Characterization of the Auditory Thalamic Nucleus of the Barn Owl

Thesis by
Larry Proctor

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1995
(Submitted June 7, 1994)

ACKNOWLEDGEMENTS

A number of people made valuable contributions to the specific work contained herein as well as the general research atmosphere which goes beyond the particular results of this thesis. Foremost, I would like to thank Dr. Mark Konishi for his guidance, support and patience. Mark allowed me to explore several facets of neurophysiology in his laboratory while deftly managing to keep me on track. His discussions have always helped me to see the big picture despite the fact that I was looking at it in 30 μm sections. Gene Akutagawa provided valuable expertise on histochemistry as well as getting me to set high standards for the quality of my histological material. Early in my graduate career, I benefited greatly from working with Drs. Catherine Carr, Hermann Wagner and Terry Takahashi, then post-doctoral fellows in the Konishi lab. Much of my understanding of auditory physiology resulted from listening to their boisterous arguments. Dr. Georg Striedter provided valuable advice on matters of anatomy, microscopy and displaying histological data. Georg also introduced me to the method of using biotinylated dextran-amine as a tracer, thereby saving me from the capriciousness of free HRP. Dr. Allison Doupe was instrumental in maintaining my sanity over the last four years, generously taking time to listen to both triumphs and tribulations and making specific suggestions which improved the quality of my research. Roian Egnor allowed me to examine the pattern of nucleus ovoidalis cell bodies which were retrogradely labelled from a BDA injection which she placed in Field L2a. Roian also provided a critical reading of portions of a draft of this thesis. I would like to thank all members of the Konishi lab (even the birdsong people) for providing a lively and interesting work environment. Finally, I would like to pay my special thanks to my wife, Patti, for seeing to it that the comparatively simple task of completing my doctorate was truly a challenge.

ABSTRACT

The barn owl has the remarkable ability to accurately localize a target on the basis of auditory cues alone. The investigation of the central nervous system mechanisms underlying the sensory aspect of this behavior led to the discovery of neurons which have auditory receptive fields restricted to small regions of the acoustic environment. Neurons with these characteristics were found both in the forebrain and in the inferior colliculus. In the inferior colliculus, neurons which were near one another had receptive fields which were near one another in the sound field; there was a physiological map of auditory space. No such organization was observed in the forebrain. The auditory map of space in the inferior colliculus projects to the optic tectum where it is preserved and placed in register with a visual map of space. The auditory space map in the inferior colliculus and optic tectum have been functionally related to sound localization in a behavioral assay. Recent experiments have demonstrated that the thalamo-telencephalic auditory pathway is sufficient for localization of sound sources, despite the fact that a topographic map of auditory space has not been demonstrated in this neural pathway. The purpose of the work reported upon here was to examine the manner in which the auditory thalamus, nucleus ovoidalis (N.Ov), transforms the neural code which represents the auditory environment. Neurons in nucleus ovoidalis were characterized with respect to their responses to stimulation with sounds presented through earphones. This dichotic stimulation allowed for the independent control of the amplitude, frequency and temporal structure of the sounds delivered to the ear. The anatomy of the afferent and efferent pathways to the auditory thalamus were also investigated.

The activity of extracellularly isolated single-neurons was recorded in the nucleus ovoidalis of the anesthetized barn owl. This nucleus contains two subdivisions based on tonotopic organization. The central and medial portion of the nucleus is organized such that neurons responding well to high frequencies tend to be located dorsally while neurons which respond well to lower frequencies are located progressively more ventral. In the lateral portion of the nucleus, neurons in a dorsoventral electrode penetration have the same best frequency, between 4.5 and 5.0 kHz. Of 114 neurons whose best frequency tuning curves were characterized completely, 14 had two or three clearly distinguishable peaks. The response to sound localization cues at one of the frequency peaks was different from those at the other frequency peak(s). Of 207 neurons which were recorded within N.Ov, all responded to auditory stimulation with broad band noise and all were sensitive to at least one sound localization parameter. In contrast to previously studied auditory nuclei in the medulla and mesencephalon of the barn owl, there was no apparent systematic mapping of either sound localization cue in the dorsoventral, mediolateral or rostrocaudal axes.

Neurons within N.Ov had response tuning curves to interaural time difference (ITD), interaural intensity difference (IID) and frequency which were most similar to those found in the lateral shell and core subdivisions of the central nucleus of the inferior colliculus (ICc). However, neurons were found in ovoidalis which had combinations of response tuning curve types not previously observed in the auditory system of the barn owl. Thirteen neurons were classified as space-specific, although their response properties were not always similar to neurons in the space maps of ICx or optic tectum. Six neurons were found which

had broad frequency tuning curves but which also had ambiguous ITD or IID tuning curves in response to white noise.

Injections of retrograde tracers into N.Ov resulted in stained cell bodies in ICc that were sparsely distributed across the three subdivisions of the nucleus. Labeled neurons were located in various positions along the dorsoventral axis of ICc, along which frequency is mapped. Retrogradely labeled somata were found bilaterally in ICc, though the number of labeled cells was higher in inferior colliculus ipsilateral to the injection site. The widely distributed pattern of retrograde staining in ICc was obtained irrespective of the location of the tracer injection within N.Ov. Anterograde tracers injected into the core/medial shell region of ICc resulted in a pattern of stained axonal terminals in the centromedial N.Ov that was relatively focal and which corresponded well with the tonotopic organization of this region of the nucleus. Injection of anterograde tracers into the lateral shell subdivision of ICc yielded labeled axon terminals in the lateral portion of N.Ov in caudal sections, while the heaviest staining was located along the dorsal aspect in the most rostral sections. Anterograde tracer studies revealed that N.Ov efferents originating from the medial portion of the nucleus have a restricted terminal field in the most medial region of the forebrain area Field L2a. The ventrolateral region of N.Ov, however, sends a wide projection pattern of efferent terminations across the mediolateral extent of caudal Field L2. Labeled axon terminals are quite dense in the lateral portion of Field L2 and rather diffuse in the medial aspect.

The unique combinations of physiological responses found in N.Ov as well as its patterns of afferent and efferent connectivity suggest that ovoidalis is reorganizing the neural information concerning acoustic stimuli. While tuning to sound localization cues is maintained, it is possible that such coding may be of

secondary consideration in the thalamo-telencephalic pathway. The foundation for the neural representation of auditory recognition may well begin at the level of the auditory thalamus.

TABLE OF CONTENTS:

Chapter 1: General Introduction

The ability to localize sound sources is a critical component of many adaptive behaviors across species. Sound localization can be used to find prey, to avoid predators or to locate conspecific individuals. Auditory cues provide information beyond the often restricted field of view provided by the visual system, and are particularly useful under environmental conditions that hinder vision. The stimulus cues that indicate the position of a sound source are not directly mapped onto the sensory epithelium of the ear. Consequently, the vertebrate nervous system has evolved mechanisms by which to transform sound localization cues into an orderly representation of space. Some species have become specialists at using sound localization to gain a selective advantage over other species over the course of evolution. The barn owl, *Tyto alba* (figure 1.1), is able to locate targets accurately using passive auditory cues exclusively (Payne, 1971). The investigation of the central nervous system mechanisms underlying the sensory aspect of this behavior led to the discovery of a map of auditory space in the inferior colliculus of the owl (Knudsen and Konishi, 1978). Following a "top-down" experimental approach, Konishi and colleagues have been working to describe the neural transformations that ultimately lead to the formation of the space map in the auditory system. The owl uses interaural time difference (ITD) to determine the horizontal coordinate of a sound source, and the interaural intensity difference (IID) to ascertain the vertical coordinate (Moiseff and Konishi, 1981a). The time and intensity cues contained in the acoustic signal are processed separately in parallel neural pathways in the brainstem (Takahashi et al., 1984).

Figure 1.1 Portrait of a barn owl *(Tyto alba)*.

Early Processing

The initial stages of processing in the auditory system are very similar across all avian species (Sachs and Sinnott, 1978; Warchol and Dallos, 1990; Sullivan and Konishi, 1984). The cochlea of birds (which is straight, rather than coiled as in mammals) has sensory receptors called hair cells which respond to narrow frequency bands of complex sound stimuli. The cochlea decomposes the acoustic signal into its component frequencies. In the cochlea, hair cells with similar frequency response characteristics are situated near one another. This tonotopic organization is maintained in the auditory nerve and the central auditory system of the brainstem. Each auditory nerve fiber codes the amplitude and timing information contained in its frequency component of the acoustic stimulus with its spike rate and pattern. Stimulus **intensity** is encoded via a sigmoid shaped amplitude vs. spike rate curve. **Time** information is coded with a phase-locked pattern of action potentials. Phase-locking refers to the high probability of a neural spike to occur near a particular phase angle of a tonal signal (Rose et al., 1967). Period histograms are used to determine the phase angle at which a neuron is most responsive. The degree of tuning of the neuron to that phase angle is called the vector strength (Goldberg and Brown, 1969).

The auditory nerve bifurcates and innervates two distinct cochlear nuclei in the brainstem of birds; nucleus angularis (NA) and nucleus magnocellularis (NM) (Boord, 1969; Carr and Boudreau, 1991). These cochlear nuclei receive auditory nerve input from the ipsilateral ear only. The neurons of nucleus angularis filter the timing information from their auditory nerve input while preserving the amplitude information. Conversely, neurons within nucleus magnocellularis preserve and enhance the phase-locked response of their afferents while discarding intensity information (Sachs and Sinnott, 1978; Sullivan and Konishi,

1984; Warchol and Dallos, 1990). This dichotomy in information processing is subserved by differences in synaptic and cellular morphology between the two nuclei. Neurons of nucleus angularis have extensive dendritic arborizations which receive "bouton-type" synaptic endings from their branch of the auditory nerve. Neurons in nucleus magnocellularis have few or no dendrites, and their cell bodies are covered with fine membrane protrusions similar to dendritic spines (Jhaveri and Morest, 1982; Carr and Boudreau, 1993). These neurons receive a large "calyx" synapse from auditory nerve afferents, called the endbulb of Held (Carr and Boudreau, 1991; Boord, 1969). Specializations in membrane physiology and cellular chemistry probably also play a major role in determining the response differences between these two nuclei which receive their input from a common source (Raman and Trussell, 1992; Takahashi et al., 1987).

The Time Pathway

To obtain the interaural time difference which is used to determine the azimuthal coordinate of a sound source, timing information from the two ears must be compared. Two timing cues are available from a particular sound source: the difference in the time of arrival of the first wave of sound at the two ears, and the ongoing difference in the phase relationship of the spectral components of sound in each ear for the duration of the acoustic stimulus. Barn owls do not use the transient onset time difference of a sound to determine its horizontal coordinate (Moiseff and Konishi, 1981a). Instead, their peripheral auditory nervous system encodes the phase of the frequency components of a sound, and the central auditory system extracts timing information by comparing the phase information from each ear. This comparison is performed in nucleus laminaris, which receives bilateral input from nucleus magnocellularis (Parks and Rubel, 1975; Carr and Boudreau, 1993). Phase-locked action potentials from

nucleus magnocellularis travel along axons which act as physical delay lines (Carr and Konishi, 1990), much like those hypothesized to be involved in the processing of interaural time delays (Jeffress, 1948). When the time delay between phase-locked neural spikes arriving from the left and right inputs is offset by an equivalent amount by the delay line on the earlier input, action potentials from left and right NM inputs will arrive at a particular laminaris neuron simultaneously. When activity from the afferents coincide in this manner, the response of an NL neuron is maximal. When the inputs from both sides are 180° out of phase, activity levels in laminaris neurons are minimal. Intermediate levels of activity are obtained when the inputs have intermediate phase relationships, or when phase-locked activity is received from one side only. Like their afferents, nucleus laminaris cells respond to auditory stimulation in a phase-locked manner. When stimulated monaurally, laminaris neurons phase-lock at the same mean phase as the efferents from the active nucleus magnocellularis. The arrival time of phase-locked spikes in many laminaris neurons differs between the ipsi- and contralateral inputs. When this difference is offset by an appropriate interaural time difference, the laminaris neurons respond maximally, phase-lock at a mean phase angle between those of the ipsilateral and contralateral inputs, and have a large vector strength. When this difference is offset by an unfavorable ITD, the laminaris neurons have a minimal response, a very low vector strength, and phase-lock (weakly) at the mean phase of both the ipsilateral and contralateral inputs (Carr and Konishi, 1990). This sensitivity to the relative timing of inputs from left and right sides is remarkable, given that, for a 2 kHz stimulus the difference in arrival time between in-phase and out-of-phase spikes is only 250 microseconds. While 2 kHz is the upper limit for phase-locking in the chick (Warchol and Dallos, 1990), the barn owl maintains phase-locking (and ITD tuning) up to 9 kHz (Sullivan, 1985). At this extreme, laminaris neurons are

discriminating time differences of less than 55 microseconds (5-7% of the duration of incoming action potentials)!

Nucleus laminaris apparently contains only one morphological cell type (Carr and Boudreau, 1993). Physiologically, these cells are distinguished by the frequency to which they respond and the interaural phase difference that they prefer. In the chick and pigeon, laminaris cell bodies are arranged in a monolayer running mediolaterally with bipolar dendrites extending dorsally and ventrally. These dendrites are of equal length. The axonal delay lines in the chick are believed to be formed by the distance ipsi- and contralateral afferents must travel along the mediolateral extent of the nucleus before making synaptic contact on the dendrites of their target cell (Parks and Rubel, 1975). The barn owl has evolved unique specializations in the anatomy of nucleus laminaris and the morphology of its constituent cells. Barn owl nucleus laminaris neurons are sparsely arrayed in a matrix-like configuration across the nucleus. These cell bodies lack dendrites, but are covered with spine-like protrusions. Afferents from nucleus magnocellularis form delay lines as the ipsilateral axons cross the nucleus from the dorsal to the ventral aspect while the contralateral axons project from the dorsal to the ventral border of the nucleus. The magnocellular afferents make synaptic contacts on laminaris cell bodies with no spatial segregation of ipsi- and contralateral inputs across the soma (Carr et al., 1989). Inputs from approximately 100 nucleus magnocellularis neurons converge upon each laminaris neuron cell body (Carr and Boudreau, 1993). Glutamic acid decarboxylase (GAD) positive synaptic terminals have been observed on laminaris cell bodies, but no GAD positive cell bodies themselves are present in the nucleus (Carr et al., 1989). The origin of these presumably inhibitory inputs to nucleus laminaris has not yet been determined. The possibility that glycinergic

inhibition is supplied to laminaris neurons has not yet been investigated. Fujita and Konishi (1991) demonstrated that GABA-mediated inhibition plays a role in interaural phase-difference processing at higher levels in the auditory system of the barn owl, and mention preliminary experiments which suggest that the same is true in nucleus laminaris. The difficulty of obtaining extracellular recordings from laminaris neurons *in vivo*, however, precluded these investigators from pursuing this topic.

Currently, the standard model for the formation of time difference tuning in nucleus laminaris asserts that the laminaris cell bodies act as **coincidence detectors** for the simultaneous arrival of action potentials from the axonal delay lines formed by magnocellular efferents (Jeffress, 1948). This model accounts for the fact that when the phase-locked inputs from the two magnocellular nuclei arrive at a particular laminaris neuron at the same time, then that neuron is driven maximally. However, the simple coincidence detection model is unable to account for the following aspects of interaural time difference tuning in nucleus laminaris:

- Input from one side only results in an intermediate level of activity in these neurons.

- Should the inputs from each magnocellularis arrive out-of-phase, then the activity of the laminaris neuron is less than it is when it is stimulated with input from one side alone. That is, the response of laminaris cells to binaural out-of-phase stimulation is suppressed relative to that to monaural stimulation (which, in turn, is less than the response to binaural in-phase stimulation).

There are currently two hypotheses to explain the response of nucleus laminaris neurons to out-of-phase binaural stimulation. The **inhibition hypothesis** asserts that inhibitory inputs suppress the activity of laminaris neurons when the phase relationship of the inputs from the two sides is not optimal. In order for the inhibitory inputs to suppress the activity specifically only at non-optimal interaural time differences in nucleus laminaris, the inhibitory neurons themselves would have to discriminate phase difference values as well as calibrate the timing of their action potentials to account for the distances their axons travel. An alternative hypothesis does not invoke a time-difference specific inhibition. The **resonance hypothesis** suggests that incoming synaptic depolarizations set off an inherent electrical membrane resonance similar to that demonstrated in saccular hair-cells (Hudspeth and Lewis, 1988). When inputs from both sides arrive simultaneously, the resonant potentials from the inputs would be in-phase and their addition would increase the total resonant amplitude in the cell, thereby increasing the probability of crossing threshold. Out-of-phase inputs would set up sinusoidal potentials that tend to cancel each other and therefore reduce the probability of crossing the spiking threshold. This scheme would require extremely short membrane time constants as well as unusually fast ion channel kinetics. Excitatory post-synaptic potentials would need to have quite short durations, perhaps being repolarized by active currents and thereby increasing the time window available for temporal summation. Raman and Trussell (1993) have demonstrated that cells in the auditory brainstem of the chick express glutamate receptors with unusually rapid kinetics of desensitization.

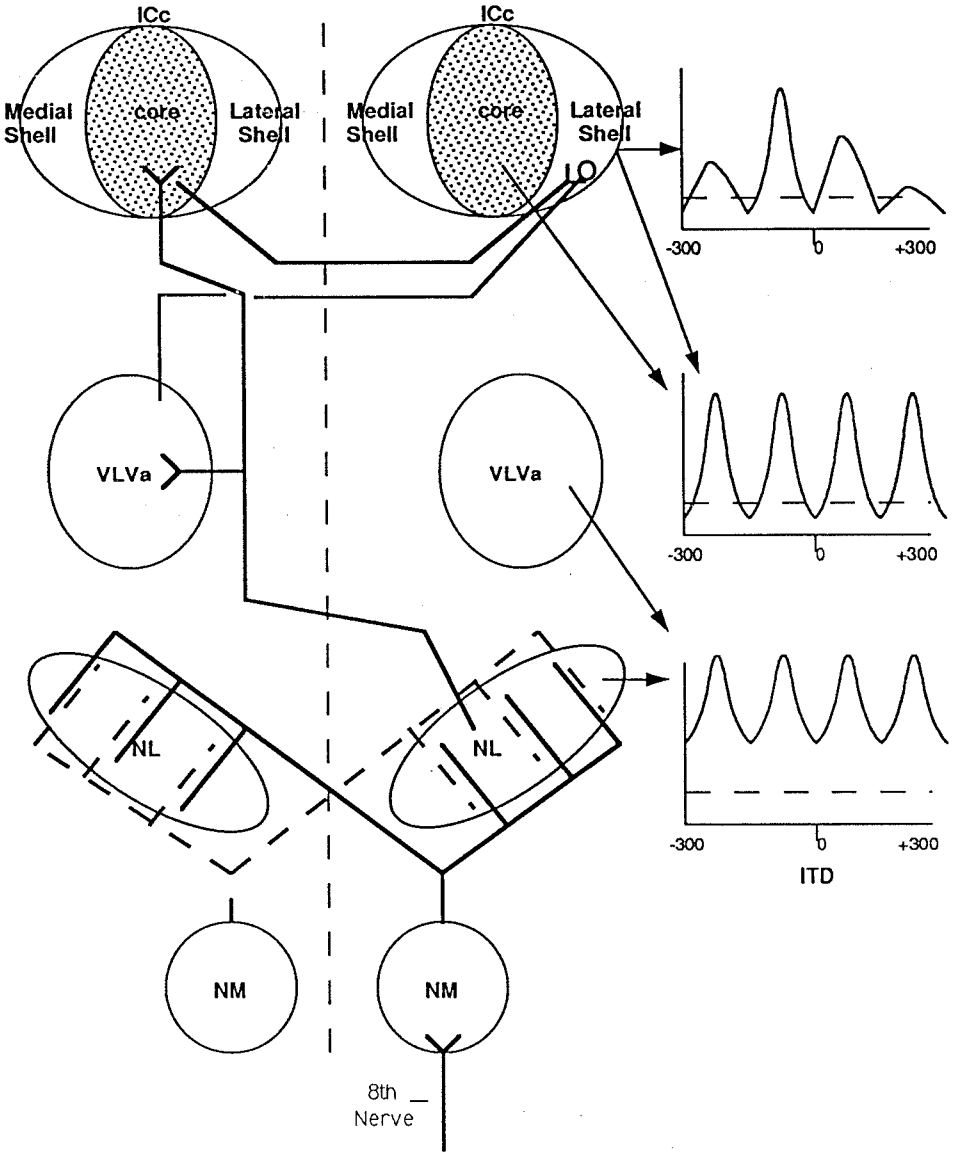
As previously stated, the cochlea decomposes a sound stimulus into its component frequencies. As a result, individual hair cells and their afferent

targets are effectively responding to sinusoidal stimuli. For complex or wideband acoustic input, the frequency of this sinusoid corresponds to the cell's best (or characteristic) frequency--the frequency to which the cell is most sensitive. For pure tones, the frequency of the sine wave corresponds to that of the stimulating tone, provided that it falls within the range of frequencies that the cell is capable of responding to. Neurons in the auditory nerve and nucleus magnocellularis tend to respond when the sine wave is at a particular phase. However, these cells have no way of conveying information about *which cycle* of the sine wave that phase is in. Consequently, in nucleus laminaris phase-locked spikes from one ear can coincide with phase-locked spikes generated by different cycles of the sinusoid in the other ear. The result is an ITD tuning curve which has multiple peaks. The distance between these peaks corresponds to the period of the laminaris neuron's characteristic frequency, or the period of the stimulating tone if a single frequency other than that of the neuron's characteristic frequency is used. An ITD tuning curve which contains multiple peaks is incapable of signaling the horizontal coordinate of a sound source unambiguously; at least one "phantom" target location will be perceived. ITD tuning curves of this type are called **phase ambiguous.** Although phase ambiguous, ITD is systematically mapped across the dorso-ventral dimension of nucleus laminaris (Sullivan and Konishi, 1986; Carr and Konishi, 1990).

Efferent fibers from nucleus laminaris decussate and synapse on two contralateral targets (figure 1.2): nucleus ventralis lemnisci lateralis, pars anterior (VLVa) and the core subdivision of the central nucleus of the inferior colliculus (ICc core) (Takahashi and Konishi, 1988a,b). NL also sends a projection to the medial portion of the ipsilateral nucleus of the superior olive (SO). These terminal fields of nucleus laminaris all display calcium binding protein-like

Figure 1.2 Schematic diagram of the time pathway in the barn owl and the ITD response tuning curves characteristic of the nuclei in the time pathway which have been investigated.

The Time Pathway



immunoreactivity (Takahashi et al., 1987). Neurons in VLVa and ICc core have ITD tuning curves which are phase ambiguous, though these cells do not display phase-locking. Cells in these nuclei have narrow frequency tuning curves and are tonotopically organized. The frequency tuning characteristics of their afferents correspond to the frequency responses of the target neurons.

Neurons in ICc core are arranged in arrays perpendicular to the isofrequency laminae of this region according to their response to ITD (Wagner et al., 1987). Only one ITD (the "array-specific ITD") is capable of activating all neurons in an array to the same relative level. That is, at the array-specific ITD, all neurons in a particular dorso-ventral region of ICc core have the same level of activity despite being tuned to different frequencies. Conversely, at all other ITDs, neurons in different frequency laminae will have disparate activity levels. This is similar to the observation that when a single, phase-ambiguous neuron is stimulated with tones of different frequencies, there will be one ITD at which its response is independent of frequency. This frequency independent ITD is called the **characteristic delay** of that neuron (Rose et al., 1966). An array-specific ITD in ICc core is equivalent to the characteristic delay of that array of neurons. Array-specific ITDs are mapped across the medio-lateral aspect of ICc core; at any particular ITD only one array is maximally activated, and different arrays are maximally excited at different ITDs. The neurons which are members of an array, though individually phase-ambiguous, are collectively capable of unambiguously representing a single ITD.

ICc core and VLVa project to the contralateral lateral shell subdivision of ICc (Takahashi et al., 1989; Mazer and Adolphs, 1991). In ICc lateral shell, ITD responses begin to be integrated across frequency channels (as well as with intensity information). ICc lateral shell sends its output to the ipsilateral external

nucleus of the inferior colliculus (ICx), which is the location where the map of auditory space is synthesized (Knudsen and Konishi, 1978). ICx cells do not respond to monaural stimuli. Their frequency tuning curves are generally broad and they are not tonotopically organized (Knudsen and Konishi, 1978c). Neurons in ICx do not change their selectivity for one sound localization cue with changes in the other cue--there is no trading between time and intensity (Moiseff and Konishi, 1981a; Takahashi et al., 1984). In ICx, the side peaks of phase-ambiguous ITD tuning curves are suppressed, yielding tuning curves which have a large primary peak and smaller or no secondary peaks. The sizes of the secondary peaks relative to the height of the primary peak vary; secondary peaks roughly half the height of the primary peak are the most usual, though secondary peaks may be non-existent or up to 90% of the height of the primary peak (Takahashi and Konishi, 1986). Side-peak suppression equivalent to that obtained with noise stimuli can be achieved in ICx by presenting sound (via earphones) consisting of the sum of the neuron's best frequency and one other frequency. The neuron's response to such 2-tone stimuli is neither the sum nor the average of the neuron's response to either tone alone, suggesting that some type of nonlinear processing is involved (Takahashi and Konishi, 1986). Fujita and Konishi (1991) demonstrated that GABAergic inhibition may be largely responsible for the nonlinearity inherent in side-peak suppression. These investigators applied a selective antagonist of GABA, bicuculline methiodide (BMI), iontophoretically in ICx and found that side-peak suppression in response to noise stimuli gave way to phase-ambiguous responses under the influence of the drug. Mazer and Adolphs (1991) found that applying BMI to VLVa resulted in a decreased response to ITD in ICx, while applying GABA in VLVa caused an increase in ITD responsiveness in ICx as well as the elimination of side-peak suppression in some cases. This suggests that VLVa provides the inhibition

necessary for the resolution of phase-ambiguity in ICx. The current hypothesis for the generation of unambiguous ITD tuning is that neurons from several frequency laminae which are members of an array with an array-specific ITD converge upon a cell. Inhibitory input reduces the activity levels caused by integrating ITD excitation across frequencies.

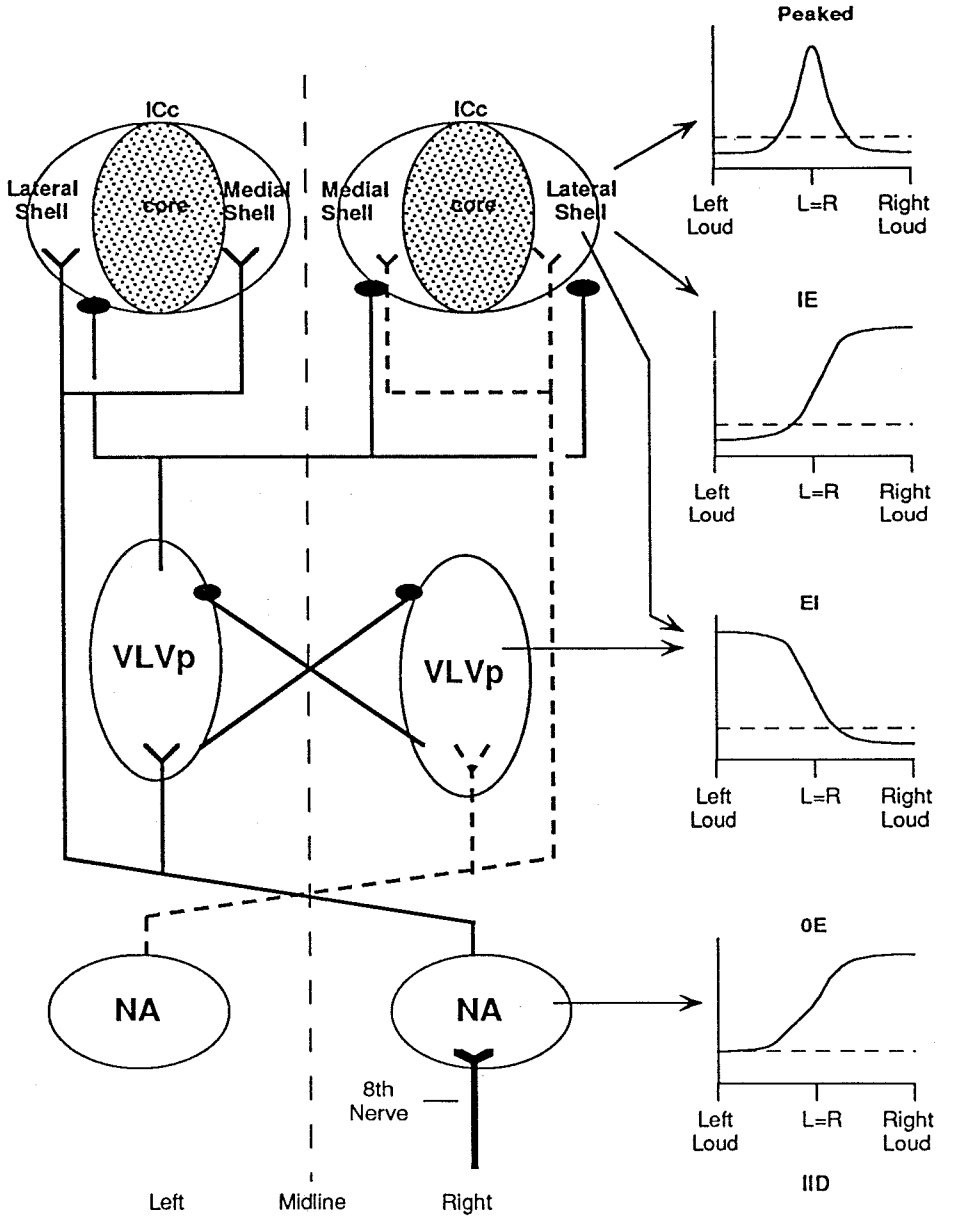
The Intensity Pathway

Having extracted intensity information from the auditory nerve, nucleus angularis relays these data to the inferior colliculus both directly and indirectly. Angularis projects bilaterally to the lateral aspect of the nucleus of the superior olive (SO), as well as to the nucleus lemnisci lateralis, pars ventralis (LLv) (Takahashi and Konishi, 1988b). NA sends efferents contralaterally to the nucleus ventralis lemnisci lateralis, pars posterior (VLVp) and both the medial and lateral shell subdivisions of the ICc (Takahashi and Konishi, 1988a; Adolphs, 1993b). The efferent projections of SO have yet to be identified. LLv projects to the ipsilateral lateral shell subdivision of ICc (Adolphs, 1993b) as well as the medial shell subdivision (unpublished personal observation). In pigeons, LLv also sends a direct projection to a subdivision of the auditory thalamus called nucleus semilunaris parovoidalis (SPO) (Wild, 1987).

Comparison of the sound levels in the two ears takes place in VLVp (figure 1.3). Nucleus angularis provides a direct, excitatory input to the contralateral VLVp (Takahashi and Konishi, 1988b; Takahashi and Keller, 1992), which also receives an inhibitory input from the opposite VLVp (Adolphs, 1993b; Manley et al., 1988; Takahashi and Keller, 1992; Mogdans and Knudsen, 1994). VLVp is the first binaural station in the intensity pathway; neurons in this nucleus are excited by contralaterally loud stimuli, inhibited by ipsilateral stimuli (EI), and their

Figure 1.3 Schematic diagram of the intensity pathway and the characteristic IID response tuning curves of those nuclei in this pathway which have been described.

The Intensity Pathway



response to systematic variation in IID is a monotonic, sigmoid shaped function (Moiseff and Konishi, 1983; Manley et al., 1988). There is a gradient of inhibitory input across the nucleus, with neurons which are strongly inhibited by ipsilaterally loud stimuli located dorsally and neurons weakly inhibited by loud sounds to the ipsilateral ear located ventrally (Manley et al., 1988). Immunocytochemical staining for GAD mirrors this distribution, with dense staining present dorsally and gradually declining towards the ventral aspect of the nucleus (Carr et al., 1989). Intensity differences are organized topographically across the nucleus, in the anterior-posterior dimension (Manley et al., 1988).

