7. Spatial ambience and sound localization programs

(This section last updated November 2012)

7.1. Sound ambience and localization

Sound *localization* and sound ambience are vital and related elements in acoustic recording, sound reproduction over loudspeakers and headphones, room design and, of course, in computer/electronic music production. A *soundfield* is the complete three dimensional space surrounding a listener. A soundfield can be small (e.g. a closet) or large (e.g. a concert hall), bounded (any room or enclosed space) or unbounded (outdoors). Any soundfield will have a particular **ambience**, or "environmental quality" in which we hear sounds, and which listeners often describe subjectively as "wet," "dry," "warm," "bright," "spacious," "harsh" and so on. This ambient quality results from many factors, including the dimensions of the soundfield, its symmetry or asymmetry, and the sound reflectivity vs. absorption characteristics of walls, boundaries and other objects around the perimeter and within the soundfield. Barometric pressure, temperature and humidity also affect these ambient qualities, usually in subtle but perceptible ways.

As sound pressure waves ripple through space at approximately 343 meters (c. 1100 feet) per second the intensity (amplitude) of the of these waves is reduced, falling off by the square of the distance traversed from the sound source. High frequencies decay more quickly than low frequencies, due in part to absorption by moisture in the air. When a sound wave strikes a flat surface, some of the energy of the wave is absorbed by the surface, reducing its intensity, while the remainder of the intensity reflects off of the surface, at a new angle (often about 90 degrees) back into the soundfield. Soft, porous materials, such as thick drapes and carpeting, are highly absorbant, often absorbing 80 % or more of the intensity of the wave, while hard surfaces, like metal or stone, reflect most of the pressure wave back into the room. High frequencies are absorbed much more than low frequencies by many materials. When sound waves strike a curved surface, the wave tends to diffract around the surface, continuing on in roughly the same direction, rather than bouncing (reflecting) off of the surface.

Walls and surfaces closest to the sound source and to the listener produce the first few reflections. Our auditory perception (which includes our physical hearing mechanism, from the pinna to the cochlea of both ears, and the manner in which our brain interprets electrical impulses received from the cochleas) is quite sensitive to the arrival times and decreasing amplitudes of **early reflections** that reach our ears between about 15 and 100 or so milliseconds after the arrival of the direct signal. At some point, depending upon the dimensions and reflectivity of the room, and the locations of the sound source and the listener, reflections that have bounced off of two, three or multiple surfaces bunch up and arrive in dense clusters, so that our brains no longer can distinguish the arrival times or intensities of individual reflections. Rather, these **late reflections** fuse together and create a quality of sound diffusion, or "spreading." **Reverberation** is a combination of our responses to discrete early reflections and to the "tail" or "wash" of late reflection diffusion. **Reverberation time** (often abbreviated *T60*), is a measurement of late reflection diffusion, and is defined as the duration it takes, after cessation of the direct signal, for the reverberant tail to decay by 60 dB. Concert halls the size of Kilbourn typically have *T60* times of about 3/4 of a second. Halls the size of the Eastman Theater have a reverberation time of between 1.5 and 3 seconds, while for rooms the size of the MIDI studio reverberation typically lasts less than half a second.

Appropriate amounts of reverberation tend to increase our enjoyment of music, subjectively making the sound "fuller," "richer," "livelier" and "warmer." Too much reverberation, or the "wrong" kind of reverberation, however, is distracting or annoying, "coloring" the sound excessively and leading to a loss in clarity and detail ("smearing"). Rooms with parallel walls and boundary dimensions with simple integer or near-integer ratios (for example, a shoebox-shaped room) often produce standing waves that create modal resonances — frequencies (and harmonic multiples of these frequencies) that can ring prominently when excited by any sound. The worst possible acoustical shape for a concert hall would be a cube.

Different types of music benefit from different reverberant characteristics; a reverberant ambience appropriate for a Josquin motet probably would sound "muddy," even oppresive, in a performance of a work for percussion ensemble. For intelligibility, speech requires a drier sonic environment than most musical genres, a fundamental problem in the design of multi-purpose halls.

Synthetic digital reverberation, whether hardware- or software-based, most often employs banks of comb and alpass filters (delay lines with feedback), along with low pass and other frequency selective filters, to produce a reverberant "wash" that simulates (more or less successfully) the "tail" of late reflections. More sophisiticated reverberation algorithms also employ multiple delay lines without feedback to simulate early reflections (although with less density than occurs in natural reverberation). High quality reverberation is among the most difficult and computationally expensive areas of audio signal processing. And even when using high quality reverberation units or algorithms, users must take care to avoid "painting" all sounds with an indistinguishable reverberant glob.

Owing to the limitations and cost (in computer cycles and/or dollars) of delay line-based reverberation algorithms, alternatives such as *convolution* are being employed with increasing frequency. Convolution techniques — which also can be complicated, tricky and time consuming to use — process (dry) input sounds against *impulse response* files created in reverberant natural environments, such as concert halls and cathedrals, attempting to impose the ambient room response qualities of these reverberant sources upon the input sounds.

Sound **localization** is the ability of our auditory perception, apart from any visual cues, to pinpoint the origin of any sound within a soundfield. If you were taken blindfolded into an unfamiliar room and heard, say, a finger snap, you probably would be able to determine instantly the approximate distance and direction of the snapping fingers ("about 10 feet to the right, a couple of feet in front of me and slightly above my head"). Additionally, you also could venture a fairly accurate guess as to the dimensions and structural materials of the rrom. The sound that reaches your ear contains information, which your brain can decode, not only about the sound source itself, but also about its location and about the physical environment around you.

In fact, our brains employ a variety of sound localization procedures which operate in different frequency bands. Several (but not all) of these localization cues involve subtle differences between the signals received by the brain from the two ears. The direct signal of a sound to our right and in front of us will reach our right ear slightly before reaching our left ear. This localization cue is called the *interaural time delay* (*ITD*). The signals that the brain receives from the two ears will be out of phase. The signal from the right ear will have a higher intensity (amplitude) than the signal from the left ear, which provides the brain with an *interaural level difference* (*ILD*) cue. The signal from our right ear also will have slightly more high frequency energy (since the signal from our left ear has been at least partially diffracted around our head, causing not only a longer path but also low pass filtering). The further the sound moves toward the right, the greater the temporal, amplitude, phase and spectral differences received from the two ears. Additionally, as the sound source moves further to the right, the overall intensity will be reduced (especially for higher frequencies) and the percentage of reverberation we hear will increase. Reverberation levels from a sound remain relatively constant no matter where it is located within a room. But as the intensity of the direct signal is reduced with increasing distance, the "wet/dry mix" betweeen direct and reverberant signals changes, another distance cue. The pattern of the early reflection arrival times also will change. As a sound moves closer to the closest wall, the difference in arrival times between the direct signal and the first reflection will be reduced, while the arrival time of the first reflection off of the oppostie wall will be increased.

The brain processes direction and distance cues to calculate the origin of a sound source, but this is a complex process that is only partially understood. Each of these cues provides useful information only for a limited range of frequencies. Phase differences between the signals received from the two ears, for example, are not useful for very low frequencies (where a single cycle may be several meters in length) nor for high frequencies with extremely short wavelengths, but only provide usable information at frequencies between about 150 hertz and 1.5 kHz. Intensity differences between the two ears (*ILD*s) are used to calculate direction and distance for frequencies between about 300 hertz and 5 kHz. Other cues, called *head related transfer function* (*HRTF*) cues, come into play above 2.5 kHz. (Note how the frequency bands of these cues overlap). *HRTF*s are unique to each individual, and are derived from the shapes and dimensions of our heads, ears and shoulders. Sounds reflect back and forth somewhat in triangular fashion between various parts of our heads, pinnae and upper bodies creating complex filtering responses at higher frequencies that vary with the direction and distance of a sound source, and thus also vary between the two ears.

Note that none of these cues provide much information for low frequencies below about 150 hertz or so. As a result, we have trouble localizing low frequencies, which in many respects are non-directional.

Also, we our much less adept at localizing sound sources behind us than to our front. For a sound directly behind us, we receive almost no direct signal (which strikes the *back* of our pinnae), but only diffracted and reflected signals.

7.2. Monophonic, stereophonic and multichannel sound reproduction systems

Monophonic sound reproduction provides very little sound localization information. We are aware of the location of the loudspeaker, of course, but cannot determine relative positions for the clarinet or violin within the music being played over the speaker. *Stereophonic* (two speaker) sound reproduction, which now has been commercially available for more than 50 years, offers the potential for localization of individual sounds along a left-right plane as we face the speakers. Stereo offers relatively poor *depth* information (the distance of a sound source in front of the microphone or listener), and even poorer location information on sounds intended to emanate from the sides of or behind the listening position. And even left-right localizationis subject to significant limitations.

Stereophonic recording of "classical" and jazz genres generally is done with a *coincident cardioid pair* of microphones. Two small condenser cardioid mics (which are considerably more sensitive to sounds coming from the front than to sounds coming from the sides or from the rear), placed very closely together, are pointed forward toward the musicians but are angled in opposing directions toward the left and toward the right, at an angle somewhere between 90 and 120 degrees. This technique attempts to capture the left-right positions of the musicians, and also a good general balance of direct (dry) and reverberant information. The distance of the mics from the musicians, the height of the mics above the performers and the angles of the mics determine the proportion of direct (dry) and reflected ("wet") signal recorded as well as the *width* of the stereo imaging (how far apart the performers seem to be). The technique provides relatively little information on how far in front of the mics each player is positioned. (Obviously, this recording technique also provides almost no locational information on any sound sources arriving from behind the mics. It is not designed for this purpose.) The goal of coincident pair recording is to attempt to capture the sound of an ensemble as it would be heard by a listener positioned in the best seat in the house.

Pop and commercial film and TV music, by contrast, is generally multi-tracked. Instrumental and vocal parts are recorded individually and/or in small groups, sometimes days or weeks apart, on separate tracks of a hard disk recording system or a sequencer program, or a hardware recording unit. Computer and electronic parts also are recorded onto separate audio or MIDI tracks. All of the acoustic and computer-generated tracks can be edited individually, and then mixed down to two tracks. During the mixing process each track can be processed with one or more "effects," such as equalization (EQ), compression and reverberation. These "effects" can provide varous type of ambience, and can be used to simulate, in a very generalized way, different close-distant locations for the sound sources.

Also during the mixing stage, although occasionally instead during the mastering stage (when all of the cuts for a compact disc are assembled and homogenized), the mixing or mastering engineer, possibly in consultation with the performing artists, determines the left-right stereo *imaging* (localization) for each track. Stereo mixing involves the creation of *phantom images*, in which sounds are perceived to emanate from a position other than their actual origins (the two speakers). Stereo mixing almost always is accomplished by means of *panpots*, which vary the amount of signal from each track that is routed to the master left and right stereo busses. This panpot-based localization technique is sometimes called *pair-wise mixing*, because it distributes varying amounts of signal between pairs of channels and, ultimately, pairs of loudspeakers. It also is called *intensity panning*, because the relative intensity of the signal coming from the left and right speakers determines the perceived location of each sound. If we want to position a sound to the left of center, we feed more of the signal to the left speaker than to the right speaker, creating an *ILD* (*interaural level difference*) cue. In the absence of other cues to the contrary, the brain will use this information to calculate a position for the sound somewhere to the left of center.

Although *pair-wise* stereo mixing simulates only one the cues (amplitude differences between the two ears) that are used by the brain to localize sounds, it can work surprisingly well, but only if the listener is in the *sweet spot*, approximately equidistant from the two speakers. If a listener is closer to the right speaker than to the left, the illusion is destroyed and the sound gets "sucked" into the right speaker. The sound reaching the listener from the right speaker will reach the listener's ears before the sound arriving from the left speaker. Because of the *Haas effect* (also called the *precedence effect*), the brain will interpret

this bogus *interaural time delay* cue as sound arriving from the right. Additionally, the sound from the right speaker will have more high frequency energy, and may actually have a higher intensity (since the signal path is shorter). The listener is receiving conflicting cues, or "crosstalk," from the two speakers: each sound (except those panned hard left or right) actually arrives at our ears from two locations (the stereo loudspeakers), and the two speakers provide incoherent phase, HRTF and other cues to our two ears. The further that listeners move from the relatively narrow sweet spot, the greater the smearing of the intended pan locations. When we sit much closer to one speaker than to the other, we begin to experience monophonic reproduction, with the other speaker providing either a reverberant wash or a slap-back echo.

