excess room noise or noise generated in the recording process. The inverse DCT was also implemented to help in the noise-filtering process, as described in the next section.

3.3 Noise filter

The sound recorded also contained a significant portion of noise. This noise is the ambient noise present in the lab. For better sound quality, this noise should be minimized. One form of noise filter using Discrete Cosine Transform was implemented.

Since the lab noise is statistically known, the noise reduction process can be made to use the characteristics of the noise. A sound sequence can be recorded prior to the actual recording, to record only the noise. Then the actual sequence is recorded. Noise reduction can be obtained by "subtracting" the first sequence from the second sequence. However, this subtraction cannot be simple subtraction in the time domain. This is because noise is a random process in the time domain, and simple subtraction will give a meaningless result. However, the frequency characteristics of the noise can be assumed to be fixed for a sufficiently large sample. And noise reduction can theoretically be achieved by subtraction in the frequency domain.

The DCT coefficients are calculated for both the noise and the recorded audio sequences. These give the desired representation in the frequency domain. The noise reduction algorithm then subtracts the two to yield a new set of DCT coefficients. Finally the inverse DCT is taken to generate the new sequence.

Using the above algorithm, it was found that the ambient noise, particularly the excessive DCT component, decreases. However, the DCT subtraction leads to additional frequency components in the final signal, and there is substantial ringing sound due to it. This can be attributed to the fact that lab noise may not be statistically constant over the finite interval chosen. Also, the noise need not be a simple additive noise, i.e., the noisy signal need not be a simple addition of the signal with the ambient noise. As a result, the above algorithm did not yield the desired result, and was not used in the final version of the program.

4 View-dependent Audio Regeneration

Once the synchronized set of audio samples has been obtained, the next requirement is the generation of view-dependent audio. For this the different audio channels have to be processed to locate the source of sound, and then the audio samples are reconstructed based on the view-points provided by the user.

4.1 Source localization

To regenerate the audio depending of the coordinates of the viewer, we need some measure of the coordinates of the sound source. We assume a single source model, i.e., there is a single source of sound positioned in space. For cases in which there are actually multiple sound sources, this can be used to find an effective single source.

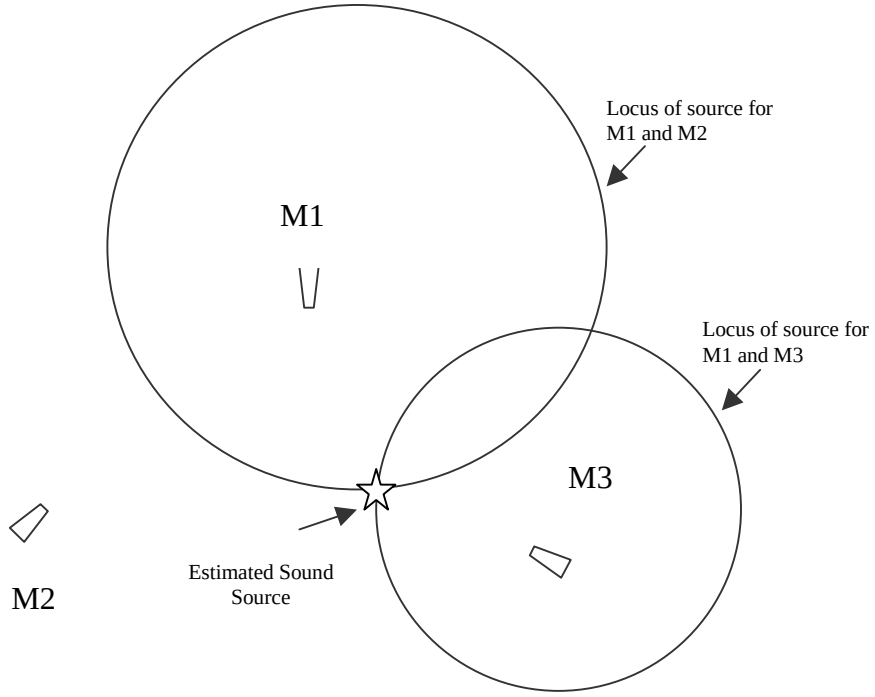

Figure 5: Source Localization using omni-directional pickup pattern for the microphones