VLVp projects bilaterally to the lateral shell subdivision of ICc, as well as contralaterally to ICc medial shell. In ICc lateral shell the VLVp efferents provide GABAergic inhibition directly at large interaural intensity differences (Adolphs, 1993b). Adolphs (1993b) proposed a model in which excitation (which does not have to be IID tuned) is combined with bilateral inhibition from VLVp to yield neurons which have a peaked IID tuning curve. This type of tuning is characteristic of units with receptive fields that are restricted in elevation (Olsen et al., 1989). The location of the peak in the IID tuning curve could be determined by the relative amount of inhibition arriving from each VLVp. Adolphs (1993b) also found several examples of neurons in the lateral shell whose inhibitory response to ipsilaterally loud sounds could not be blocked by local application of BMI. This suggests that inhibition due to ipsilateral sounds arrives at these cells indirectly, via a polysynaptic pathway. Indeed, it appears that neurons located medially in lateral shell respond to IID in the same manner as neurons in VLVp while cells located at more lateral positions have IID responses similar to those in ICx (Adolphs, 1993b; J. Mazer, personal communication). Like the elimination of ambiguity in the time pathway, the mechanism for the

elimination of ambiguity in the intensity pathway has yet to be elucidated unequivocally. Unlike the model for the resolution of phase-ambiguity, however, Adolph's model does not address the role of the integration of information across frequencies in the resolution of spatial ambiguity inherent in intensity difference cues at the periphery (Brainard et al., 1992).

The processing of auditory information in the medial shell subdivision of ICc remains largely unexplored, primarily due to the existence of a large plexus of arteries in the IVth ventricle overlying this brain region; repeated electrode penetrations in this area put the animal's life at risk. The medial shell apparently lacks ITD tuning (R. Adolphs, personal communication) and its efferent projections have been heretofore unknown.

In the lateral shell of ICc, the time and intensity pathways converge for the first time since they were separated at the cochlear nuclei. The responses of neurons become more space-specific progressing from the medial to the lateral aspect of the subdivision, and many combinations of IID and ITD tuning types are found there (J. Mazer, personal communication). The ICc lateral shell projects to the ICx on the same side (Takahashi et al., 1989), where the responses are organized according to the position in space of sound sources in the contralateral hemifield with respect to the head. ICx, in turn, projects to the superficial layers of the optic tectum where the auditory map of space is aligned with a visual map of space (Knudsen, 1982; Knudsen and Knudsen, 1983). The superficial layers of the optic tectum, containing the sensory maps, sends efferents to the deep layers of the optic tectum which contains a motor map. This motor map is used to drive the head turning movements which the owl uses to direct its gaze (du Lac and Knudsen, 1990), since the owl's eyes are fixed in position in the eye sockets.

Plasticity in the Auditory Space Map

As a barn owl grows from a hatchling, the size of its head changes, increasing the distance between the ears and therefore changing the location in space that a particular interaural time difference signals. Also, the facial ruff, which serves to amplify and direct sounds as they reach the face (Konishi, 1973a), does not grow in until the owl is several weeks old. Despite these changes, owls are able to localize sounds fairly accurately as they mature. Consequently, the spatial locations that the owl associates with particular sound localization cues are altered during development. Barn owls which have their normal sound localization cues altered by chronic occlusion of one ear with an earplug make systematic errors in localizing sound sources. If the earplug is inserted before the owl is 60 days old, then, after a few weeks, these birds recover the ability to accurately localize sounds (Knudsen et al., 1984a). When owls older than 60 days have one of their ears plugged, they are unable to readjust their interpretation of the sound localization cues to match the correct position of a sound source. This early period of plasticity is called the sensitive period. Another plastic period, called the critical period, occurs between 60 and 200 days of age. Owls which have one ear plugged during the sensitive period accomodate for the change in localization cues, and then have the plug removed during the critical period are able to regain normal sound localization (Knudsen et al., 1984b). During the critical period owls are able to adjust the association between sound localization cue values and sound locations, but only towards a normal correspondence; the auditory system can no longer compensate for abnormal cues.

The visual space map in the optic tectum is responsible for directing the plastic changes in the auditory space map during the sensitive and critical

periods (Knudsen and Brainard, 1991; Knudsen and Knudsen, 1985, 1989, 1990). Barn owls which have one ear plugged during the sensitive period make no corrections in orienting to sound sources if they are unable to see. Young owls fitted with goggles which alter their perception of visual space adjust their auditory localization to match the deviation produced by their eyewear. Knudsen et al. (1991) demonstrated that the auditory space map can be generated in the absence of instruction from the visual system, although the topography of these maps is abnormal. Blind-reared owls have space maps in the optic tectum which are stretched out in the dorso-ventral direction, the plane in which elevation is mapped, or the representation of elevation is inverted from that observed in sighted birds. Interestingly, the mapping of the horizontal coordinate of space in the optic tectum was unaffected by blind-rearing.

Visually instructed calibration of the auditory space map in the optic tectum is effected by reorganization of the space map in ICx (Brainard and Knudsen, 1993). In owls raised wearing prismatic goggles which shift the visual field in azimuth only, ITD values in the auditory space map in the optic tectum were shifted to agree with the neurons' optically displaced visual receptive fields. The representation of ITD in ICx was similarly adjusted, but no shift was observed in ITD representations in ICc lateral shell or core. In ICx, the altered ITD values were apparent at short latencies (7-8 msec) suggesting that the adjustment in the space map is a result of plastic change in ICx itself rather than descending activity from telencephalic auditory areas.

Currently, it is unknown how the error signal coding the disparity between the visual receptive fields and the auditory receptive fields in the optic tectum are relayed to the inferior colliculus. Although the optic tectum would logically be the originator of such a signal, Knudsen and co-workers have been unable to

demonstrate a feedback pathway from the optic tectum to ICx. The optic tectum projects to the telencephalon via nucleus rotundus, but rotundus responds only to visual stimulation and is unresponsive to auditory stimuli (Knudsen, personal communication). The auditory forebrain areas remain as the most likely candidates to provide the instructive signal for plasticity in the inferior colliculus, though this possibility has yet to be demonstrated experimentally.

The Colliculo-thalamo-telencephalic Pathway

In the barn owl, the central nucleus of the inferior colliculus provides the input to the space map in ICx (Knudsen, 1983). In other species of birds the central nucleus of the inferior colliculus is known as the nucleus mesencephalicus lateralis, pars dorsalis, or MLd. In pigeons (Karten, 1967) and ring doves (Durand et al., 1992), MLd projects to the auditory thalamus, nucleus ovoidalis (N.Ov). All auditory input to the telencephalon is provided via N.Ov., which projects to the forebrain area Field L (Karten, 1968; Wild et al., 1993). Neurons recorded in Field L of the barn owl have receptive fields restricted both in elevation and azimuth and have broad frequency tuning curves (Knudsen et al., 1977). However, no systematic organization of these space-specific neurons have been found in Field L. In a more recent study, the optic tectum and the auditory thalamus were lesioned either independently or together in order to assess the relative contribution that tectum and forebrain make to sound localizing behavior (Knudsen et al., 1993). These investigators obtained evidence suggesting that sound localization and gaze control can be mediated through either the optic tectum or the thalamo-telencephalic auditory pathway. They present the hypothesis that sound localization is mediated in parallel pathways in the midbrain and forebrain of the owl, similar to the parallel neural

pathways of the retino-tectal and retino-thalamo-cortical pathways in the visual system.

Specific Aims of the Present Work

This thesis examines the possible role that N.Ov plays in transforming the neural code which represents the auditory environment. The physiological responses of N.Ov neurons to frequency and sound localization parameters are examined with respect to the response types described in the brainstem and mesencephalic auditory nuclei. The anatomy of the afferent and efferent pathways to N.Ov are also described. Taken together, the input/output connectivity and the characteristic response functions suggest that N.Ov is involved in reorganizing the auditory code, and that perhaps sound source location is not the primary feature of the stimulus which it is involved in extracting.

Chapter 2: Responses of Neurons in Nucleus Ovoidalis to Dichotic Auditory Stimulation in the Anesthetized Owl

"If your experiment needs statistics, you ought to have done a better experiment."

- Lord Ernest Rutherford, in The Mathematical Approach to Biology and Medicine, N.T.J. Bailey, 1967.

Methods

Adult barn owls (*Tyto alba*) were anesthetized with intramuscular injections of ketamine hydrochloride (Ketaset, Aveco; 10 mg/hr) and an initial supplemental dose of diazepam (Diazepam Injection, Steris Labs; 0.25 mg). Anesthetized owls were placed into a stereotaxic device that held the head tilted downward at an angle of 45° from the horizontal plane, a stainless steel plate was attached to the rostral portion of the cranium with dental cement and a reference pin was glued to the cranium on the midline in the plane between the two ears. A hole approximately 1.0 cm^2 was opened in the skull with rongeurs and mineral oil was applied to the exposed dura to prevent it from drying out over the course of the experiment. Body temperature was maintained between 38° and 39° C with a circulating-water heating pad. After the experiment, the exposed dura was coated with a topical antibiotic ointment (Neosporin, Burroughs Wellcome Co.) and the craniotomy closed with dental cement. The scalp was sutured and the incision line covered with more topical antibiotic. The owl was returned to its recovery cage and monitored until the anesthetic had worn off. Owls usually ate 1-2 mice the following morning, but those who refused food were administered 10 to 20 ml of 5% dextrose in lactated Ringer's solution intravenously and 10,000 units of penicillin by intramuscular injection. All birds treated in this manner resumed normal feeding within 24 hours.

The activity of single neurons was recorded extracellularly with glass electrodes filled with either Woods Alloy metal plated with gold and platinum or 0.5 M sodium acetate containing 2% pontamine sky blue. Electrode impedances were between 2 and 5 MΩ. Electrodes were placed in a holder which was attached to a mechanical microdrive capable of advancing the electrode in

discrete steps of from 1 to 200 μm. The microdrive was held by a manipulator which allowed the electrode to be positioned with micrometer accuracy with respect to the reference pin in both the medio-lateral and rostro-caudal dimensions. Electrodes were usually lowered through the intact dura in order to minimize edema, though on very few occasions the dura was resected over a very limited area after the target nucleus had been located. Action potentials were amplified, filtered (300 Hz high pass, 10 kHz low pass) and sent to both an analog oscilloscope and an audio monitor. The criteria for single neuron isolation were that action potentials not occur within 1 msec of each other, and that the amplitude of all discriminated action potentials was very nearly constant. Under these conditions the output from the audio monitor consisted of distinct "pops" with practically no background broadband noise responding to the stimulus. Level-discriminated action potentials were converted into 5 V TTL pulses whose times of occurrence were recorded with microsecond resolution by a custom made event timer board (Beckman Electronic Shop) on the STD bus of a Masscomp 5600 minicomputer. Custom software (see Appendix) delivered stimuli and synchronously acquired spike timing data in real time. Data analysis was carried out separately (though not in real time) during the course of the experiment. Neural spikes were recorded for 100 msec prior to the stimulus, for the duration of the stimulus and for 100 msec after the stimulus was complete. Information about spontaneous activity was obtained on each repetition from the 100 msecs preceding the stimulus. The number of stimulus repetitions at each point in the tuning curve was selectable, although in the majority of cases 5 repetitions with a given set of parameters was used. At each point in the tuning curve, the number of spikes was averaged over the number of stimulus repetitions and the mean and standard error was calculated and plotted for the evoked activity and the mean was plotted for the spontaneous activity. Electrode

tracks were histologically verified and recording sites were marked with either electrolytic lesions when Woods Alloy metal was used (5 μA DC current for 10 sec) or by electrophoresing pontamine sky blue when NaOAc electrodes were used (-10 μA DC current pulsed 7 sec on/7 sec off for 5 min).

Stimulus Generation. In a soundproof chamber, stimuli were delivered through calibrated earphones (Takahashi and Konishi, 1986; Wagner et al., 1987) that provided power over the frequency range used by barn owls for sound localization (1-10 kHz). Auditory stimuli were digitally synthesized tone, pseudo-random noise, and narrow band noise of various pass band and center frequencies. Two channels of output from a 12 bit D/A converter (EF12M, Masscomp) were passed through a digital reconstruction filter, amplified and sent to a pair of 16 bit digital attenuators (Beckman Electronic Shop) which controlled the intensity level at each earphone. Bandpass noise was obtained by filtering the pseudo-random noise in the frequency domain to avoid phase distortion. The D/A conversion rate was 100 kHz allowing for a minimum ITD between the two channels of 10 μsec. The beginning, ending and incremental step parameters for frequency and sound localization cues were selectable by the investigator. Stimuli were 100 msec in duration, had 5 msec linear rise/fall times, and were presented every 1-2 seconds.

Definition of Terms. Interaural intensity difference (IID) is defined to be the sound intensity in the right ear minus the sound intensity in the left ear (in dB). The average binaural intensity (ABI) is the mean of the sound intensity in the two ears. Thus, at zero IID the sound intensity in both ears is the same and equal to the ABI. IID can be varied either by holding the ABI constant, or by holding the intensity to one ear constant while varying the intensity to the other ear. In the former case, as the intensity to one ear increases, the intensity to the other ear

decreases by the same amount. In the other case, the ABI varies with the IID. IID at constant ABI was used unless otherwise noted.

Interaural time difference (ITD) is the temporal offset between identical portions of the stimulus being delivered to the two ears. Negative values mean that the left stimulus precedes the right by the ITD value, while positive values indicate that the right stimulus leads the left.

Results

A total of 207 neurons from 18 owls of both sexes were recorded in nucleus ovoidalis over the course of these investigations. Of these neurons, 153 (74%) were chosen for inclusion into the data set presented here on the bases of high quality single-neuron isolations, significant responses (enough spikes for statistics to be meaningful) and histological localization to the nucleus. Neurons were obtained from both the left and right N.Ov, though the majority of physiologically characterized neurons were isolated on the left side. Of the recorded neurons 100% responded to auditory stimulation with white noise. Furthermore, all isolated neurons displayed selective tuning to at least one sound localization cue (interaural time difference and/or interaural intensity difference).

Responses to Tone

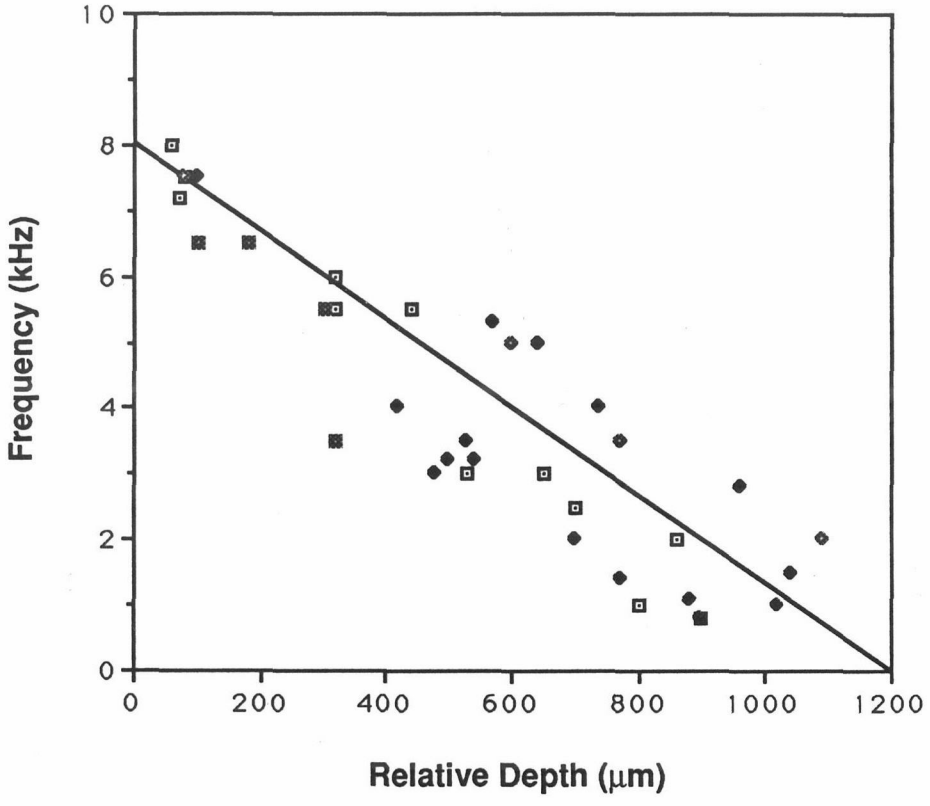
The central and medial portions of N.Ov contained neurons which were tonotopically organized, with neurons having high best frequencies located dorsally while those with decreasing best frequencies were located progressively more ventral (Figure 2.1a), as has been reported in the starling (Bigalke-Kunz et al., 1987) and the zebra finch (Diekamp and Margoliash, 1991). This tonotopic organization is inverted from the map of frequency occurring in the inferior colliculus (Wagner et al., 1987; Takahashi et al., 1989; Fujita and Konishi, 1992). The frequency representations in the centro-medial portion of N.Ov of the barn owl consist of isofrequency laminae which extend mediolaterally and rostrocaudally, lying roughly perpendicular to the dorso-ventral plane of electrode penetration (figure 2.1b). Auditory nuclei in the brainstem and midbrain of the owl have a tonotopic organization which is best fit by regression lines of different slope, with frequencies between 4 and 8 kHz being overrepresented in the

Figure 2.1 Frequency organization of centro-medial portion of nucleus ovoidalis.

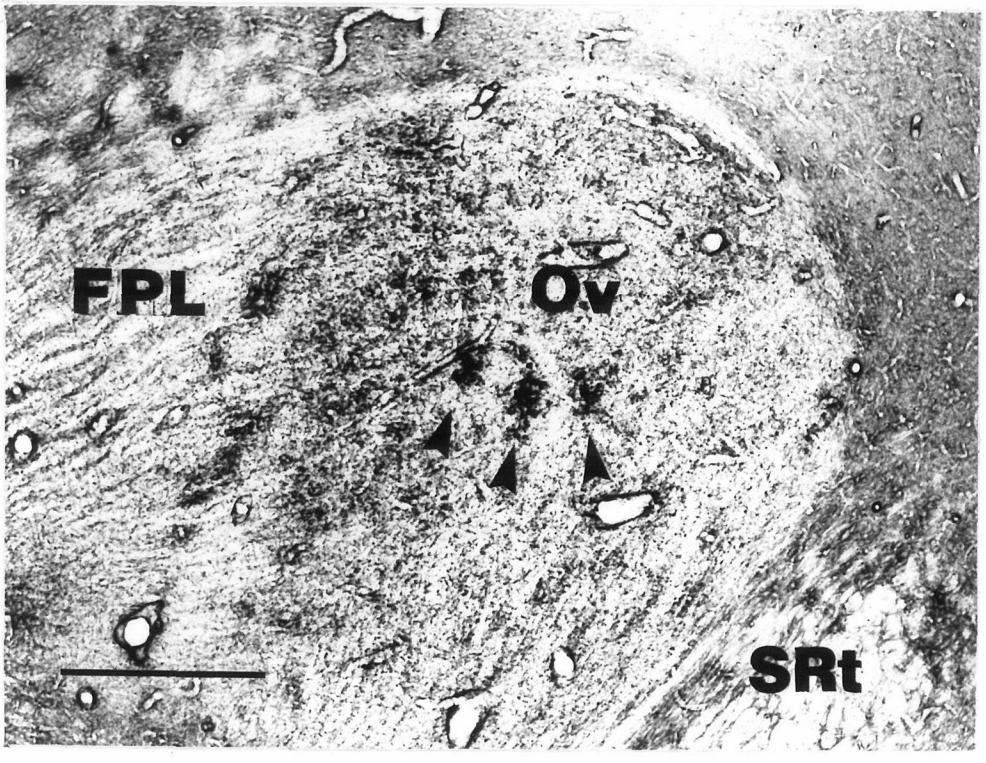
A) Plot of neuronal best frequency vs. the location of the neuron in the nucleus along the dorso-ventral axis. Positions along the x-axis are relative to the dorsal border of the nucleus as determined by the location where auditory activity was first discernible. Data for this figure were compiled from four different owls: Owl #395 (open triangles) - 2 penetrations; Owl #415 (open squares) - 2 penetrations; Owl #440 (closed triangles) - 3 penetrations; Owl #449 (closed squares) - 2 penetrations. Electrode penetrations from a given owl may have occurred during one experiment or during separate experiments. The criterion for inclusion into this data set was that at least 4 neurons were isolated in different frequency regions during a single electrode penetration.

B) Coronal section showing fiducial marks in the centro-medial portion of N.Ov. Three locations across the mediolateral extent of the nucleus were marked in the 4.5 kHz frequency lamina. Each mark is approximately 200 μm apart. The leftmost mark borders closely on the lateral portion of the nucleus which has a different frequency organization. Dorsal is towards the top and medial is to the right. Scale bar = 500 μm.

A


10
8
6
4
2
0
Frequency (kHz)
0
200
400
600
800
1000
1200
Relative Depth (μm)

B


FPL
Ov
SRt

nucleus while frequencies less than about 3 kHz are relegated to narrow regions of the nucleus (Wagner et al., 1987; personal observation). In nucleus ovoidalis, however, all frequencies between 500 Hz and 9 kHz are represented by approximately equal areas of the centro-medial portion of the nucleus. The frequency organization was not absolutely strict, however. On occasion a neuron with a low best frequency would be isolated in a high frequency region, or a neuron with a high best frequency would be isolated in a low frequency region. Thereafter, the predominant tonotopic trend would resume. The extracellular recording techniques employed in this study preclude identifying the type of these neurons with best frequencies that are "out of order," and they could be either inhibitory interneurons, primary projection neurons, or (less likely) fibers of passage.

The organization of best frequency in the lateral portion of the nucleus was quite different. In a relatively narrow portion of the nucleus on the lateral aspect, neurons in a given dorso-ventral penetration have the same best frequency. This region corresponds with that portion of the nucleus which contains distinctly larger cell bodies when viewed in Nissl stained sections (figure 3.1), and which also receives a characteristic input from the inferior colliculus (see Chapter 3). All of the physiologically characterized neurons which were localized to this region had best frequencies between 4.5 and 5.0 kHz. Although it is possible that this region has a tonotopic organization in the rostro-caudal dimension, I was unable to demonstrate such a mapping. Anteriorly, this lateral portion of the nucleus runs into the fasciculus presencephali lateralis (FPL) (Karten, 1968) which carries the efferent axons from nucleus ovoidalis to the forebrain. The FPL fiber bundle is not tonotopically organized. Posteriorly N.Ov becomes quite small and the frequency representation in the centro-medial portion is compressed. I

was unable to obtain multiple single-neuron isolations in the caudal N.Ov which could be unambiguously assigned to the lateral division of the nucleus.

In the lateral shell of ICc, multiple ITD peaks are suppressed as ITD information is integrated across several frequencies and combined with inhibitory input. While phase ambiguous neurons in ICc core have narrow frequency tuning curves, neurons which respond specifically to a single ITD tend to have relatively broad frequency tuning curves. Some space-specific neurons in ICx cannot be driven by single tones at all, but require broad band stimuli. In order to determine the amount of possible cross-frequency integration occurring in N.Ov as well as to compare with processing occurring in the subdivisions of the inferior colliculus, the width of the frequency tuning curves at one-half the peak value were measured in cells from N.Ov (figure 2.2). The majority of cells had narrow frequency tuning curves, comparable to those obtained in the core of ICc or the cochlear nuclei (Carr and Konishi, 1990; Wagner et al., 1987). While some neurons were found with very broad frequency tuning curves, this characteristic did not necessarily correlate with the elimination of phase ambiguity in the ITD tuning curve, or specificity in the IID tuning curve (see Response to Sound Localization Cues below). Of neurons with a single peak in the best frequency tuning curve 18% exhibited a pronounced inhibition below spontaneous levels at frequencies flanking the best frequency (figure 2.3), which has also been reported in the N.Ov of starlings (Bigalke-Kunz et al., 1987) and zebra finches (Diekamp and Margoliash, 1991).

A number of cells in N.Ov (12.3%) had frequency tuning curves with multiple distinct peaks (Figure 2.4). The location of these peaks varied between neurons, with one peak corresponding to the tonotopic organization of the nucleus. Usually, one peak was quite narrow and the other peak relatively broad.

Figure 2.2 Distribution of width of frequency tuning curves measured at half of the peak value (best frequency). Neurons which had more than one peak in their frequency tuning curve did not have their widths measured and are tallied separately. The great majority of neurons were narrowly tuned to frequency or were tuned to more than one frequency, although cells that were broadly tuned were fairly common as a group (3-7 kHz wide at half peak).

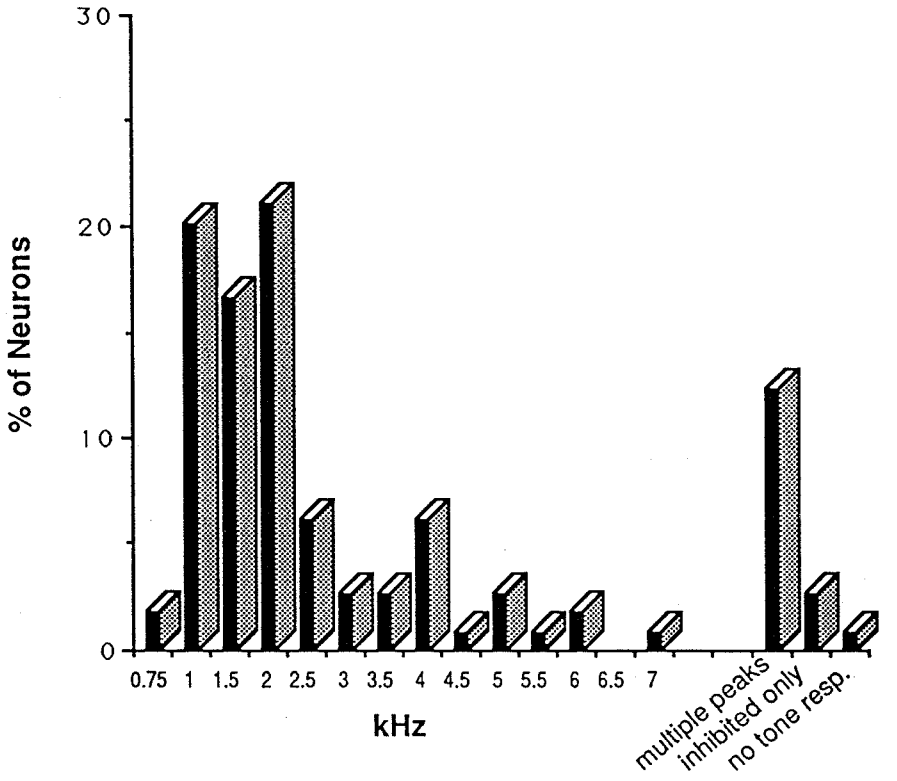
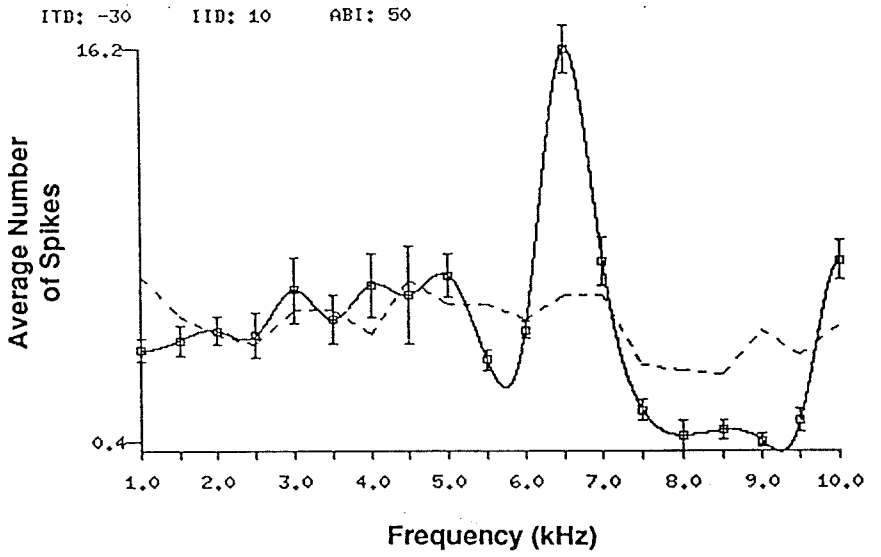

30
20
10
0
% of Neurons
0.75 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7
kHz
multiple peaks
inhibited only
no tone resp.

Figure 2.3 Examples of neurons whose frequency tuning curves consisted of an excitatory peak flanked by regions of inhibition (below the spontaneous rate). Symbols used for all response tuning curves contained herein: boxes indicate data points consisting of the mean of 5 stimulus repetitions at a particular frequency; error bars indicate standard error. Solid line is the cubic spline interpolation between data points. Dashed line indicates spontaneous activity level in the 100 msec preceding the stimulus.


ITD: -30
IID: 10
ABI: 50
16.2
0.4
Average Number of Spikes
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
Frequency (kHz)

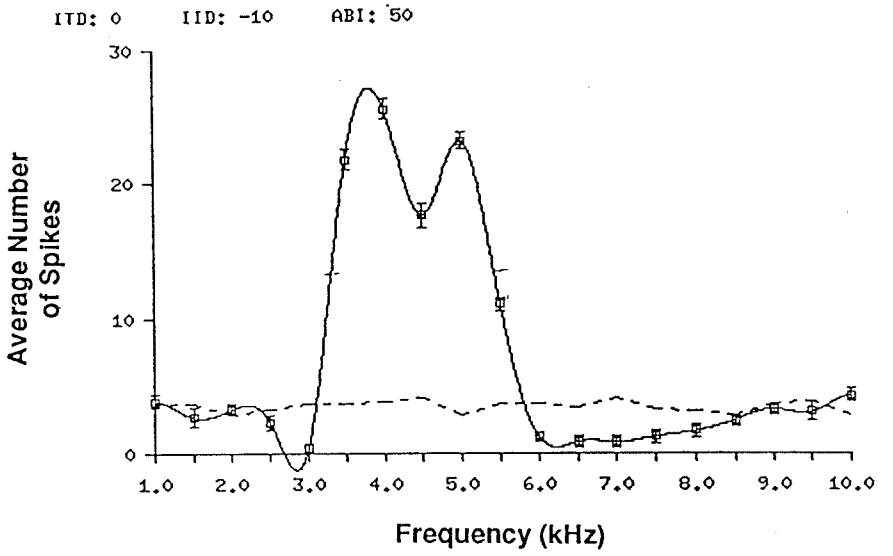
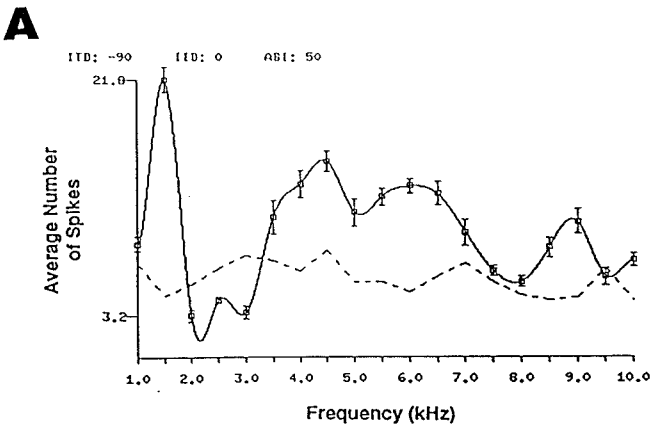
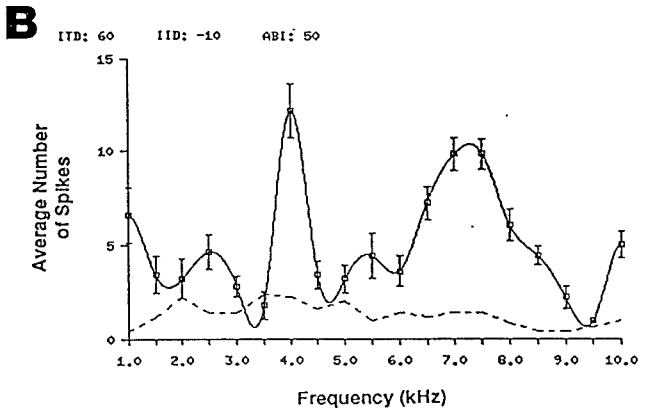
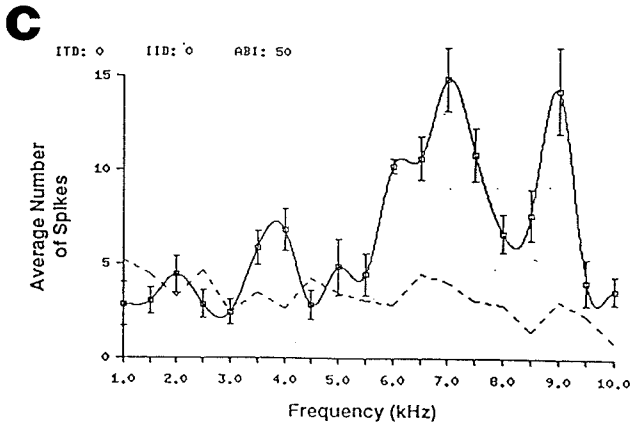

ITD: 0
IID: -10
ABI: 50
30
20
10
0
Average Number of Spikes
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
Frequency (kHz)

Figure 2.4 Examples of neurons whose frequency tuning curves contained more than one peak, or best frequency. Each curve shown was taken at the values of ITD and IID which elicited the maximal response; the location of the peaks did not vary with ITD and IID but the size of the peaks did. As seen in A-C, the peaks could be located anywhere in the frequency range investigated (1-10 kHz). In some cases the peaks were separated by an inhibitory frequency region (A). The distance between peaks varied from as much as 3.5 kHz as in (B) to as little as 1 kHz (data not shown). Frequency tuning curves with multiple peaks separate by more than 3.5 kHz were not observed.