For stereo to work well, not only must all listeners sit in the sweet spot, but also the speakers should be facing the center of the room at an angle of between 45 and 60 degrees (measured from the position of the listener looking forward toward the speakers). As this angle approaches 90 degrees — something that easily can happen when the speakers are widely spaced on the sides of a stage — a "hole in the middle" results; sounds panned toward the center instead are perceived to come either from the left or right, and may sound "softer." Issues of speaker placement become especially problematic in larger performing spaces (e.g. halls such as Kilbourn and larger), since if the speakers are positioned properly for those in the sweet spot, listeners near the far left and right of the hall may receive no direct signal. Compromises often are necessary.

During the 1960s some composers and bands, dissatisfied with the one dimensional left-right imaging and the limitations of stereophonic reproduction, and seeking to "envelope listeners in sound," began exploring an obvious solution: add more loudspeakers. Practical limitations, and the need for some standardization, led in the early 1970s to the commercial development of **quadraphonic** reproduction systems, with four speakers placed in the corners of a room or hall. Often the left front and rear speakers, and the right front and rear speakers, were positioned facing each other, although sometimes the speakers were angled toward the center. The goal was to be able to position sounds at any two dimensional (horizontal) location within the performing space. Joystick panpots were used in attempts to move sounds in circles around the perimeters of halls or in diagonal paths. The technical method used was still intensity panning, now attempted by varying the speaker feeds sent to various combinations of between two and four speakers rather than between a single stereo pair.

Once the novelty wore off — usually after about ten minutes — it became clear to almost everyone that the sonic results of this type of "joystick quad" audio were very disappointing. Yes, you could bounce sounds around between the four speakers like a racketball, but it proved impossible to create convincing phantom images at points between the speakers. Even imaging between the two front speakers generally was poorer and less stable than with stereo, owing to the large angle between these speakers and the resulting lack of any sweet spot. Production of commercial quadraphonic mixers and other gear was discontinued within a few years.

The most important problem of the many problems with traditional quad mixing and playback is that intensity panning does not work with rear speakers, or with combinations of front and rear speakers. Our ears receive little or no direct signal, or at best a confusing diffracted signal, from sources behind our ears, and very conflicting signal time arrival, phase, timbral and HRTF information between the ears. Instead of providing a rich two dimensional soundfield, quad often reduced the soundfield to four points.

Still the conundrum persisted: our brains can perceive three dimensional holographic images by interpreting differing phase information received by our two eyes. Why can’t we perceive holophonic images received by our two ears? In fact, we can. As research continued, isolated recordings made under controlled conditions, usually employing headphones rather than loudspeakers, provided some tantalizing examples of effective three dimensional sound localization. And concurrent with the rise and fall of commercial quad recordings during the 1970s, a group of English academics and engineers led by Michael Gerzon began devising sound recording and reproduction theories and systems they termed *ambisonic*. Although little known outside of England until the mid 1990s, *ambisonic* processing today offers composers some of the most effective techniques for sound localization and ambience.

The next widespread use of multi-channel audio, however, occurred in motion picture theaters during the 1980s. While stereo speakers, placed behind the screen, were still used for music, a center speaker, also behind the screen, was added to stabilize the localization of dialogue in the center of the screen for moviegoers seated anywhere in the theater. Side and/or rear speakers were added to place the sounds of off-

camera events, such as distant traffic or forest sounds, "in the distance," and to envelope the audience in "surround" effects, such as storms. Rear speakers also were employed for reverberation and echos. As special effects (train wrecks, explosions, dinosaur footsteps, alien spacecraft and so on) became an increasingly important selling point of commerical Hollywood films, one or more elephantine subwoofers also were added to produce very low frequencies (less than 50 hertz or so) that cannot be produced at all (or at least not cleanly at high amplitudes) on standard sound reinforcement speakers.

7.3. 5.1 surround sound

By the mid 1990s **5.1** channel "**surround sound**" systems had become the "industry standard"[1] for cinema-based multichannel audio. With the subsequent marketing of "home entertainment systems, " 5.1 systems also have become increasingly prevalent in homes, often replacing stereo systems. Like it or not, 5.1 systems are now widely used not just for film and video, but also for music reproduction, and cannot be ignored by musicians. (That is why we have such a system in our MIDI studio.)

Despite the term "surround sound," however, cinema-style 5.1 systems do not attempt to position sounds at precise two dimensional locations within a room. Rather, the six speakers have particular functions in film and video production. For this reason, the speakers often are not matched, but instead often differ in size and shape, have different frequency responses and different sound radiation characteristics.

Conventions of 5.1 cinema-style audio

- The two *mains* (front pair of speakers) are used for most or all of the music, which thus is generally still mixed to stereo (except for — here comes that term again — special effects).
- The center channel is the dialogue channel, and rarely contains much music.
- The "*surrounds*" (rear speakers, which ideally should be only slightly behind the listener) are used for sound effects and for ambience (reverberation and other effects that create the aural illusion of a particular environment or space)
- The *LFE* ("*low frequency effects*") channel is just that. Because only very low frequencies are recorded to this channel, which is intended for a subwoofer, the 5.1 specifications limit the frequency bandwidth of this channel to about 1/10 that of the other channels (hence its sobriquet, the *.1* channel).

With 5.1 systems now available in a variety of venues besides motion picture theaters, and 5.1 distribution media also readily available — DVD video, DVD-A (DVD Audio, which provides much higher audio quality), SACD, computer playback and multitrack tape and disk systems — one question for musicians is how to use this six channel speaker array effectively for purely musical purposes. The answers often are not easy, since it is highly unlikely that the 5.1 specifications would have become a standard if purely musical considerations had been paramount. (In fact, 6.1 and 7.1 alternatives have been proposed for musical purposes.) How closely should the cinema-style conventions be followed? Should the rear speakers be used only for reverberant ambience? When mixing contrabass or kickdrum sounds, some of the signal must be sent to the sub, since the fundamentals of very low pitched tones are below the range of the other speakers. However, these sound sources also must be sent to the other speakers to create the higher harmonics that the sub cannot produce, and without which the double bass or kick drum will sound pallid and mushy.[2] Some improvement over stereo in the precision and stability of frontal left-right imaging is possible for sounds intended to be heard "sort of from the left" or "a little to the right" by panning the sound between one of the mains and the center speaker. But the center speaker raises complications as well, especially since it normally will be positioned along an axis with the two mains, and thus will be somewhat closer to the sweet spot than the left and right front speakers. Additionally, we now have two center positions to deal with — the center speaker, and the phantom center created by routing equal signal levels to the two mains.

The most important caveat, however, is that 5.1 systems cannot be used successfully with standard *intensity panning* techniques to position sounds at arbitrary points between the front and rear speakers or behind the listener. For reasons discussed earlier, this just does not work.

[1] a term I loathe, as many of you know

[2] In fact, built-in *bass management* circuitry of many 5.1 systems will do this automatically if we don't. Bass management systems employ crossover networks that route all frequencies below about 80 hertz from all other channels to the LFE channel. But not all 5.1 systems include bass management circuits.

7.4. Ambisonic sound processing

Ambisonics is a mathematically-based signal processing system that attempts to capture and reproduce information about a complete three dimensional soundfield, including the preceise location of each sound source and the ambient characteristics of a real or simulated performance and performance environment. Ambisonic theories and technology originally were developed during the 1970s for the recording and playback of acoustic music (especially of "classical" and avant garde rock genres) using custom-made analog equipment, which was (and remains) expensive. Today, digital software is more often used for ambisonic processing of both the recording and playback stages. And although ambisonic recording continues as a rather niche audiophile market in England,[3] these techniques presently are not widely used elsewhere in commercial recording. However, ambisonic procedures are increasingly common in computer music production, and are beginning to make inroads in the commerical recording industry.[4]

Ambisonics is a encoding and decoding process. During recording — whether of live acoustic music or of computer-based synthesis or mixing — information about the location of sound sources and room ambience is processed mathematically and encoded into a number of discrete channels of data. Unlike traditional stereo, 5.1 and other multichannel formats, however, the data channels of ambisonic formats are not speaker feeds, and cannot be played directly. Ambisonic channels have nothing to do with loudspeakers, but rather describe what are called in ambisonic terminology *spherical harmonics*. The principal spherical harmonics are sound planes, such as the front/rear placement of sounds, their left/right placement, and their height (up/down placement). Encoding (recording and/or mixing) and decoding (playback) are de-coupled, and are entirely distinct processes. Ambisonic encoding is not tailored to a specific number of playback speakers Rather, the encoding process is designed to capture, as accurately and completely as possible, information about what is happening within a soundfield. For playback, these data channels must be decoded for a particular speaker configuration. Thus, an ambisonic recording can be decoded for a standard stereo system or for a playback system with 4, 5, 6, 8, 16 or any number of loudspeakers. In ambisonic terminology, a particular speaker configuration, which includes not only the number of loudspeakers but also their placement within a room, is called a *rig*, or a *speaker array*. In general, the more (high quality) speakers that are available for playback, the more accurate and detailed the reproduction of the encoded data.

Decoding ambisonic signals

The original source of ambisonic recordings, still in use (although not widely) was a specially designed array of seven very closely spaced microphonic capsules contained within a single enclsure and called a *soundfield microphone*. The seven capsules are designed to capture sound from a single point,[5] but with different orientations. Six of the mics are *cardioids*, which are oriented

- forwards (facing the front wall)
- backwards (facing the rear wall)
- to the left (facing the left wall)
- to the right (facing the right wall)
- upwards (facing the ceiling), and
- downwards (facing the floor)

These six mics "hear" the sound "from the front," "from the rear," "from the left and right," and from "above" and "below" the microphone (and the listener's) position. The seventh mic has an *omnidirectional* response (equally sensitive to sounds arriving from any direction and angle), and captures the overall amplitude intensity of the sound.

The original, simplest and still most widely used ambisonic encoding format, called **first order** or **B** format, applies mathematical phase shifting and amplitude scaling equations to convert the signals from these seven mics into four channels of data, comprising four *spherical harmonics*, or components, of the soundfield.

[3] In fact, technically the term *ambisonic* is a registered trademark of the small English recording company *Nimbus Records*, the most fervent proponent of these techniques. In common usage, however, the term is applied widely to the mathematical techniques first developed by Michael Gerzon and his associates that are now being employed and extended by engineers and musicians throughout the world.

[4] For example, options for certain types of ambisonic decoding have been included in the DVD-Audio specifications.

[5] Physically, this is impossible of course, but with some mathematical processing it can be simulated.

• The omnidirectional signal, termed the **W** component, describes the time varying amplitude intensity of each source sound.
• Signals from the front and rear facing mics are processed to encode a *depth* harmonic, recorded to channel 2, that describes how the *W* (intensity) component is distributed along an **X** axis between the front and rear of the soundfield.
• Signals from the left and right facing mics are processed to encode a *width* component, recorded to channel 3, that describes how the *W* component is distributed along a **Y** axis between the left and right of the soundfield.
• Signals from the upward and downward facing mics are processed to encode *heighth* information, recorded to channel 4, that describes how the *W* component is distributed along a **Z** axis running vertically between the floor and ceiling.

(Note that the ambisonic labeling of these axes is non-standard. In algebraic graphs, the *X* axis usually runs left to right, whereas in ambisonic diagrams the left-right axis is the *Y* axis.)

The original decoders for ambisonic recordings were hardware units that performed amplitude matrixing to send the correct amount of each ambisonic channel component (*W, X, Y* and *Z*), with correct phase information, to each loudspeaker. Hardware ambisonic decoders still are manufactured and sold in England, primarily for audiophile home systems. Today, however, most ambisonic decoding is done in software. To do its work properly, a decoder must know the configuration of a speaker array — not just the number of speakers, but also their placement within a room. Decoder hardware and software typically includes programming for a number of common rigs of symmetrically placed speakers — stereo, 4 speakers, and so on.