A simple algorithm for source localization can be implemented using the spatial variation of the intensity of the received signal. This is a function of the distance from the sound source. The intensity at a distance r varies as $1/r^2$. Hence, if the microphones are located at co-ordinates (x_1,y_1) .. (x_m,y_m), and the sound source is located at co-ordinate (x,y), the microphone equations are:

$$\frac{I_0}{\left(\left(x-x_i\right)^2+\left(y-y_i\right)^2\right)}=I_i \qquad i=1..m \tag{4}$$

Here, I_i is the calculated intensity of sound at the i'th microphone and I_0 is the effective source intensity. If we have 6 microphones, we will have 6 such equations. Using 3 of these, ideally the exact location of the source in the x-y plane can be found.

Solving equations for microphones 1,2 and 3, we get

$$\begin{aligned} I_1\left((x-x_1)^2+(y-y_1)^2\right) &= I_2\left((x-x_2)^2+(y-y_2)^2\right) \\ I_1\left((x-x_1)^2+(y-y_1)^2\right) &= I_3\left((x-x_3)^2+(y-y_3)^2\right) \end{aligned} \quad (5)$$

$$\begin{aligned} &(I_1-I_2)x^2+(I_1-I_2)y^2-(2x_1I_1-2x_2I_2)x-(2y_1I_1-2y_2I_2)y+x_1^2I_1+y_1^2I_1-x_2^2I_2-y_2^2I_2=0 \\ &(I_1-I_3)x^2+(I_1-I_3)y^2-(2x_1I_1-2x_3I_3)x-(2y_1I_1-2y_3I_3)y+x_1^2I_1+y_1^2I_1-x_3^2I_3-y_3^2I_3=0 \end{aligned} \quad (6)$$

Normalizing the equations, we get:

$$\begin{aligned} &x^2+y^2-\frac{(2x_1I_1-2x_2I_2)}{(I_1-I_2)}x-\frac{(2y_1I_1-2y_2I_2)}{(I_1-I_2)}y+\frac{x_1^2I_1+y_1^2I_1-x_2^2I_2-y_2^2I_2}{(I_1-I_2)}=0 \\ &x^2+y^2-\frac{(2x_1I_1-2x_3I_3)}{(I_1-I_3)}x-\frac{(2y_1I_1-2y_3I_3)}{(I_1-I_3)}y+\frac{x_1^2I_1+y_1^2I_1-x_3^2I_3-y_3^2I_3}{(I_1-I_3)}=0 \end{aligned} \quad (7)$$

Subtracting the two, we get the equation of a straight line

$$\begin{aligned} &\left(\frac{(2x_1I_1-2x_2I_2)}{(I_1-I_2)}-\frac{(2x_1I_1-2x_3I_3)}{(I_1-I_3)}\right)x-\left(\frac{(2y_1I_1-2y_2I_2)}{(I_1-I_2)}-\frac{(2y_1I_1-2y_3I_3)}{(I_1-I_3)}\right)y \\ &\quad = \frac{x_1^2I_1+y_1^2I_1-x_2^2I_2-y_2^2I_2}{(I_1-I_2)}-\frac{x_1^2I_1+y_1^2I_1-x_3^2I_3-y_3^2I_3}{(I_1-I_3)}=0 \end{aligned} \quad (8)$$

This equation can be used to get an expression for x as a function of y. Substituting this in the previous equation gives a quadratic equation in x. This can be solved to yield two estimates of the sound source. Of this, the one that falls inside the triangle bounded by the 3 microphones can be detected. This is used as the measure of the sound source.