A
ITD: -90
IID: 0
ABI: 50
21.8
3.2
Average Number of Spikes
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
Frequency (kHz)


B
ITD: 60
IID: -10
ABI: 50
15
10
5
0
Average Number of Spikes
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
Frequency (kHz)


C
ITD: 0
IID: 0
ABI: 50
15
10
5
0
Average Number of Spikes
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
Frequency (kHz)

The narrow peak could be above (Figure 2.4c) or below (Figure 2.4a,b) the broad peak. Two of these neurons had two peaks that were the same width (data not shown). In this study, the neurons with multiple frequency peaks were isolated in the centro-medial portion of the nucleus which had the dorso-ventral tonotopic organization. Variations in ITD and IID had no effect on the location of these multiple peaks. The location of the peaks could be anywhere in the range of frequencies tested (1-10 kHz). The distance between peaks could be as small as 1 kHz or as much as 3.5 kHz (figure 2.4b). In two cases the peaks were separated by a frequency region in which tones inhibited the neuron below spontaneous rates. In all but one case (a space-specific neuron), cells which had more than one peak had phase-ambiguous ITD tuning curves in response to white noise. When the peaks were located within approximately 2 kHz of each other, ITD tuning was obtained in response to tones with frequencies located at either peak. The distance between peaks in the ITD tuning curves in response to noise, however, corresponded to the period of the frequency of only one of the peaks. In those cases where the peaks in the frequency tuning curves were more than 2 kHz apart, ITD tuning could be obtained at one of the best frequencies but not the other. Futhermore, the distance between peaks in the ITD tuning curve in response to noise corresponded to the period of that frequency to which the neuron had an ITD response. Interaural intensity difference responses in neurons with multiple best frequencies was variable. In several cases, IID tuning did not change significantly in response to stimulation with different frequencies corresponding to multiple peaks in the frequency tuning curve. One neuron, however, had an IID tuning curve which had a peak at 0 dB IID when stimulated with tone corresponding to one peak in the frequency tuning curve, and had an IID tuning curve with the maximum value at +50 dB IID when stimulated with tone corresponding to the other peak in the frequency tuning

curve. The IID tuning curve in response to noise for this neuron displayed two maxima - one at 0 dB IID and one at +55 dB IID. Another neuron had one best frequency at 3.0 kHz and a second best frequency at 6.0 kHz (figure 2.5). At 3.0 kHz the IID tuning was monotonic while at 6.0 kHz the cell had a narrowly peaked IID tuning curve. The IID tuning curve of this cell in response to noise appeared to approximate the sum of the responses at the two frequency peaks. Several of these neurons with multiple frequency peaks had different response latencies for tone stimuli with frequencies located at the different peaks. Many of these neurons had an onset response for one best frequency but lacked an onset response at the other best frequency. Bigalke-Kunz et al. (1991) did not report finding neurons which had frequency tuning curves with multiple peaks in the N.Ov of the starling, but Banks and Margoliash (1993) mention finding such neurons in the N.Ov of zebra finches.

Approximately 3% of the neurons responded to pure tone with an inhibition below spontaneous activity levels despite having a vigorous excitatory response to white noise. Rarely did a neuron show no deviation from background activity in response to tone while still responding well to white noise.

Response to Sound Localization Cues

All cells obtained in N.Ov responded in a tuned manner to at least one sound localization cue. In contrast to all midbrain and brainstem auditory nuclei studied thus far in the barn owl, N.Ov has no apparent topographic organization of either sound localization cue in the dorso-ventral, rostro-caudal or medio-lateral planes. Furthermore, there is no apparent clustering or grouping of neurons with similar responses to sound localization cues in nucleus ovoidalis. Most neurons (62%) responded in a tuned manner to both ITD and IID, but a

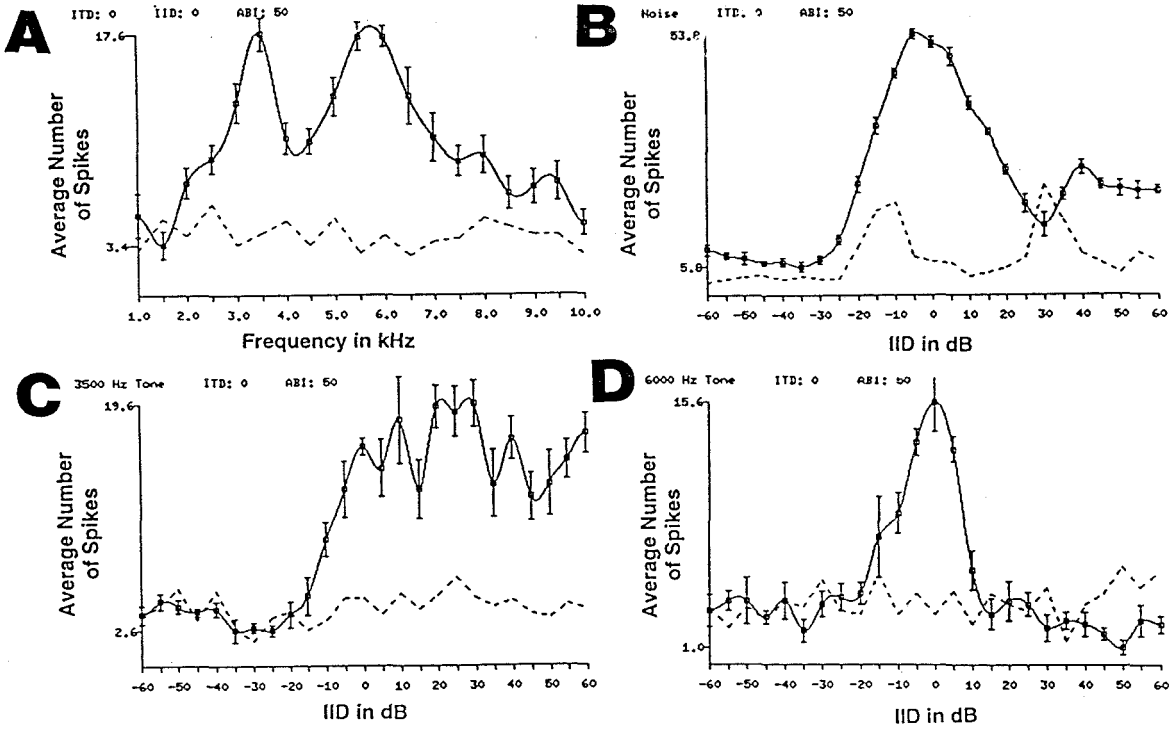
Figure 2.5 Response tuning curves from an isolated neuron in N.Ov which displayed different types of IID tuning at each of two best frequencies contained in its frequency tuning curve.

A) Frequency tuning curve displaying two maxima; one peak occurs at 3.5 kHz and the other occurs at 6.0 kHz.

B) IID tuning curve in response to noise; this type of curve is classified as peaked (or IID specific). This curve is roughly the sum of those contained in (C) and (D).

C) Monotonic tuning curve obtained with a stimulating tone of 3.5 kHz, corresponding to the lower peak of the two contained in the frequency tuning curve.

D) Narrowly peaked IID tuning curve obtained by stimulating with a tone of 6.0 kHz (the location of the second peak in the frequency tuning curve) at various IID values.


A
ITD: 0
IID: 0
ABI: 50
17.6
3.4
Average Number of Spikes
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
Frequency in kHz
B
Noise
ITD: 0
ABI: 50
53.8
5.8
Average Number of Spikes
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60
IID in dB
C
3500 Hz Tone
ITD: 0
ABI: 50
19.6
2.6
Average Number of Spikes
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60
IID in dB
D
6000 Hz Tone
ITD: 0
ABI: 50
15.6
1.0
Average Number of Spikes
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60
IID in dB

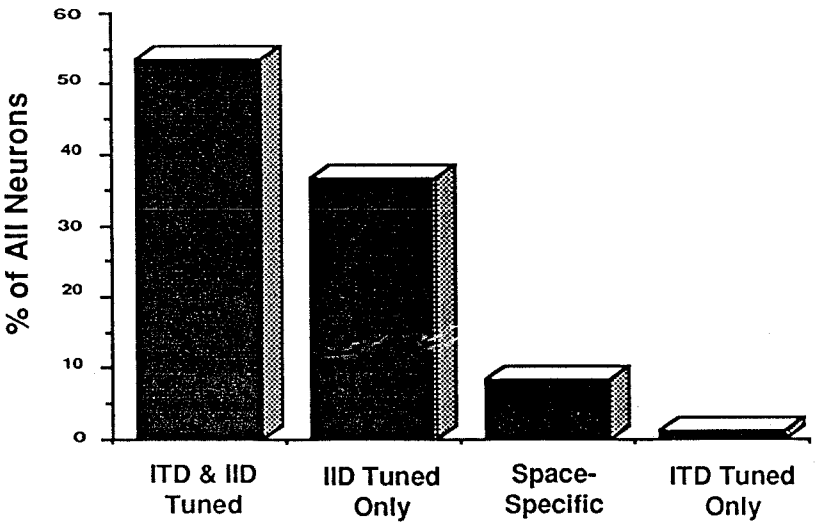
large number (36.6%) of cells were unresponsive to ITD (figure 2.6a - IID tuned only). Only 4 neurons were found which were tuned to ITD but not IID.

Of the neurons which were tuned to IID, nearly half (49.7%) had a tuning curve which was specific for a particular IID (peaked IID tuning curves, figure 2.7a). Slightly more than half of the neurons which were tuned to IID had monotonic, sigmoid type response functions when stimulated with noise. The distinction between peaked and sigmoid IID tuning curves was not always absolutely clear. Many neurons in N.Ov exhibited IID tuning curves which displayed a distinct maximal peak, but which had significantly elevated activity levels on one side of the peak in response to broadband stimuli. Cells which had such a "shoulder" in their IID tuning curves appear to be composites of the sigmoid and peaked IID tuning curve types. Such IID tuning curves possibly represent intermediate processing stages in the transformation from ambiguous to unambiguous sound elevation coding. Although Adolphs (1993b) makes no mention of this type of IID tuning, it has been observed repeatedly in ICc lateral shell (Jamie Mazer, personal communication). No observation of this type of tuning has been made in ICx. For the purposes of this study, IID response types which had a peak with a shoulder when stimulated with noise were classified as peaked if the height of the shoulder was less than 75% of the value of the peak. Those which had a shoulder with an amplitude 75% or more of the value of the peak were classified as monotonic.

Of the neurons which had peaked IID tuning curves, 69.3% (52/75) also responded to varying ITD in a phase-ambiguous manner when stimulated with noise. Neurons with this combination of IID and ITD tuning curves have been previously observed only in the lateral shell subdivision of the ICc (Jamie Mazer, personal communication). Of the neurons with peaked IID tuning

Figure 2.6 A) Distribution of responses to sound localization cues recorded in nucleus ovoidalis. B) Distribution of combinations of response types from among all neurons recorded in N.Ov which responded in a tuned manner to both interaural time and intensity differences. All possible combinations of IID and ITD tuning are found in N.Ov with the exception of EI and side peak suppressed.

A


% of All Neurons
60
50
40
30
20
10
0
ITD & IID Tuned
IID Tuned Only
Space-Specific
ITD Tuned Only

B

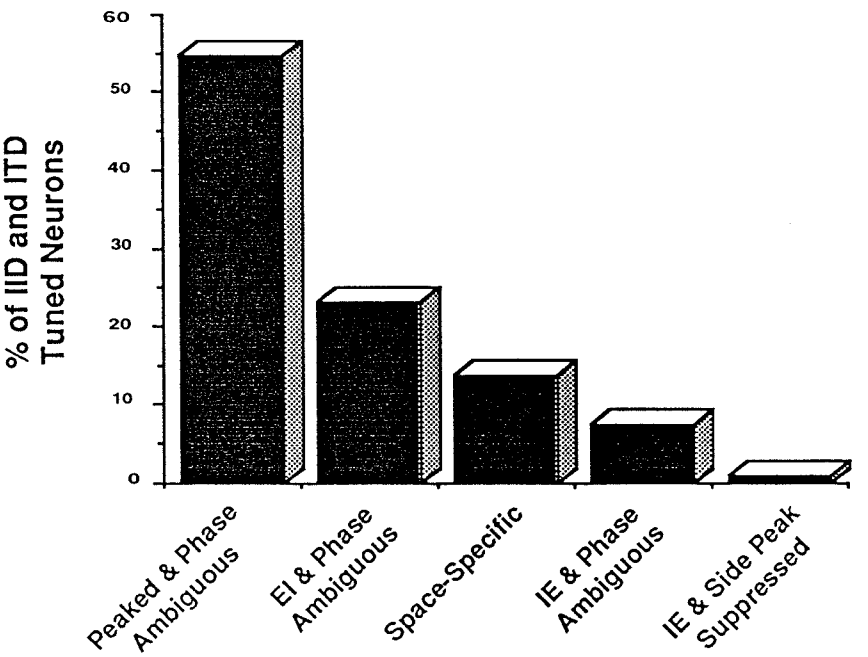
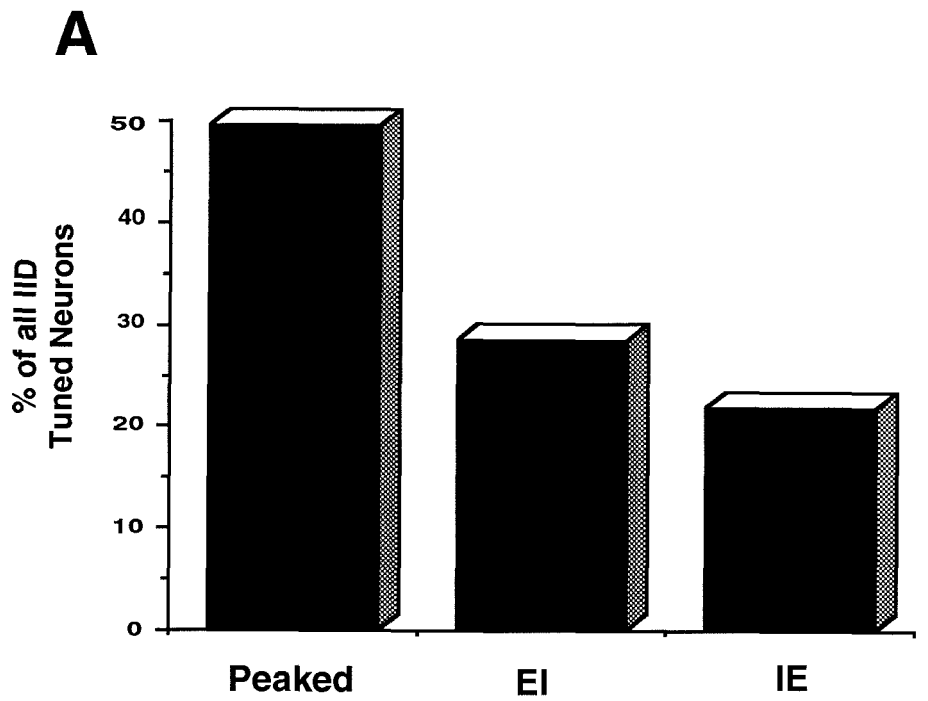
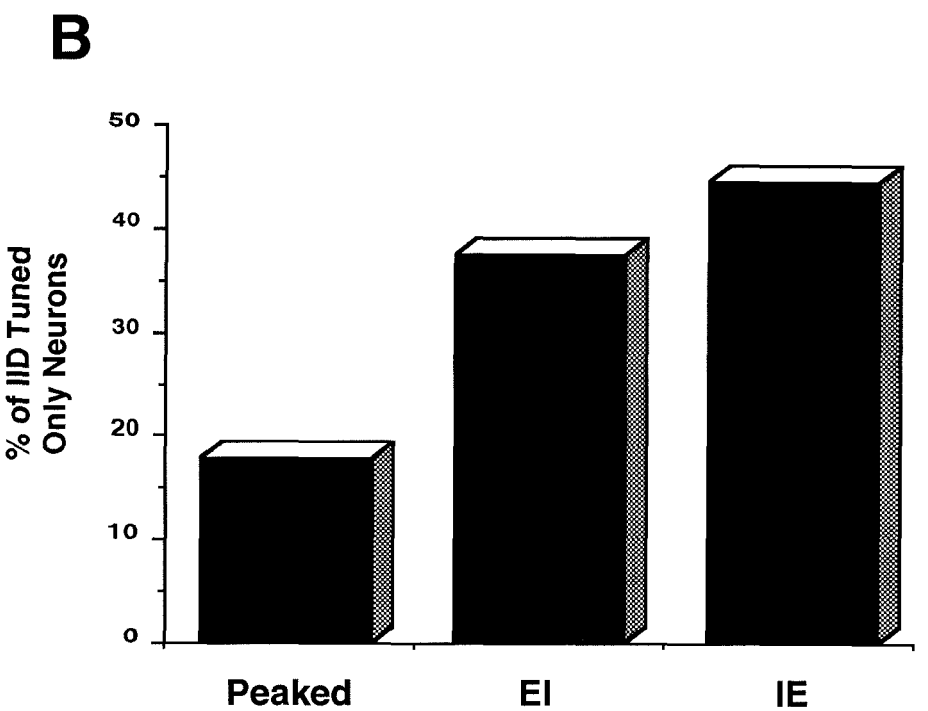

% of IID and ITD Tuned Neurons
60
50
40
30
20
10
0
Peaked & Phase Ambiguous
EI & Phase Ambiguous
Space-Specific
IE & Phase Ambiguous
IE & Side Peak Suppressed

Figure 2.7 A) Distribution of IID tuning curve types among all neurons which responded to interaural intensity differences in a tuned manner (includeding those which responed to ITD as well). B) Summary of IID response types among neurons which were unresponsive to ITD. While a neuron which is tuned to intensity differences is most likely to have a peaked IID tuning curve, if that neuron is unresponsive to ITD it is very likely that it has a monotonic, sigmoid type IID tuning curve.


A
% of all IID Tuned Neurons
50
40
30
20
10
0
Peaked
EI
IE


B
% of IID Tuned Only Neurons
50
40
30
20
10
0
Peaked
EI
IE

curves 17.3% (13/75) had ITD tuning curves which had a prominent central peak and suppressed side peaks. These are characteristics of neurons which respond selectively to unique combinations of interaural time and level differences - space-specific neurons. Finally, 13.3% (10/75) of IID specific neurons had no tuned response to ITD; the ITD response curves for these cells were essentially flat at a given level above the spontaneous background activity. Neurons which have peaked IID tuning curves yet are insensitive to variations in ITD have not previously been described at any level of the barn owl's auditory system.

Of the neurons which were tuned to interaural intensity differences 50.3% were catagorized as having monotonic, sigmoid type response functions. These neurons were either excited by stimulation of the contralateral ear and inhibited by stimulation of the ipsilateral ear (EI cells), or they were inhibited by contralateral stimulation and excited by ipsilateral stimulation (IE cells). Neurons which have IE type IID tuning curves are quite rare in the barn owl's auditory system, although some have been observed in the ICc lateral shell (Jamie Mazer, personal communication). Of the EI cells, 51.2% were tuned to ITD in a phase ambiguous manner while 48.8% were not responsive to ITD variations. Conversely, 75.8% of the IE cells displayed no ITD tuning while 21.2% had phase ambiguous ITD tuning curves. One IE neuron was specific for a single ITD - its ITD tuning curve had a single large peak with suppressed side peaks. This particular combination of IID and ITD characteristics has not been described in any other auditory nuclei of the barn owl.

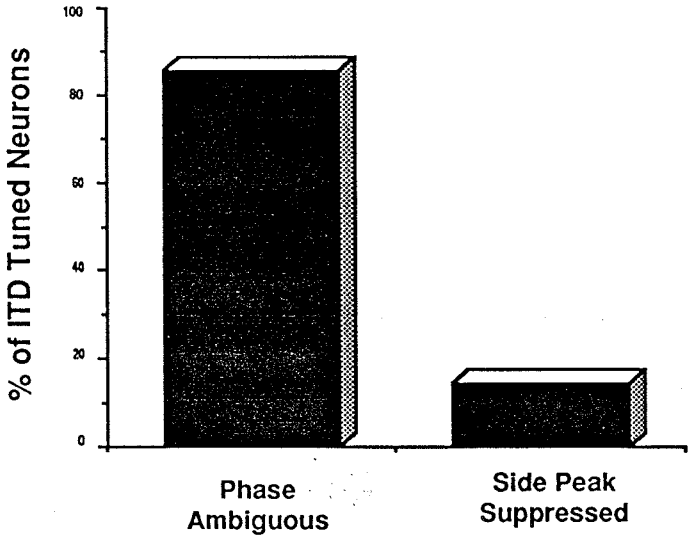
Of the neurons which responded to ITD, 85.6% did so in a phase ambiguous manner while the remainder (14.4%) were classified as having side peak suppression (figure 2.8a). Only one neuron was found which displayed the nearly complete lack of side peaks as is commonly found in ICx. The remainder

of the cells which exhibited side peak suppression in their ITD tuning curves in response to noise had incomplete suppression of the secondary peaks, as is commonly found in ICc lateral shell. Takahashi and Konishi (1986) surveyed neurons in ICx and found that the degree of side peak suppression varied widely. These investigators found that the distribution of the ratio of side peaks to main peak in ICx was roughly gaussian with a mean of 0.47. Neurons with ratios of 0 (complete lack of side peaks) and 100% (fully phase-ambiguous) were reported in ICx. The criteria that was employed to ascertain that the neurons recorded from were confined to ICx, however, was not discussed. Furthermore, this paper predates the finding that ICc contains a lateral shell subdivision, so it is possible that a number of the neurons in the sample were in fact in the central nucleus. Neurons in the optic tectum are commonly found which have only 50% side peak suppression (Eric Knudsen, personal communication). Since it has yet to be ascertained what degree of side peak suppression is necessary in order to render a neuron "space-specific," all neurons which had side peaks smaller than 75% of the main peak were classified as space-specific for the purposes of this study.

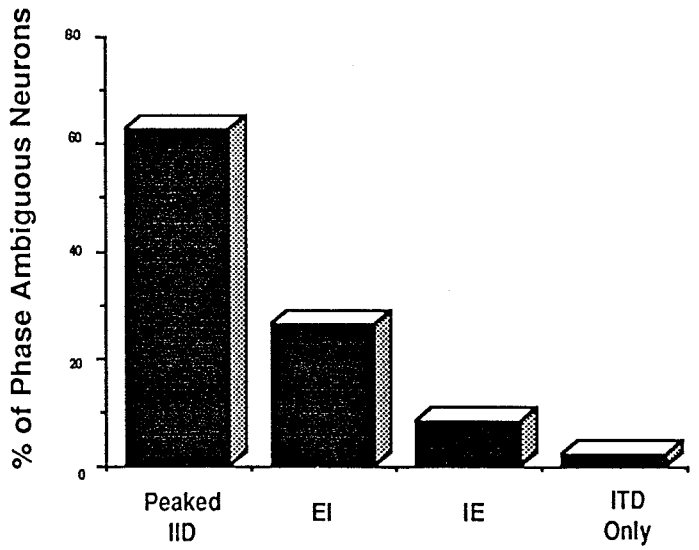
Neurons which had phase ambiguous ITD tuning curves could have either monotonic or peaked IID curves (Figure 2.8b). The two neurons which were tuned only to ITD and not IID had phase ambiguous tuning curves. Of the 83 neurons which responded to ITD variations in a phase ambiguous manner, 51 (61.4%) had the trough of the ITD curve well above the spontaneous activity level. The remaining 32 (38.6%) had troughs in their ITD curves at or below the spontaneous activity level. Fujita and Konishi (1991) observed both types of ITD tuning curves in ICc core. Additionally, they demonstrated that neurons which had troughs in their ITD curves at or below the spontaneous activity level could

Figure 2.8 A) Distribution of the types of interaural time difference tuning curves among all neurons in N.Ov which responded to ITD in a tuned manner. B) Distribution of combinations of ITD and IID tuning from among those neurons whose ITD tuning curves were phase ambiguous.

A


% of ITD Tuned Neurons
100
80
60
40
20
0
Phase Ambiguous
Side Peak Suppressed

B


% of Phase Ambiguous Neurons
80
60
40
20
0
Peaked IID
EI
IE
ITD Only

have those troughs raised well above spontaneous activity by the iontophoretic application of the GABAergic antagonist BMI in ICc core. It is possible that ITD tuning curves which have troughs above or below the spontaneous activity level represent successive processing stages in the time pathway.

The presence of space-specific neurons in Field L, the efferent target of N.Ov, raises the question of whether this response type is synthesized de novo in the thalamus or forebrain, or is relayed via N.Ov from ICx or the optic tectum. Space-specific neurons in ICx and optic tectum typically have very broad frequency tuning curves (or are unresponsive to tones and require broad band stimuli), have peaked IID tuning curves and ITD tuning curves with a single, dominating peak (Takahashi and Konishi, 1986; Wagner et al., 1987; Olsen et al., 1989; Fujita and Konishi, 1991). In the N.Ov, neurons were classified as space-specific if they had a peaked IID tuning curve and their ITD tuning curve in response to noise showed a single large peak with smaller sides peaks. Some such space-specific neurons in N.Ov had frequency tuning characteristics not found in neurons with similar IID and ITD responses in the midbrain. The neuron in figure 2.9a, for example, had a quite narrow frequency tuning curve with the best frequency at a relatively low value of 1.5 kHz. Despite the fact that barn owls are unable to use frequencies lower than about 3.0 kHz for sound localization in the vertical plane (Payne, 1971; Moiseff and Konishi, 1981b), this neuron displays IID and ITD tuning typical of space-specific neurons. Figure 2.9b shows a neuron which also had fairly narrow frequency tuning, although the degree of side peak suppression in the ITD tuning curve in response to noise was fairly small and the IID tuning curve was quite broad. The cell documented in figure 2.10a displayed IID and ITD tuning curves which are typical of neurons in the space map in ICx, yet this neuron had two peaks in its frequency tuning

Figure 2.9 Frequency and sound localization cue tuning curves from "space-specific" neurons in N.Ov.

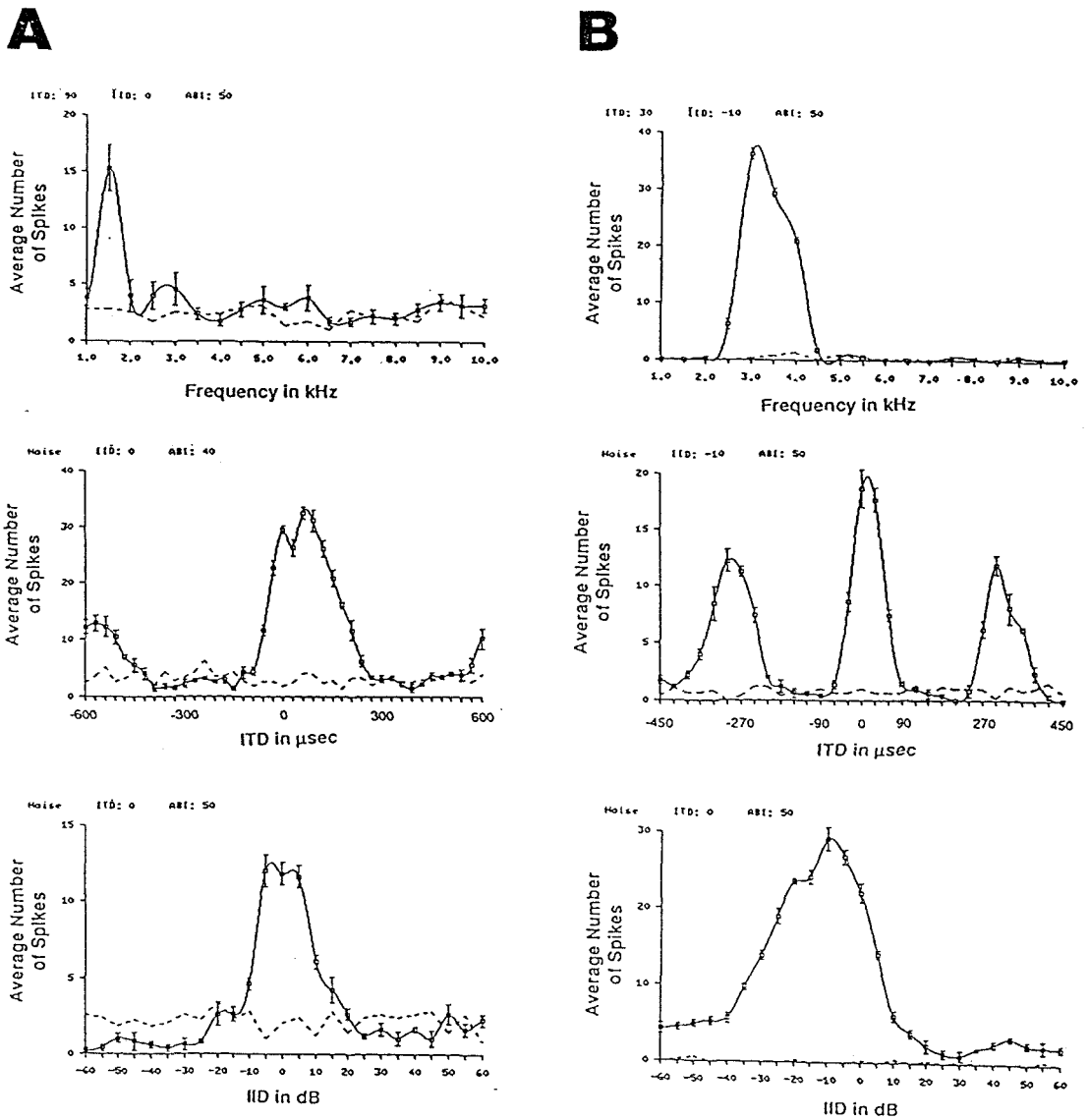
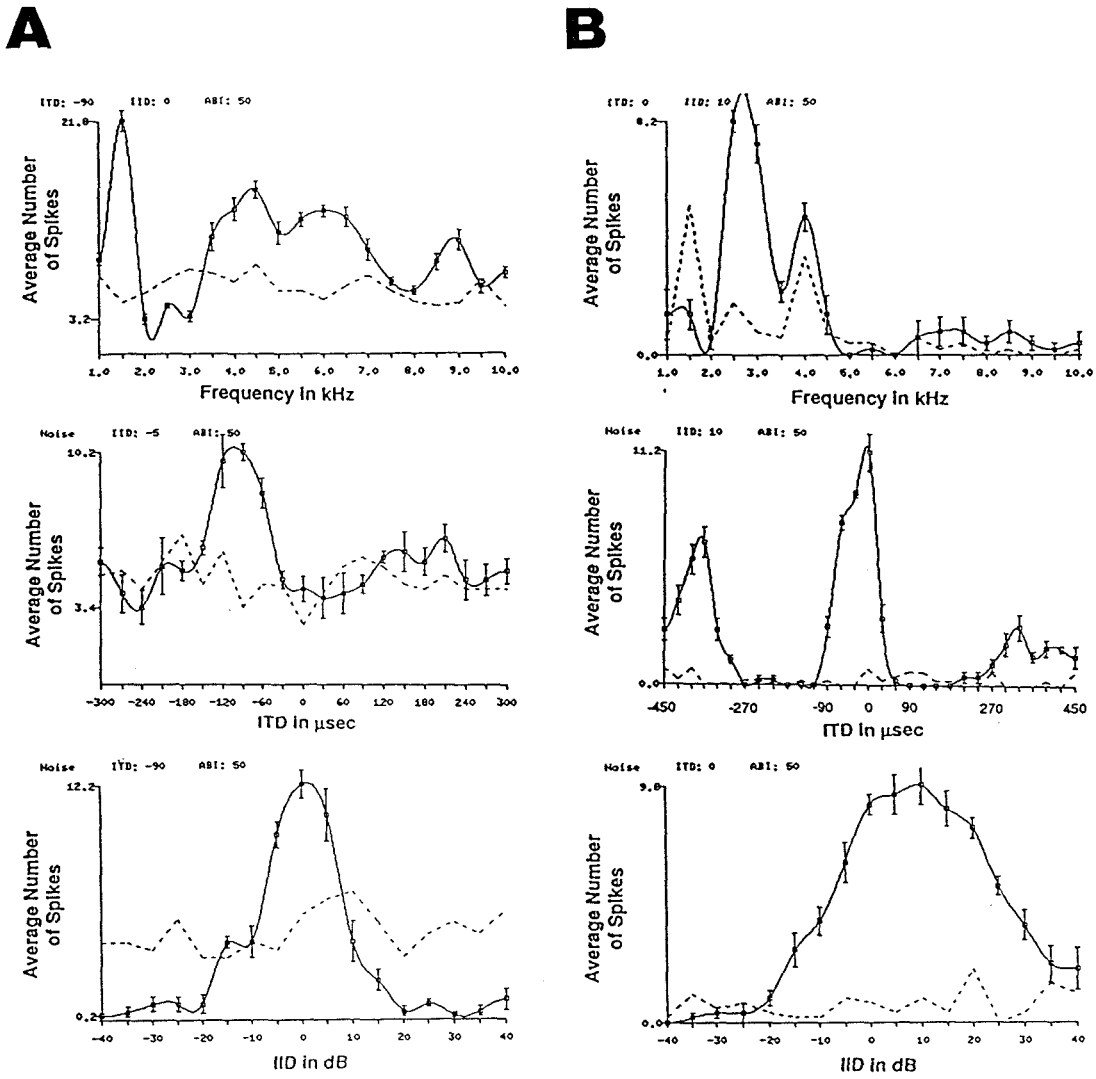

A
B
Average Number of Spikes
Frequency in kHz
ITD in µsec
IID in dB

Figure 2.10 Additional tuning curve responses from space-specific neurons in N.Ov.