The signals sent to opposing speakers — front and rear, left and right, or high and low) — contain not only amplitude differences (similar to standard *intensity panning*), but also, and more importantly, phase differences (opposite speakers often are 180 degrees out of phase), as well as differences in reflections and other locational cues. In order to reproduce any of the three principal dimensions (or "spheres") of the recorded soundfield adequately, it is necessary to have at least one speaker on opposing perimeters of the sphere. Thus, decoding the left/right *Y* component requires at least one speaker to the left of the listener and another speaker to the right. Accurate representation of the full depth (*X*) sphere requires at least one speaker in front and one in back of the listener. However, to reproduce both the *X* and *Y* spheres fully, so that listeners receive correct amplitude, phase and other cues at both ears, we need at least 2 front, 2 rear, 2 left and 2 right speakers. This can be accomplished by placing four speakers near the corners of a room, each angled slightly toward the center.

This might look like a quad setup, but the speakers are functioning quite differently here than in quad recordings, "pushing and pulling" the air in concert to provide localization cues for each sound source. Ambisonic reproduction does not recreate the original soundfield (something that would require thousands of very closely spaced speakers). But by providing coherent (non-conflicting) information about each sound source from multiple speakers, it can provide a very convincing illusion of the originally recorded or, in the case of computer-generated music, an intended soundfield.

If an ambisonic recording is decoded for stereo speakers, the left-right stereo imaging usually will be very good. Depth (*X*) localization, however, will be only partial. The reproduction will convey some information — generally much better than standard stereo reproduction — about the distances of sound sources in front of the listening position (since there are two speakers to our front). However, because there are no rear speakers, the imaging of forward sounds may not always be precise, and sounds intended to emanate from the rear may be perceived instead in mirror-like fashion to come from the front, perhaps at reduced amplitude. Height information (the *Z* component) will be discarded by the decoder. Since 2 stereo speakers (or, for that matter, 4 quad speakers) are at the same height, there is no way for the decoder to distribute the out-of-phase "push-pull" height data to complementary transducers.

Four speakers in a rectangular configuration is the minimum "*rig*" necessary for full horizontal (left/right/front/rear) ambisonic imaging. Five speakers outlining the points of a pentagon may provide somewhat better horizontal imaging, and a hexagonal array of six speakers perhaps better still, especially for side and rear imaging and with idiophonic sounds. However, such configurations are not common. Speaker configurations designed for ambisonic reproduction tend to grow by powers-of-two.

Periphony: the height dimension

As noted earlier, decoding of the *heighth* (*Z*) component requires that speakers be placed above and below ear level. Whereas horizontal ambisonic decoding provides a two dimensional soundfield, in which sounds can be localized at any point along a flat horizontal plane within the performing space, the addition of the heighth componenet turns the performance space into a full three dimensional sphere. In ambisonic terminology this is called *periphony*, from the Greek roots for "around the edges."

It is possible to achieve usable 3D ambisonic imaging with six speakers: a horizontal quad array and two additional speakers positioned front/center and rear/center, elevated 3 or 4 meters above the horizontal array. Full, accurate periphony, however, requires a minimum of eight speakers, generally a quad array below ear level (near or on the floor) and a second quad array above ear level, with the speakers near the eight corners of a cube. (Sixteen speakers work even better, of course.)

For a recording of, say, a solo piano work, or of any music in which all sound sources are at approximately the same height, there might appear to be little to be gained to justify the added expense and complication required to decode the ambisonic heighth dimension (especially in halls such as Kilbourn, where speakers and cables must be set up before each concert). However, listeners who have experienced high quality "periphonic" reproduction often have observed that not only does the music seem "fuller" and more "life-like" (due in part to the different reflections and ambient qualities coming from the elevated and lowered speakers), but also that they seem to be able to "hear more detail" in the music (an observation similar to comparisons between stereo and monophonic reproduction). For computer-generated music, of course, true 3D imaging of sounds, and the ability to move sounds in *any* direction and at any angle, raises many new formal, coloristic and expressive possibilities.

Some advantages of ambisonic reproduction

Besides offering improved imaging of sound sources compared with standard stereo and multichannel reproduction systems, ambisonic encoding/decoding provides other significant benefits. The sweet spot is much larger, and imaging remains accurate and stable over a wide area that often encompasses most of a medium sized room or concert hall. Listeners seated to the left or right, forwards or toward the rear of a hall receive essentially the same sonic information. Additionally, speaker placement and angles are not nearly as critical as in stereo and 5.1 reproduction; the speakers can be moved around somewhat with considerably less degradation in the quality of the reproduction.

Caveats and limitations

Since ambisonic decoding can be rather complex, and requires accurate reproduction, it is important that high quality speakers be used. The speakers should be matched, (identical models work best) so that they have identical gain and phase responses. The speakers must be time aligned, and ideally should cover the full frequency range, or at least as wide a frequency range as possible. Ambisonic recordings can sound quite bad when played back on cheap, unmatched speakers.

Since recordings and computer-generated mixes realized with ambisonic techniques provide their own ambient and spatial information, and the goal of decoding and playback is accuracy in the reproduction of this information, the performance space should be relatively dry and neutral, and add little of its own coloration. Kilbourn Hall provides a wonderful ambience for the performance of a string quartet. But when this hall ambience is overlaid on top of localization cues and ambient qualities already present in the ambisonic source material, some smearing of imaging can result, and the overall ambient quality may become too "wet" and diffuse.

Furthermore, megaphone-shaped halls such as Kilbourn, in which the floor ramps upwards from the front to the rear of the hall, can raise serious problems in speaker placement, since listeners' ears are at varying heights. This is a greater problem with ambisonic recordings than with traditional stereo and multichannel reproduction. Even for horizontal ambisonic reproduction, rear speakers that are close to ear level for the late-arriving patrons in the back of the hall will be well over the heads of listeners seated in the first few rows, while the front speakers will be"submerged" below foot level for listeners in the rear seats. Periphonic reproduction becomes even more problematic. There is really no way to position 8 speakers in the corners of a cube in a hall such as Kilbourn. With some ingenuity, such problems need not remain insoluble, especially since ambisonic reproduction does tolerate some fudging in speaker placement. However, composers need to take such practical considerations into account when "choreographing" sound

localization in a composition. Otherwise, sound localization procedures that may sound electrifying in one of our studios may translate poorly to larger performing halls.

Alternative ambisonic formats

Although *first order* (*B*) format is the most widely used ambisonic encoding/decoding scheme, a number of alternative ambisonic formats also are available. *Higher order* formats provide additional ambisonic signal components, beyond the common *WXYZ* set, in attempts to achieve greater accuracy in localization. *Second order* formats, which include 9 channels of encoded data, can be encoded on ECMC Linux systems with the *vspace* program and decoded with the *ambidec* program. (However, I still recommend B format for your work.) A variety of *UHJ* formats are designed for compatibility with two channel stereo reproduction systems. *G formats* provide pre-decoding of B format for playback on 5.1 systems, but currently no *G* format software is available on any of the ECMC systems. For more information on these and other aspects of ambisonic theory and practice, see the *Links* page on the ECMC web site.

7.5. Using vspace

A good introductory program that I recommend for applying spatial localization to monophonic input soundfiles on ECMC Linux systems is *vspace* by Richard Furse. *vspace* can create output soundfiles in both first order or second order ambisonic format, and also in standard stereo WAVE format. These encoded output files can be decoded for particular speaker arrays — usually for a stereo or quad rig — with Furse's companion *ambidec* program or with various other ambisonic decoding utilities.

vspace is a soundfile localization, spatial ambience and mixing program. It accepts one or more monophonic WAVE format input soundfiles. These input soundfiles can have a sampling rate of 44.1k, 48k or 96k. However, all input soundfiles must have the same sampling rate, which also will be the sampling rate for output soundfiles. Unfortunately, while supporting three common sampling rates, *vspace* is **limited to 16 bit resolution**: 24 bit and 32 bit input soundfiles cannot be read nor written by the program, nor by its companion *ambidec* decoding utility. Since we rarely deal with 96k 16 bit soundfiles — 24 or 32 bit resolution generally is more important to high audio resolution that higher sampling rates), *vspace* is for all intents a cd quality 16 bit program.

vspace processes and mixes the input soundfile(s) with ambisoninic formulae to apply simulated spatial localization and room ambience cues, and writes the processed samples to one or more output soundfiles. When two or more output soundfiles are created, they share many common features, including:

- the same fixed or time varying (moving) spatial localization of each source soundfile
- the same mixing specifications for the input soundfiles
- the same dimensions and reverberant qualities of the "virtual" (simulated) room or hall into which the source soundfiles are placed

However, two or more output soundfiles will differ in the following important respects:

☞ Format: *vspace* can produce output soundfiles in first order or second order ambisonic formats, and also in standard stereo or mono WAVE format

☞ Simulated microphones with different polar patterns (sensitivity to sounds coming from various directions) are used for different output formats, and these mics often are placed at different locations within the virtual room. WAVE output soundfiles will not produce exactly the same reverberant ambience, nor the same wet/dry mix, as ambisonic output. Ambisonic output is designed to capture the complete wet/dry soundfield, whereas WAVE output generally is optimized for either the direct or the reverberant signal.

vspace is a non-graphical command line program, run from a shell, but it contains far too many arguments to be typed in on a command line. Instead, these arguments are consolidated within an input *parameter file*. The ECMC utility **vspacetp** ("*vspace* **t***em***p***late*") provides a template for creating this ASCII parameter file. The template includes all of the required and most of the optional parameter arguments available within the *vspace* program. For parameters with default values in *vspace*, these defaults are included in the template. However, for some values, such as as file names and room dimensions, there are no defaults, and the template includes a *??* symbol for these arguments.

vspace shares with many other mixing programs the concept of *tracks*. Every input soundfile must be assigned to a particular track. Unlike many mixing programs, however, *vspace* allows us to mix together

two or more input soundfiles so that they are processed and sound *simultaneously* on a track. Sound localization is done by track rather than in terms of individual soundfiles. The spatial localization of all input soundfiles assigned to a given track can be either fixed (remaining stationary) or moving (changing location over time). However, at any given instant, all soundfiles sounding on a track will emanate from the same virtual location. To place two simultaneous sounds at two different locations, the two sounds must be assigned to to diferent tracks.

After creating a parameter file with *vspacetp* you must edit this file with a text editor, changing the *??* symbols and, often, some of the default values as well to the values you want. When the template file is ready, it is used as the input file to *vspace*. The entire process of creating soundfiles with *vspace* runs like this:

(1) Obtain a parameter template file with *vspacetp*.
(2) Edit this template file, changing some of the argument values within the template in order to obtain the result you want.
(3) Run the *vspace* program with this parameter file:

vspace [flag options] parameterfilename

(4) If you have created an output soundfile in ambisonic format, decode this soundfile for a particular speaker array (most often for stereo or quad) and play it.
(6) If you are not satisfied with the result, loop back to step 2.

Example vspace parameter files

Example *vspace* input parameter files, used to create the *vspace* example soundfiles in */sflib/x*, are available on the ECMC Linux systems. To list these example parameter files, type

lsex vs or else *lsexvs*

To display one or more of these example files, type:

getex <filename>

Example: *getex vspace1*

7.5.1. Creating an input parameter file with *vspacetp*

For full details on using *vspacetp* consult the *man* page for this utility. The syntax for obtaining an input parameter template file is:

vspacetp [-v or -s] inputsoundfile(s) [T inputsoundfile(s) T inputsoundfile(s) etc.] [> filename]

or

vspacetp [-v or -s] - [> filename]

Flag options: The default *vspacetp* template includes brief usage and suggested value comments, preceded by the *vspace* # (pound sign) comment symbol, for most parameters. If a -v ("verbose") flag, designed primarily for new users, is included on the command line, *vspacetp* will provide more detailed usage comments and examples. By contrast, *-s* ("succinct") flag will instead provide a bare-boned template with only a few comments designed for advanced users.

Soundfile argument(s): Your *vspacetp* command line should include the names of all input soundfiles to be used (although you can always add more soundfile inputs later while editing the template file). If these soundfiles are located in your current working soundfile directory (*$SFDIR*) or in any of the *sflib* directories you need only type the name of the soundfile, and *vspacetp* will locate the soundfile and fill in its full path, which is required by *vspace*. You can omit *.wav* extensions if you wish, and *vspacetp* will supply them. If an input soundfile is located in some other directory, type in the subdirectory name or full path as well as the soundfile name. If you simply want to display a *vspacetp* template without specifying an input soundfile, substitute a **-** (minus sign) for the *inputsoundfile* argument.