Geometrically, this derivation can be understood as follows. Consider the three microphones as three vertices of a triangle. We know the ratio of intensities of recorded sound between the 3 points. This ratio of intensity between microphones 1 and 2 gives the locus of the possible points as a circle with center on the line joining the two microphones. We get another such circle for a second pair of microphones. The intersection of the two circles that falls inside the triangle gives the estimated source. We can always find such a source for any ratio of intensities. For the trivial case in which the intensities are same at two microphones, the circle tends to a straight line passing midway between the two microphones.

This derivation assumes that the microphones are omni-directional, i.e., the pickup intensity is independent of the direction of source with respect to the microphone. However, normally for a microphone, the sound pickup is a function of the angle that the source makes with the microphone. Hence, the above derivation can be extended to incorporate this effect.

The directional variation can be expressed as a function of the angle θ that the source makes with the axis of the microphone, i.e., θ is zero for points directly in front of the microphone, and 2π for points behind the microphone. The attenuation due to directional variation can be expressed by incorporating the function f(θ) in (4) to give:

$$I_i' = \frac{I_i}{f(\theta)} = \frac{I_0}{\left((x - x_i)^2 + (y - y_i)^2\right) f(\theta)}$$
$$\theta = \tan^{-1}\left(\frac{x - x_i}{y - y_i}\right) \quad (9)$$

Now the equations are solved taking the additional factor into account. This yields more accurate results for source localization.

4.2 View dependent normalization

Once the source co-ordinates have been estimated, the audio samples can be normalized on the basis of the viewpoint coordinates specified by the user. These coordinates are in the Cartesian form, with the same center as the center being used for the microphone coordinates. Now, for the estimation of the sound at different points in space, we have the information regarding these coordinates and the sound samples obtained at the various microphones.

4.2.1 Normalizing on the basis of direction

One strategy for signal estimation at any point in space is by doing a weighed average of the samples obtained from the various microphones. If the microphones are located at coordinates (x_i, y_i), and the sound intensity is to be computed at the point (x_0, y_0), then we can estimate the weights as the inverse of the distances of the microphone from the viewpoint. i.e.,

$$w_i = \left((x_0 - x_i)^2 + (y_0 - y_i)^2\right)^{-1/2} \quad (10)$$

The weighing factors need to be normalized, so that their sum is unity. A microphone that is closer to the viewpoint will have samples closer to the actual value at that point, and hence it should be a given a higher weight. If the viewpoint is the same as the location of a microphone, then the sound sample recorded at that microphone can be used. In that case, the weight for that microphone becomes 1, while it is now zero for the rest of the microphones.

This strategy works for points along the perimeter of the polygon bounded by the microphones, as this takes care of the directionality of sound. However, it does not make use of the locations of the source of sound. A point at the center of the microphones will have the same sample values as a point infinitely away (in both cases, the weights for all the microphones will be equal). This is incorrect, since the point at the center of the microphones would have a high intensity of sound. And the point infinitely away should have zero intensity. Hence, another strategy is to use the distance of the viewpoint from the estimated sound source.

4.2.2 Normalization on basis of distance from source

Since the intensity of sound varies as $1/r^2$ outwards from the source, this can be used to estimate the intensity at each point in space. We need to express the intensity at each point as a function of the distance r, such that the following constraints are specified

- The sound intensity at the source is theoretically "infinite", but the actual maximum sample value is bounded by the microphone. Hence, the average source intensity is assumed to correspond to this maximum value.
- The sound intensities at the microphones should be equal to the actual sound intensities recorded earlier.
- The sound intensity should be zero at a point infinitely away from the source.

If the distance from the first microphone is r_0, the spatial distribution factor for the first microphone is:

$$\begin{aligned} f &= f_1 && r < r_{\min} \\ f &= 1 - \left(f_1 / r_0 \right) + \left(f_1 / r \right) && r < r_0 \\ f &= r_0 / r && r > r_0 \end{aligned} \qquad (11)$$

where f is the weighing factor for the sound sample from Microphone 1, f_1 is the ratio of the maximum intensity to the average recorded intensity for the microphone, and r is the distance from the source.