A
B
Average Number of Spikes
Frequency in kHz
ITD in µsec
IID in dB
Noise

curve - a narrow peak centered at 1.5 kHz, and broad peak from 3.5-7.0 kHz. This neuron showed no ITD response at all when stimulated with a 1.5 kHz tone, but displayed a typical phase ambiguous ITD tuning curve when stimulated with a 4.0 kHz tone. The IID tuning curve was peaked in either case. Figure 2.10b shows a neuron very similar to that of figure 2.9b, though its best frequency was at 2.5 kHz. The remainder of the space-specific neurons had frequency tuning curves which were 3 to 4 kHz wide at half of the peak value. Interestingly, none of the neurons which had very broad frequency tuning curves (5-7 kHz wide at half-peak) were space-specific. Furthermore, none of the space-specific neurons were unresponsive to tones, as is frequently seen in ICx.

In contrast to the neurons described above which displayed side peak suppression and yet had narrow frequency tuning curves, three neurons were found in N.Ov which had broad frequency tuning curves but responded to white noise with a phase ambiguous ITD tuning curve (figure 2.11). All three such neurons had peaked IID tuning curves. Such neurons may be integrating information across frequency channels, but ITD information may be carried on only one of those channels. Alternatively, ITD information at various frequencies might be summed by these neurons which then lack the specific inhibition necessary to suppress the ITD side peaks. Fujita and Konishi (1991) showed that blocking GABAergic inhibition in ICx resulted in ICx neurons responding to noise in a phase ambiguous manner. This type of neural processing may also explain those phase ambiguous neurons in N.Ov which have the troughs in their ITD curves well above the spontaneous activity levels. Three other neurons isolated in N.Ov had ambiguous (monotonic) IID tuning curves in response to noise despite having very broad frequency tuning curves (figure 2.12). Two of these three had flat ITD tuning curves, while the third had a phase-ambiguous

Figure 2.11 Response tuning curves from an N.Ov neuron which displayed phase-ambiguous ITD tuning in response to white noise despite having a broad frequency tuning curve. The IID tuning is specific (peaked) as is usual for cells in the inferior colliculus which integrate sound localization cue information over several frequencies.

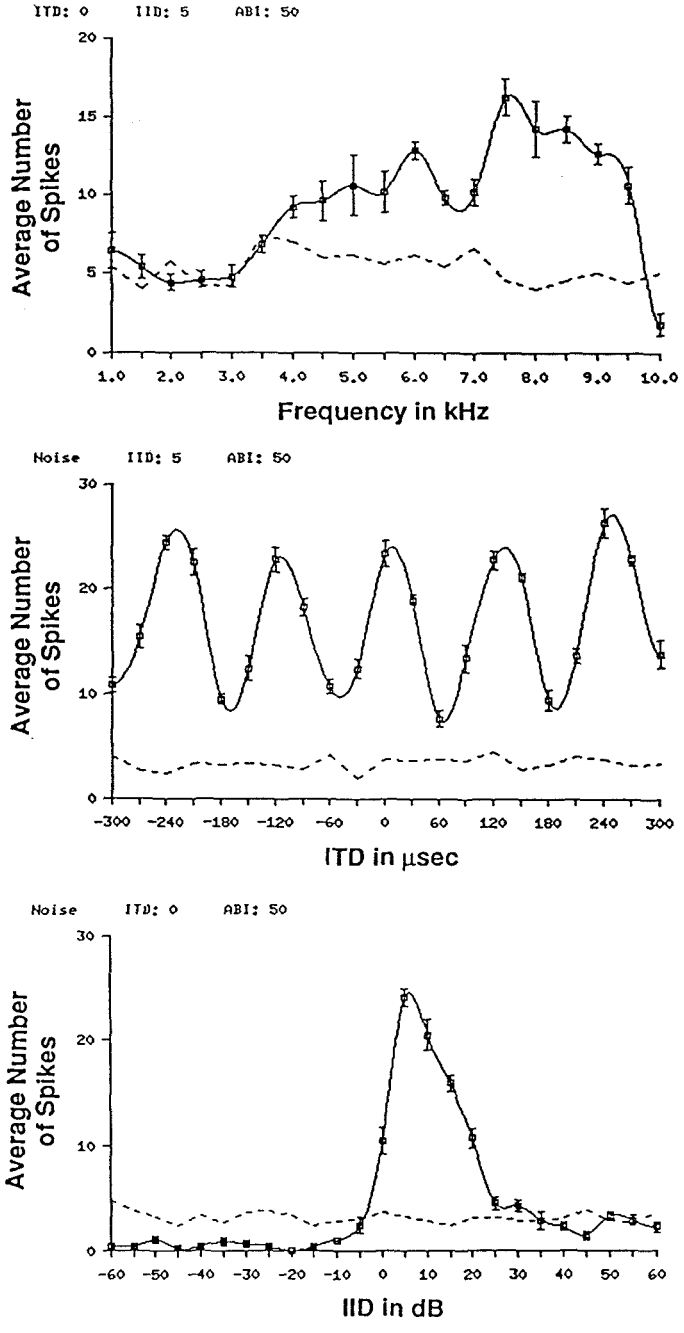
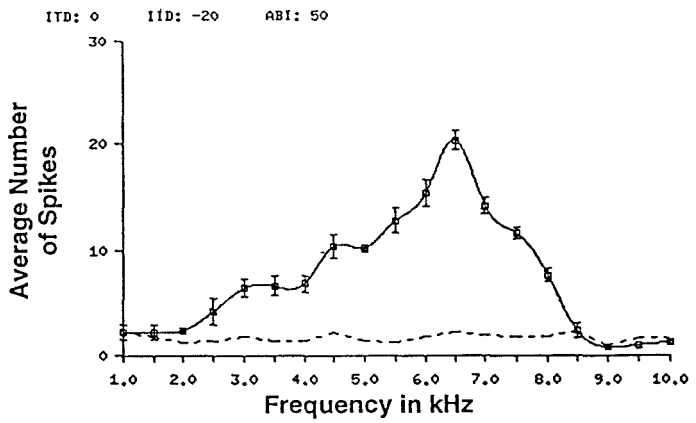
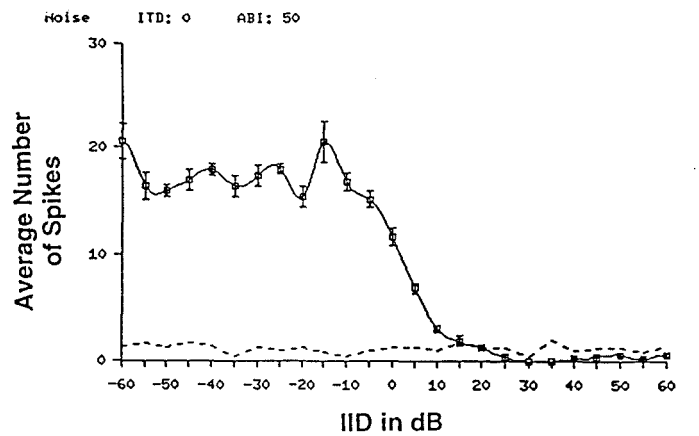
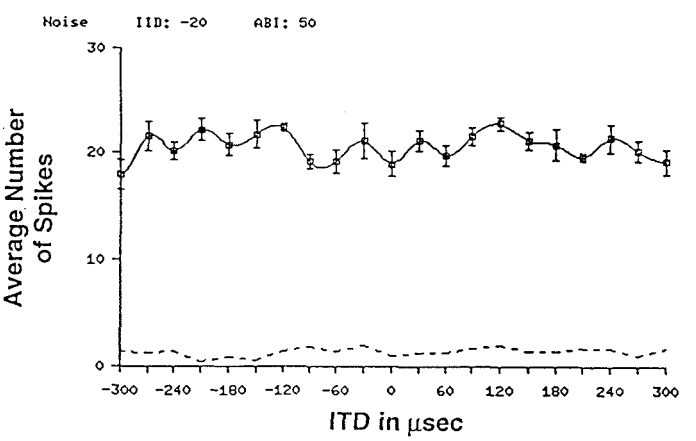

ITD: 0 IID: 5 ABI: 50
Average Number of Spikes
Frequency in kHz
Noise IID: 5 ABI: 50
Average Number of Spikes
ITD in µsec
Noise ITD: 0 ABI: 50
Average Number of Spikes
IID in dB

Figure 2.12 Response tuning curves from an N.Ov neuron which displayed ambiguous IID tuning in response to white noise despite having a broad frequency tuning curve. This particular cell also showed no modulation in its activity level in response to varying interaural time differences.


ITD: 0 IID: -20 ABI: 50
Average Number of Spikes
30
20
10
0
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
Frequency in kHz


Noise ITD: 0 ABI: 50
Average Number of Spikes
30
20
10
0
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60
IID in dB

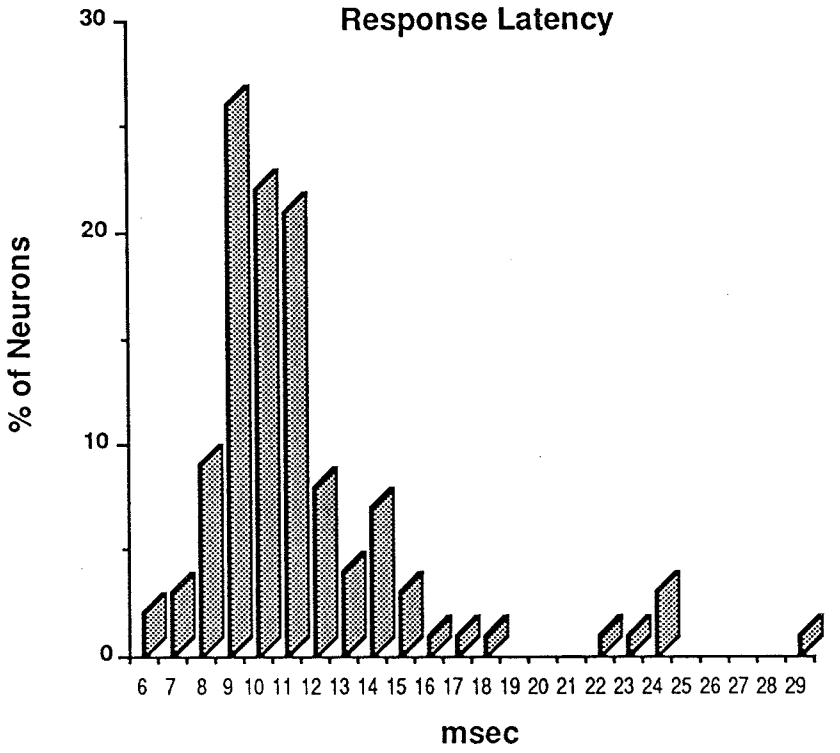

Noise IID: -20 ABI: 50
Average Number of Spikes
30
20
10
0
-300 -240 -180 -120 -60 0 60 120 180 240 300
ITD in μsec

response to ITD when stimulated with noise. Such neurons must be receiving very similar (ambiguous) localization cue coding from across a wide range of frequencies.

Latency

The latency of isolated neurons in N.Ov was measured at the neuron's best (or one of the best) IID and ITD with noise as the stimulus. It was readily apparent from a neuron's raster plot that the response latency varied with tone frequency as well as with IID and ITD. Figure 2.11 shows that the response latency histogram contains one large peak centered around 10 msec, which is consistent with neurons located one to two synapses from ICc. There are a considerable number of neurons with much longer response latencies, however. Some of these are attributable to neurons with delayed onset responses, but a relatively rapid return to spontaneous activity levels when the stimulus ends (pauser neurons). The remainder may indicate neurons which are integrating descending information from the forebrain. Neurons which have response latencies of 6-8 msec could well be the targets of neurons located in LLv.

Figure 2.13 Distribution of response latencies measured from neurons in N.Ov in response to white noise stimulation at the best (or one of the best) values of ITD and IID.


Response Latency
% of Neurons
30
20
10
0
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
msec

Chapter 3: Anatomy and Connectivity of Nucleus Ovoidalis in the Barn Owl

Methods

Normal Histology. Following physiological experiments, owls were overdosed with pentobarbital (8 ml i.m. Nembutal, Abbot Laboratories, 50 mg/ml) and perfused through the heart with 0.1 M phosphate buffer (pH 7.4) followed by a solution of 1% paraformaldehyde and 1.25% glutaraldehyde (in 0.1 M phosphate buffer, pH 7.4). The brains were blocked stereotaxically in a transverse plane parallel to that of the electrode penetrations, removed from the skull and sunk in 30% sucrose for cryoprotection. A sliding microtome was used to cut frozen sections 30 μm thick which were then cleared, dehydrated, stained with cresyl violet and examined for electrode tracks and fiducial marks.

Tracer Studies. Free HRP. A 2.5% solution of HRP in 0.9% NaCl was iontophoresed into nucleus ovoidalis in five owls. The target was physiologically identified and then a non-capillary glass electrode with a tip of approximately 10 μm o.d. was filled with the HRP solution and lowered into the nucleus. HRP was iontophoresed with 7-10 μA positive DC current pulsed 7 sec on/7 sec off for 15-30 minutes. Following iontophoresis the electrode was held in place for an additional 30 minutes before being withdrawn. After a five day survival period, owls were overdosed with pentobarbital, perfused through the heart with 0.1 M phosphate buffer (pH 7.4) followed by a cold (15° C) solution of 1 % paraformaldehyde and 1.25% glutaraldehyde (in 0.1 M phosphate buffer, pH 7.4). Following fixation the owl was perfused with a cold 10% sucrose solution and then a cold 20% sucrose solution. The brain was blocked, removed, sunk in cold 30% sucrose (0.025 M PB) and cut into 30 μm sections on a freezing microtome. Sections were rinsed in cold acetate buffer (pH 3.3), incubated with tetramethylbenzidine (Mesulam, 1978), reacted with 0.3% H_2O_2, rinsed three

times with acetate buffer, mounted and counterstained with neutral red. Dextran-Amines. Biotinylated (M.W. 10,000) or fluorescent (M.W. 3000) dextran-amines were iontophoresed into the central nucleus of the inferior colliculus (ICc) or N.Ov. A solution of 10% dextran-amine in 0.2 M KCl with 0.1% Triton X-100 was loaded into a capillary glass electrode with a tip size of 7-10 μm, which was then lowered into the physiologically identified target. Positive, pulsed (7 sec on/7 sec off) DC current 5-8 μA was used to electrophorese the tracer for a period of 30 minutes. Following electrophoresis the electrode was held in place for an additional 30 minutes. After a survival period of two to eight days (depending on the M.W. of the tracer and the estimated distance of transport), the owls were exsanguinated with 0.1 M PB and perfused with a fixative solution of 4% paraformaldehyde, 0.1 M lysine and 0.01 M Na-periodate in 0.1 M PB (pH 7.4). Brains were blocked, removed, post-fixed for one day and sunk in 30% sucrose, 10% PLP fix in 0.1 M PB. Sections were cut on the freezing microtome into 0.1 M PB. For fluorescent dextran-amine injections, sections were mounted, cleared, coverslipped and viewed with a fluorescent microscope with the proper filters. Sections containing biotinylated dextran-amine (BDA) were processed with a Vectastain ABC Elite kit (Vector Laboratories) (Leonardus Veenman et al., 1992), incubated with a nickel/cobalt/DAB solution and reacted with 0.3% H_2O_2. Sections were then mounted and counterstained with neutral red.

Results

Figure 3.1 is a Nissl stained transverse section through the diencephalon which shows the relationship of nucleus ovoidalis to other diencephalic structures. N.Ov is a large, distinct cell group encapsulated by fibers and lying lateral to the third ventricle and medial to the dorsolateral thalamic nucleus (DL). In the barn owl, N.Ov is approximately 1.2 mm in diameter, stereotaxically located roughly 2.0 mm anterior and 2.0 mm lateral of the reference pin. The afferent fiber bundle from the inferior colliculus, the tractus nucleus ovoidalis (TOv), enters the ventral aspect of the nucleus from a ventrolateral projection, passing medial to the nucleus rotundus (Rot) and extending down along the lateral border of the lateral hypothalamus. The nucleus subrotundus is a narrow layer of cell bodies just ventromedial to N.Ov which separates ovoidalis from the occipitomesencephalic fiber tract. Figure 3.2 shows higher power photomicrographs of coronal sections through nucleus ovoidalis at three rostrocaudal positions. At the caudal levels, neurons of average size are tightly packed ventrally but become more widely separated and interspersed with smaller neurons dorsally. At middle and more rostral levels, the large cell bodies tend to be located in a fairly distinct region of the lateral nucleus, running from the dorsal border down into the ventral aspect of the nucleus and into the TOv. In the central and medial portions of the nucleus, the cell bodies tend to be smaller and oriented to follow the convex curve of the nucleus. Anteriorly, the efferents exit nucleus ovoidalis in a large fiber bundle lateral to the nucleus, which Karten (1968) called the fasciculus prosencephali lateralis (FPL). At the rostral end of N.Ov, cell bodies from the nucleus extend laterally to form a narrow cap over the dorsal aspect of the efferent fiber bundle, giving the nucleus the shape of an inverted comma. The neuronal cell bodies at these levels are of the larger

Figure 3.1 Low power photomicrograph of the left side of a Nissl-stained transverse section through the diencephalon at the level of nucleus ovoidalis. Dorsal is towards the top and lateral is to the left.

Abbreviations:

DL - nucleus dorsalis lateralis thalami

DM - nucleus dorsalis medialis thalami

N.Ov - nucleus ovoidalis

OM - occipitomesencephalic fiber tract

OT - optic tectum

PV - nucleus posteroventralis

Rot - nucleus rotundus

SGC - stratum griseum centrale

T - nucleus triangularis

TIO - tractus isthmo-opticus

TOv - tractus nucleus ovoidalis

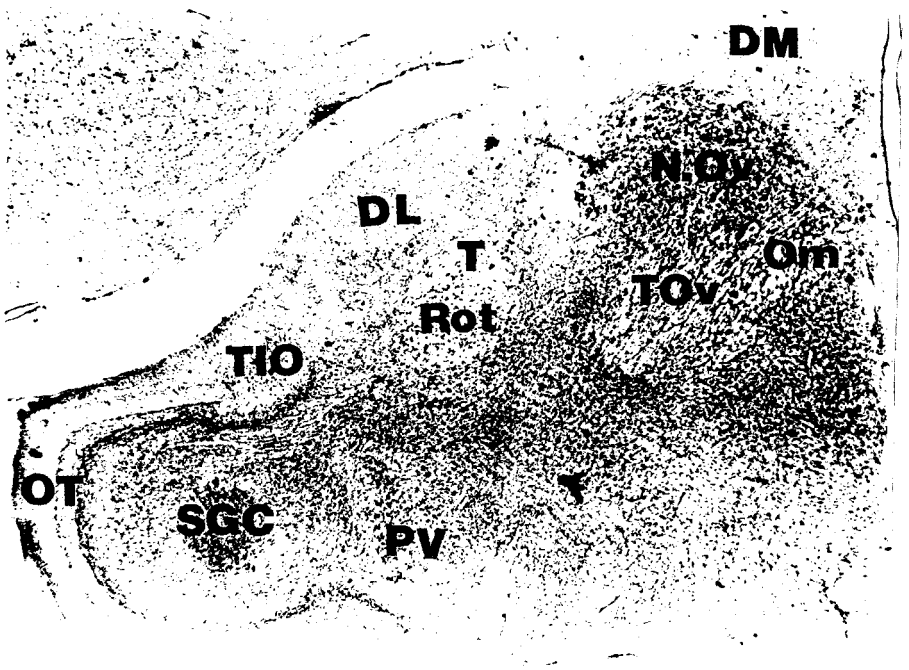

DM
N.Dv
DL
T
Om
TOv
Rot
TIO
OT
SGC
PV

Figure 3.2 A-C: Normal Nissl-stained transverse sections through the left nucleus ovoidalis and corresponding camera lucida drawings identifying the visible structures. A is rostral, C is caudal and B is roughly half-way between A and C. Note particularly the lateral extension of cell bodies dorsally and the large cell bodies imbedded in the tractus nucleus ovoidalis ventrally in A, the subdivisions of the nucleus based on cell size and orientation in B, and the gradient of cell density across the dorsovental extent of the nucleus in C.

Abbreviations:

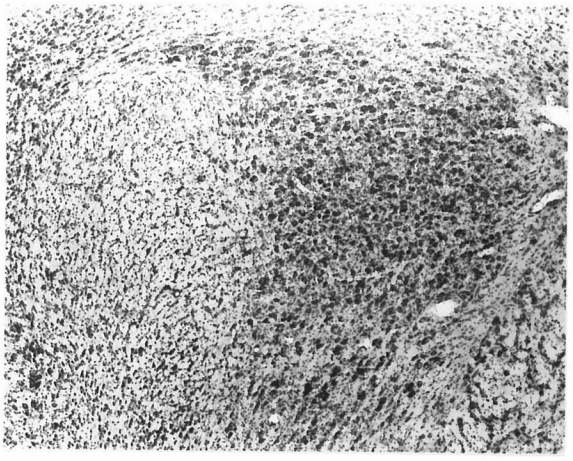
FPL - fasciculus prosencephali lateralis

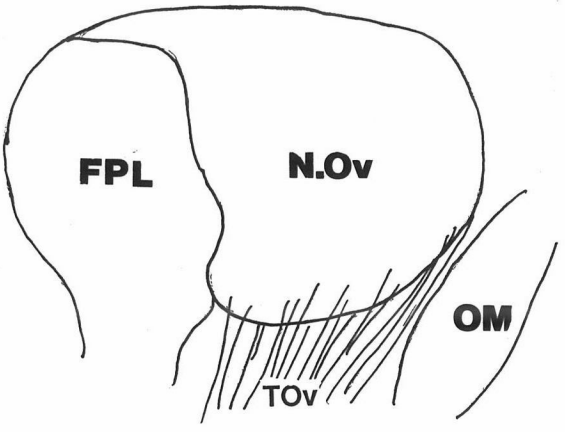
OM - occipitomesencephalic fiber tract

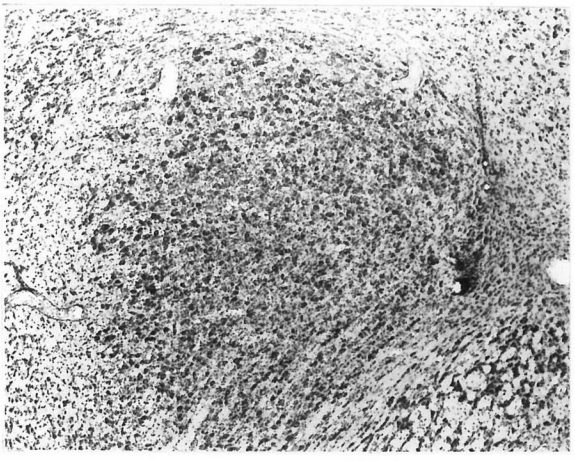
N.Ov - nucleus ovoidalis

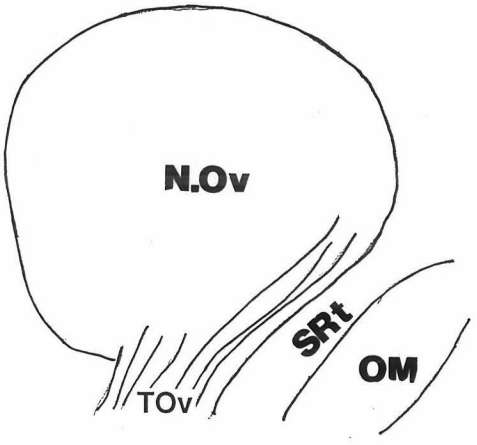
SRt - nucleus subrotundus

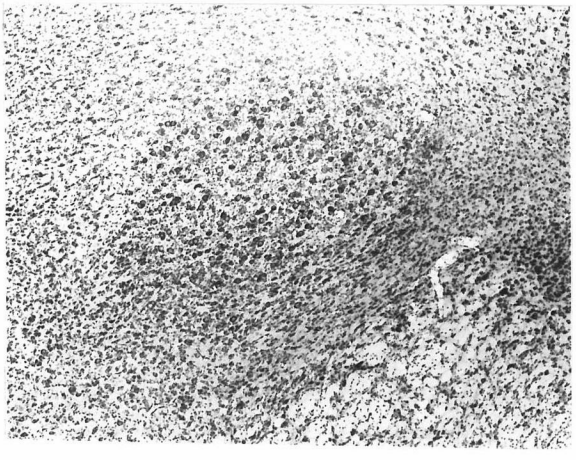
TOv - tractus nucleus ovoidalis

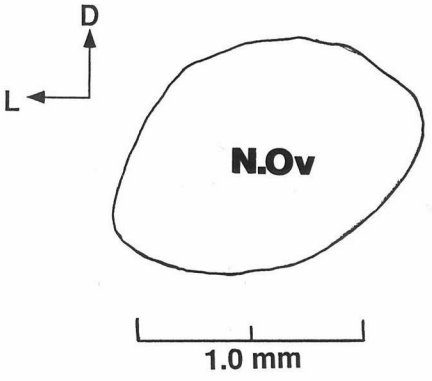



FPL
N.Ov
OM
TOv




N.Ov
SRt
OM
TOv




D
L
N.Ov
1.0 mm

variety and again are densely packed ventromedially and more spread out dorsolaterally. In the pigeon (Wild, 1987; Wild et al., 1993) and ring dove (Durand et al., 1992) nucleus ovoidalis contains a subnucleus called the nucleus semilunaris parovoidalis (SPO) which is readily apparent in normal, Nissl-stained transverse sections. In these species, SPO stands out as the medial two-thirds of this region are separated from N.Ov by a fibrous lamina running horizontally. The lateral portion of SPO extends through the axonal fibers of TOv. In the barn owl, no such ventromedial subdivision corresponding to SPO is discernible in Nissl-stained sections through ovoidalis. There are, however, a set of large cell bodies imbedded in the TOv just ventral to the nucleus at rostral levels.

Connectivity

Projections from the midbrain.

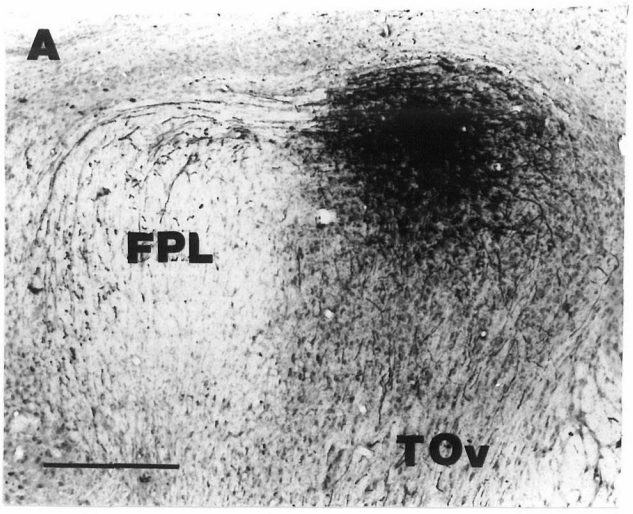
In order to determine the pattern of input to nucleus ovoidalis from the inferior colliculus, a number of small retrograde tracer injections were made in various locations throughout N.Ov. Figure 3.3 shows the results from a free HRP injection placed anteriorly in the dorsolateral portion of the nucleus. Retrogradely labeled cell bodies were sparsely located across the dorsoventral extent of all three subdivisions of the ipsilateral central nucleus of the inferior colliculus. Labeled neurons were evident across all frequency laminae of ICc. The distribution of these labeled cells was so sparse that adequate visualization of these HRP filled neurons required the highly sensitive TMB development technique (Mesulam, 1978) and sections had to be left uncounterstained in order to avoid obscuring the cells containing tracer (figure 3.3c). Labeled neurons were also found in the contralateral ICc, with the same sparse distribution though far fewer cells were labeled. Localized HRP injections placed caudally, medially

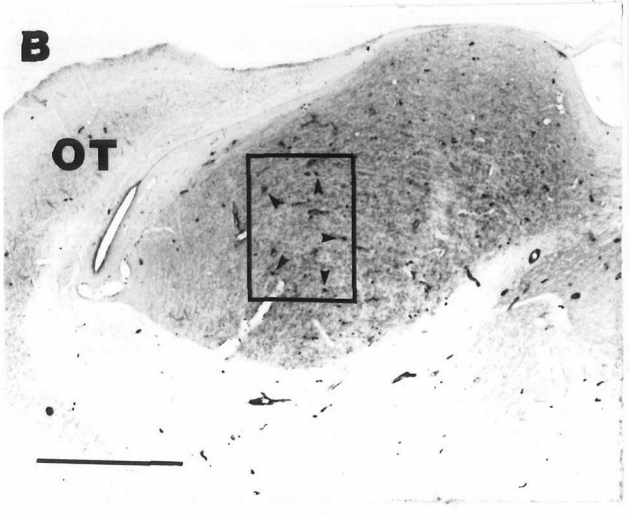
Figure 3.3 Pattern of retrograde labeled neurons in the inferior colliculus following tracer injection in N.Ov. Dorsal is up and lateral is to the left in all photographs.

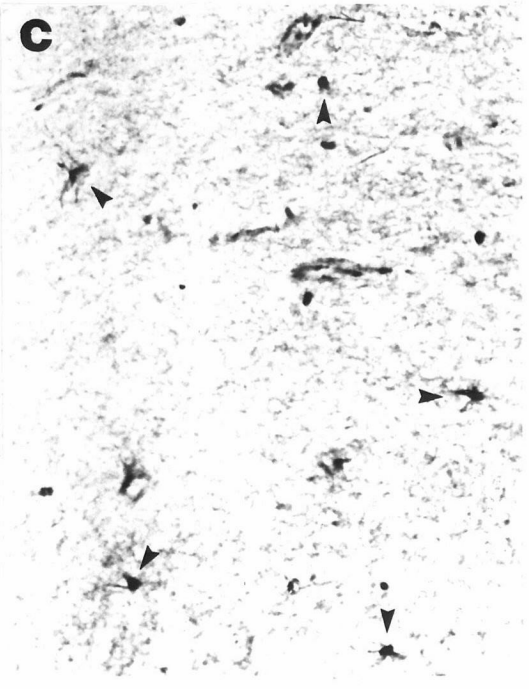
A) Free HRP injection site in the dorsolateral portion of the rostral end of nucleus ovoidalis. Filled axons can be seen exiting the nucleus anterogradely in the FPL and retrogradely in the TOv. Scale bar = 500 μm.