By default, all of the input soundfiles that you specify will be assigned to the same track (track 1), and thus will share identical spatial placements. To assign input soundfiles to separate tracks, include a *T* ("track") argument whenever you want to create a new track for the following soundfile(s). However, do not specify a *T* for the first track — only for subsequent tracks.

Example vspacetp command lines

(1) *vspacetp -v -*

Result: A verbose *vspacetp* template, with no input soundfile specifications, is displayed.

(2) *vspacetp clockticks3.wav > clockticks3.vsp*

Result: A template for running *vspace* with your soundfile *clockticks3.wav* is created and is written to a file named *clockticks3.vsp* in your current working Unix directory.

(3) *vspacetp -v bottlefilling gristmill*

Result: A verbose template that places the two *sflib/env* soundfiles *bottlefilling.wav* and *gristmill.wav* on a single track will be displayed in your shell window.

!! > vspace4

Result: Repeat the previous command, but this time write the template to a file named *vspace4*.

vspacetp -s monomix1 T monomix2 monomix3 T monomix4 T cym1 > ambisonicmix1

A succinct template is created and written to a file named *ambisonicmix1* in your current Unix directory. Within this template soundfile *monomix1.wav* is placed on track 1, soundfiles *monomix2.wav* and *monomix3.wav* are placed on track 2, soundfile *monomix4.wav* is placed on track 3, and the *sflib/perc* soundfile *cym1.wav* is assigned to track 4.

A sample template

The command

vspacetp whisper.ah.wav gu.c4.roll.wav T iceskate.wav

will produce the template file shown below, incorporating three input *sflib* soundfiles on two tracks. Line numbers, not a part of *vspace* parameter files, have been added here to simplify the discussion that follows.

```
################ example vspace parameter file produced by vspacetp : ###########
1  # all uncommented parameter values shown as ?? must be changed to a float value
2  # use full floats like this: 3.0 , not abbreviations like this: 3.
3  ##### [1] GENERAL parameters and EARLY & LATE REFLECTION parameters #####
4  tempo 60;
5  skip 0 ;
6 #  disable early reflections; # uncomment to disable EARLY REFLECTIONS
7 early reflection time 0.25;   # default .25, normal range .25 - .75
8 early reflection minimum gain 0.005;

9 # disable late reflections  # uncomment to disable LATE REFLECTIONS
10  # late reflection time default: calculated from room dimensions
11 #late reflection time ?? ;          # Normal range: .75 - 3.0
12 late reflection gain 1;           # > 1.0 = wetter, < 1.0 = drier
13 late reflection echo density 1.0;   # > 1.0 = wetter, < 1.0 = drier
14 late reflection frequency density 1.0;   # > 1.0 = wetter, < 1.0 = drier
15 # Low pass filter cutoff frequency : by default calculated from room dimensions
16 # late reflection cutoff ?? ; # in hertz, normally between 1000 & 2000
17 # ================================================================
18 ##### [2]  ROOM DIMENSIONS & REFECTIVITY parameters  #####
19 distance unit metre;    # metre (sic), yard or foot
20 room {
21 dimensions     -??, ??  ,     -?? , ?? ,        -?? , ?? ;
22 #            -rear, front,   -right,  left ,    -down,  up
23 # Set the reflectivity of each of the 6 walls above: Ranges 0 - 1.0:
24 reflections     0.5,  0.5,   0.5, 0.5,      0.5, 0.5 ;
25 }
26 # ================================================================
27 ##### [3] RECORDING devices (mics) : each "recording" produces an output soundfile #####
28  # A recording device consists of: (1) a soundfile name (whose extension specifies
29  # the output format); (2) a mic device, which determines mono or
30   # stereo WAVE or 1st or 2nd order ambisonic output; (3) mic location;
31   # (4) for cardiod mics, mic direction; (5) optional core radius
32  # mic device:       resulting output soundfile format:   filename extension:
33  # ambisonic          1st order B format 4 channel ambisonic   .wxyz
```

7.5.1: A sample *vspace* template

```
34  # second order ambisonic  2nd order 9 channel ambisonic       .fmh
35  # cardiod                 mono WAVE                   .wav
36 # cardiod pair             stereo WAVE                 .wav
37 # figure-of-eight          mono WAVE                    .wav
38  # omindirectional          mono WAVE                    .wav
39 # You MUST uncomment and edit all of the parameters for at least one
40 # "mic device" below to create one or more output soundfiles
41  # Mic locations are often in the center of the room <0,0,0> but can be placed
42  # anywhere within the room
43 # ---------------------
44 # (1) 1st order B-format 4 channel ambisonic file: Playable only after decoded.
45 # recording ???.wxyz  {
46 #  device ambisonic;
47 #  location <0, 0, 0>;
48 #  gain 1.0 ;
49 #  core radius 1;
50 # }
51 # ---------------------------------------------------
52 # (2) Second-order ambisonic 9 channel ambisonic file. Playable only after decoded.
53 # recording ???.fmh {
54 # device second order ambisonic;
55 # location <0, 0, 0>;
56 # gain 1.0 ;
57 # core radius 1;
58 # }
59 # ---------------------------------------------------
60 # (3) Coincident stereo pair for stereo WAVE output soundfile
61  # Direction vectors specify where the cardiod pair mics are pointing:
62  # Recommended:  direction <1, 1, 0>, <1, -1, 0>;
63 # recording ???.wav {
64 # device cardioid pair;
65 # location <0, 0, 0>, <0, 0, 0>;
66 # direction <1, 1, 0>, <1, -1, 0>;
67 # gain 1.0 ;
68 # core radius 1;
69 # }
70 # ---------------------------------------------------
71 # (4) An omnidirectional or cardiod pair mic near rear of the hall to isolate & test
72 # reverberant qualities.

73 # recording ???.wav {  # use EITHER omnidirectional (mono) or else cardioid pair (stereo)
74 # device omnidirectional;
75 # location <??, ??, ??>;   # use for omnidirectional

76 # device cardioid pair;
77 # location <??, ??, ??>, <??, ??, ??>;  #  use for cardiod pair
78 # direction <0, 1, 0>, <0, -1, 0>;  #  use for cardiod pair
79 # gain 1;
80 # }
81 # ================================================================
82 ##### [4] TRACK parameters (MIX and MOTION) #####
83 track {    #  Track 1:
84   mix     {   # soundfile inputs for track 1:
```

```
85     0  /sflib/africa/whisper.ah.wav  gain 1.0 ;
86     0  /sflib/chinaperc/gu.c4.roll.wav  gain 1.0 ;
87        }
88   motion  {   #  add fixed, line or arc locations for this track below:
89        ???
90        }
91 }
92 track {         #  Track 2 : ----------------
93    mix      {   # input soundfile(s) for track 2 :
94      0  /sflib/env/iceskate.wav  gain 1.0 ;
95        }
96   motion  {   #  add fixed, line or arc locations for this track below:
97        ???
98        }
99 }
100   ## --- Template for adding more tracks: ---
101 # track {
102 #   mix      {   # add input soundfile(s) for track below:
103
104 #         }
105 #  motion  { # add fixed, line or arc locations for this track below:
106
107 #        }
108 #}
#################### end of sample vspace parameter file created by vspacetp ######
```

7.5.2. Editing vspace parameter files

The template is divided into four major sections:

[1] *GENERAL parameters and EARLY & LATE REFLECTION parameters* (lines 3 through 16 above);

[2] *ROOM DIMENSIONS & REFECTIVITY parameters* (lines 18 through 25 above)

[3] *RECORDING devices (mics)* (lines 27 through 80 above); and

[4] *TRACK parameters (MIX and MOTION)* (lines 82 through 99, and also commented lines 101 through 108; the commented blank track lines at the end are provided in case you want to add one or more new tracks to the template at some point in the future)

Comments begin with a **#** pound sign and can be placed either on separate lines or after parameter values. Most *vspace* argument values are floating point or integer numbers, but some parameters are file names and paths, and some parameters are vectors. A *vector* usually is a group of three floating point or integer coordinates that control a single parameter. The three values are separated by commas and are enclosed within < and > delimiters, like this:

location <0, 0, 0>;

For parameters with default values in *vspace*, these defaults are included in the template. For file name and vector parameters with no default value, a *???* symbol (which must be changed) is included instead.

All parameter value lines must end with a semicolon. Major subroutines, including the *room* specifications (lines 20 through 25), each "*recording device*" (lines 45-50, 53-58, 63-69 and 73-80) and each *track* specification (lines 83-91 and lines 92-99) include several parameters, and these subroutines must be enclosed within matching **{** and **}** curly braces. Each *track*, in turn, includes two subroutines: *mix* (with input soundfile values for the track) and *motion* (with spatial localization values for the track). These *mix* and *motion* subroutines within each track also must be enclosed within matching curly braces (lines 84-87, 88-90, 93-95 and 96-99 above). Make sure that you do not inadvertantly delete any of these curly braces when editing the template or the job will not run.

In fact, you should exercise greater than normal care when editing *vspace* parameter files. The *vspace* parser is fairly primitive, easily becoming confused, and its error messages often are not very helpful,

frequently stating simply that there is a "parse error near line xx." When editing parameter values, always use either full floating point notation, like this: *8.00*, or else integers. Do not abbreviate floats by leaving out digits after the decimal point, (like this: *8.*) or you likely will get a "parser error." Other frequent causes of "parse error" aborts include

- omitting the semicolon at the end of a paramter definition
- mis-spelling parameter names or arguments (e.g. mis-spelling *cardioid* as "*cardoid*")
- using the wrong type of *location* and/or *direction* specifications for a particular types of microphone
- specifying parameter arguments in the wrong order

GENERAL parameters and EARLY & LATE REFLECTION parameters

These parameters are changed less frequently than the parameters within the other three sections of the template. Often it is possible to skip over these parameters when editing the template, employing the default values with good results. Still, there may be occasions when you wish to change some of these values.

tempo : The start times for all soundfiles are given in beats, and the default beat is 60 bpm (beats per minute), so that each beat equals one second. Changing the tempo to, say, 120 will double the tempo, and the rhythm of soundfile entrances will be twice as fast.

skip : It is possible to skip into a mix by a specified number of beats. If the first four seconds of a *vspace* job sound fine but you are not happy with the ending, temporarily set the *skip* value to *4.0* to save computation time while you fix the end.

vspace employs one algorithm to calculate and produce early reverberant reflections, and a separate algorithm to create late reflections. The **early reflection time** parameter (line 7) sets the time during which the computationally more expensive early reflections algorithm (which caluculates and creates delay times from the sound sources off of the six walls) will be used. By default, the first quarter of a second (0.25) of the hall reverb is produced by the early reflections algorithm, and the rest of the reverberation is created by the late reflection algorithm, which produces denser (but, in terms of spatial localization, less "accurate") reverberation. When the amplitude of the input signal(s) is less than the **early reflection minimum gain** value (line 8, default value .005 of maxamp), no early reflections will be created, saving computation time for reflections that lkely will not be heard anyway.

The **late reflection gain** (line 12), **late reflection echo density** (line 13) and **late reflection frequency density** (line 14) provide a means of adjusting the prominence and the "wetness" of the reverberation. Values greater than 1.0 for these parameters increase the "wetness," while values less than 1.0 create a "drier" ambience. By default the reverberation time for the reverb "tail" is calculated from the room dimension and reflectivity coefficients, but one can substitute a reverberation time by uncommenting and editing the **late reflection time** parameter. Late reflections are sent through a low pass filter with a default frequency cutoff value again determined by the room specifications. It is possible, however, to use the **late reflection cutoff** parameter, generally with a value of between 1000 and 2000 (hertz), to manually adjust the "brightness" (or "warmth") of the reverberation.

Finally, when you are performing tests with elements other than reverbation, it is possible to turn off the early and/or the late reflection algorithms to save computation time by uncommenting the **disable early reflections** (line 6) and/or the **disable late reflections** (line 9) parameters, which have no argument.

ROOM DIMENSIONS & REFECTIVITY parameters

By default, the unit of measurement for the room dimensions (line 21), for microphone locations (lines 47, 55, 65 and 77) and for the *motion* parameters for each track is the *metre* (note the English rather than American spelling). However, the **distance unit** parameter (line 19) enables you to change this to the *yard* or to the *foot* if you prefer.