4.2.3 Normalization using both distance and directional variation

The normalization on the basis of distance alone does not consider the directional properties of audio. For example, if a unidirectional speaker is located at a particular point in space, the points in front of the speaker should have a higher intensity of sound compared to equidistant points behind the speaker. Weighing on the basis of microphone distances can handle this, as the microphone in front of the speaker will have a higher intensity of sound. Since the points in front of the speaker will have a higher contribution from that particular microphone, their intensity would be higher.

Hence, better results can be obtained by combining both the algorithms – weighing on the basis of the inverse microphone distances, and the distance from the sound source.

Now the intensity of a point is a function of both the distance from the estimated source, and the angle it makes with the axis of the estimated source.

4.3 Generation of the audio sequence

The audio sequence has to be modified depending on the viewpoint at each instant of time. For this, the user-defined viewpoints are taken as input. This consists of a series of coordinates denoting the viewpoint for each frame of the corresponding video sequence.

The time corresponding to each viewpoint is calculated knowing the initial time stamp. This is then used to find the corresponding audio samples associated with each frame. These audio samples are then normalized based on the equation given in the previous section. Since each set of samples is normalized by a different factor, simply combining them will lead to abrupt boundaries and ringing sounds.

Hence, some kind of smoothing filter has to be used. It is assumed that the viewpoints gradually change from one frame to the other. Hence, the actual sound samples are linear combinations of the normalized sound samples between two frames. This gives a better quality of audio by reducing abrupt changes.

Once the audio samples have been obtained, they are saved onto a file, and can now be combined with the processed video sequence. The video sequence is obtained using the same set of coordinates. The audio and video can now be combined in any movie format like MPEG or QuickTime, to yield the final view dependent scene reconstruction.

5 Conclusions and Future Work

The audio recording and processing techniques outlined in the report have been implemented and are found to yield satisfactory results. Six microphones are used to record six audio streams in synchronization with the multiple cameras. Synchronization is based on use of the VITC time stamp. The source is then localized using a simple single source model. This yields the virtualized model of the audio accompanying the scene. The sound is then dynamically regenerated using the viewer coordinates. Finally, the processed audio and video is combined to yield the final view-dependent scene reconstruction.

Here, we are assuming a single source of model. This gives only approximate results when there are more sources, as would be found in most practical situations. Hence, a possible enhancement would be the isolation of the multiple sources, and their separate analysis. The regeneration process would also consider the sources separately.

Another scope for improvement is better identification of the directional properties of the source. Using more microphones to sample the sound at more locations in space, we can find the directional nature of the source itself, rather than the extending the effect observable on the microphones. Also, another more complicated extension can be the inclusion of obstruction and reflection effects caused due to physical objects in space. For

example, a person on the field would block the sound, and view-coordinates behind the person would receive a more attenuated version of sound coming from a source in front of the person. This effect would require using the 3D models generated from video processing.

6 References

1. P.J. Narayanan, P. Rander, and T. Kanade. Constructing Virtual Worlds Using Dense Stereo. *Proceedings of Sixth IEEE International Conference on Computer Vision (ICCV'98)*, January, 1998.
2. T. Kanade, P. Rander, and P.J. Narayanan. Virtualized Reality : Constructing Virtual Worlds from Real Scenes. *IEEE Multimedia, Immersive Telepresence*, Vol. 4, No. 1, January, 1997.
3. T. Kanade, P. Rander, and P.J. Narayanan. Virtualized Reality: Concepts and Early Results, *IEEE Workshop on the Representation of Visual Scenes*, June, 1995.
4. Microsoft MSDN Library http://msdn.microsoft.com/library © 2000 Microsoft Corporation.
5. P.J. Narayanan, T. Kanade and P. Rander. Synchronous Capture of Image Sequences from Multiple Cameras, *CMU Technical Papers CMU-RI-TR-95-25*, December 1995.
6. K.R. Rao and P. Yip. Discrete Cosine Transform: Algorithms, Advantages, Application, *Academic Press, London,* ISBN 0-12-580203-X, 1990