B) Low power photomicrograph of the inferior colliculus ipsilateral to the injection site in N.Ov. HRP fillled neurons are scattered widely across the mediolateral and dorsoventral extent of the nucleus. Arrows within the box point to neurons seen more clearly in (C). OT - optic tectum. Scale bar = 1 mm.

C) High power photomicrograph of the area contained within the box in (B). Five retrogradely labeled neurons can be discerned (arrows - the topmost neuron has its neuritic extensions located out of the plane of focus). The smallest distance between the labeled neurons is well over 100 μm.


A
FPL
TOv


B
OT


C

and centrally yielded similar results. Figure 3.4 is a camera lucida drawing of the distribution of cell bodies in the inferior colliculus retrogradely labeled from an injection of biotinylated dextran-amine in the dorsomedial portion of the ipsilateral N.Ov. The number of neurons containing tracer in ICc is quite low. Labeled somata were located in the lateral, central and medial portions of ICc, though the medial aspect contained the most such cells. An injection of fluorescein-conjugated dextran-amine was placed in the dorsolateral portion of N.Ov (approximately at the midpoint of the rostrocaudal extent of the nucleus) together with an injection of rhodamine-conjugated dextran-amine placed approximately 500 μm more ventral. Both red and yellow fluorescent cell bodies were sparsely distributed across ICc, but only two cells were double labeled. An injection of BDA into the ventrolateral portion of N.Ov labeled a small population of cells in ipsilateral (but not contralateral) LLv in addition to the widespread pattern of neurons in ICc (figure 3.5).

Based upon the results of the retrograde labeling experiments, I expected that anterograde tracers placed into ICc would yield a diffuse pattern of tracer filled axon terminals throughout nucleus ovoidalis. This, however, was not the case. Figure 3.6 shows the results of BDA injected into the 4.5 kHz region of ICc physiologically identified as lateral shell. In caudal and middle sections of the ipsilateral N.Ov, the anterogradely labeled terminals were largely restricted to the lateral aspect of the nucleus, corresponding roughly to the region with constant frequency in a dorsoventral electrode penetration and the region which contains large cell bodies oriented vertically. At the rostral end of N.Ov, the pattern of labeled terminals moved dorsally to form a cap on the nucleus, though here too the heaviest label was located more laterally. The central and medial portions of the nucleus were not devoid of labeled axon terminals (particularly noticeable in

Figure 3.4 Camera lucida drawings of transverse sections containing a BDA injection site in N.Ov (topmost drawing) and retrogradely labeled cell bodies in the ipsilateral inferior colliuculus. Dorsal is towards the top and lateral is to the left. The numbers to the upper left of the sections indicate the normalized anterior-posterior position of the section within the nucleus. The stipples lines medial and ventral to ICc in sections 0.35-0.475 indicate the pathway of efferent fibers projecting to N.Ov which contained tracer filled axons. The rostrocaudal extent of labeled somata is seen to cover roughly the second quarter of the inferior colliculus. Scale bars = 1 mm.

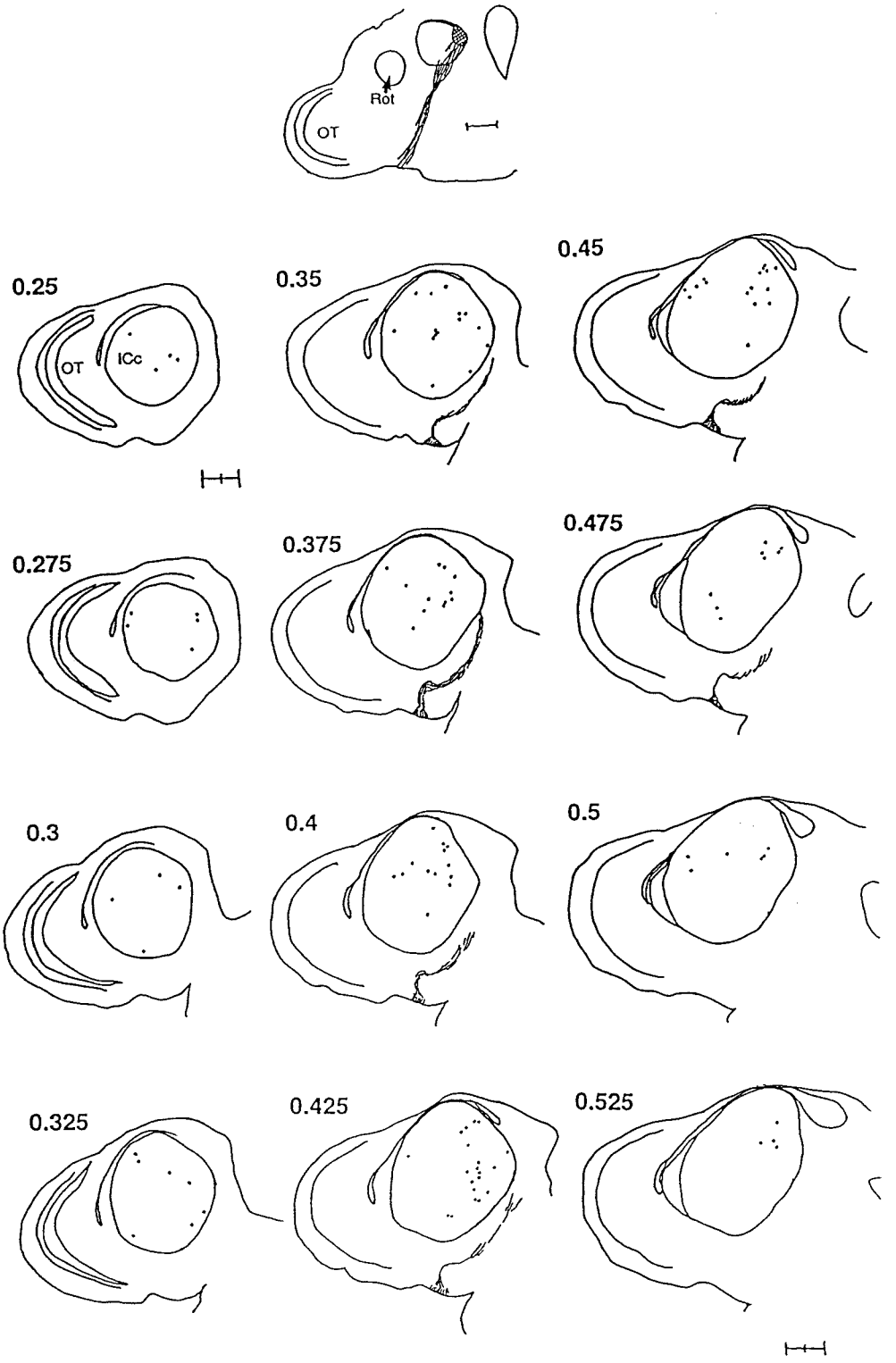

Rot
OT
0.25
OT
ICc
0.35
0.45
0.275
0.375
0.475
0.3
0.4
0.5
0.325
0.425
0.525

Figure 3.5 Neurons in LLv which were retrogradely labeled by a BDA injection in ventrolateral N.Ov. No labeled somata were observed in the contralateral LLv.

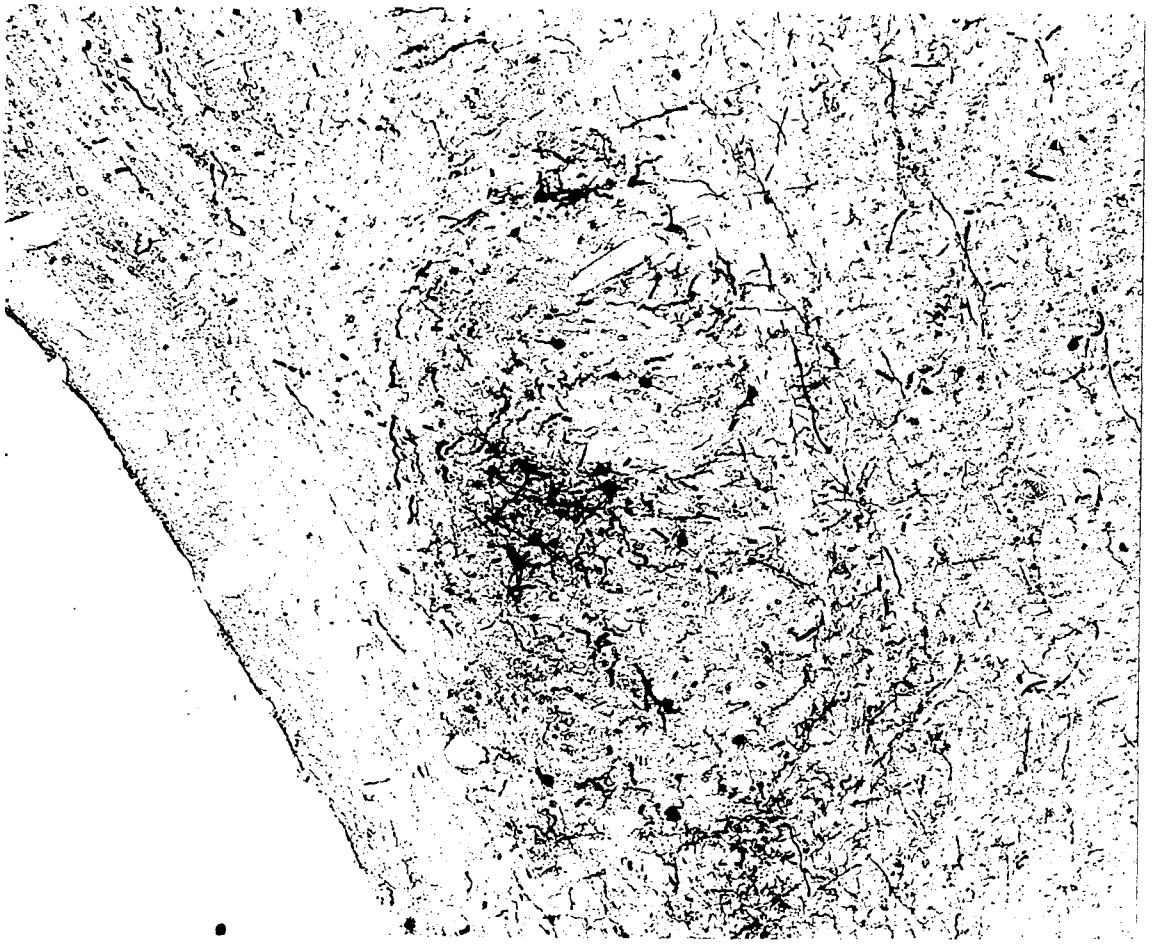


Figure 3.6 Anterograde projection pattern from ICc lateral shell to N.Ov.

A) BDA injection site in the 4.5 kHz region of the left ICc lateral shell. Labeled efferents course laterally towards their targets in ipsilateral ICx, and medially to join the fiber tract which becomes the TOv at more rostral levels. In all sections, dorsal is up and lateral is to the left. Scale bar = 1mm.

B) Transverse section through caudal ipsilateral N.Ov showing labeled axon terminals along the lateral aspect of the nucleus, arcing medially along the dorsal edge. Neutral red counterstain. Scale bar = 500 μm.

C) Neutral red counterstained transverse section through the middle of N.Ov ipsilateral to the injection site. Dense, labeled efferent terminations are confined mostly to the lateral aspect of the nucleus though some labeled axon terminals are found more medially. Particularly evident in this section is a small, dense plexus of terminations almost exactly in the center of the nucleus. Scale bar = 500 μm.

D) Rostral transverse section through N.Ov ipsilateral to the BDA injection site. Labeled efferent terminals are now mostly confined to the dorsal aspect of the nucleus, although the amount of label medial is still smaller than that located laterally. Labeled fibers can be seen coursing up through the nucleus from the TOv. Some fibers make occasional terminations in the ventral portions of the nucleus. Neutral red counterstain. Scale bar = 500 μm.

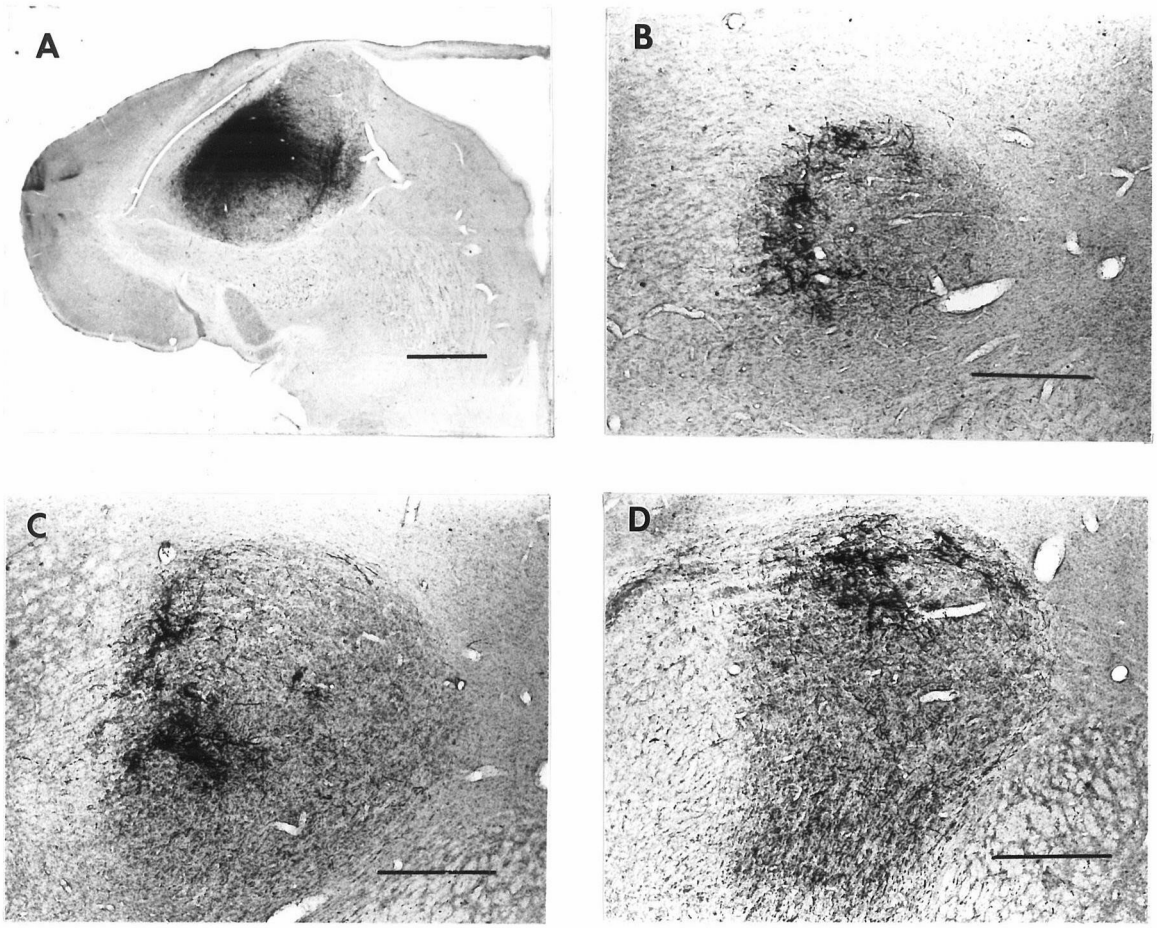

A
B
C
D

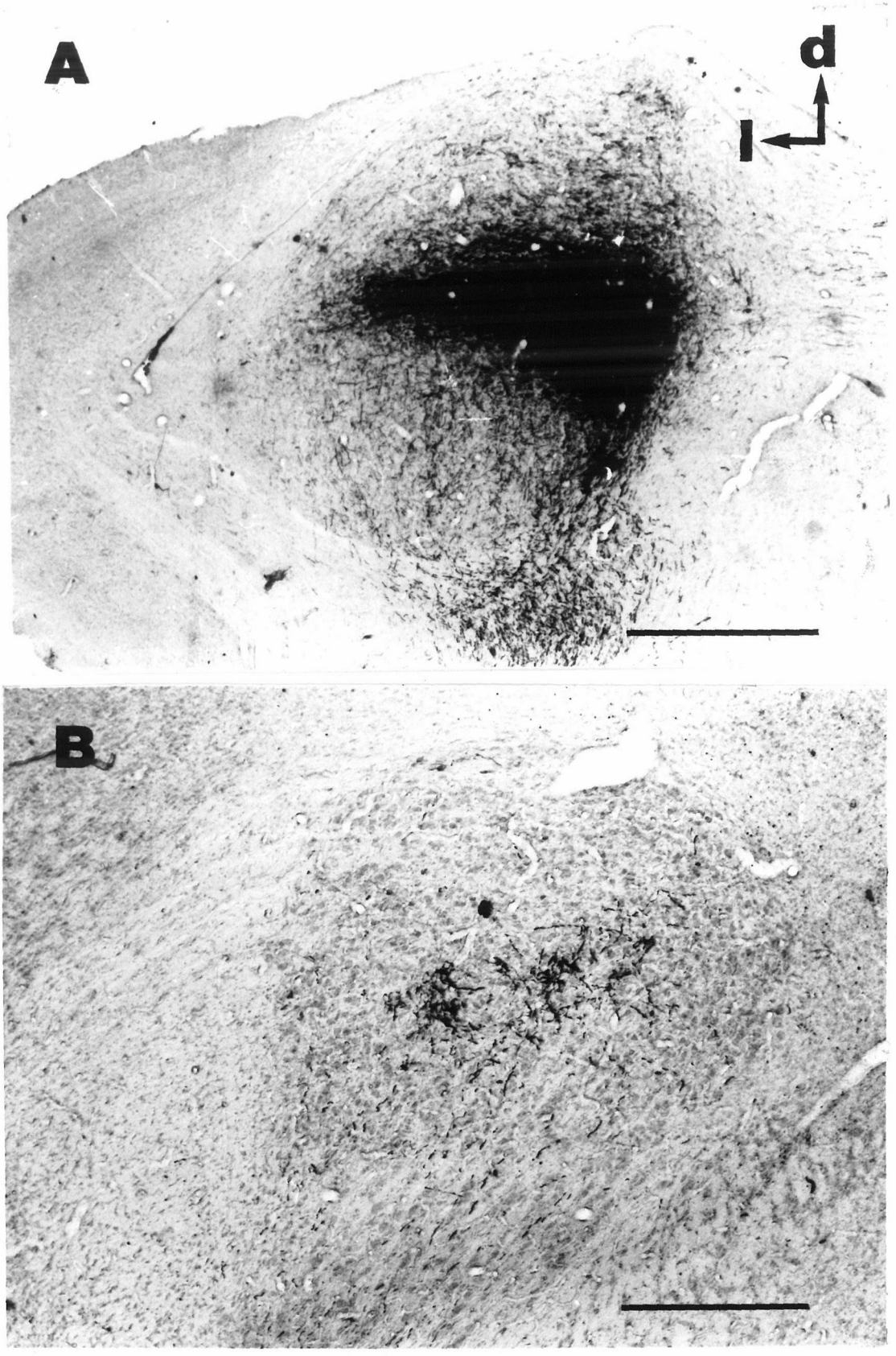
figure 3.6c), though such excursions were quite small compared to the main pattern of efferent terminations. The contralateral N.Ov contained a mirror image of the staining pattern, though the amount of label was much smaller. A similar BDA injection placed in the 3.5 kHz region of physiologically identified ICc core is displayed in figure 3.7a. Though no IID tuning was obtained at the site of the injection, some tracer leaked into the medial shell subdivision from this injection as evidenced by retrogradely labeled cell bodies in contralateral nucleus angularis and contralateral VLVp (as well as contralateral nucleus laminaris). The anterograde projection pattern was largely contained within the central and medial portion of ipsilateral nucleus ovoidalis (figure 3.7b), with the axons terminals having a horizontal staining pattern rather than the predominantly vertical orientation as in the case of the projection from lateral shell. Again, the contralateral N.Ov contained a nearly identical pattern of anterograde label with a lower density.

The patterns of retrogradely labeled cell bodies in ICc resulting from tracer injections in N.Ov and the patterns of anterogradely labeled axon terminals in N.Ov arising from tracer injections in ICc at first appeared to be contradictory. However, these results are consistent with the hypothesis that only a small subpopulation of the neurons in the inferior colliculus project to ovoidalis. These thalamic projection neurons diverge considerably upon reaching their target nucleus in a manner which depends on their location within ICc. Efferents from the lateral shell subdivision of ICc make considerable axon collateral terminations along the dorsoventral and rostrocaudal axes of N.Ov, particularly along the lateral aspect. Axons arriving from the core and medial shell subdivisions of ICc diverge in the mediolateral and rostrocaudal axes of the auditory thalamus, with

Figure 3.7 Anterograde projection pattern from ICc core/medial shell to nucleus ovoidalis.

A) BDA injection site into the 3.5 kHz region of ICc physiologically characterized as core (phase-ambiguous ITD tuning with no IID tuning). Leakage of some tracer into medial shell was discovered by the appearance of retrogradely labeled cell bodies in the contralateral nucleus angularis. Neutral red counterstain. Scale bar = 1 mm.

B) Tracer filled efferent terminations in the central portion of N.Ov. The dorsoventral location and limit of the axon terminals corresponds well with the tonotopic organization of N.Ov and ICc (which have their frequency representations inverted from one another). In contrast to the more vertical orientation of the efferent projections from lateral shell (figure 3.6), the anterograde projection pattern from core/medial shell is oriented horizontally.


A
d
l
B

most of their terminations occurring in the central and medial portions of the nucleus.

Comparison of the pattern and density of the anterograde label obtained from tracer injections in the subdivisions of ICc, along with careful examination of the pattern of retrogradely labeled cell bodies in ICc from several tracer injections in various portions of N.Ov, strongly suggest that lateral shell makes the largest contribution of efferents to the auditory thalamus. The medial shell subdivison of ICc makes the second largest contribution of inputs while ICc core sends the smallest number of axon terminals to N.Ov. Furthermore, the anatomical segregation of sound localization cue processing which occurs in ICc is somewhat maintained at the level of the thalamus; lateral shell projects most heavily to lateral ovoidalis, and core and medial shell project most heavily to the central and medial portion of the nucleus. However, it is also clear that this segregation is not strictly maintained as the lateral shell does provide efferent terminations in the central and medial portions of N.Ov and the core and medial shell do send a small number of axonal terminals to the lateral portion of ovoidalis. These projections patterns demonstrate the possible convergence of efferents from different regions of the inferior colliculus (and LLV) and thereby provide the anatomical support for the (sometimes unique) combinations of physiological response types found in nucleus ovoidalis.

Of particular interest with respect to the observation of space-specific neurons in N.Ov, as well as the findings of Knudsen and Knudsen (1993) concerning parallel pathways for sound localization, is whether or not neurons of the auditory space maps in either the optic tectum or ICx project to N.Ov. No retrogradely labeled cell bodies were ever observed in the optic tectum for any tracer injected into ovoidalis. In this study, no retrogradely labeled neurons were

unequivocally localized to ICx, though this observation is subject to considerable interpretation. The border between ICx and the lateral shell of ICc is not clearly defined. Histochemically, the subdivisions of ICc are visualized by differential staining with either antibodies to vitamin D-dependent calcium binding protein (Takahashi et al., 1987), or acetycholinesterase (Adolphs, 1993a). Visualization of the retrogradely labeled cell bodies in ICc precluded counterstaining the tissue sections with either of these agents. Furthermore, the pattern of retrogradely labeled neurons varied between sections making the use of stained alternate sections of dubious value. For these reasons, the unequivocal assignment of retrogradely labeled neurons to particular subnuclei near the borders of the subdivisions of ICc or near the border of lateral shell and ICx was not possible. However, while the projection of an extremely small number of neurons along the medial aspect of ICx to N.Ov cannot be ruled out with certainty, it is quite clear that the auditory space map as a whole does not project to the auditory thalamus. No retrogradely labeled cell bodies were ever located in the central or lateral portions of ICx or in the most anterior sections of the inferior colliculus where the extent of ICx is most predominant. The use of anterograde tracer injections in ICx for demonstrating a projection to N.Ov is of questionable value, since most tracers currently available for anterograde use are in fact bidirectional (PHAL, BDA, HRP, cholera toxin B subunit) and neurons in lateral shell could pick up the tracer retrogradely and have axon collaterals projecting to N.Ov filled, thereby clouding the interpretation of the results. Tritiated proline is a strictly anterograde tracer which could be used, but since the results of this study indicate that any projection of ICx to N.Ov is likely to be extremely small, the probability of detecting such a projection autoradiographically is also small (i.e., a negative result would not assure the absence of such a projection).

Ascending Projections to the Telencephalon

Biotinylated dextran-amine injections were placed in the dorsomedial and ventrolateral portions of nucleus ovoidalis in order to determine the pattern of projections to the auditory forebrain area, Field L. Figure 3.8 shows the results of the BDA injection into the dorsomedial area of N.Ov. Labeled axonal fibers could be seen leaving N.Ov laterally in the FPL. In the telencephalon, tracer filled fibers passed through the paleostriatum primitivum and paleostriatum augmentatum and crossed the lamina medullaris dorsalis which forms the boundary between paleostriatum and neostriatum. In the neostriatum, stained axon terminals were located most densely in the medial most aspect of Field L2a, with a few fibers coursing dorsally towards the overlying hyperstriatum ventrale (HV) (figure 3.8c). The topography of this projection is apparently in accord with the tonotopic organization of Field L, in which high frequencies are represented ventromedially with low frequencies and broadband responses being found dorsolaterally (Wild et al., 1993).

Efferents labeled by the BDA injection in the ventrolateral portion of N.Ov followed the same route to the neostriatum through the paleostriatum, but terminated most densely in the caudolateral portion of Field L2. A smaller number of axonal coursed medially in Field L, making relatively few terminations across the extent of the area medial to the dense fiber terminations located laterally (figure 3.9). The location of heaviest anterograde staining is consistent with the tonotopic organizations of Field L and nucleus ovoidalis. The sparse amount of labeled axons across the rest of Field L may indicate that a small amount of tracer leaked into the efferent fiber bundle exiting N.Ov laterally. Alternatively, this pattern of staining in the forebrain may indicate that the lateral portion of N.Ov provides a more widespread input to Field L than the medial

portion of N.Ov. My colleague S.E. Roian Egnor placed a small BDA injection into Field L2 at a position approximately 1 mm lateral of the location of stained afferents displayed in figure 3.8 (considerably medial to the most densely labeled fibers in figure 3.9) as well as slightly caudal to it. Of the neurons which were retrogradely labeled in N.Ov, the highest percentage appeared in the lateral portion of the nucleus (personal observation and interpretation of data collected by S.E.R. Egnor).

Figure 3.8 Nucleus ovoidalis projects to the telencephalic area Field L.

A) BDA injection site in N.Ov (same as that drawn in figure 3.4). Neutral red counterstain. Scale bar = 500 μm.

B) Brightfield photomicrograph of tracer filled axon terminals in the medial most portion of Field L2a. Scale bar = 500 μm.

C) High power darkfield photomicrograph of the area outlined by the box in (B) showing axon terminals with synaptic swellings more clearly. Notice the fibers coursing dorsally towards HV in the upper righthand corner.

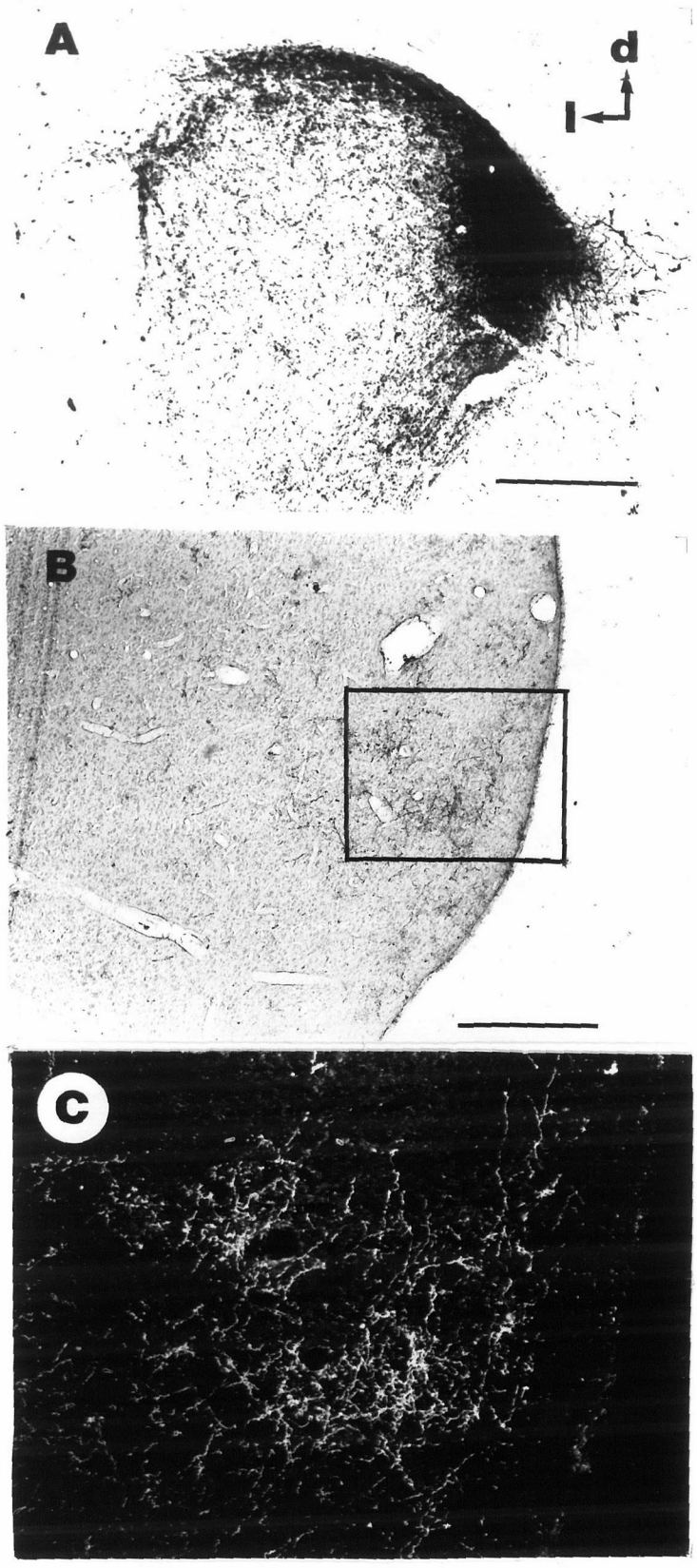
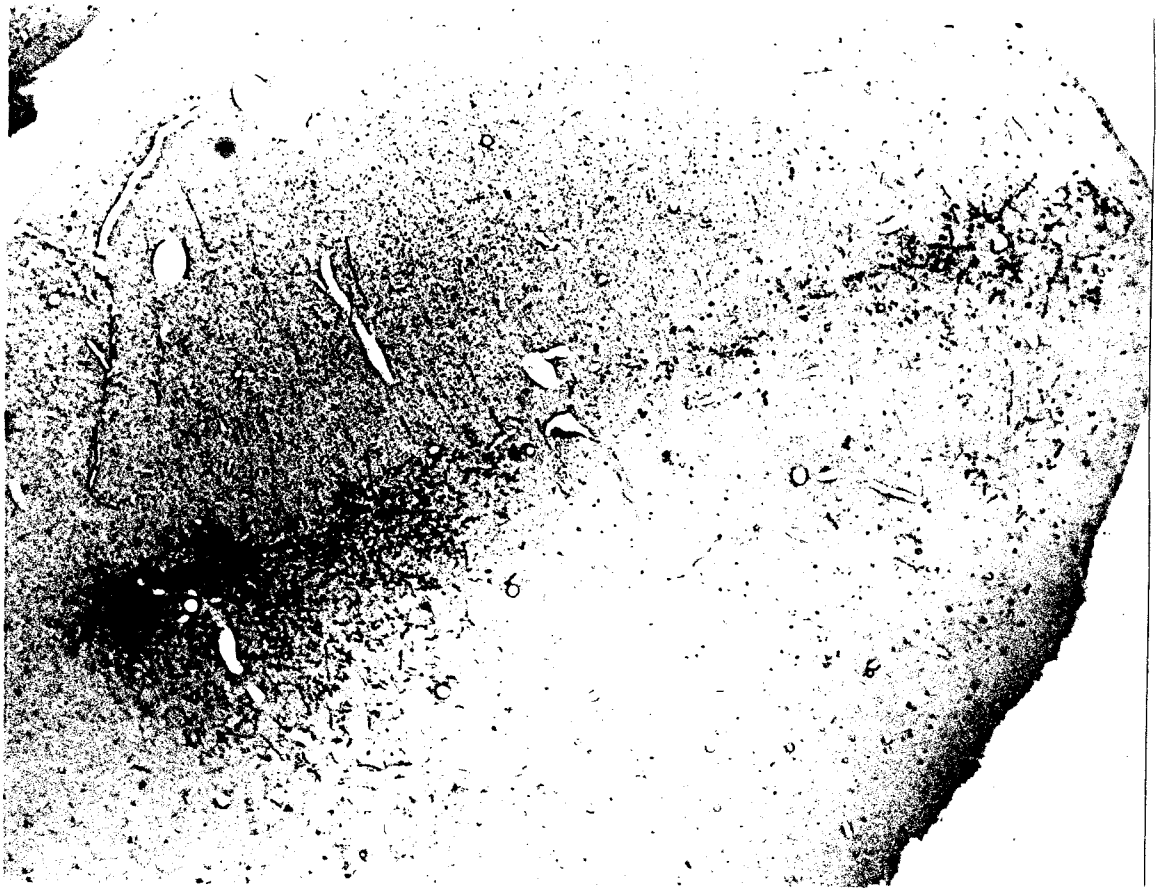

A
d
l
B
C

Figure 3.9 Anterograde label in caudal Field L resulting from a BDA injection in ventrolateral N.Ov. Dense label is located laterally, but a small amount of label is present across the mediolateral extent of the area. In this section the medial staining is most apparent near the midline.