The room **dimensions** vector values (line 21) are among the most important parameters in the template,setting the size of the simulated room or hall. The two ambisonic *depth* (*X*) dimensions (the distances from the center of the room to the rear wall and to the front wall) are given first, followed by the two ambisonic *width* (*Y*) dimensions (the distances from the center of the hall to the right and to the left walls), followed by the *vertical* (ambisonic *Z*) distances from the center to the floor and to the ceiling. The rear,right and down distances must be preceded by a - (minus) sign, and the 6 coordinates, separated by commas, MUST

be specified in the rather non-standard order

-rear, front, -right, left, -down, up

Remember that only two stereo speakers are necessary to adequately convey *Y* (left-right) information. Effective front-rear placement, however, generally requires at least four loudspeakers positioned in the corners of a square. Effective reproduction of *height* (*Z*) data generally requires at least eight speakers positioned in the 8 corners of a cube. If you try to move sounds up or down with *vspace*, but the output soundfiles are played on a rig (speaker configuration) in which all of the speakers are at the same height, listeners may hear amplitude changes and changes in the reverberant quality, but not a realistic simulation of different heights.

The **reflections** vector (line 24) enables us to set the sound reflectivity vs. absorption qualities for each of the six walls, often the most imortant factor in determining the dry/wet mix within the output sound. The six values within this vector are mapped to the corresponding six walls defined in the *dimensions* vector immediately above:

rear wall, front wall, right wall, left wall, floor,ceiling

RECORDING devices (mics)

This is the most complex and, ultimately, the most imortant section of the template. *vspace* can produce output soundfiles in one of three formats:

- first order (*B format*) ambisonic, written to 4 channels
- second order ambisonic, written to 9 channels in *Furse-Malham Higher Order* format
- standard stereo WAVE format soundfiles

Remember that the channel data of ambisonic soundfiles is not mapped to particular loudspeakers. Rather, the data is encoded, and is not playable by standard soundfile *play* applications such as the *play* command or by standard soundfile editors or mixers. Ambisonic soundfiles must be *decoded* for a particular *rig* (loudspeaker setup) before they can be played. Second order ambisonic files can be more accurate in certain cases, especially when played through a rig with more than four speakers, but are slower to compute and are more than twice as large.

The *Recording Devices* portion of the *vspacetp* template (lines 27 through 69) provides "mini-templates" for creating four types of output soundfiles:

(1) parameter lines for creating B format ambisonic output soundfiles (lines 44 through 50);
(2) parameter lines for creating second order ambisonic output soundfiles (lines 52 through 58);
(3) parameter lines for creating standard WAVE format soundfiles that capture the direct signal and some of the reverberation (lines 60 through 69); and
(4) parameters designed for isolating and testing the reverberation (lines 71 through 80)

Often, you will need to employ only one of these four *recording* subroutines to create a single output soundfile. At times, however, you may wish to employ two or more of these subroutines simultaneously to create different types of output soundfiles.

To create a *first order ambisonic* format soundfile, remove the leading comment # symbols from lines 45 through 50 and edit these lines. To create a *second order ambisonic* format soundfile, remove the leading comment # symbols from lines 53 through 58 and edit these lines. To create a *stereo WAVE* format soundfile that captures the direct signal and some of the reverberation, remove the leading comment # symbols from lines 63 through 69 and edit these lines. To isolate and/or test the reverberation, use the fourth "recording template," uncommenting and editing lines 73, 79 and 80 and EITHER lines 74 and 75 (to create a monophonic output) or else lines 76,77 and 78 (to create a stereo output).

The **recording** parameter (lines 45, 53, 63 and 73) specifies the name of an output soundfile, which on the ECMC systems will be written to your current working soundfile directory (*$SFDIR*) unless you specify an alternate path. The names of first order ambisonic soundfiles should end with a *.wxyz* extension; the names of second order ambisonic soundfiles should end with a *.fmh* extsions; WAVE format soundfiles should include a *.wav* extension.

Next one must specify what type of simulated microphone **device** will be used to make the "recording." For first order ambisonic the *device* should be set to *ambisonic* (line 46). This simulates an *ambisonic soundfile* microphone capsule, consisting of three coincident pairs (pointed forward, back, left, right up and down) and an omnidirectional mic. When creating second order ambisonic soundfiles, make sure that the

device is be set to *second order ambisonic* (line 54). For WAVE format outputs, however, *vspace* presents several options, depending upon what we are trying to capture:

- a *cardiod* mic (much more sensitive to sounds arriving from the front than from the back or the sides) creates a monophonic soundfile;
- a coincident *cardiod pair* (two very closely spaced mics placed at an angle somewhere between 90 and 180 degrees) creates a stereo soundfile
- a *figure-of-eight* (bi-directional) mic, sensitive to sounds coming from both the front or rear but not from the sides) produces a monophonic soundfile
- an *omnidirectional* mic, more or less equally sensitive to sounds arriving from all directions, creates a monophonic soundfile

In addition to specifying a *type* of microphone, we also must specify a **location** within the hall for the mic(s). The *location* vector has three values spearated by commas: *X* (front/rear), *Y* (left/right) and *Z* (up/down), measured from the *origin* (0,0,0) position from which the room *dimensions* are measured. Positive *X* values position the mic in the front of the room while negative values place it towards the rear; positive *Y* locations position the mic to the left of the center of the room, negative values position the mic towards the right; positive *Z* values position the mic above the center height of the room, negative values place it lower, below the center height of the room. Most often, when we are trying to capture a good mix of both direct and reverberant sound with a single mic or mic pair, the *location* will be at the *origin* location (the absolute center of the "hall"):

location <0, 0, 0> ;

However, it is possible to place a mic anywhere within the room, like this:

location <-2, 1.5, 3.3> ; # 2 meters behind, 1.5 meters to the left & 3.3 meters above the origin

When testing the reverberation the mic (or cardioid mic pair) usually are placed near the rear of the hall. (See ECMC example file *vspace1* for an example).

Cardiod pair mics require a double vector, with three values (*X*, *Y* and *Z*) for each mic:

location <X, Y, Z>, <X, Y, Z>;

Often, a *cardiod pair* is placed at the room *origin* and used to make coincident stereo pair recordings, as suggested on line 65 of the template:

location <0, 0, 0>, <0, 0, 0>;

Cardiod and *cardiod pair* mics also require a **direction** vector that specifies which way these (highly directional) mics are facing. A single *cardiod* mic requires a three digit (*<x,y,z>*) vector specifying whether the mic is pointed forward or backward (*X*), to the left or right (*Y*), up or down (*Z*). A *cardiod pair* of mics, however, require two three-digit vectors, one for each mic:

direction <x, y, z>, <x, y, z>; # direction vectors for stereo mics

The *direction* vector values usually are either *1* (pointing forward, to the left or up), *-1* (pointing to the rear, to the right or down) or else *0* (neutral), but any value between -1.0 and 1.0 can be used to angle a mic to a particular orientation. Unless you want to try something unusual you generally can employ the default *direction* values provided in the template.

Each of the four *recording* mini-templates also provides a **gain** (amplitude multiplier) parameter (lines 48, 56, 67 and 79) to adjust the amplitude of the output soundfile. Values greaster than 1.0 increase the output amplitude, values less than 1.0 decrease the amplitude.

An optional **core radius** parameter is useful in certain cases. The output amplitude of signals is determined in large part by their "closeness" to the microphone. When we "move sounds around" with programs such as *vspace*, it is possible to do things that would rarely or never happen in the real world, such as move a sound extremely close to (or even "inside" of) a microphone. This can result in extremely high (even infinite) amplitudes and severe clipping. If the

core radius 1;

parameter is uncommented from the template, the amplitude of sounds closer that one meter from the microphone will be adjusted and clipping likely will be avoided. This will have no effect on sounds greater than one meter away. You can adjust the *1* meter value to a higher or lower value if desired.

TRACK parameters (MIX and MOTION)

Within the *mix* subroutine for each track *vspacetp* has filled in all input soundfiles (lines 85, 86 and 94) requested on our command line with full paths and *.wav* extensions:

85 *0 /sflib/africa/whisper.ah.wav gain 1.0 ;*
86 *0 /sflib/chinaperc/gu.c4.roll.wav gain 1.0 ;*

94 *0 /sflib/env/iceskate.wav gain 1.0 ;*

Each soundfile is given a default *starting beat* of *0* and a default *gain* of *1.0*. You MUST include starting time and gain arguments for each input soundfile, and remember to make the gain argument a true floating point value (like *1.0* rather than *1.*). We could edit these lines to make the whisper softer, make the *gu* roll louder and start it at beat 2.5, and start the *iceskate* at beat 4.0, like this:

0 /sflib/africa/whisper.ah.wav gain 0.6 ;
2.5 /sflib/chinaperc/gu.c4.roll.wav gain 1.4 ;

4.0 /sflib/env/iceskate.wav gain 1.0 ;

To read in only a portion of a soundfile, include *from* ("skip") and/or *to* ("end") parameters at the end of these lines:

2.5 /sflib/chinaperc/gu.c4.roll.wav gain 1.4 from 1.5 to 5.2 ;

Here, we skip the first 1.5 seconds of the *gu roll* and stop reading in samples 5.2 seconds into the soundfile, so that the sounding duration will be 3.7 seconds. However, since no amplitude fade-ins or fade-outs are available in *vspace*, this likely will cause clicks unless our 3.7 second "selection" fortuitously begins and ends at zero amplitude (which is not the case with the *gu* roll soundfile).

After setting up the input soundfile parameters for a track, we must supply a fixed or time varying spatial localization for the sounds on this track (lines 89 and 97 in the sample template above). All sounds MUST be positioned WITHIN the room boundaries (as defined in the *room dimensions* vector). Three **motion** subroutines are available to place sounds within the soundfield of the hall: (1) one or more *fixed* positions; (2) *linear* motion; and (3) motion by curved *arcs*

(1) **FIXED placement**: The required syntax here is:

motion {
(beat) fixed <x,y,z> ; # initial location
[(beat) fixed <x,y,z> ; # location 2]
[(beat) fixed <x,y,z> ; # location 3] etc.
}

Example 1: *0 fixed <2.5, -3.3, 1.0> ;*
Result: All soundfiles on this track will be placed 2.5 meters forward, 3.3 meters to the right and one meter up from the *origin* of the room.

Example 2:

0 fixed <0, 3.0, -.5> ;
3.5 fixed <-1.2, 1.5, .5> ;
13.2 fixed <0, 3.0, -.5> ;

Result: Between beat 0 and beat 3.5 sounds on this track will be located 3 meters to the left and 1/2 meter below the *origin* point. At beat 3.5 the spatial placement of this track instantly shifts to a new position, 1.2 meters behind, 1.5 meters to the right and .5 meter up. At beat 13.2 the track position again shifts abruptly, back to the initial position. Shifts in *fixed* positions should only be made at points where nothing is sounding on the track. Otherwise you likely will get clicks and/or artifacts.

(2) **LINEAR motion** : To move the sounds on a track linearly (by straight line segments) from one position to another over time, use the **line** subrouteine, which has this syntax:

motion {
(start_time) to (end_time) line (x,y,z location1> to (x,y,z location2> ;
[(start_time) to (end_time) line (x,y,z location3> to (x,y,z location4> ;]
[optional additional lines specifying further movement)
}

Example: *5.5 to 15.0 line <3, 10, 0> to <1.5, -9.5, 0>;*
Result: Between beat 5.5 and beat 15.0 move the sounds from a position 3 meters forward, 10 meters to the

left to a position 1.5 meters forward, 9.5 meters to the right.

(3) **Movement by ARCS** : Arcs are curved segments — portions of a circle — which at a specified distance from a center (centre) locus point. If the distance from the center point remains constant (which is not a requirement), the arc will outline a portion of a circle. The syntax for using the *motion* **arc** subroutine is:

(start_time) to (end_time) arc centre <centervector>
start <startvector> to <tovector> in (time) ;

Example:

```
0 to 7.5 arc centre  <0,0,0>  # Note: no semicolon at end of this line
     start <6,0,0> ;
      to     <0, 6, 0> ;
      in     1.5 ;
```

Alternatively, the last three lines could be consolidated into a single line, like this:

```
0 to 7.5 arc centre  <0,0,0> ;
   start <6,0,0>  to   <0, 6, 0>    in   1.5;
```

Result: Between beats 0 and 7.5 the sound will rotate counterclockwise, starting 6 meters in front. By beat 1.5 the sound will have moved 1/4 of the way around a circle to a position 6 meters to the left. In 6 beats (4 * 1.5 beats) the sound will have traveled a full 360 degrees, making a circle. Since we have specified a 7.5 beat duration for the arc, the sound will make one and 1/4 revolutions around the circle.

Fixed, *line* and *arc* subroutines can be combined within the *motion* specifications of a track. Example:

```
motion  {
 0 to 7.5 arc centre  <0,0,0>
      start <6,0,0> to     <0, 6, 0> in     1.5;     # circular motion at first
7.5 fixed <-2.0, -3.0,0> ;              #  then for 2.5 beats a fixed location
10.0 to 12.0  line  <1,3,0> to <3,-1,0> ;      # linear motion at the end
   }
```

Result: During the first 7.5 beats the sound rotates around a circle, as in the *arc* example above. At beat 7.5 the sound shifts abruptly to a fixed position 2 meters behind and 3 meters to the right of the room *origin*. Then, from beat 10 to beat 12 the sound moves linearly from a position 1 meter forward, 3 meters to the left to a position 3 meters forward and 1 meter to the right. (The abrupt shifts at beats 7.5 and 10.0 may cause clicks or artifacts if the track amplitude is not 0 at these points.)

Running vspace

When your parameter file is ready, it is used as the input file to *vspace*:

vspace [flag options] filename

There is no *man* page for *vspace*, but typing the command name with no arguments will display a usage summary, including flag options. I recommend including a *-v* ("verbose") flag, which will cause output peak amplitude values to be displayed. Three other flag options also are available and sometimes are useful:

-e	Disable early reflections.
-l	Disable late reflections.
-s <secs>	Skip a number of seconds of output.

Multiple flags can be provided individually or concatenated. To use all four flag options, skipping the first 2.4 seconds of output processing, either of the following commands would work:

vspace -l -e -s 2.4 -v parameterfile
or
vspace -lev -s 2.4 parameterfile

If you have created one or more ambisonic output files, the utility *sfinfo* can display header information for *.wxyz* and *.fmh* format soundfiles. (However, *sfinfo* will mis-interpret the format of these ambisonic format files as standard WAVE format). The *sfpeak* utility wil correctly display the maximum amplitude, and a soundfile editor such as *rezound* can display all of the encoded channels correctly (but not play the soundfile correctly, since the soundfile editor will not decode the four channels).

Important note: vspace often has trouble reading the headers of WAVE soundfiles created by applications such as *Cubase* and *Csound*. If *vspace* abruptly aborts when it encounters an input soundfile, this is

probably the reason. To correct this problem, use the ECMC utility *fixheader* to re-write the input soundfile with a header that all audio applications, even fussy applications like *vspace*, should be able to read without difficulty. The source soundfile will not be altered or damaged in any way. The command

fixheader infile1.wav infile2.wav

will rewrite these two input soundfiles. The "corrected" versions will be named *fix.infile1.wav and fix.infile2.wav*. After playing these two "fixed" soundfiles to assure yourself that nothing has gone wrong, you can overwrite the original, "bad header" soundfiles:

mv fix.infile1.wav infile1.wav ; mv fix.infile2.wav infile2.wav

7.5.3. When you have trouble getting good results with vspace

vspace jobs run slowly. A lot of number crunching is required to compute distance coordinates, mixing balances, Doppler shifts, low pass filtering, multiple early reflection delay lines and "late reflections" reverberant diffusion. The more input soundfiles and (especially) the more output soundfiles you specify, and the more complex the spatial movement, the slower the job will run.

Ambisonic output format (usually first order B format) can provide more detailed and accurate spatialization and ambience information than standard WAVE format, but suffers from some practical drawbacks. You cannot listen to these soundfiles directly, but must decode them into a WAVE format soundfile before they can be auditioned. As you seek to perfect all aspects of a *vspace* job — the placement, movement and mixing balance of the input soundfiles, reverberant qualities, wet/dry mix, and so on — the cycle of editing the parameter file, running *vspace*, decoding the ambisonic output and finally listening to the result can become laborious. If you are not happy with the results you are achieving, may be more efficient or fruitful to break this complex, multilayered task down into more manageable, discrete steps, like this:

(1) Forget ambisonic format and reverberation for the moment. Turn off early and late reflection reverberation, either by uncommenting the *disable early reflections* and *disable late reflections* parameters in your parameter file, or else by running *vspace* with the *-e* and *-l* flags:

vspace -e -l paramterfile or else *vspace -el paramterfile*

(For a complete list of all *vspace* command line options, including the *-s* (skip) and *-v* (verbose) flags, type *vspace* with no arguments in a shell window.) In the *RECORDING devicces* section of the template, uncomment, edit and then use the *Coincident stereo pair for stereo WAVE output soundfile* mini-template to focus on the direct signal. Get the spatial localization and soundfile mixing parameters working to your satisfaction.
Remember that if you attempt to move a sound too rapidly, you may get unwanted chorusing, pitch clusters or artifacts.

(2) Now shift your attention to the reverberation, re-enabling early and late reflections, commenting out the recording device used above, and editing the fourth "*recording*" "mini-template" to create either an mono or stereo WAVE output soundfile that emphasizes the reverberant signal, with the *mic(s)* placed near the rear of the room. When auditioning the resulting soundfile, listen for unnatural coloration, hissing, "boinginess," artifacts and other problems frequently encountered with reverberation (especially of moving sound sources). Get the general quality of the reverberation to your liking.

(3) If both the localization of sound sources and the reverberation now are working better, return to ambisonic processing. Comment out the temporary RECORDING devices that you used to test spatial placement and reverberation, reactivate the B-format "recording device" (lines 44-50 in the template several pages back), create a B-format output soundfile, listen to it and try to perfect it.

\- - - - - - -

Other ambisonic encoding and decoding applications:

The *vspace* program is by no means the only (or, in many cases, even the easiest or best) way to generate ambisonic encoded soundfiles. For my own compositional work I have created many Csound instruments, including extensions to the *samp*, *tsamp* and *gran* Eastman Csound Library instrument algorithms, that produce B format output soundfiles. *Csound5* includes new unit generators (which we have not yet had time to test) for performing second order ambisonic encoding and decoding. Externals for first and higher order

encoding and decoding are available for *PD* and MAX.[6] Matt Barber has written PD encoding and decoding patches.

Note: *vspace* is NOT an open source program, but rather is copyrighted by its author, Richard Furse. Thus it cannot be included in the ECMC *Turnkey* package, but rather must be downloaded and installed separately.

7.6. Decoding and playing ambisonic soundfiles for stereo and quad

Ambisonic encoded soundfiles cannot be played accurately with soundfile editors and mixers such as *rezound*, *sweep*, *audacity* and *ardour*. The encoded channels of the ambisonic soundfile first must be decoded for a particular loudspeaker "rig," or array — most often for a stereo or quad speaker system.

☞ Playing B format encoded soundfiles in stereo or quad: *play -b*, *playbs* and *playbq*

To decode a four channel B format ambisonic encoded soundfile and send the decoded samples directly to the audio output of an ECMC Linux system, WITHOUT writing these decoded samples to a new soundfile, you can either

- use the standard *play* command with a *-b* flag immediately followed by a *2* (for stereo decoding) or else a *4* (for quad decoding; or else
- use the ECMC utilities *playbs* and *playbq*

play -b2 or the equivalent command **playbs** (short for "*play b* format in stereo") converts 4 channel B format *.wxyz* soundfiles to stereo and plays them with 16 bit UHJ decoding. UHJ decoding is an alternative to standard ambisonic stereo decoding algorithms. UHJ algorithms attempt to provide at least *some* front-rear locational definition, even with stereo playback. This is not nearly as effective as quad decoding in providing front-rear locational information, but (to my ears, at least), provides somewhat better depth information than standard stereo ambisonic decoding procedures.

play -b4 or the equivalent command **playbq** (short for "*play b* format in *q*uad") decodes 4 channel B format *.wxyz* soundfiles for quad playback.

Any number of B format soundfiles can be specified for succesive decoding/playing by these commands.

Example: *playbs stringsounds.wxyz storm.wxyz bushisanidiot.wxyz*
or the equivalent *play -b2 stringsounds.wxyz storm.wxyz bushisanidiot.wxyz*
Result: Three B format soundfiles, *stringsounds.wxyz*, *storm.wxyz*, and *georgewbushisanidiot.wxyz* are played in succession with stereo decoding.

To play these same three soundfiles with quad decoding, we would use the command:

playbq stringsounds.wxyz storm.wxyz bushisanidiot.wxyz
or the equivalent *play -b4 stringsounds.wxyz storm.wxyz bushisanidiot.wxyz*

Wildcard characters such as ***** can be used to specify groups of soundfiles. For example, to decode/play the soundfiles *a1mix.wxyz, a2mix.wxyz* and *a3mix.wxyz*, we could type:
*play -b2 a*mix.wxyz* or *playbs a*mix.wxyz*

play -b, *playbs* and *playbq* can decode input soundfiles at any sampling rate and 16, 24 or 32 bit depth. However, owing to a current limitation, they will always play with 16 bit resolution, regardless of the bit depth of the input soundfile. For more information on these utilities, consult the *playbs* or *playbq man* page.

☞ Decoding ambisonic-encoded soundfiles into standard stereo and quad WAVE soundfiles: *decb2* and *decb4*

play -b2 and *play -b4* (or *playbs* and *playbq*) are useful for quick auditioning of ambisonic soundfiles while they are still in an encoded state. Ultimately, however, we will need to decode ambisonic soundfiles into standard WAVE soundfiles for a particular loudspeaker configuration. For stereo and quad decoding, this can be accomplished

- with the ECMC utilities *decb2* and *decb4*; usually this is the recommended method

[6] For examples, see: (1) *http://www.bek.no/Members/lossius/lostblog/718*
(2) *http://iem.at/~zmoelnig/publications/ambisonic/ambisonic_en.html*
(3) *http://iem.at/Members/noisternig/bin_ambi* and
(4) *http://blog.soundsorange.net/?cat=7.*

• with the utility *ambidec*

A third potential method is to use ambisonic decoding utilities available within the *LADSPA CMT* plugin packages. However, most of these plugins are of marginal utility at best. The ambsionic *encoding* plugins do not include reverberation, nor filtering and time delay based upon the location of the sound source, and thus in my opinion are pretty much worthless. Plugins exist for decoding existing B format *.wxyz* and second order *.fmh* soundfiles into playable stereo, quad and cube (8 channel) WAVE soundfile. However, as of this writing the quad decoding plugin is broken, and the stereo decoding plugin is much more cumbersome to use than *decb2* or *ambidec*.

The ECMC utilities *decb2* (short for "decode a B format soundfile for 2 output channels") and *decb4* (short for "decode a B format soundfile for 4 output channels") decode a four channel B format input soundfile at any sampling rate and any bit depth into a standard WAVE format output soundfile with the same sampling rate and word size. **decb2** employs the same UHJ decoding algorithm as *playbs* to convert the B format input soundfile into a playable WAVE format **stereo** output soundfile. **decb4**, by contrast, converts the B format input soundfile into a **quad** WAVE output soundfile. The syntax for these two utilities is identical:

decb2 [-b <bits> -g <gain> -op <outputpeak>] inputfile [outputfile]
decb4 [-b <bits> -g <gain> -op <outputpeak>] inputfile [outputfile]

where *inputfile* is the name of (and, if necessary, path to) a single B format input soundfile and the optional *outputfile* argument is the name you wish to give to the output soundfile. If no *outputfile* argument is provided, *decb2* and *decb4* will supply a default name for this output soundfile, which will be the name of the input soundfile, with "ST" (*decb2*) or else "quad" ((*decb4*) prepended, and with the *.wxyz* extension of the input file changed to a *.wav* extension.

Optional arguments: -b , -g and *-op*

- *-b <output_bits>* : specifies the word size of the output soundfile if this differs from the input soundfile; valid arguments are 24, 16 or 32 (32 bit floats)
- *-g <gain>* : specifies a gain multiplier for the output samples
- *-op <outputpeak> : specifies a desired peak amplitude for the output soundfile; jobs run with this argument will take longer to decode*

Examples:

(1) *decb2 whistles.wxzy*

Result: Input B format soundfile "whistles.wxzy" is decoded with UHJ decoding into an output stereo soundfile named "STwhistles.wav"

(2) *decb2 -op 32000 happytimes.wxyz happytimes.stereo.wav*

Result: Input B format soundfile "happytimes.wxyz" is decoded into output stereo soundfile "happytimes.stereo.wav". Regardless of its word size, the output soundfile will have a peak amplitude of 32000 on an integer scale of 0 to 32767.