Chapter 4: Summary and Discussion

"What really makes science grow is new ideas, including false ideas."

-Karl Popper

This work was intended to examine the role of the auditory thalamus, nucleus ovoidalis, in processing auditory information, with particular attention to sound localization cues. All auditory input to the forebrain is received via nucleus ovoidalis (Karten, 1968). Owls which have the auditory space map in the optic tectum removed or inactivated have a diminished capacity for sound localization, rather than the complete loss of localizing ability. The inactivation of both the optic tectum and nucleus ovoidalis results in owls which are no longer able to localize sound sources (Knudsen and Knudsen, 1993). Additionally, deficits in head orienting behavior to free field sounds resulting from restricted lesions in ICx usually disappear over a time course of several hours (Wagner, 1993). These observations indicate that the thalamo-telencephalic auditory pathway is capable of subserving at least some sound localization ability.

The results of this work demonstrate that neurons in the auditory thalamus of the barn owl respond to auditory stimulation, are organized topographically with respect to sound frequency, and respond in a tuned manner to at least one sound localization cue. All neurons investigated in nucleus ovoidalis responded to stimulation with white noise. The nucleus contains at least two parts based on tonotopic organization: the centromedial portion of the nucleus has high frequencies mapped dorsally and low frequencies located ventrally (figure 2.1) while neurons on the lateral aspect of the nucleus have the same best frequency. The frequency tuning of most neurons in N.Ov is rather narrow, though a significant number have more than one peak in their frequency tuning curves (figure 2.2).

Although all ovoidalis neurons responded to at least one sound localization cue, there was no apparent systematic mapping of sound localization parameters in the nucleus. Furthermore, in contrast to the organization of the central nucleus of the inferior colliculus, neurons with similar response types were not segregated in distinct portions of the nucleus. All possible combinations of response tuning curves to sound localization cues were found in ovoidalis except one - neurons with EI IID tuning curves and side-peak suppressed ITD tuning curves. Two of the combinations of IID and ITD tuning observed in N.Ov have not been reported in other nuclei of the owl's auditory system: neurons with peaked IID tuning curves and flat ITD tuning curves, and neurons with EI IID tuning and side-peak suppressed ITD tuning. Additionally, neurons with IE IID tuning curves were quite common in N.Ov though this type of IID tuning is rarely observed elsewhere in the auditory system. Cells that had peaked IID tuning curves yet were unresponsive to variations in ITD could be the result of inputs selectively restricted to ICc medial shell. Whether such response types exist in medial shell has yet to be determined. This type of response indicates that IID information can be integrated across frequencies independently of the integration of ITD information across frequencies. Alternatively, this type of processing could arise from the convergence of inputs which cause a specific response to stimulus parameters which were not investigated. The cell which had ambiguous, IE IID tuning curves and side-peak suppressed ITD tuning indicates that ITD information can be integrated across frequencies and then combined with IID information which is still ambiguous. The IE IID response category in general could be the result of input from the contralateral medial or lateral shells.

Thirteen neurons were found in N.Ov which responded to both ITD and IID in an unambiguous manner. Such response properties are characteristic of the space-specific neurons found in the space maps of ICx and the optic tectum, though some neurons with these specific responses are also located in the lateral shell subdivision of ICc. Anterograde and retrograde tracer injections in ICc and N.Ov respectively demonstrated that a subset of ICc lateral shell neurons sends a divergent projection of efferents to ovoidalis. Subsets of the core and medial shell subdivisions of ICc also provide axon terminations across broad regions of N.Ov. Efferent input to the auditory thalamus from the medial aspect of ICx cannot be completely ruled out, though anterior and lateral portions of ICx make no efferent contribution to N.Ov. No neurons were retrogradely labeled in the optic tectum from any tracer injected in N.Ov. Consequently, the space-specific neurons found in ovoidalis could simply reflect activity from similarly tuned inputs from ICc lateral shell or (less likely) ICx. Alternatively, these tuned responses could arise as the result of independent integration of ITD and IID information from ICc medial shell, core and lateral shell. The responses of space-specific neurons in Field L, in turn, could be simply relayed from space-specific neurons in N.Ov or synthesized de novo from the appropriate integration of the ambiguous information which is maintained in the thalamus. There is some evidence that the space-specific neurons isolated in N.Ov are not simply reflecting processes which have occurred in the inferior colliculus. The neuron documented in figure 2.10a, for example, has a frequency tuning curve with at least three peaks. This type of frequency tuning has not been reported in either ICx or the optic tectum. This neuron had different sound localization tuning depending on the frequency of the stimulating tone: no response to ITD at the 1.5 kHz peak and phase ambiguous ITD tuning at 4.0 kHz. Apparently, the frequency channels

converging upon this neuron are carrying different types of information concerning sound localization cues. Since these properties have not been described at the level of the inferior colliculus, it is quite likely that they reflect transformations in the neural code that take place in ovoidalis.

As information from frequency channels converge in the inferior colliculus, ambiguity in the response to sound localization cue tuning diminishes. In the thalamus, neurons were found in which the breadth of frequency tuning and resolution of ambiguity were not correlated in this manner. These cells had quite broad frequency tuning and completely ambiguous ITD or IID tuning. In the neuron documented in figure 2.11, it appears that the ambiguous cue is arriving from only a limited number of frequency channels, and that the remaining frequency channels contributing to the response of the neuron are not tuned to that localization parameter. Another possibility is indicated by figure 2.12, in which broad frequency tuning is accompanied by ambiguous IID tuning and the complete lack of ITD tuning. In this case, excitation due to intensity differences only must be responsible for the response of the neuron across frequency channels. Consequently, very similar IID information must occur on each frequency channel.

A general characteristic of neurons that were found in N.Ov which had tuning properties not previously observed in the auditory system was that information concerning sound localization cues can be quite different depending on the frequency tested. Cells which had monotonic IID curves at one frequency had peaked IID curves at others (figure 2.5), or cells which had phase-ambiguous ITD responses at one frequency and no response to ITD at others were rather common. In some cases the integration of such information resulted in responses to wide band stimuli which made little sense for localizing

sounds, such as having two peaks in the IID curve, phase-ambiguous ITD curves (despite broad frequency tuning), or IID curves with a single trough near zero IID. These types of auditory responses together with the lack of topographic organization with respect to sound localization cues in nucleus ovoidalis suggests that auditory information might possibly be reorganized in the thalamus with respect to a behaviorally relevant feature of the stimulus other than sound source location. However, information concerning sound localization is clearly preserved in ovoidalis, and could be used to support sound localizing behavior. The physiological and hodological results of this study suggest that information concerning auditory space which is conveyed from the inferior colliculus to N.Ov is incomplete. Consequently, sound localization subserved only by the thalamo-telencephalic pathway would be rather inaccurate and unreliable. This is in fact what Knudsen et al. (1993) found when they lesioned the optic tectum of a barn owl. In such owls, both the response probability and accuracy of sound localizing behavior (gaze orienting) decreased significantly. Though response probability improved over a period of weeks following the lesion, the accuracy of sound localization remained impaired.

Comparison with other species

Physiological studies in nucleus ovoidalis are rare, and no previous work has been published on sound localization responses in this nucleus. Diekamp and Margoliash (1991) reported that neurons within N.Ov of the zebra finch differ in their response strength to conspecific songs. These investigators also found that the best frequencies at some recording sites differed by more than an octave, which could be similar to the out-of-order neurons found in the tonotopic organization of N.Ov in this study. Biederman-Thorson (1970) described

ovoidalis neurons that had "w-shaped" frequency tuning curves in the ring dove, though Bigalke-Kunz et al. (1987) did not find any units with more than a single excitatory frequency band in the starling. Bigalke-Kunz et al. (1987) tested the latency to tonal stimuli (from a loudspeaker located at a fixed position) in 49 neurons of the N.Ov in the starling and found that 80% had latencies between 11 and 22 msec with an average of 13.5 msec (the values of the other 20% were not reported).

Stanford et al. (1992) compared the ITD responses of neurons in the inferior colliculus and auditory thalamus (medial geniculate body) of the unanesthetized rabbit and found that the location of characteristic delay was more likely to be located at non-peak ITD values in the thalamus than in the inferior colliculus. Olsen and Suga (1991) found that neurons within the MGB of the mustached bat encode range information of echolocated targets. It is unclear what acoustic cues the barn owl uses for range information, or whether such information is mapped in any of the auditory nuclei. One possibility is that the owl uses changes in the apparent location of a sound source at different distances as it flies towards the target. Alternatively, the owl could disregard long range information completely, simply making corrections to the internal representation of the location of a target as it approaches until it is within striking range. Payne (1971) produced some evidence for the latter case; owls which flew from some distance to strike a target usually missed if that target (mice running on leaves or mice with leaves attached to their tails) became silent during the approach of the owl. One facet of the prey striking behavior that the thalamo-telencephalic pathway may be particularly well suited to supporting is the manner in which barn owls orient their talons to parallel the long axis of the body of their prey just prior to striking.

The afferent and efferent pattern of connectivity described in this thesis is consistent with the results of Karten (1967, 1968) obtained in the pigeon. Durand et al. (1992) reported a dense projection from the laminaris input zone of the mesencephalicus lateralis pars dorsalis (MLD) (equivalent to ICc core) to a shell region surrounding nucleus ovoidalis in the ring dove. In the dove, the laminaris input zone of MLD borders on a region medial to MLD called the nucleus intercollicularis (ICo). No projections from the inferior colliculus (also known as the MLD) to the regions surrounding the auditory thalamus were observed in this study. None of the tracer injections, however, were placed near ICo. It is possible that the projections to the areas bordering nucleus ovoidalis in the dove were the result of tracer leakage into ICo. Karten (1967), using techniques to visualize degenerating axons from a lesion of the extent of MLD in the pigeon described no projections to shell regions surrounding nucleus ovoidalis. Wild (1987) found that nucleus LLv projects directly to nucleus ovoidalis, and particularly to a ventrolateral subdivision called semilunaris parovoidalis (SPO). Furthermore, Wild et al. (1993) found that SPO projects preferentially to Field L2b, rather than Field L2a as the main portion of N.Ov does. SPO is not apparent in Nissl-stained coronal sections through the diencephalon in the barn owl, as it is in the pigeon. Injection of BDA into the ventrolateral aspect of N.Ov retrogradely labeled some neurons in LLv and anterogradely labeled efferent terminations heavily in the lateral aspect of Field L (though more sparse anterograde labeling was observed in medial Field L). This lateral portion of Field L could be Field L2b. These results indicate that SPO is a less distinct subdivision of N.Ov in the barn owl than it is in the pigeon or ring dove. The physiological role of this separate pathway from LLv through the ventrolateral aspect of N.Ov to Field L2b has yet to be determined.

BIBLIOGRAPHY

Adolphs, R. (1993a). Acetylcholinesterase staining differentiates functionally distinct auditory pathways in the barn owl. *J. Comp. Neurol.* **329**:365-377.

Adolphs, R. (1993b). Bilateral inhibition generates neuronal responses tuned to interaural level differences in the auditory brainstem of the barn owl. *J. Neurosci.* **13**:3647-3668.

Banks, S.C. and Margoliash, D. (1993). Parametric modeling of the temporal dynamics of neuronal responses using connectionist architectures. *J. Neurophysiol.* **69**:980-991.

Biederman-Thorson, M. (1970). Auditory responses of units in the ovoid nucleus and cerebrum (Field L) of the ring dove. *Brain Res.* **24**:247-256.

Bigalke-Kunz, B. Rübsamen, R., and Dörrscheidt, G. (1987). Tonotopic organization and functional characterization of the auditory thalamus in a songbird, the european starling. *J. Comp. Physiol. A* **161**:255-265.

Boord, R.L. (1968). Ascending projections of the primary cochlear nuclei and nucleus laminaris in the pigeon. *J. Comp. Neurol.* **133**:523-542.

Boord, R. (1969). The anatomy of the avian auditory system. *Ann. N.Y. Acad. Sci.* **167**:186-198.

Boord, R.L. and Rasmussen, G.L. (1963). Projection of the cochlear and lagenar nerves on the cochlear nuclei of the pigeon. *J. Comp. Neurol.* **120**:463-475.

Brainard, M.S., Knudsen, E.I., and Esterly, S.D. (1992). Neural derivation of sound source location: resolution of spatial ambiguities in binaural cues. *J. Acoust. Soc. Am.* **91**:1015-1027.

Brainard, M.S. and Knudsen, E.I. (1993). Experience-dependent plasticity in the inferior colliculus: a site for visual calibration of the neural representation of auditory space in the barn owl. *J. Neurosci.* **13**:4589-4608.

Braun, K., Scheich, H., Heizmann, C.W., and Hunziker, W. (1991). Parvalbumin and calbindin-D28K immunoreactivity as developmental markers of auditory and vocal motor nuclei of the zebra finch. *Neuroscience* **40**:853-869.

Carr, C.E. (1993a). Delay line models of sound localization in the barn owl. *Amer. Zool.* **33**:79-85.

Carr, C.E. (1993b). Processing of temporal information in the brain. *Ann. Rev. Neurosci.* **16**:223-243.

Carr, C.E. and Boudreau, R. (1991). Central projections of auditory nerve fibers in the barn owl. *J. Comp. Neurol.* **314**:306-318.

Carr, C.E. and Boudreau, R. (1993). Organization of the nucleus magnocellularis and the nucleus laminaris in the barn owl - encoding and measuring interaural time differences. *J. Comp. Neurol.* **334**:337-355.

Carr, C., Fujita, I., and Konishi, M. (1989). Distribution of GABAergic neurons and terminals in the auditory system of the barn owl. *J. Comp. Neurol.* **286**:190-207.

Carr, C.E. and Konishi, M. (1988). Axonal delay lines for time measurement in the owl's brainstem. *Proc. Natl. Acad. Sci. USA* **85**:8311-8315.

Carr, C.E. and Konishi, M. (1990). A circuit for detection of interaural time differences in the brainstem of the barn owl. *J. Neurosci.* **10**:3227-3246.

Diekamp, B. and Margoliash, D. (1991). Auditory responses in the nucleus ovoidalis are not so simple. *Soc. Neurosci. Abstr.* **17**:446.

du Lac, S. and Knudsen, E.I. (1990). Neural maps of head movement vector and speed in the optic tectum of the barn owl. *J. Neurophysiol.* **63**:131-146.

Durand, S.E., Tepper, J.M., and Cheng, M-F. (1992). The shell region of the nucleus ovoidalis: a subdivision of the avian auditory thalamus. *J. Comp. Neurol.* **323**:495-518.

Fujita, I. and Konishi, M. (1991). The role of GABAergic inhibition in processing of interaural time difference in the owl's auditory system. *J. Neurosci.* **11**:722-739.

Goldberg, J.M. and Brown, P.B. (1969). Response of binaural neurons of dog superior olivary complex to dichotic tonal stimuli: some physiological mechanisms of sound localization. *J. Neurophysiol.* **32**:613-636.

Ivarsson, C., De Ribaupierre, Y., and De Ribaupierre, F. (1988). Influence of auditory localization cues on neuronal activity in the auditory thalamus of the cat. *J. Neurophysiol.* **59**:586-606.

Jeffress, L. (1948). A place theory of sound localization. *J. Comp. Physiol. Psychol.* **41**:35-39.

Jhaveri, S. and Morest, D. (1982). Sequential alterations of neuronal architecture in nucleus magnocellularis of the developing chicken: a golgi study. *Neuroscience* **7**:837-853.

Joseph, A. and Hyson, R. (1993). Coincidence detection by binaural neurons in the chick brainstem. *J. Neurophysiol.* **69**:1197-1211.

Karten, H. (1967). The organization of the ascending auditory pathway in the pigeon (*columbia livia*) i. diencephalic projections of the inferior colliculus (nucleus mesencephali lateralis pars dorsalis). *Brain Res.* **6**:409-427.

Karten, H. (1968). The ascending auditory pathway in the pigeon (*columbia livia*) ii. telencephalic projections of the nucleus ovoidalis thalami. *Brain Res.* **11**:134-153.

Knudsen, E.I. (1982). Auditory and visual maps of space in the optic tectum of the owl. *J.Neurosci.* **2**:1177-1194.

Knudsen, E.I. (1983). Subdivisions of the inferior colliculus in the barn owl (*Tyto alba*). *J. Comp. Neurol.* **218**:174-186.

Knudsen, E.I. and Brainard, M.S. (1991). Visual instruction of the neural map of auditory space in the developing optic tectum. *Science* **253**:85-87.

Knudsen, E.I., Esterly, S.D., and du Lac, S. (1991). Stretched and Upside-down maps of auditory space in the optic tectum of blind-reared owls; acoustic basis and behavioral correlates. *J. Neurosci.* **11**:1727-1747.

Knudsen, E.I., Esterly, S.D., and Knudsen, P.F. (1984a). Monaural occlusion alters sound localization during a sensitive period in the barn owl. *J. Neurosci.* **4:**1001-1011.

Knudsen, E.I., Esterly, S.D., and Olsen, J.F. (1994). Adaptive plasticity of the auditory space map in the optic tectum of adult and baby barn owls in response to external ear modification. *J. Neurophysiol.* **71**:79-94.

Knudsen, E.I. and Knudsen, P.F. (1983). Space-mapped auditory projections from the inferior colliculus to the optic tectum in the barn owl (*Tyto alba*). *J. Comp. Neurol.* **218**:187-196.

Knudsen, E.I. and Knudsen, P.F. (1985). Vision guides the adjustment of auditory localization in young barn owls. *Science* **230**:545-548.

Knudsen, E.I. and Knudsen, P.F. (1989). Vision calibrates sound localization in developing barn owls. *J. Neurosci.* **9**:3306-3313.

Knudsen, E.I. and Knudsen, P.F. (1990). Sensitive and critical periods for visual calibration of sound localization by barn owls. *J. Neurosci.* **10**:222-232.

Knudsen, E.I., Knudsen, P.F., and Esterly, S.D. (1984b). A critical period for the recovery of sound localization accuracy following monaural occlusion in the barn owl. *J. Neurosci.* **4**:1012-1020.

Knudsen, E., Knudsen, P., and Masino, T. (1993). Parallel pathways mediating both sound localization and gaze control in the forebrain and midbrain of the barn owl. *J. Neurosci..* **13**:2837-2852.

Knudsen, E.I. and Konishi, M. (1978a). A neural map of auditory space in the owl. *Science* **200**:795-797.

Knudsen, E.I. and Konishi, M. (1978b). Center-surround organization of auditory receptive fields in the owl. *Science* **202**:778-780.

Knudsen, E.I. and Konishi, M. (1978c). Space and frequency are represented separately in the auditory midbrain of the owl. *J. Neurophysiol.* **41**:870-884.

Knudsen, E.I. and Konishi, M. (1979). Mechanisms of sound localization in the barn owl. *J. Comp. Physiol.* **133**:13-21.

Knudsen, E.I. and Konishi, M. (1980). Monaural occlusion shifts receptive-field locations of auditory midbrain units in the owl. *J. Neurophysiol.*. **44**:687-696.

Knudsen, E.I., Konishi, M., and Pettigrew, J.D. (1977). Receptive fields of auditory neurons in the owl. *Science* **198**:1278-1280.

Knudsen, E.I. and Mogdans, J. (1992). Vision-independent adjustment of unit tuning to sound localization cues in response to monaural occlusion in developing owl optic tectum. *J. Neurosci.* **12**:3485-3493.

Konishi, M. (1973a). How the owl tracks its prey. *Am. Sci.* **61**:414-424.

Konishi, M. (1973b). Locatable and nonlocatable acoustic signals for barn owls. *Am. Naturalist* **107**:775-785.

Konishi, M. (1986). Centrally synthesized maps of sensory space. *Trends Neurosci.* **4**:163-168.

Konishi, M. (1990). Similar algorithms in different sensory systems and animals. *Cold Spring Harbour Symp. Quant. Biol.* **55**:575-584.

Konishi, M. (1991). Deciphering the brain's codes. *Neural Computation* **3**:1-18.

Konishi, M., Sullivan, W.E., and Takahashi, T. (1985). The owl's cochlear nuclei process different sound localization cues. *J. Acoust. Soc. Am.* **78**:360-364.

Konishi, M., Takahashi, T., Wagner, H., Sullivan, W.E., and Carr, C.E. (1987). Neurophysiological and anatomical substrates of sound localization in the owl. In *Auditory Function*, G.M. Edelman, W.E. Gall and W.M. Cowan, eds., John Wiley, New York.

Leonardus Veenman, C., Reiner, A., and Honig, M.G. (1992). Biotinylated dextran-amine as an anterograde tracer for single- and double-labeling studies. *J. Neurosci. Methods* **41**:239-254

Manley, G.A., Köppl, C., and Konishi, M. (1988). A neural map of interaural intensity differences in the brain stem of the barn owl. *J. Neurosci.* **8**:2665-2576.

Masino, T. and Knudsen, E.I. (1993). Orienting head movements resulting from electrical microstimulation of the brainstem tegmentum in the barn owl. *J. Neurosci.* **13**:351-370.

Mazer, J.A. and Adolphs, R. (1991). Inhibition shapes response to interaural time differences in the inferior colliculus of the barn owl. *Soc. Neurosci. Abstr.* **17**:444.

Mesulam, M.-M. (1978). Tetramethyl benzidine for horseradish peroxidase neurohistochemistry: a non-carcinogenic blue reaction-product with superior sensitivity for visualizing neural afferents and efferents. *J. Histochem. Cytochem.* **26**:106-117.

Mogdans, J. and Knudsen, E.I. (1992). Adaptive adjustment of unit tuning to sound localization cues in response to monaural occlusion in developing owl optic tectum. *J. Neurosci.* **12:**3473-3484.

Mogdans, J. and Knudsen, E.I. (1994). Representation of interaural level difference in the VLVp, the first site of binaural comparison in the barn owl's auditory system. *Hearing Res.* **74**:148-164.

Moiseff, A. and Konishi, M. (1981a). Neuronal and behavioral sensitivity to binaural time differences in the owl. *J. Neurosci.* **1**:40-48.

Moiseff, A. and Konishi, M. (1981b). The owl's interaural pathway is not involved in sound localization. *J. Comp. Physiol. A.* **144**:299-304.

Moiseff, A. and Konishi, M. (1983). Binaural characteristics of units in the owl's brainstem auditory pathway: precursors of restricted spatial receptive fields. *J. Neurosci.* **3**:2553-2562.

Moiseff, A. and Konishi, M. (1989a). Binaural disparity cues available to the barn owl for sound localization. *J. Comp. Physiol. A* **164**:629-636.

Moiseff, A. and Konishi, M. (1989b). Bi-coordinate sound localization by the barn owl. *J. Comp. Physiol. A* **164**:637-644.

Olsen, J.F., Knudsen, E.I., and Esterly, S.D. (1989). Neural maps of interaural time and intensity differences in the optic tectum of the barn owl. *J. Neurosci.* **9**:2591-2605.

Olsen, J.F. and Suga, N. (1991). Combination-sensitive neurons in the medial geniculate body of the mustached bat: encoding of target range information. *J. Neurophysiol.* **65**:1275-1296.

Overholt, E.M., Rubel, E.W., and Hyson, R.L. (1992). A circuit for coding interaural time differences in the chick brainstem. *J. Neurosci.* **12**:1698-1708.

Parks, T. and Rubel, E. (1975). Organization of projections from n. magnocellularis to n. laminaris. *J. Comp. Neurol.* **164**:435-448.

Payne, R. (1962). How the barn owl locates prey by hearing. *The Living Bird, First Annual of the Cornell Laboratory of Ornithology.* Chapter 34, pp.151-159.

Payne, R. (1971). Acoustic location of prey by barn owls (*tyto alba*). *J. Exp. Biol.* **54**:535-573.

Proctor, L. and Konishi, M. (1992). Responses of cells in an auditory thalamic nucleus of the barn owl to sound localization cues. *Soc. Neurosci. Abstr.* **18**:840.

Raman, I.M. and Trussell, L.O. (1992). The kinetics of the response to glutamate and kainate in neurons of the avian cochlear nucleus. *Neuron* **9**:173-186.

Raman, I.M. and Trussell, L.O. (1993). Pathway specific variants of ampa receptors may subserve physiological roles of auditory neurons. *Soc. Neurosci. Abstr.* **297**:3.

Rose, J.E., Brugge, J., Anderson, D., and Hind, J. (1967). Phase-locked response to low-frequency tones in single auditory nerve fibers of the squirrel monkey. *J. Neurophysiol.* **30**:769-793.

Rose, J.E., Grass, N.G., Geisler, C.D., and Hind, J.E. (1966). Some neural mechanisms in the inferior colliculus of the cat which may be relevant to localization of a sound source. *J. Neurophsyiol.* **29**:288-314.

Sachs, M. and Sinnott, J. (1978). Responses to tones of single cells in nucleus magnocellularis and nucleus angularis of the redwing blackbird (*agelaius phoenicens*). *J. Comp. Physiol.* **126**:347-361.

Stanford, T.R., Kuwada, S., and Batra, R. (1992). A comparison of the interaural time sensitivity of neurons in the inferior colliculus and thalamus of the unanesthetized rabbit. *J. Neurosci.* **12**:3200-3216.

Sullivan, W.E. (1985). Classification of response patterns in cochlear nucleus of barn owl: correlation with functional response properties. *J. Neurophysiol.* **53**:201-216.

Sullivan, W.E. and Konishi, M. (1984). Segregation of stimulus phase and intensity coding in the cochlear nucleus of the barn owl. *J. Neurosci.* **4**:1787-1799.

Sullivan, W.E. and Konishi, M. (1986). Neural map of interaural phase difference in the owl's brainstem. *Proc. Nat. Acad. Sci.* **83**:8400-8404.

Takahashi, T., Carr, C.E., Brecha, N., and Konishi, M. (1987). Calcium binding protein-like immunoreactivity labels the terminal field of nucleus laminaris of the barn owl. *J. Neurosci.* **7**:1843-1856.

Takahashi, T. and Keller, C. (1992). Commissural connections mediate inhibition for the computation of interaural level difference in the barn owl. *J. Comp. Physiol. A* **170**:161-169.

Takahashi, T. and Konishi, M. (1986). Selectivity for interaural time difference in the owl's midbrain. *J. Neurosci.* **6**:3413-3422.

Takahashi, T. and Konishi, M. (1988a). Projections of the cochlear nuclei and nucleus laminaris to the inferior colliculus of the barn owl. *J. Comp. Neurol.* **274**:190-211.

Takahashi, T. and Konishi, M. (1988b). Projections of nucleus angularis and nucleus laminaris to the lateral lemniscal nuclear complex of the barn owl. *J. Comp. Neurol.* **274**:212-238.

Takahashi, T., Moiseff, A., and Konishi, M. (1984). Time and intensity cues are processed independently in the auditory system of the owl. *J. Neurosci.* **4**:1781-1786.

Takahashi, T., Wagner, H., and Konishi, M. (1989). Role of commissural projections in the representation of bilateral auditory space in the barn owl's inferior colliculus. *J. Comp. Neurol.* **281**:545-554.

Wagner, H. (1993). Sound-localization deficits induced by lesions in the barn owl's auditory space map. *J. Neurosci.* **13**:371-386.

Wagner, H., Takahashi, T., and Konishi, M. (1987). Representation of interaural time difference in the central nucleus of the barn owl's inferior colliculus. *J. Neurosci.* **7**:3105-3116.

Warchol, M.E. and Dallos, P. (1990). Neural coding in the chick cochlear nucleus. *J. Comp. Physiol. A.* **166**:721-734.

Wild, J.M. (1987). Nuclei of the lateral lemniscus project directly to the thalamic auditory nuclei in the pigeon. *Brain. Res.* **408**:303-307.

Wild, J.M., Karten, H.J., and Frost, B.J. (1993). Connections of the auditory forebrain in the pigeon (*columbia livia*). *J. Comp. Neurol.* **337**:32-62.

Yin, T.C. and Chan, J.C. (1990). Interaural time sensitivity in medial superior olive of cat. *J. Neurophysiol.* **64:**465-488.

APPENDIX: Source code for OASys, the X11 based Owl Auditory System data acquisition and analysis software.

```
# Makefile for oasys: Owl Auditory SYStem data acquisition & analysis program.

OBJECTS = main.o setup.o gui.o plotutils.o \
          compress.o atten.o uhoh.o \
          file.o plots.o loadstim.o \
          sound.o fstnoise.o analysis.o
DSP = /usr/local/s56dsp
INCLUDES = includes.h defines.h globals.h

LIBS = -g -lnr -lm -lXcu -lXaw -lXmu -lXext -lXt -lX11 -L$(DSP)/lib -ldrp -lqckMon
#LIBS = -lm -lnr -lXcu -lXaw -lXmu -lXext -lXt -lX11 -L$(DSP)/lib -ldrp -lqckMon

CFLAGS  = -g -I$(DSP)/include -I$(DSP)/libdrp
#CFLAGS  = -O -I$(DSP)/include -I$(DSP)/libdrp

CC = cc

oasys:  $(OBJECTS)
          $(CC) $(OBJECTS) $(LIBS) -o oasys

main.o: $(INCLUDES)

setup.o:  $(INCLUDES)

gui.o: $(INCLUDES)

plotutils.o: $(INCLUDES)

compress.o: $(INCLUDES)

atten.o:

uhoh.o: $(INCLUDES)

file.o: $(INCLUDES)

plots.o: $(INCLUDES)

loadstim.o: $(INCLUDES)

sound.o: $(INCLUDES)

fstnoise.o:

analysis.o: $(INCLUDES)

clean:
   rm -f $(OBJECTS) core
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

char description[80];
char stimparams[80];

void analyze_data()
{
   FILE *readfile;
   void plot_curve();
   int slparam,  i, j, k, index, size, num_pts, stimbegin, stimend;
   int *count, *spont_count;
   int total_spikes;
   int start, stop, step, nreps, total_duration;
   float *abscissa, *ordinate, *standard_err, *vector();
   float *spont, *spont_err, factor;
   float variance, std_deviation;
   unsigned int *event_list;

   if ((readfile = fopen(datafilename, "r")) == NULL) {
      uhoh("Unable to fopen \"%s\" for reading.", datafilename);
      return;
   }
   read_header(readfile);

   stimbegin = Analysis.soundStim.prestim_delay*1000;
   stimend = Analysis.soundStim.stim_dur*1000 + stimbegin;

   switch(Analysis.tuningCurve) {
      case Iid:
         start = Analysis.uval.iid_TC.start;
         stop = Analysis.uval.iid_TC.end;
         step = Analysis.uval.iid_TC.step;
         break;
      case Itd:
         start = Analysis.uval.itd_TC.start;
         stop = Analysis.uval.itd_TC.end;
         step = Analysis.uval.itd_TC.step;
         break;
      case Fiid:
         start = Analysis.uval.fiid_TC.start;
         stop = Analysis.uval.fiid_TC.end;
         step = Analysis.uval.fiid_TC.step;
         break;
      case Freq:
         start = Analysis.uval.freq_TC.start;
         stop = Analysis.uval.freq_TC.end;
         step = Analysis.uval.freq_TC.step;
         break;
   }
   num_pts = ((stop - start)/step)+1;
   abscissa = vector(0, num_pts-1);
   ordinate = vector(0, num_pts-1);   /* y axis */
   standard_err = vector(0, num_pts-1);   /* standard error */
   spont = vector(0, num_pts-1);   /* spontaneous rate */
   nreps = Analysis.soundStim.nreps;
   count = (int *)malloc((unsigned)nreps*sizeof(int));
   spont_count = (int *)malloc((unsigned)nreps*sizeof(int));

   index = 0;
   /* for each value */
   for (slparam=start; slparam<=stop; slparam+=step) {
      for (j=0; j<nreps; j++) spont_count[j] = count[j] = 0;
      for (j=1; j<=nreps; j++) {
         fread((char *)&size, sizeof(int), 1, readfile);
         if (size > 0) {
            if ((event_list = (unsigned *)malloc(size)) == NULL) {
               uhoh("Out of core - Analysis()");
               return;
             }
            if (fread((char *)event_list, size, 1, readfile) < 1) {
               free(event_list);
               uhoh("Error reading events.\n");
               return;
```

```
            }
            /* use only events which occur during the stimulus */
            for (i=1; i<=event_list[0]; i++) {
               if ((event_time(event_list[i]) <= stimend) &&
                   (event_time(event_list[i]) >= stimbegin))
                  ++count[j-1];
               if (event_time(event_list[i]) < stimbegin)
                  ++spont_count[j-1];
            }
         } else {
            uhoh("Size of event list = 0\n");
            return;
         }
      }
      total_spikes = 0;
      for (i=0; i<nreps; i++) total_spikes += count[i];
      abscissa[index] = (float)slparam;
      /* calc the average */
      ordinate[index] = (float)total_spikes/(float)nreps;
      /* calc the std deviation */
      std_deviation = variance = 0.0;
      for (i=0; i<nreps; i++)
         variance += SQR((float)count[i] - ordinate[index]);
      variance /= (float)(nreps-1);
      std_deviation = (float)sqrt((double)variance);
      standard_err[index] = std_deviation/(float)sqrt((double)nreps);
      total_spikes = 0;
      for (i=0; i<nreps; i++) total_spikes += spont_count[i];
      spont[index] = (float)total_spikes/(float)nreps;
      if (Analysis.soundStim.prestim_delay > 0)
         factor = (float)Analysis.soundStim.stim_dur/(float)Analysis.soundStim.prestim_delay;
      else factor = 0.0;
      spont[index] *= factor;
      index++;
      free((char *)event_list);
   }
   /* for labeling the plot with the data file name */
   sprintf(description, "File: %s", datafilename);

   switch(Analysis.tuningCurve) {
      case Iid:
         if (Analysis.soundStim.stimtype == WNoise)
            sprintf(stimparams, "Noise     ITD: %d     ABI: %d ",
               Analysis.uval.iid_TC.itd, Analysis.uval.iid_TC.abi);
         else if (Analysis.soundStim.stimtype == Tone)
            sprintf(stimparams, "%d Hz Tone     ITD: %d     ABI: %d ",
               Analysis.soundStim.freq, Analysis.uval.iid_TC.itd,
               Analysis.uval.iid_TC.abi);
         break;
      case Freq:
         sprintf(stimparams, "ITD: %d     IID: %d     ABI: %d",
            Analysis.uval.freq_TC.itd, Analysis.uval.freq_TC.iid,
            Analysis.uval.freq_TC.abi);
         break;
      case Itd:
         if (Analysis.soundStim.stimtype == WNoise)
            sprintf(stimparams, "Noise     IID: %d     ABI: %d ",
               Analysis.uval.itd_TC.iid, Analysis.uval.itd_TC.abi);
         else if (Analysis.soundStim.stimtype == Tone)
            sprintf(stimparams, "%d Hz Tone     IID: %d     ABI: %d ",
               Analysis.soundStim.freq, Analysis.uval.itd_TC.iid,
               Analysis.uval.itd_TC.abi);
         break;
      case Fiid:
         if (Analysis.soundStim.stimtype == WNoise)
            sprintf(stimparams, "Noise     ITD: %d     Fixed: %d",
               Analysis.uval.fiid_TC.itd, Analysis.uval.fiid_TC.fixed);
         else if (Analysis.soundStim.stimtype == Tone)
            sprintf(stimparams, "%d Hz Tone     ITD: %d     Fixed: %d ",
               Analysis.soundStim.freq, Analysis.uval.fiid_TC.itd,
               Analysis.uval.fiid_TC.fixed);
         break;
   }

   plot_curve(abscissa, ordinate, standard_err, spont, num_pts);
```

```
    free_vector(abscissa, 0, num_pts-1);
    free_vector(ordinate, 0, num_pts-1);
    free_vector(standard_err, 0, num_pts-1);
    free((char *)count);
    fclose(readfile);
}

void plot_curve(xvals, yvals, std_error, sr, numvals)
int  numvals;
float xvals[], yvals[], std_error[];
float sr[];      /* spontaneous rate */
{
    register int i;
    int  x, y, x2, y2, n;
    int textwidth;
    int screen = DefaultScreen(display);
    float min, max;
    Drawable d;
    Window window;
    XWindowAttributes info;
    WC_window canvas_wc;
    float x_value_min, y_value_min, xmin, ymin;
    float x_value_max, y_value_max, xmax, ymax;
    float x_range, y_range;
    float y_hash;
    float loc, step, sy;
    float *ddy;              /* y''  - for cubic spline interpolation */
    float *vector();
    void set_world_coords();
    void user_to_out();
    void spline(), splint();
    char buf[15];
    char xtitle[64];

    XRaiseWindow(display, XtWindow(XtParent(freqCanvas)));
    switch(Analysis.tuningCurve) {
        case Iid:
            d = window = XtWindow(iidCanvas);
            strcpy(xtitle, "IID in dB\0");
            break;
        case Itd:
            d = window = XtWindow(itdCanvas);
            strcpy(xtitle, "ITD in usec\0");
            break;
        case Fiid:
            d = window = XtWindow(fiidCanvas);
            strcpy(xtitle, "Fixed ILD in dB\0");
            break;
        case Freq:
            d = window = XtWindow(freqCanvas);
            strcpy(xtitle, "Frequency in KHz\0");
            break;
    }
    /* erase the pixmap */
    XSetForeground(display, plot_gc, WhitePixel(display, screen));
    XFillRectangle(display, d, plot_gc, 0, 0, 400, 350);
    XSetForeground(display, plot_gc, BlackPixel(display, screen));

    XGetWindowAttributes(display, window, &info);

    /* find the min & max */
    max = 0.0;
    for (i=0; i<numvals; i++)
        if (max < yvals[i]) max = (float)yvals[i];
    min = max;
    for (i=0; i<numvals; i++)
        if (min > yvals[i]) min = (float)yvals[i];
    x_value_min = (float)xvals[0];
    x_value_max = (float)xvals[numvals-1];
    y_value_min = 0.0;
    y_value_max = max;
    x_range = x_value_max - x_value_min;
    y_range = y_value_max - y_value_min;

    /* scale window to the data */
```

```
xmin = x_value_min - 0.2*x_range;  /* 20% space */
ymin = y_value_min - 0.2*y_range;
xmax = x_value_max + 0.2*x_range;
ymax = y_value_max + 0.2*y_range;
set_world_coords(&canvas_wc, xmin, ymin, xmax, ymax);

/* draw x and y axis */
user_to_out(canvas_wc, x_value_min, y_value_min, &x, &y,
    info.width, info.height);
user_to_out(canvas_wc, x_value_max, y_value_min, &x2, &y2,
   info.width, info.height);
XDrawLine(display, d, plot_gc, x, y, x2, y2);
user_to_out(canvas_wc, x_value_min, y_value_max, &x2, &y2,
   info.width, info.height);
XDrawLine(display, d, plot_gc, x, y, x2, y2);

/* put hash marks on the x axis */
for (i=0; i<numvals; i++) {
   user_to_out(canvas_wc, (float)xvals[i], 0.0, &x, &y,
      info.width, info.height);
   user_to_out(canvas_wc, (float)xvals[i], -(y_value_max/50.0), &x2, &y2,
      info.width, info.height);
   XDrawLine(display, d, plot_gc, x, y, x2, y2);
}

/* label hash marks */
switch(Analysis.tuningCurve) {
   case Iid:
      if (Analysis.uval.iid_TC.step < 10)
         n = 10/Analysis.uval.iid_TC.step;
      else n = 1;
      for (i=0; i<numvals; i+=n) {
         user_to_out(canvas_wc, xvals[i], 0.0, &x, &y,
            info.width, info.height);
         sprintf(buf, "%d", (int)xvals[i]);
         textwidth = XTextWidth(font_info, buf, strlen(buf));
         XDrawString(display, d, plot_gc, x-(textwidth/2), y+20,
            buf, strlen(buf));
      }
      break;
   case Itd:
      if (Analysis.uval.itd_TC.step < 40)
         /* label hash marks which are multiples of 60 */
         n = 60/Analysis.uval.itd_TC.step;
      else n = 1;
      for (i=0; i<numvals; i+=n) {
         user_to_out(canvas_wc, xvals[i], 0.0, &x, &y,
            info.width, info.height);
         sprintf(buf, "%d", (int)xvals[i]);
         textwidth = XTextWidth(font_info, buf, strlen(buf));
         XDrawString(display, d, plot_gc, x-(textwidth/2), y+20,
            buf, strlen(buf));
      }
      break;
   case Fiid:
      n = 1;
      for (i=0; i<numvals; i+=n) {
         user_to_out(canvas_wc, xvals[i], 0.0, &x, &y,
            info.width, info.height);
         sprintf(buf, "%d", (int)xvals[i]);
         textwidth = XTextWidth(font_info, buf, strlen(buf));
         XDrawString(display, d, plot_gc, x-(textwidth/2), y+20,
            buf, strlen(buf));
      }
      break;
   case Freq:
      if (Analysis.uval.freq_TC.step < 1000)
         /* label hash marks that are multiples of 1000 */
         n = 1000/Analysis.uval.freq_TC.step;
      else
         n = 1;
      for (i=0; i<numvals; i+=n) {
         user_to_out(canvas_wc, xvals[i], 0.0, &x, &y,
            info.width, info.height);
         sprintf(buf, "%.1f", xvals[i]/1000.0);
```

```
            textwidth = XTextWidth(font_info, buf, strlen(buf));
            XDrawString(display, d, plot_gc, x-(textwidth/2),
               y+20, buf, strlen(buf));
         }
         break;
   }

   /* title the x-axis */
   user_to_out(canvas_wc, (float)xvals[numvals/2], 0.0, &x, &y,
      info.width, info.height);
   textwidth = XTextWidth(font_info, xtitle, strlen(xtitle));
   XDrawString(display, d, plot_gc, x-(textwidth/2),
      y+40, xtitle, strlen(xtitle));

   ddy = vector(1, numvals);
   spline(xvals-1, yvals-1, numvals, 0.0, 0.0, ddy);
   /* draw a cubic spline interpolated iid tuning curve */
   step = 1.0;
   loc = xvals[0];
   while (loc < xvals[numvals-1]) {
      splint(xvals-1, yvals-1, ddy, numvals, loc, &sy);
      user_to_out(canvas_wc, loc, sy, &x, &y, info.width, info.height);
      loc += step;
      splint(xvals-1, yvals-1, ddy, numvals, loc, &sy);
      user_to_out(canvas_wc, loc, sy, &x2, &y2, info.width, info.height);
      XDrawLine(display, d, plot_gc, x, y, x2, y2);
      /*loc += step; */
   }
   free_vector(1, numvals, ddy);
   /* draw the data points */
   for (i=0; i<numvals; i++) {
      user_to_out(canvas_wc, xvals[i], yvals[i],
         &x, &y, info.width, info.height);
      XDrawRectangle(display, d, plot_gc, x-2, y-2, 4, 4);
   }
   /* draw the standard errors of the  values */
   for (i=0; i<numvals; i++) {
      user_to_out(canvas_wc, xvals[i], yvals[i]+std_error[i], &x, &y,
         info.width, info.height);
      user_to_out(canvas_wc, xvals[i], yvals[i]-std_error[i], &x2, &y2,
         info.width, info.height);
      XDrawLine(display, d, plot_gc, x, y, x2, y2);
      XDrawLine(display, d, plot_gc, x-2, y, x+2, y);
      XDrawLine(display, d, plot_gc, x-2, y2, x+2, y2);
   }
   /* draw the spontaneous rate line */
   for (i=0; i<numvals-1; i++) {
      user_to_out(canvas_wc, xvals[i], sr[i], &x, &y,
         info.width, info.height);
      user_to_out(canvas_wc, xvals[i+1], sr[i+1],
         &x2, &y2, info.width, info.height);
      XDrawLine(display, d, dashed_gc, x, y, x2, y2);
   }

   /* mark & label the y axis */
   /* highest peak */
   user_to_out(canvas_wc, (float)xvals[0], max, &x, &y,
      info.width, info.height);
   user_to_out(canvas_wc, (float)xvals[0]-(x_value_max/50.0), max,
      &x2, &y2, info.width, info.height);
   XDrawLine(display, d, plot_gc, x, y, x2, y2);
   sprintf(buf, "%.1f", max);
   textwidth = XTextWidth(font_info, buf, strlen(buf));
   XDrawString(display, d, plot_gc, x2-(textwidth), y+4, buf, strlen(buf));

   /* lowest trough */
   user_to_out(canvas_wc, (float)xvals[0], min, &x, &y, info.width, info.height
);
   user_to_out(canvas_wc, (float)xvals[0]-(x_value_max/50.0), min,
      &x2, &y2, info.width, info.height);
   XDrawLine(display, d, plot_gc, x, y, x2, y2);
   sprintf(buf, "%.1f", min);
   textwidth = XTextWidth(font_info, buf, strlen(buf));
   XDrawString(display, d, plot_gc, x2-textwidth, y+4, buf, strlen(buf));
```

```
   XDrawString(display, d, plot_gc, 20, 20, description, strlen(description));
   XDrawString(display, d, plot_gc, 20, 30, stimparams, strlen(stimparams));
   XFlush(display);
}

void iidExposed(w, event, params, numparams)
Widget w;
XEvent *event;
String *params;
Cardinal *numparams;
{
   XCopyArea(display, iidPixmap, XtWindow(iidCanvas), plot_gc,
      0, 0, 400, 350, 0, 0);
}

void itdExposed(w, event, params, numparams)
Widget w;
XEvent *event;
String *params;
Cardinal *numparams;
{
   XCopyArea(display, itdPixmap, XtWindow(itdCanvas), plot_gc,
      0, 0, 400, 350, 0, 0);
}

void fiidExposed(w, event, params, numparams)
Widget w;
XEvent *event;
String *params;
Cardinal *numparams;
{
   XCopyArea(display, fiidPixmap, XtWindow(fiidCanvas), plot_gc,
      0, 0, 400, 350, 0, 0);
}

void freqExposed(w, event, params, numparams)
Widget w;
XEvent *event;
String *params;
Cardinal *numparams;
{
   XCopyArea(display, freqPixmap, XtWindow(freqCanvas), plot_gc,
      0, 0, 400, 350, 0, 0);
}
```

```
#include "includes.h"
#include <sys/filio.h>
#include <fcntl.h>
#include <termios.h>

#define MAXATTEN 99

#define TTYDEV    "/dev/ttya"

/* BAUDRATE can be B9600, B19200, B38400 etc.. */
#define BAUDRATE  B38400
#define BAUDRATE_STR "38400"

static void xb1_sendstr(n, buf)
     int n;
     char *buf;
{
  int fd;
  static struct termios *t = NULL;

  if ((fd = open(TTYDEV, O_RDWR, 0777)) < 0) {
    perror(TTYDEV);
    exit(1);
  }
  if (t == NULL) {
    t = (struct termios *) malloc(sizeof(struct termios));
  }
  tcgetattr(fd, t);
  t->c_oflag &= ~(OPOST | OCRNL | XTABS);    /* clear */
  t->c_cflag &= ~(CBAUD | CSIZE | PARENB);      /* clear */
  t->c_cflag |= (BAUDRATE | CS8);          /* set   */
  tcsetattr(fd, TCSANOW, t);
  if (write(fd, buf, n) < 0)  {
    perror(TTYDEV);
    exit(1);
  }
  close(fd);
}

void PA4atten(id, atten)
int id; /* 0x04 - left; 0x05 - right */
float atten;
{
  char buf[6];
  int bitpat;

  bitpat = (int)(atten * 10.0 + 0.05);

  buf[0] = 0x7f & id;       /* device address */
  buf[1] = 0x44;      /* instruction length */
  buf[2] = 0x20;      /* PA4_ATT op code */
  buf[3] = 0xff & (bitpat >> 8);
  buf[4] = 0xff & bitpat;
  buf[5] = 0xff & (buf[2] + buf[3] + buf[4]);
  xb1_sendstr(6, buf);
}

int setRack(freq, latten, ratten)
     int freq;        /* hz */
     int latten;      /* db */
     int ratten;      /* db */
{
#ifndef STANDALONE
  extern void label_set();
#endif
  static int disable = -1;
   void PA4atten();

  if (disable < 0) {
    if (getenv("ATTOFF") != 0) {
      disable = 1;
      fprintf(stderr, "setRack: attenuator's disabled\n");
    } else {
      disable = 0;
    }
```

```
  }

  if (disable) {
    PA4atten(0x04, (float)MAXATTEN);
    PA4atten(0x05, (float)MAXATTEN);
  } else {
    if (latten < 0 || latten < 0)
      return(True);

    if (freq != 0) {
      fprintf(stderr, "\007setRack: can't set frequency!!\n");
      return(False);
    }
    if (latten > MAXATTEN) {
      fprintf(stderr, "\007setRack: %ddb on LEFT too big, using %ddB\n",
         latten, MAXATTEN);
      latten = MAXATTEN;
    }
    if (ratten > MAXATTEN) {
      fprintf(stderr, "\007setRack: %ddb on RIGHT too big, using %ddB\n",
         ratten, MAXATTEN);
      ratten = MAXATTEN;
    }

    PA4atten(0x04, (float)latten);
    PA4atten(0x05, (float)ratten);
  }
  return(True);
}
```

```
/* Deals with the back-asswards way the event timer board puts information
 * into the data buffer.  Blame Ted Sullivan, John Power or Jack Wathey.
 */

#include "includes.h"
#include "defines.h"
#include "globals.h"

unsigned int *CompressEventBuffer(size)
int *size;
{
   int i;
   unsigned int Count, *CompBuffer, *event_ptr, *first_event;
   unsigned int TriggerBug;
   unsigned short low_word, high_word, *short_ptr;

   TriggerBug = 100;

   /* transpose low and high words in each event */
   for (event_ptr = first_event =
         (unsigned int *)(short_ptr = Events.list);
         *event_ptr; event_ptr++) {
      low_word = *short_ptr++;
      high_word = *short_ptr;
      *((short *)event_ptr) = high_word;
      *short_ptr++ = low_word;
   }

   Count = 0;
   for (event_ptr = first_event; event_id(*event_ptr); event_ptr++) {
      if (event_ptr != first_event
            &&
         event_id(*event_ptr) == event_id(*(event_ptr-1))
            &&
         (event_time(*(event_ptr)) - event_time(*(event_ptr-1))) < TriggerBug)
         continue;
      else
         Count++;
   }

   *size = (Count+1) * sizeof(unsigned int);
   if ((CompBuffer = (unsigned int *)malloc(*size)) == NULL)
      fprintf(stderr, "Out of memory in CompressEventBuffer()\n");
   else {
      CompBuffer[0] = Count;
      Count = 0;
      for (event_ptr = first_event; event_id(*event_ptr); event_ptr++) {
         if (event_ptr != first_event &&
               event_id(*event_ptr) == event_id(*(event_ptr-1)) &&
            (event_time(*event_ptr)-event_time(*(event_ptr-1))) < TriggerBug)
            continue;
         else {
            CompBuffer[1+Count] = *event_ptr;
            Count++;
         }
      }
   }
   return(CompBuffer);
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

#include <sys/types.h>
#ifdef mc700
#include <ndir.h>
#else
#include <sys/dir.h>
#endif

BOOLEAN FILEOK = False;;

void getFilename(w, client, call)
Widget w;      /* the begin data acquisition command widget */
caddr_t client, call;
{
   XcuDeckRaiseWidget(acqDeck, XtNameToWidget(acqDeck, "file"));
}

getFile(widget, event, params, numparams)
Widget widget;
XEvent *event;
String *params;
Cardinal *numparams;
{
   Widget dialog = XtParent(widget);
   struct stat file_info;
   char *filename;
   char buf[80];
   void start_run();

   filename = XawDialogGetValueString(dialog);
   if (stat(filename, &file_info) == -1) {
      if (errno == ENOENT) {
         if ((datafile = fopen(filename, "w")) == NULL) {
            sprintf(buf, "Unable to open file \"%s\".", filename);
            uhoh(buf);
            FILEOK = False;
         }
         else {
            XcuDeckRaiseWidget(acqDeck, begin);
            strcpy(datafilename, filename);
            start_run();
         }
      }
      else {
         sprintf(buf, "File stat error. Errno: %d", errno);
         uhoh(buf);
         FILEOK = False;
      }
   }
   else if ((file_info.st_mode & S_IFMT) == S_IFDIR) {
      uhoh("Requested file is a subdirectory!");
      FILEOK = False;
   }
   else
      overwrite(filename);
}

overwrite(filename)
char *filename;
{
   Widget OverwritePopup;
   Widget owTbl, owLbl;
   Widget yes, no;
   Position x, y;
   char buf[80];
   Arg args[5];
   Cardinal i;
   void yesCB(), noCB();

   i = 0;
   XtSetArg(args[i], XtNinput, True); i++;
   XtSetArg(args[i], XtNallowShellResize, True); i++;
```

```
    XtTranslateCoords(acqDeck, (Position)0, (Position)0, &x, &y);
    XtSetArg(args[i], XtNx, x); i++;
    XtSetArg(args[i], XtNy, y); i++;
    OverwritePopup = XtCreatePopupShell("Overwrite?", transientShellWidgetClass,
        toplevel, args, i);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c\n c c."); i++;
    owTbl = XtCreateManagedWidget("owTbl", xcuTblWidgetClass,
        OverwritePopup, args, i);

    sprintf(buf, "File \"%s\" exists.\n Overwrite?", filename);
    i = 0;
    XtSetArg(args[i], XtNlabel, buf); i++;
    owLbl = XtCreateManagedWidget("owLbl", xcuLabelWidgetClass,
        owTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "Yes"); i++;
    yes = XtCreateManagedWidget("yes", xcuCommandWidgetClass,
        owTbl, args, i);
    XtAddCallback(yes, XtNcallback, yesCB, (caddr_t)filename);

    i = 0;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "No"); i++;
    no = XtCreateManagedWidget("no", xcuCommandWidgetClass,
        owTbl, args, i);
    XtAddCallback(no, XtNcallback, noCB, (caddr_t)NULL);

    XtPopup(OverwritePopup, XtGrabNonexclusive);
}

write_header()
{
    fprintf(datafile, "OASys Data File\n");
    if (curvetype == Freq) {
        fprintf(datafile, "Frequency Tuning Curve\n");
        fprintf(datafile, "Start: %d\n", FreqCurve.start);
        fprintf(datafile, "End: %d\n", FreqCurve.end);
        fprintf(datafile, "Step: %d\n", FreqCurve.step);
        fprintf(datafile, "ITD: %d\n", FreqCurve.itd);
        fprintf(datafile, "IID: %d\n", FreqCurve.iid);
        fprintf(datafile, "ABI: %d\n", FreqCurve.abi);
    }
    else if (curvetype == Itd) {
        fprintf(datafile, "ITD Tuning Curve\n");
        fprintf(datafile, "Start: %d\n", ItdCurve.start);
        fprintf(datafile, "End: %d\n", ItdCurve.end);
        fprintf(datafile, "Step: %d\n", ItdCurve.step);
        fprintf(datafile, "IID: %d\n", ItdCurve.iid);
        fprintf(datafile, "ABI: %d\n", ItdCurve.abi);
        write_stimtype();
    }
    else if (curvetype == Iid) {
        fprintf(datafile, "IID Tuning Curve\n");
        fprintf(datafile, "Start: %d\n", IidCurve.start);
        fprintf(datafile, "End: %d\n", IidCurve.end);
        fprintf(datafile, "Step: %d\n", IidCurve.step);
        fprintf(datafile, "ITD: %d\n", IidCurve.itd);
        fprintf(datafile, "ABI: %d\n", IidCurve.abi);
        write_stimtype();
    }
    else if (curvetype == Fiid) {
        fprintf(datafile, "Fixed IID Tuning Curve\n");
        fprintf(datafile, "Start: %d\n", FiidCurve.start);
        fprintf(datafile, "End: %d\n", FiidCurve.end);
        fprintf(datafile, "Step: %d\n", FiidCurve.step);
        fprintf(datafile, "Fixed: %d\n", FiidCurve.fixed);
        fprintf(datafile, "ITD: %d\n", FiidCurve.itd);
        write_stimtype();
    }
    fprintf(datafile, "Reps: %d\n", stimulus.nreps);
    fprintf(datafile, "Recording duration: %d\n", stimulus.total_dur);
```

```
   fprintf(datafile, "PreStimulus Delay: %d\n", stimulus.prestim_delay);
   fprintf(datafile, "Stimulus duration: %d\n", stimulus.stim_dur);
   fprintf(datafile, "END HEADER\n");
}


write_stimtype()
{
   register int i;

   if (stimulus.stimtype == Tone)
      fprintf(datafile, "Tone: %d\n", stimulus.freq);
   else if (stimulus.stimtype == WNoise)
      fprintf(datafile, "WNoise\n");
   else if (stimulus.stimtype == BPNoise) {
      fprintf(datafile, "BPNoise\n");
      fprintf(datafile, "Low Freq: %d\n", stimulus.bpNoiseStim.startfreq);
      fprintf(datafile, "High Freq: %d\n", stimulus.bpNoiseStim.endfreq);
   }
   else if (stimulus.stimtype == ToneCombo) {
      fprintf(datafile, "ToneCombo\n");
      fprintf(datafile, "Num Freqs: %d\n", stimulus.tcStim.numfreqs);
      for (i=0; i<stimulus.tcStim.numfreqs; i++)
         fprintf(datafile, "Freq: %d\n", stimulus.tcStim.comp[i].freq);
   }
}

read_header(readfile)
FILE *readfile;
{
   register int i, j;
   char oasys[80];
   char buf[80], ident[80], value[80];
   int val;
   int start, end, step, itd, iid, abi, fixed;

   for (i=0; i<80; i++) buf[i] = '\0';
   fgets(oasys, 80, readfile);
   if (strcmp("OASys Data File\n", oasys) != 0) {
      uhoh("Specified file is not an OASys data file.");
      return;
   }
   for (i=0; i<80; i++) buf[i] = '\0';
   fgets(buf, 80, readfile);
   while (strcmp(buf, "END HEADER\n") != 0) {
      /* parse the data file header info */
      if (strcmp("Frequency Tuning Curve\n", buf) == 0)
         Analysis.tuningCurve = Freq;
      else if (strcmp("ITD Tuning Curve\n", buf) == 0)
         Analysis.tuningCurve = Itd;
      else if (strcmp("IID Tuning Curve\n", buf) == 0)
         Analysis.tuningCurve = Iid;
      else if (strcmp("Fixed IID Tuning Curve\n", buf) == 0)
         Analysis.tuningCurve = Fiid;
      else if (strcmp("WNoise\n", buf) == 0)
         Analysis.soundStim.stimtype = WNoise;
      else if (strcmp("BPNoise\n", buf) == 0)
         Analysis.soundStim.stimtype = BPNoise;
      else if (strcmp("ToneCombo\n", buf) == 0)
         Analysis.soundStim.stimtype = ToneCombo;
      else {  /* field contains a value */
         for (i=0; i<80; i++) ident[i] = value[i] = '\0';
         i = 0;
         while (buf[i] != ':') {
            ident[i] = buf[i]; i++;
            if (i > 79) {
               uhoh("Error reading data file - no \':\'");
               return(-1);
            }
         }
         i++;  /* skip the colon */
         while (isspace(buf[i]) != 0) i++;  /* skip to the number */
         if (i > 79) {
            uhoh("Error reading data file - no value found");
            return(-1);
```

```
        }
        j = 0;
        while (buf[i] != '\n') {
           value[j] = buf[i];
           if (isdigit(value[j]) == 0) {
              if (value[j] != '-') {
                 uhoh("Error reading data file- digit expected");
                 return(1);
              }
           }
           i++, j++;
        }
        val = atoi(value);
        if (strcmp(ident, "Start") == 0) start = val;
        else if (strcmp(ident, "End") == 0) end = val;
        else if (strcmp(ident, "Step") == 0) step = val;
        else if (strcmp(ident, "ITD") == 0) itd = val;
        else if (strcmp(ident, "IID") == 0) iid = val;
        else if (strcmp(ident, "ABI") == 0) abi = val;
        else if (strcmp(ident, "Fixed") == 0) fixed = val;
        else if (strcmp(ident, "Tone") == 0) {
           Analysis.soundStim.stimtype = Tone;
           Analysis.soundStim.freq = val;
        }
        else if (strcmp(ident, "Low Freq") == 0)
           Analysis.soundStim.bpNoiseStim.startfreq = val;
        else if (strcmp(ident, "High Freq") == 0)
           Analysis.soundStim.bpNoiseStim.endfreq = val;
        else if (strcmp(ident, "Reps") == 0)
           Analysis.soundStim.nreps = val;
        else if (strcmp(ident, "Recording duration") == 0)
           Analysis.soundStim.total_dur = val;
        else if (strcmp(ident, "PreStimulus Delay") == 0)
           Analysis.soundStim.prestim_delay = val;
        else if (strcmp(ident, "Stimulus duration") == 0)
           Analysis.soundStim.stim_dur = val;
     }
     /* read in next line */
     for (i=0; i<80; i++) buf[i] = '\0';
     fgets(buf, 80, readfile);
  }
  /* end of header */
  if (Analysis.tuningCurve == Freq) {
     Analysis.uval.freq_TC.start = start;
     Analysis.uval.freq_TC.end = end;
     Analysis.uval.freq_TC.step = step;
     Analysis.uval.freq_TC.itd = itd;
     Analysis.uval.freq_TC.iid = iid;
     Analysis.uval.freq_TC.abi = abi;
  }
  else if (Analysis.tuningCurve == Itd) {
     Analysis.uval.itd_TC.start = start;
     Analysis.uval.itd_TC.end = end;
     Analysis.uval.itd_TC.step = step;
     Analysis.uval.itd_TC.iid = iid;
     Analysis.uval.itd_TC.abi = abi;
  }
  else if (Analysis.tuningCurve == Iid) {
     Analysis.uval.iid_TC.start = start;
     Analysis.uval.iid_TC.end = end;
     Analysis.uval.iid_TC.step = step;
     Analysis.uval.iid_TC.itd = itd;
     Analysis.uval.iid_TC.abi = abi;
  }
  else if (Analysis.tuningCurve == Fiid) {
     Analysis.uval.fiid_TC.start = start;
     Analysis.uval.fiid_TC.end = end;
     Analysis.uval.fiid_TC.step = step;
     Analysis.uval.fiid_TC.itd = itd;
     Analysis.uval.fiid_TC.fixed = fixed;
  }
}

/* Callbacks of Yes and No response buttons of the overwrite popup */
```

```
void yesCB(widget, name, call)
Widget widget;
caddr_t call, name;
{
   Widget popup = XtParent(XtParent(widget));
   char buf[80];
   void start_run();

   if ((datafile = fopen((char *)name, "w")) == NULL) {
      sprintf(buf, "Unable to open file \"%s\".", (char *)name);
      uhoh(buf);
      FILEOK = False;
   }
   else FILEOK = True;
   XtPopdown(popup);
   XtDestroyWidget(popup);
   XcuDeckRaiseWidget(acqDeck, begin);
   strcpy(datafilename, (char *)name);
   start_run();
}

void noCB(widget, client, call)
Widget widget;
caddr_t client, call;
{
   Widget popup = XtParent(XtParent(widget));

   FILEOK = False;
   XtPopdown(popup);
   XtDestroyWidget(popup);
}
```

```
/*-------------------------------------------------------------
|                                                             |
|   PROGRAM:                                                  |
|                                                             |
|   MODULE:   fstnoise.c                                      |
|                                                             |
|   RELATED                                                   |
|   MODULES:  fstnoise.h                                      |
|                                                             |
|   MACHINE:  Any unix machine with 32-bit word size          |
|                                                             |
|   STARTED:  24-AUG-89        BY:  J.C. Wathey               |
|                                                             |
|   REVISED:  06-SEP-89        BY:  JCW                       |
|                                                             |
|   STATUS:      incomplete or untested                       |
|                     compiles; partly tested                 |
|                     runs; revisions in progress             |
|                  -> runs; stable version                    |
|                                                             |
|        CONTAINS: routine fast_noise() for generating        |
|                  Gaussian white noise with a binary search  |
|                  look-up table algorithm developed by JCW;  |
|                  also includes 2 initialization routines.   |
|                                                             |
-------------------------------------------------------------*/


/*--------------------------------- GLOBAL DEFINITIONS --------------*/