(3) *decb4 -op .98 picasso.wxzy*

Result: Input B format soundfile "picasso.wxzy" is decoded into a 4 channel output soundfile named "quad.picasso.wav". The peak amplitude of "quad.picasso.wav" will be 98 % of maxamp.

(4) *decb4 -g 1.2 -b 24 happytimes.wxyz happytimes.quad.wav*

Result: Input B format soundfile "happytimes.wxyz" is decoded into a quad output soundfile named "happytimes.quad.wav", which will have a 24 bit word size. All output samples are multiplied by 1.2, so the output amplitude will be 20 % higher than with default decoding.

For more usage details consult the *man* page for *decb2* or *decb4*.

Alternatively, Richard Furse's **ambidec** utility can be used to decode a 44.1k, 48k or 96 k B format or a second order 9-channel *.fmh* encoded soundfile into standard WAVE format output soundfiles for any of about 10 speaker configurations, including stereo, quad and 8 channel. However, *ambidec* has a significant limitation: the input soundfiles MUST have 16 bit resolution, and the decoded output soundfile also will have a 16 bit word size, making this utility unusable for high resolution audio. The syntax for this utility is:

ambidec [flag options and arguments] inputambisonicfile [outputWAVEfile]

If no output WAVE format soundfile is specified, *ambidec* will supply one, using the base name (minus the *.wxyz* or *.fmh* filename extension) of the input soundfile and appending a *.wav* extension.

There is no *man* page for *ambidec*, but typing the command name with no arguments will produce a usage summary that includes flag options. By default, *ambidec* will produce a stereo decoding. To decode an ambisonic file for a

horizontal quad rig, rather than for stereo, use the *-r square* option:

ambidec -r square inputfile.wxyz outputfile.wav

Also by default, *ambidec* and will normalize the output soundfile so that its peak amplitude is maxamp. This normalization increases the length of time required to perform the conversion, since the program must scan the ambisonic file for its peak amplitude. To bypass this normalization, include a *-g* flag and a gain (amplitude multiplier value):

ambidec -g 1 inputfile (no adjustment made to amplitude)
ambidec -g 2.3 inputfile (output peak amplitude will be 2.3 times the input peak amplitude)

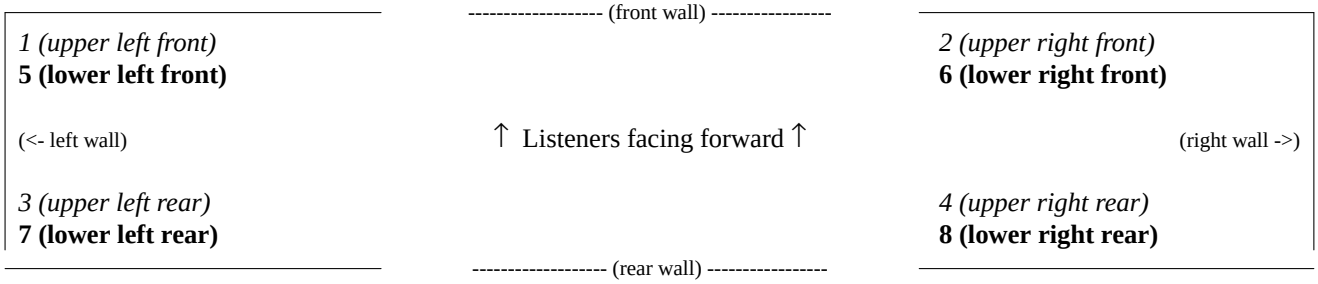
7.7. Cubic 8 channel decoding (decb8) and playback (8play)

Hardware and software installed in the Linux sound room in 2006 now make it possible to decode B format soundfiles for eight audio channels, and then to play these 8 channel soundfiles over a secondary audio system that leads to the eight Mackie *824* loudspeakers in room 52. Decoding a B format soundfile for eight channels is accomplished by using the ECMC utility **decb8**. In simple usage, without flag options, *decb8* works almost exactly like *decb2* and *decb4*, but instead creates an 8 channel WAVE output soundfile. Most often, the output soundfile is designed for playback over a cubic 8 speaker rig, with an upper quad speaker array (four speakers in a square or rectangle above ear level) and a lower quad array (four speakers in a square or rectangle below ear level). The eight Mackie speaker array in room 52 is arranged in this format.

Example: *decb8 mix1.wxyz mix1.8ch.wav*
Result: Input B format soundfile *mix1.wxyz* is decoded into 8 channel output soundfile *mix1.8ch.wav*.

The eight audio channels usually are designed for routing to the following speaker locations:

Standard 8 channel cubic loudspeaker rig (American numbering of channels)



Speakers in *italics* are placed above ear level (upper quad speaker array)
Speakers in **boldface** are placed below ear level (lower quad speaker array)

Note: The figure above is not "drawn" to scale. In most rooms the left and right walls are longer than the front and rear walls.
Note also that in European channel numbering the numbering of channels 3 and 4, and of channels 7 and 8, are often reversed, so that audio channel 3 = upper right rear, channel 4 = upper left rear, 7 = upper right rear, 8 = upper left rear

Unlike *decb2* and *decb4*, however, *decb8* includes options for additional ambisonic processing and modification of the input B format soundfile, and also an option for an alternative speaker setup. The full syntax for *decb8*, including flag options, is:

decb8 [-l <level>] [-o <speaker_orientation>] [-r <rotation>] [-t <tilt>] [-p <pitch>] inputfile [outfile]

where:

- *inputfile* is the name of the B format input soundfile (any sampling rate and bit depth)
- *outfile* is the name of the decoded 8 channel output soundfile

  If no output soundfile name is provided, the output soundfile will have the same name as the input soundfile, but with *oct.* prepended and with the *.wxyz* extension replaced by *.wav*

  Example: *decb8 finalmix.wxzy*

  Result: Input B format soundfile *finalmix.wxzy* is decoded into an 8-channel output soundfile named *oct.finalmix.wav*

- the optional *-l <level>* flag, followed by an argument such as 1.2 or .75, specifies an amplitude mul-

tiplier for the output samples, much like the *-g* flag and argument available in *decb2* and *decb4*.[7]

• the optional *-o* flag and its "speaker_orientation" argument, which currently must be either an *a* or a *b*, specifies an intended 8 channel speaker rig configuration

• the optional *-r*, *-t* and *-p* flags, and their integer arguments, in degrees, specify a "turning" or "rotation" of the entire soundfield along either the *x*, *y* or *z* axis, so that all or most sound sources will be shifted to a new locational position.

Typing: *decb8* with no arguments will display a usage summary with examples.

The *-o speaker orientation* flag:

The *-o* flag, followed by a space and a lower case *a*, specifies that standard ambisonic cubic speaker assignments (shown in the figure above) be used. This is the default speaker-feed decoding, and thus generally is not specified. However, if instead the *-o* flag is given an argument of *b*, an alternative output channel decoding matrix is employed in which channels 1 through 4 are designed to feed a front 4 speaker array of two raised and two lowered speakers (like the front four Mackie *824* speakers in the studio), while channels 5 through 8 are inteded to feed a rear 4 speaker array (such as the rear 4 Mackie speakers).

Normal ambisonic cubic rig *8play* orientation **a**: (corresponds to the preceding figure)			An alternative ambisonic cubic rig *8play* orientation **b**:		
Soundfile channel		Output speaker	Soundfile channel		Output speaker
Upper quad array:			*Front quad array:*		
1	-->	upper left front	1	-->	upper left front
2	-->	upper right front	2	-->	upper right front
3	-->	upper left rear	3	-->	lower left front
4	-->	upper right rear	4	-->	lower right front
Lower quad array:			*Rear quad array:*		
5	-->	lower left front	5	-->	upper left rear
6	-->	lower right front	6	-->	upper right rear
7	-->	lower left rear	7	-->	lower left rear
8	-->	lower right rear	8	-->	lower right rear

Note: If you decode a B format soundfile with the *-o b* option, and then play the decoded 8 channel soundfile in room 52, the *x*, *y* and *z* coordinates will not be correct. Rather, this option could be useful in case you ever need to decode a B format soundfile for an 8 channel playback system that is configured with hardware channel routings like those in the right hand columns in the chart above, or if you simply want to "fool around" with the decoding matrix and hear what results you get, even though these results are not "correct." In the future, we may implement additional *-o* speaker assignment options.

Rotate, tilt and tumble (pitch) soundfield manipulations, and some fun with a basketball

[As a visual aid to the following discussion, you may find it helpful to look at the large *Figure 1* diagram about 2/3 of the way down in the online article *Spatial Hearing Mechanisms and Sounds Reproduction* by D.G. Malham at:

http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm

This figure also is reproduced near the top of this web page: *http://ccrma.stanford.edu/~isak/220c/math.html*]

The *-r*, *-t* and *-p* options to *decb8*, which can be employed individually or in combination, enable us to turn, or rotate, the entire soundfield along some axis or combination of axes. This requires some explanation.

Imagine that you are inside of a very large basketball. Imagine, too, that all of the sound sources are located at various points around you inside the basketball, and that they are attached by invisable, stiff rods to the surface of the basketball, so that turning the basketball will result in a corresponding change in the position of each sound source. Bear with me here. Imagine further that a very large pair of invisible hands appear and twist, or turn, the basketball to a new position, rotating it to the right or left, or forward or backwards, while you remain immobile. Everything has moved except you, and all of the sounds now seem

[7] Unlike *decb2* and *decb4*, *decb8* currently has no *-op* flag option for specifying a desired peak amplitude output, and no *-b* option for changing the word size.

positioned in a new location, although the distance and angle between any sound source and any other sound source has not changed. The orientation of the entire soundfield — the left-right, front-rear and up-down positioning of each sound — has been shifted. This is what happens when we perform ambisonic soundfield manipulations.

If this fantasizing of invisible rods and hands is beginning to irritate you, think of *soundfield manipulation* in a simpler way. While listening, you turn your head to the right or to the left, or tilt your head up or down. In this case, although the soundfield has remained constant, your *orientation* to this soundfield (your sense of "up-down," "left-right" and "front-rear") has changed, producing a similar perceptual result.

A *soundfield manipulation* is defined by three angles, measured from 0 to 360 degrees, that describe (in the three *x, y* and *z* dimensions) the direction in which we have "turned the basketball" or "turned our head." To describe any possible movement of the ball or of our head, we need to specify an angle of displacement around the front-rear *x* axis, a displacement around the *y* axis, and a displacement around the *z* axis.

Imagine that our basketball ("Oh no! Not again!") has been painted with a white dot that currently is positioned directly to our right (at 3 o'clock), equadistant from the top and bottom of the ball. If the impalpable hands rotate the basketball so that the white dot is now directly in front of us (at 12 o'clock), but at the same height (still equadistant from the top and bottom of the ball), the soundfield has been rotated 90 degrees counter-clockwise around the *z* axis. This is called ambisonic *rotation*. The *x* (front-rear) and *y* (left-right) coordinates of each sound source have changed, but the *z* coordinate, or axis of rotation, has not changed. To accomplish this 90 degree rotation with *decb8*, we would use the command line:

decb8 -r 90 infile.wxyz

If the incorporeal hands now turn the basketball again so that the white dot is re-positioned directly above us, the soundfield has been shifted 90 degrees counter-clockwise (to the "left") around the *y* axis. This type of soundfiled manipulation is called *tumble* or, in the case of *decb8, pitch*. The *x* and *z* coordinates of each sound source have shifted, but the *y* coordinate is unchanged. (The white dot is still neither to our left nor to our right, but rather remains centered on the *y* axis. To perform this manipulation with *decb8*, we would use the command line:

decb8 -p 90 infile.wxyz

As if by magic, the intangible hands reappear yet again and twist the basketball so that the white dot is restored to its original position (3:00, directly to our right, equadistant between the top and bottom of the ball). In this case the soundfield has been turned -90 degrees (or 270 degrees) around the *x* axis, a manipulation called *tilt*. The *y* and *z* coordinates of each sound source have changed, but their front-rear *x* co-ordinate remains unchanged.

Thus far we have "turned" the soundfield ("basketball") in only one Cartesian direction at a time. To describe any possible three-dimensional (*x, y, z*) respositioning of the white dot (and of the soundfield), we need to specify a rotation angle, a tilt angle and a pitch (tumble) angle between the original and the new positions of the white dot. If one or two of these angles is zero, it can be omitted. In ambisonic processing, these angles by convention are measured counter-clockwise (toward the left). Negative angles create clockwise movement (toward the right). Note that when more than one soundfile manipulation is performed (e.g. rotation and then tilt), the order in which these are specified on the command line (and thus in which order they are performed) may be important.

Example: *decb8 -t 120 -p 60 -r -45 polka.wxyz*
Result: B format soundfile is decoded for 8 channel output soundfile *oct.polka.wav*. The soundfield is tilted 120 degrees counter-clockwise, then a 60 degree *pitch* (tumble) is performed, and then the soundfield is rotated 45 degrees clockwise.

For more information and examples, type: *decb8* with no arguments.

7.7.1. Eight channel playback: play -8

In the Linux sound room 8 channel decoded ambisonic soundfiles are played over the eight Mackie 824 speakers with the command **play -8**. The alias abbreviations *8play*, *play8*, *8p* and *p8* also can be used. The large rotary black knob on the Blue Sky BMC controller is used to control the listening level volume, as with all *play* commands.

Example command lines:

(1) *play -8 schnittke.oct.wav*

Result: 8 channel soundfile *schnittke.oct.wav* is played over the 8 Mackie speakers.

(2) *p8 mix1.oct.wav mix2.oct.wav testmix.quad.wav*

Result: The three specified soundfiles are played. (The third soundfile, *testmix.quad.wav*, is presumably a quad soundfile *play -8* is capable of playing stereo and quad soundfiles through the Mackie speakers).

play -8 also include a flag option to produce playback with speaker orientation *b* (see section 7.7), althugh this option is rarely used:

play -8 -8o b pizzline2-1.wxyz

7.8. Mixing B format soundfiles: mixb and mixbsc

Soundfiles encoded in B format cannot be mixed together properly with standard soundfile mixing programs such as *qrt*, *audacity*, *ardour* or *Cubase*. However, you can use the ECMC scripts *mixb* and *mixbsc* for this purpose. This section provides quickstart instructions for using these two scripts to mix two or more B format 4 channel soundfiles. These instructions do not cover all aspects of using these two scripts, but can get you started with the most common and simplest tasks for mixing B format soundfiles.

mixb runs Csound in the background to mix two or more B format soundfiles together. All of the input soundfiles MUST

(1) be in B format, with 4 encoded channels

(2) have the same sampling rate (96k, 44.1k or 48 k)

(3) have the same word size (24 bit ints, 16 bit short ints or 32 bit floats)

Csound requires an orchestra file and a score file. Three steps are required to mix B format soundfiles with *mixb* and *mixbsc* :

(1) First run *mixbsc*. This will create a Csound score file template (similar to the *sout* files produced by *score11*) for the mix.

(2) Edit this score file template, filling in a start time and an amplitude multiplier for each soundfile, and also, if necessary, a global amplitude multiplier that affects all input soundfiles. When the score file is ready,

(3) Run *mixb* with appropriate flags for the type of output you want. This will

- ☞ create a temporary, hidden Csound orchestra file, and
- ☞ run Csound to mix the soundfiles in your score into an output soundfile, or else to mix the input soundfiles in real time and play the result

Typing *mixbsc* or *mixb* with no arguments will produce a usage summary. However, both of these scripts have many options that I occasionally use but you probably will never need, and so the usage summary may seem overly complicated. In addition to adjusting the start times and amplitudes of each input soundfile -- changes that are almost always necessary -- *mixbsc* and *mixb* also enable you to alter the amplitude envelopes of the input soundfiles (e.g. with fade-ins or fade-outs), to transpose them in pitch, and to introduce time varying transpositions (glissandi). These more advanced features are not covered in the quickstart instructions presented here, because you probably will not need them, and because this section of the *Users' Guide* already is too long. However, these features are discussed in the *man* page for *mixbsc*. In the discussion that follows, it is assumed that you simply want to mix two or more B format soundfiles, adjust their start times and scale their amplitudes, without introducing new amplitude envelopes, skipping into these soundfiles, or transposing them.

7.8.1. Using mixbsc

To simply mix 2 or more B format input soundfiles, a note/event in the score file with 5 parameters is required for each input soundfile. (To alter the amplitude envelope, skip a portion of a soundfile, or transpose a soundfile, 11 additional p-fields are required. These 11 additional p-fields are not covered here.) *mixbsc* has 2 flag options:

- *-s* produces a "short" or "succinct" score file, without comments and with only 5 p-fields. I recommend that you use this flag to create "simple" score files.
- *-v* produces a "verbose" score file with all 16 p-fields are lots of comments.

If neither a *-s* nor a *-v* flag is included, a score file is created with all 16 p-fields but only a few comments.

If I want to mix 3 soundfiles named *a1mix.wxyz*, *a2mix.wxyz* and *a3mix.wxyz* I would type:

mixbsc -s a1mix.wxyz a2mix.wxyz a3mix.wxyz

This will display the following score file template:

```
; SamplingRate 96000  WordSize 32 bits ; DO NOT ALTER OR DELETE THIS LINE
;f60 0 1025 5 .005 1024 1.; exponential function for glissandi
 ; a 0 0 0 ; to skip into the output uncomment & change the last 0 to a skip time

i1  0  1.  1.   ; p4 = GLOBAL AMP. MULTIPLER {0=1.}
i2  0  7.29  "/snd/allan/SUBMIXES/a3mix1.wxyz"   1
i2  0  6.11  "/snd/allan/SUBMIXES/a3mix2.wxyz"   1
i2  0  21.08  "/snd/allan/SUBMIXES/a3mix3.wxyz"   1

e ;  end of score; this must be the last line
```

If this template looks okay, and no error messages appear, repeat the command and capture the output into a score file:

!! > Amix.bm or

mixbsc -s a1mix.wxyz a2mix.wxyz a3mix.wxyz > Amix.bm

This will create a score file called *Amix.bm*. You can call the outputfile anything you like. I customarily append the extension *.bm* (for "b format mix") to the names of my B mix scorefiles so that I know that this file is a score file for *mixb*, but this is certainly not required.

Do not change the top line the top line in the score or add any lines before it:

; SamplingRate 96000 WordSize 32 bits ; DO NOT ALTER OR DELETE THIS LINE

This line is used by *mixb* to set the default output sampling rate and bit depth for the mix.

Semi-colons (;) are comment symbols, and when Csound parses this input score and encounters a semi-colon it will ignore the rest of the line. Note that "notes" for 2 instruments are created in this score. Instrument 1 (i1) provides a global amplitude multiplier and has 4 p-fields. Instrument 2 (i2) reads in the input soundfiles:

(p1) (instrument number)	(p2) (start time)	(p3) (duration)	(p4) (input soundfile)	(p5) (amplitude multiplier)
i1	0	1.	1.	; p4 = GLOBAL AMP. MULTIPLER
i2	0	7.29	"/snd/allan/SUBMIXES/a3mix1.wxyz"	1

Now you must edit this scorefile with a text editor such as vi or nedit. Here is an example of an edited version of this score, in which I have changed the global amplitude multiplier and the start times and amplitude multipliers for each of the three input soundfiles. These changes are shown here in bold font:

Example of edited score file:

```
; SamplingRate 96000  WordSize 32 bits ; DO NOT ALTER OR DELETE THIS LINE

;f60 0 1025 5 .005 1024 1.; exponential function for glissandi
 ; a 0 0 0 ; to skip into the output uncomment & change the last 0 to a skip time

i1  0  1.  .85   ; p4 = GLOBAL AMP. MULTIPLER {0=1.}
i2  0  7.29  "/snd/allan/SUBMIXES/a3mix1.wxyz"   .35
i2  4.75  6.11  "/snd/allan/SUBMIXES/a3mix2.wxyz"   .72
i2  9.33  21.08  "/snd/allan/SUBMIXES/a3mix3.wxyz"   .65
e ;  end of score; this must be the last line
```

In this mix:

- Soundfile "a3mix1" will begin at time 0 (p2) , and has an amplitude multiplier of .35 (p5).
- Soundfile "a3mix2" will begin at time 4.75 (p2) , and has an amplitude multiplier of .72 (p5).
- Soundfile "a3mix1" will begin at time 9.33 (p2) , and has an amplitude multiplier of .65 (p5).

Note that p5 acts as an amplitude "fader" or multiplier for each input soundfile. There is also a "global" fader, or amplitude multiplier, that affects ALL soundfiles in the mix. This global fader is found in p4 of instrument 1:

i1 0 1. **.85** *; p4 = GLOBAL AMP. MULTIPLER {0=1.}*

In this case, I have set the global amplitude multiplier to .85 Thus, the samples in input soundfile a3mix1.wxyz will be multiplied by .35 and then by .85, for a total gain of .2975. If the soundfile is at max-amp (32767), its output within the mix will have a peak amplitude of 9478 (.2975*32767).

If at some time in the future I decide that I want to add more input soundfiles to this mix — say, *bigmix.wxyz* and *littlemix.wxyz* — I could run this command in a shell window:

mixbsc -s bigmix.wxyz littlemix.wxyz

and then, from the resulting display, copy and paste the two lines

i2 0 29.33 "/snd/allan/SUBMIXES/bigmix.wxyz" 1

i2 0 5.84 "/snd/allan/SUBMIXES/smallmix.wxyz" 1

into my score file, probably near the end but above the line

e ; end of score; this must be the last line

which MUST be the last line of text in the file. I then can edit the file again and adjust the start times and amplitude multipliers for the two new input soundfiles.

7.8.2. Using mixb

When the score file is ready it is time to create the mix by running *mixb*. The syntax for running *mixb* is:

mixb [flags] scorefile [OUTFILE]

where *scorefile* is the name of the score file we just created and *OUTFILE* is the name of the output soundfile. One or more *flag* options determine the format of the output. In the case of our new score file, we first want to test it by playing the mix, sending the output samples directly to the soundcard DACs rather than writing these samples into an output soundfile. To play a mix with quad decoding, use the *p4* ("**p**lay in 4 channels") flag:

mixb p4 Amix.bm

To play the mix with UHJ stereo decoding instead, include either *p2* flag:

mixb p2 Amix.bm

If we are not completely satisfied with the result we can continue editing score file *Amix.bm*, playing it again, editing again, possibly adding more input soundfiles, listening to how the mix sounds in both quad and stereo, and so on until we are satisfied with the output and ready to write it to a soundfile.

The default output soundfile format is B format. Flag options that cause *mixb* to write to other formats include:

- **s** : standard ambisonic stereo decoding into a two channel WAVE file
- **2** : UHJ decoding, an alternative stereo decoding method that I believe creates SOME sense of front-rear placement; I recommend using the *U* flag (UHJ stereo decoding) rather than the *2* (standard stereo) flag whenever writing a stereo output file. However, you can try both flags and determine which decoding you prefer.
- **4** : decodes the output for quad and writes the samples to a four channel WAVE file
- [• **8** : decodes for 8 channels and writes the output to an 8 channel WAVE file (not yet implemented as of this writing)]

The sampling rate and bit depth of the output soundfile will match that of the input soundfiles. However, the *word size* can be changed by including a *WS* or *WwsfR flag followewd by 16, 24 or 32.*

Examples:

mixb 2 Amix.bm Amix.wav

Result: The mix samples are written are written to a stereo (UHJ decoded) output soundfile named *Amix.wav*.

mixb 4 Amix.bm Amix.quad.wav

Result: The mix samples are decoded for four speakers and are written are written to a quad soundfile named *Amix.quad.wav*.

mixb Amix.bm

Result: The output samples will be in B format. Since we have not provided a name for the output soundfile, *mixb* will create one for us: *Amix.bm.wxyz* (the name of the input score file plus an extension that indicates the output format).

mixb 4 ws 24 Amix.bm Amix.quad24.wav

Result: Quad decoding with 24 bit word size written into the output soundfile *Amix.quad24.wav*.