#include <stdio.h>
#include <math.h>

/* The noise values will vary between +/- MAX_DA. */
#define MAX_DA          8192

   /* MAX_WIDTH_IN_SIGMAS is the maximum width bell curve which
   can be represented in the Gaussian look-up table.  It depends
   upon MAX_DA and MAX_RAND in a complex way and, as far as I
   know, cannot be solved for analytically.  The value 4.1608
   was determined empirically (with a binary search) to be
   appropriate for MAX_DA = 2047 and MAX_RAND = 1771874. */

#define MAX_WIDTH_IN_SIGMAS   4.1608

void init_random_sequence();

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

#define UNIFORM_TABLE_SIZE 100
#define GAUSSIAN_TABLE_SIZE   (MAX_DA+2)

   /* The next four constants are appropriate for
   machines using 32-bit integers.  See Numerical
   Recipes in C, section on portable random number
   generators, for values appropriate to other word
   sizes. */

#define IA     2416
#define IC     374441
#define IM     1771875
#define MAX_RAND  (IM-1)

#define random() (uniform_deviate=(uniform_deviate*IA+IC)%IM)

#define Int4      long
#define Int2      short
#define Int1      char

#define Int4u     unsigned long
```

```
#define Int2u     unsigned short
#define Int1u     unsigned char

/*-------------------------------- GLOBAL DECLARATIONS --------------*/

static Int4u   gaussian_table[ GAUSSIAN_TABLE_SIZE ],
         uniform_table[   UNIFORM_TABLE_SIZE ],
       uniform_deviate = 0;

static int  gaussian_table_initialized = FALSE,
       uniform_table_initialized  = FALSE;


/*-------------------------------------------------------------------*/
int fast_noise( noise_buffer, num_noise_pts, frame_size)

        /*  On entry,  noise_buffer points to a buffer large enough  to
        contain  num_noise_pts  short integers.    This  routine  fills
        that  buffer with Gaussian white noise,   varying  between  +/-
        MAX_DA.    The argument frame_size is the number of shorts  per
        frame  and must be > 0.   If frame_size = 1,  every element  of
        noise_buffer  is  filled;  if frame_size  =  2,  only  elements
        0,2,4...  are filled; if frame_size = 3, only elements 0,3,6...
        are  filled,  etc.   For generating 2-channel interlaced noise,
        call  this  routine with frame_size=2 and then copy  the  even-
        index   values  to  the  odd-indexed   values,   shifting   and
        interpolating  as  necessary to create the  desired  interaural
        time difference.  The sequence of values is uniquely determined
        by  the  arguments to the routines  init_random_sequence()  and
        init_gaussian_table(), both of which must be called before this
        routine  can  be  used.   If  either  of  these  initialization
        routines  has  not  been called,  this routine gives  an  error
        message  and  returns  TRUE without changing  the  contents  of
        noise_buffer.  Returns FALSE if no error occurs. */

    short        * noise_buffer;
    Int4u      num_noise_pts;

    register unsigned int frame_size;


{
                                        /*----- functions called -----*/

                                        /*----- extern variables -----*/
    extern Int4u gaussian_table[];
    extern Int4u uniform_deviate;
                                        /*----- local  variables -----*/
    int        error;
    register Int4u low,
                   high,
                   middle,
                   target,
         uniform_i;

    register short * noise_ptr;

                                        /*----- start function -------*/

    if ( error = (frame_size<=0) ) {
   fprintf(stderr,
   "fast_noise: frame_size must be > 0\n");
   return(error);
    }

    if (error = !gaussian_table_initialized) {
   fprintf(stderr,
   "fast_noise: init_gaussian_table() must be called first\n");
   return(error);
    }

    if (error = !uniform_table_initialized) {
   fprintf(stderr,
   "fast_noise: init_random_sequence() must be called first\n");
   return(error);
```

```
    }

    for( noise_ptr =  noise_buffer;
         noise_ptr <  noise_buffer + num_noise_pts;
         noise_ptr += frame_size ) {

        low    = 0;
        high   = GAUSSIAN_TABLE_SIZE;
        middle = (high + low) >> 1;

   uniform_i = (UNIFORM_TABLE_SIZE-1)
          * uniform_deviate / MAX_RAND;

        target = uniform_table[ uniform_i ];
   uniform_table[ uniform_i ] = random();

        while (high-low > 1) {
            if (target > gaussian_table[middle])
                low = middle;
            else
                high = middle;
            middle = (high + low) >> 1;
        }

        if (target & 1)
            low = -low;

        *noise_ptr = (short) low;
    }
    return(error);
}

/*---------------------------------------------------------------------*/
void init_random_sequence( seed_ptr )

        /* Initializes uniform_table[] using *seed_ptr as a seed if  it
        is between 0 and MAX_RAND,  inclusive.   If *seed_ptr is out of
        this  range,  the  routine generates a seed using  the  time(2)
        system call and copies that seed to *seed_ptr.  */

    Int4      * seed_ptr;

{
                                        /*----- functions called -----*/

                                        /*----- extern variables -----*/

    extern Int4u  uniform_table[],
         uniform_deviate;

    extern int    uniform_table_initialized;

                                        /*----- local  variables -----*/
    long  time_value;
    Int4u low_16;
    int   i;
                                        /*----- start function -------*/

    if (*seed_ptr < 0 || *seed_ptr > MAX_RAND) {
   time(&time_value);
   low_16 = time_value & 0xFFFF;
   *seed_ptr = ((unsigned)(low_16 | (low_16<<16))) % IM;
    }

    uniform_deviate = *seed_ptr;

    for( i=0; i < UNIFORM_TABLE_SIZE; i++ )
   random();

    for( i=0; i < UNIFORM_TABLE_SIZE; i++ )
   uniform_table[i] = random();

    uniform_deviate = random();

    uniform_table_initialized = TRUE;
```

```
}

/*-----------------------------------------------------------------*/
int init_gaussian_table( width_in_sigmas )

        /*  Initializes  noise table for generation  of  Gaussian
        noise.  Argument width_in_sigmas gives the width of the bell
        curve used, expressed as the number of standard deviations.
        This routine need only be called once by the application
        program.  The routine fast_noise() can then be used as many
        times as desired, without calling this routine, as long as
        there is no need to change the width of the bell curve.
        Returns  TRUE,  without  initializing  the  table,  if
        width_in_sigmas is not in the useable range; otherwise returns
        FALSE. */

    double width_in_sigmas;

{
                                        /*----- functions called -----*/
    double exp();
                                        /*----- extern variables -----*/
    extern Int4u gaussian_table[];
    extern int  gaussian_table_initialized;

                                        /*----- local  variables -----*/
    int  error,
      i;
    double  bell_curve_integral[ GAUSSIAN_TABLE_SIZE ],
      sum,
      x;
                                        /*----- start function -------*/

    gaussian_table_initialized = FALSE;

    if (error = (width_in_sigmas <= 0.0
       ||
       width_in_sigmas > MAX_WIDTH_IN_SIGMAS )) {

   fprintf(stderr,
   "init_gaussian_table: width_in_sigmas must be >0 and <=%g\n",
   MAX_WIDTH_IN_SIGMAS);
   return(error);
    }

    bell_curve_integral[ 0 ] = 0.0;
    bell_curve_integral[ 1 ] = 0.5;
    for (i = 2; i < GAUSSIAN_TABLE_SIZE; i++) {
   x = (i-1) * width_in_sigmas / MAX_DA;
        bell_curve_integral[i] = exp(-x*x/2.0) +
   bell_curve_integral[i-1];
    }
    sum = bell_curve_integral[ GAUSSIAN_TABLE_SIZE-1 ];

    for (i = 1; i< GAUSSIAN_TABLE_SIZE; i++)
        bell_curve_integral[i] /= sum;

    for (i = 0; i< GAUSSIAN_TABLE_SIZE; i++)
        gaussian_table[i] =
   (Int4u) (MAX_RAND * bell_curve_integral[i] + 0.5);

    if (error = ( gaussian_table[GAUSSIAN_TABLE_SIZE-1]
           <=
           gaussian_table[GAUSSIAN_TABLE_SIZE-2])) {
   fprintf(stderr,
   "init_gaussian_table: use a smaller width_in_sigmas\n");
   return(error);
    }

    gaussian_table_initialized = TRUE;

    return(error);
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

Widget rasterTbl;
Widget stimdeck, curvedeck;
Widget freqTunTbl, itdTbl, iidTbl, fiidTbl;

void getBPlow(), getBPhi();
void getFile(), getFreq(), getITD(), getIID(), getABI();
void itdExposed(), iidExposed(), freqExposed(), fiidExposed();
XtActionsRec actionTable[] = {
   {"getBPlow", getBPlow},
   {"getBPhi", getBPhi},
   {"getFile", getFile},
   {"getFreq", getFreq},
   {"getITD", getITD},
   {"getIID", getIID},
   {"getABI", getABI},
   {"itdExposed", itdExposed},
   {"freqExposed", freqExposed},
   {"iidExposed", iidExposed},
   {"fiidExposed", fiidExposed}
};

makeGUI(argc, argv)
int *argc;
String *argv;
{
   Widget mainForm;
   XGCValues values;
   char *fontname = "6x10";
   Arg args[10];
   Cardinal i;

   XtToolkitInitialize();
   app_context = XtCreateApplicationContext();
   XtAppAddActions(app_context, actionTable, XtNumber(actionTable));
   display = XtOpenDisplay(app_context, NULL, "OASys", "OASys",
      NULL, 0, argc, argv);

   /* define cursors */
   dot = XCreateFontCursor(display, XC_dot);
   crosshair = XCreateFontCursor(display, XC_crosshair);

   i = 0;
   XtSetArg(args[i], XtNinput, True); i++;
   XtSetArg(args[i], XtNallowShellResize, True); i++;
   toplevel = XtAppCreateShell("OASys", "OASys", applicationShellWidgetClass,
      display, args, i);

   i = 0;
   XtSetArg(args[i], XtNformatString, "c."); i++;
   mainForm = XtCreateManagedWidget("mainForm", xcuTblWidgetClass,
      toplevel, args, i);

   makeControlPanel(mainForm);
   makeDisplayPanel(mainForm);

   XtRealizeWidget(toplevel);
   makePlotPanel(mainForm);

   /* now make the GC's & stuff */
   values.foreground = BlackPixel(display, DefaultScreen(display));
   plot_gc = XCreateGC(display, XtWindow(toplevel), GCForeground, &values);
   values.line_style = LineOnOffDash;
   dashed_gc = XCreateGC(display, XtWindow(toplevel),
      GCForeground | GCLineStyle, &values);

   if (!(font_info = XLoadQueryFont(display, fontname))) {
      fprintf(stderr, "Cannot open %s font\n", fontname);
      return;
   }
   XSetFont(display, plot_gc, font_info->fid);
}
```

```
makeDisplayPanel(parent)
Widget parent;
{
   Arg args[10];
   int i;

   i = 0;
   XtSetArg(args[i], XtNformatString, "c."); i++;
   rasterTbl = XtCreateManagedWidget("rasterTbl", xcuTblWidgetClass,
      parent, args, i);

   i = 0;
   XtSetArg(args[i], XtNwidth, 500); i++;
   XtSetArg(args[i], XtNheight, 400); i++;
   raster = XtCreateManagedWidget("raster", simpleWidgetClass,
      rasterTbl, args, i);

   i = 0;
   XtSetArg(args[i], XtNwidth, 500); i++;
   XtSetArg(args[i], XtNheight, 100); i++;
   stimCanvas = XtCreateManagedWidget("stimCanvas", simpleWidgetClass,
      rasterTbl, args, i);

}

makePlotPanel(parent)
Widget parent;
{
   Widget canvasTbl;
   Arg args[10];
   int i, x, y;

   x = XDisplayWidth(display, DefaultScreen(display))/2 - 400;
   y = XDisplayHeight(display, DefaultScreen(display))/2 - 350;

   i = 0;
   XtSetArg(args[i], XtNallowShellResize, True); i++;
   XtSetArg(args[i], XtNx, x); i++;
   XtSetArg(args[i], XtNy, y); i++;
   plotPopup = XtCreatePopupShell("OASys Plots", transientShellWidgetClass,
      parent, args, i);

   i = 0;
   XtSetArg(args[i], XtNformatString, "c c\n c c."); i++;
   canvasTbl = XtCreateManagedWidget("canvasTbl", xcuTblWidgetClass,
      plotPopup, args, i);

   i = 0;
   XtSetArg(args[i], XtNwidth, 400); i++;
   XtSetArg(args[i], XtNheight, 350); i++;
   freqCanvas = XtCreateManagedWidget("freqCanvas", simpleWidgetClass,
      canvasTbl, args, i);
   freqPixmap = XCreatePixmap(display, XtWindow(toplevel), 400, 350,
      DefaultDepth(display, DefaultScreen(display)));
   {
      String trans =
         "<Expose>:  freqExposed()\n";
      XtTranslations table;
      table = XtParseTranslationTable(trans);
      XtOverrideTranslations(freqCanvas, table);
   }

   i = 0;
   XtSetArg(args[i], XtNwidth, 400); i++;
   XtSetArg(args[i], XtNheight, 350); i++;
   itdCanvas = XtCreateManagedWidget("itdCanvas", simpleWidgetClass,
      canvasTbl, args, i);
   itdPixmap = XCreatePixmap(display, XtWindow(toplevel), 400, 350,
      DefaultDepth(display, DefaultScreen(display)));
   {
      String trans =
         "<Expose>:  itdExposed()\n";
      XtTranslations table;
      table = XtParseTranslationTable(trans);
```

```
      XtOverrideTranslations(itdCanvas, table);
   }

   i = 0;
   XtSetArg(args[i], XtNwidth, 400); i++;
   XtSetArg(args[i], XtNheight, 350); i++;
   iidCanvas = XtCreateManagedWidget("iidCanvas", simpleWidgetClass,
      canvasTbl, args, i);
   iidPixmap = XCreatePixmap(display, XtWindow(toplevel), 400, 350,
      DefaultDepth(display, DefaultScreen(display)));
   {
      String trans =
         "<Expose>:  iidExposed()\n";
      XtTranslations table;
      table = XtParseTranslationTable(trans);
      XtOverrideTranslations(iidCanvas, table);
   }

   i = 0;
   XtSetArg(args[i], XtNwidth, 400); i++;
   XtSetArg(args[i], XtNheight, 350); i++;
   fiidCanvas = XtCreateManagedWidget("fiidCanvas", simpleWidgetClass,
      canvasTbl, args, i);
   fiidPixmap = XCreatePixmap(display, XtWindow(toplevel), 400, 350,
      DefaultDepth(display, DefaultScreen(display)));
   {
      String trans =
         "<Expose>:  fiidExposed()\n";
      XtTranslations table;
      table = XtParseTranslationTable(trans);
      XtOverrideTranslations(fiidCanvas, table);
   }
}

makeControlPanel(parent)
Widget parent;
{
   Widget overallTbl, stimTbl, curveTbl, utilTbl;
   Arg args[4];
   int i;

   i = 0;
   XtSetArg(args[i], XtNformatString, "c c c c."); i++;
   utilTbl = XtCreateManagedWidget("utilTbl", xcuTblWidgetClass,
      parent, args, i);
   makeUtilTbl(utilTbl);

   i = 0;
   XtSetArg(args[i], XtNformatString, "c c."); i++;
   overallTbl = XtCreateManagedWidget("overallTbl", xcuTblWidgetClass,
      parent, args, i);

   i = 0;
   XtSetArg(args[i], XtNformatString, "c."); i++;
   stimTbl = XtCreateManagedWidget("stimTbl", xcuTblWidgetClass,
      overallTbl, args, i);
   makeStimTbl(stimTbl);

   i = 0;
   XtSetArg(args[i], XtNformatString, "c."); i++;
   curveTbl = XtCreateManagedWidget("curveTbl", xcuTblWidgetClass,
      overallTbl, args, i);
   makeCurveTbl(curveTbl);
}

makeUtilTbl(parent)
Widget parent;
{
   Widget zero, file, quit;
   Arg args[10];
   int i;
   String translations;
   XtTranslations table;
   void Quit(), getFilename(), stop_data_acq(), zeroAtten();
```

```
    acqDeck = XtCreateManagedWidget("acqDeck", xcuDeckWidgetClass,
       parent, NULL, 0);

    i = 0;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "Start Data\n Acquisition"); i++;
    begin = XtCreateManagedWidget("begin", xcuCommandWidgetClass,
       acqDeck, args, i);
    XtAddCallback(begin, XtNcallback, getFilename, (caddr_t)NULL);

    i = 0;
    XtSetArg(args[i], XtNvalue, ""); i++;
    XtSetArg(args[i], XtNlabel, "Data File:"); i++;
    file = XtCreateManagedWidget("file", dialogWidgetClass,
       acqDeck, args, i);
    translations = "#override\n   <Key>Return: getFile()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(XtNameToWidget(file, "value"), table);
    XtSetKeyboardFocus(file, XtNameToWidget(file, "value"));

    i = 0;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "Stop Data\n Acquisition"); i++;
    XtSetArg(args[i], XtNsensitive, False); i++;
    interrupt = XtCreateManagedWidget("interrupt", xcuCommandWidgetClass,
       parent, args, i);
    XtAddCallback(interrupt, XtNcallback, stop_data_acq, (caddr_t)NULL);

    i = 0;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "Zero\n Attenuators"); i++;
    zero = XtCreateManagedWidget("zero", xcuCommandWidgetClass,
       parent, args, i);
    XtAddCallback(zero, XtNcallback, zeroAtten, (caddr_t)NULL);

    i = 0;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "QUIT"); i++;
    quit = XtCreateManagedWidget("quit", xcuCommandWidgetClass,
       parent, args, i);
    XtAddCallback(quit, XtNcallback, Quit, (caddr_t)NULL);
}

makeCurveTbl(parent)
Widget parent;
{
    Widget curveButtonMgr, curveBtnTbl;
    Widget freqBtn, itdBtn, iidBtn, fiidBtn;
    Widget managed_buttons[4];
    caddr_t managed_values[4];
    Arg args[10];
    int i;
    void curve_changer();

    /* buttons which control which tuning curve is to be taken (and
     * therefore which tuning curve parameters are on top of the deck.
     */
    i = 0;
    XtSetArg(args[i], XtNbmgrType, XcuBMGR_ONE_OF_MANY); i++;
    curveButtonMgr = XtCreateManagedWidget("curveButtonMgr", xcuBmgrWidgetClass,
       parent, args, i);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c c\n c c."); i++;
    curveBtnTbl = XtCreateManagedWidget("curveBtnTbl", xcuTblWidgetClass,
       parent, args, i);

    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "Freq Tuning"); i++;
    XtSetArg(args[i], XtNset, True); i++;
    freqBtn = XtCreateManagedWidget("freqBtn", xcuButtonWidgetClass,
       curveBtnTbl, args, i);
```

```
    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "ITD Tuning"); i++;
    itdBtn = XtCreateManagedWidget("itdBtn", xcuButtonWidgetClass,
       curveBtnTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "IID Tuning"); i++;
    iidBtn = XtCreateManagedWidget("iidBtn", xcuButtonWidgetClass,
       curveBtnTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "FIID Tuning"); i++;
    fiidBtn = XtCreateManagedWidget("fiidBtn", xcuButtonWidgetClass,
       curveBtnTbl, args, i);

    managed_buttons[0] = freqBtn; managed_values[0] = (caddr_t)"freqBtn";
    managed_buttons[1] = itdBtn;  managed_values[1] = (caddr_t)"itdBtn";
    managed_buttons[2] = iidBtn;  managed_values[2] = (caddr_t)"iidBtn";
    managed_buttons[3] = fiidBtn; managed_values[3] = (caddr_t)"fiidBtn";

    curvedeck = XtCreateManagedWidget("curvedeck", xcuDeckWidgetClass,
       parent, NULL, 0);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c."); i++;
    freqTunTbl = XtCreateManagedWidget("freqTunTbl", xcuTblWidgetClass,
       curvedeck, args, i);
    makeFreqTbl(freqTunTbl);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c."); i++;
    itdTbl = XtCreateManagedWidget("itdTbl", xcuTblWidgetClass,
       curvedeck, args, i);
    makeItdTbl(itdTbl);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c."); i++;
    iidTbl = XtCreateManagedWidget("iidTbl", xcuTblWidgetClass,
       curvedeck, args, i);
    makeIidTbl(iidTbl);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c."); i++;
    fiidTbl = XtCreateManagedWidget("fiidTbl", xcuTblWidgetClass,
       curvedeck, args, i);
    makeFiidTbl(fiidTbl);

    XcuBmgrManage(curveButtonMgr, managed_buttons, managed_values, FOUR);
    XtAddCallback(curveButtonMgr, XtNsetCallback, curve_changer, NULL);
}

/* Make frequency tuning curve parameter table */
makeFreqTbl(parent)
Widget parent;
{
    Widget start, end, step, itd, iid, abi;
    Widget val;
    Arg args[10];
    int i;
    String translations;
    XtTranslations table;
    void getFreq();
    char buf[16];

    i = 0;
    sprintf(buf, "%d", FreqCurve.start);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "Starting Freq (Hz):"); i++;
    start = XtCreateManagedWidget("start", dialogWidgetClass,
```

```
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(start, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
XtSetArg(args[i], XtNwidth, 75); i++;
val = XtNameToWidget(start, "value");
XtSetValues(val, args, i);
translations = "#override\n   <Key>Return: getFreq()\n";
table = XtParseTranslationTable(translations);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(start, val);

i = 0;
sprintf(buf, "%d", FreqCurve.end);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "Ending Freq (Hz):  "); i++;
end = XtCreateManagedWidget("end", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(end, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
XtSetArg(args[i], XtNwidth, 75); i++;
val = XtNameToWidget(end, "value");
XtSetValues(val, args, i);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(end, val);

i = 0;
sprintf(buf, "%d", FreqCurve.step);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "Step Freq (Hz):    "); i++;
step = XtCreateManagedWidget("step", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(step, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(step, "value");
XtSetValues(val, args, i);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(step, val);

i = 0;
sprintf(buf, "%d", FreqCurve.itd);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "ITD (usec):        "); i++;
itd = XtCreateManagedWidget("itd", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(itd, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(itd, "value");
XtSetValues(val, args, i);
translations = "#override\n   <Key>Return: getITD()\n";
table = XtParseTranslationTable(translations);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(itd, val);

i = 0;
sprintf(buf, "%d", FreqCurve.iid);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "IID (dB):          "); i++;
iid = XtCreateManagedWidget("iid", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(iid, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(iid, "value");
```

```
    XtSetValues(val, args, i);
    translations = "#override\n    <Key>Return: getIID()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(iid, val);

    i = 0;
    sprintf(buf, "%d", FreqCurve.abi);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "ABI (dB):          "); i++;
    abi = XtCreateManagedWidget("abi", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(abi, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(abi, "value");
    XtSetValues(val, args, i);
    translations = "#override\n    <Key>Return: getABI()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(abi, val);
}

makeItdTbl(parent)
Widget parent;
{
    Widget start, end, step, iid, abi;
    Widget val;
    Arg args[10];
    int i;
    String translations;
    XtTranslations table;
    char buf[16];

    i = 0;
    sprintf(buf, "%d", ItdCurve.start);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "Starting ITD (usec):"); i++;
    start = XtCreateManagedWidget("start", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(start, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(start, "value");
    XtSetValues(val, args, i);
    translations = "#override\n    <Key>Return: getITD()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(start, val);

    i = 0;
    sprintf(buf, "%d", ItdCurve.end);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "Ending ITD (usec):  "); i++;
    end = XtCreateManagedWidget("end", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(end, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(end, "value");
    XtSetValues(val, args, i);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(end, val);

    i = 0;
    sprintf(buf, "%d", ItdCurve.step);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "ITD step (usec):    "); i++;
    step = XtCreateManagedWidget("step", dialogWidgetClass,
```

```
      parent, args, i);
   i = 0;
   XtSetArg(args[i], XtNwidth, 75); i++;
   XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(step, "label")); i++;
   XtSetArg(args[i], XtNfromVert, NULL); i++;
   XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
   val = XtNameToWidget(step, "value");
   XtSetValues(val, args, i);
   XtOverrideTranslations(val, table);
   XtSetKeyboardFocus(step, val);

   i = 0;
   sprintf(buf, "%d", ItdCurve.iid);
   XtSetArg(args[i], XtNvalue, buf); i++;
   XtSetArg(args[i], XtNlabel, "IID (dB):           "); i++;
   iid = XtCreateManagedWidget("iid", dialogWidgetClass,
      parent, args, i);
   i = 0;
   XtSetArg(args[i], XtNwidth, 75); i++;
   XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(iid, "label")); i++;
   XtSetArg(args[i], XtNfromVert, NULL); i++;
   XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
   val = XtNameToWidget(iid, "value");
   XtSetValues(val, args, i);
   translations = "#override\n   <Key>Return: getIID()\n";
   table = XtParseTranslationTable(translations);
   XtOverrideTranslations(val, table);
   XtSetKeyboardFocus(iid, val);

   i = 0;
   sprintf(buf, "%d", ItdCurve.abi);
   XtSetArg(args[i], XtNvalue, buf); i++;
   XtSetArg(args[i], XtNlabel, "ABI (dB):           "); i++;
   abi = XtCreateManagedWidget("abi", dialogWidgetClass,
      parent, args, i);
   i = 0;
   XtSetArg(args[i], XtNwidth, 75); i++;
   XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(abi, "label")); i++;
   XtSetArg(args[i], XtNfromVert, NULL); i++;
   XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
   val = XtNameToWidget(abi, "value");
   XtSetValues(val, args, i);
   translations = "#override\n   <Key>Return: getABI()\n";
   table = XtParseTranslationTable(translations);
   XtOverrideTranslations(val, table);
   XtSetKeyboardFocus(abi, val);
}

makeIidTbl(parent)
Widget parent;
{
   Widget start, end, step, itd, abi;
   Widget val;
   Arg args[10];
   int i;
   String translations;
   XtTranslations table;
   char buf[16];

   i = 0;
   sprintf(buf, "%d", IidCurve.start);
   XtSetArg(args[i], XtNvalue, buf); i++;
   XtSetArg(args[i], XtNlabel, "Starting IID (dB):"); i++;
   start = XtCreateManagedWidget("start", dialogWidgetClass,
      parent, args, i);
   i = 0;
   XtSetArg(args[i], XtNwidth, 75); i++;
   XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(start, "label")); i++;
   XtSetArg(args[i], XtNfromVert, NULL); i++;
   XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
   val = XtNameToWidget(start, "value");
   XtSetValues(val, args, i);
   translations = "#override\n   <Key>Return: getIID()\n";
   table = XtParseTranslationTable(translations);
   XtOverrideTranslations(val, table);
```

```
    XtSetKeyboardFocus(start, val);

    i = 0;
    sprintf(buf, "%d", IidCurve.end);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "Ending IID (dB):  "); i++;
    end = XtCreateManagedWidget("end", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(end, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(end, "value");
    XtSetValues(val, args, i);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(end, val);

    i = 0;
    sprintf(buf, "%d", IidCurve.step);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "IID step (dB):    "); i++;
    step = XtCreateManagedWidget("step", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(step, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(step, "value");
    XtSetValues(val, args, i);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(step, val);

    i = 0;
    sprintf(buf, "%d", IidCurve.itd);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "ITD (usec):       "); i++;
    itd = XtCreateManagedWidget("itd", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(itd, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(itd, "value");
    XtSetValues(val, args, i);
    translations = "#override\n   <Key>Return: getITD()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(itd, val);

    i = 0;
    sprintf(buf, "%d", IidCurve.abi);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "ABI (dB):         "); i++;
    abi = XtCreateManagedWidget("abi", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(abi, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(abi, "value");
    XtSetValues(val, args, i);
    translations = "#override\n   <Key>Return: getABI()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(abi, val);
}

makeFiidTbl(parent)
Widget parent;
{
    Widget start, end, step, fixed, itd;
```

```
Widget val;
Arg args[10];
int i;
String translations;
XtTranslations table;
char buf[16];

i = 0;
sprintf(buf, "%d", FiidCurve.start);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "Starting IID (dB):"); i++;
start = XtCreateManagedWidget("start", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(start, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(start, "value");
XtSetValues(val, args, i);
translations = "#override\n   <Key>Return: getIID()\n";
table = XtParseTranslationTable(translations);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(start, val);

i = 0;
sprintf(buf, "%d", FiidCurve.end);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "Ending IID (dB):  "); i++;
end = XtCreateManagedWidget("end", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(end, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(end, "value");
XtSetValues(val, args, i);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(end, val);

i = 0;
sprintf(buf, "%d", FiidCurve.step);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "IID step (dB):    "); i++;
step = XtCreateManagedWidget("step", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(step, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(step, "value");
XtSetValues(val, args, i);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(step, val);

i = 0;
sprintf(buf, "%d", FiidCurve.fixed);
XtSetArg(args[i], XtNvalue, buf); i++;
XtSetArg(args[i], XtNlabel, "Fixed Level (dB): "); i++;
fixed = XtCreateManagedWidget("fixed", dialogWidgetClass,
   parent, args, i);
i = 0;
XtSetArg(args[i], XtNwidth, 75); i++;
XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(fixed, "label")); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
val = XtNameToWidget(fixed, "value");
XtSetValues(val, args, i);
translations = "#override\n   <Key>Return: getIID()\n";
table = XtParseTranslationTable(translations);
XtOverrideTranslations(val, table);
XtSetKeyboardFocus(fixed, val);
```

```
    i = 0;
    sprintf(buf, "%d", FiidCurve.itd);
    XtSetArg(args[i], XtNvalue, buf); i++;
    XtSetArg(args[i], XtNlabel, "ITD (usec):       "); i++;
    itd = XtCreateManagedWidget("itd", dialogWidgetClass,
       parent, args, i);
    i = 0;
    XtSetArg(args[i], XtNwidth, 75); i++;
    XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(itd, "label")); i++;
    XtSetArg(args[i], XtNfromVert, NULL); i++;
    XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
    val = XtNameToWidget(itd, "value");
    XtSetValues(val, args, i);
    translations = "#override\n   <Key>Return: getITD()\n";
    table = XtParseTranslationTable(translations);
    XtOverrideTranslations(val, table);
    XtSetKeyboardFocus(itd, val);
}

Widget psDelayLbl, recDurLbl, stimDurLbl, repsLbl;
Widget freqTbl, freqLbl, wnoiseLbl, BandpassTbl;

/* interface to fill the STIM struct */
makeStimTbl(parent)
Widget parent;
{
    Widget buttonMgr, buttonTbl;
    Widget toneBtn, whiteNoiseBtn, bpNoiseBtn, toneComboBtn;
    Widget managed_buttons[4];
    caddr_t managed_values[4];
    Widget freqSBHund, freqSBTens;
    Widget yaTbl;
    Widget psDelaySB;
    Widget recDurSB;
    Widget stimDurSB;
    Widget incrReps, decrReps;
    Arg args[5];
    char buf[80];
    int i;
    void stim_changer();
    void freqJumpProcHund(), freqJumpProcTens();
    void stimDurJumpProc(), recDurJumpProc(), psDelayJumpProc();
    void addRep(), subRep();

    /* buttons which control which stimulus is to be used (and therefore
     * which stimulus parameters are on top of the deck.
     */
    i = 0;
    XtSetArg(args[i], XtNbmgrType, XcuBMGR_ONE_OF_MANY); i++;
    buttonMgr = XtCreateManagedWidget("buttonMgr", xcuBmgrWidgetClass,
       parent, args, i);

    i = 0;
    XtSetArg(args[i], XtNformatString, "c c c c."); i++;
    buttonTbl = XtCreateManagedWidget("buttonTbl", xcuTblWidgetClass,
       parent, args, i);

    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "Tone"); i++;
    XtSetArg(args[i], XtNset, True); i++;
    toneBtn = XtCreateManagedWidget("toneBtn", xcuButtonWidgetClass,
       buttonTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    XtSetArg(args[i], XtNlabel, "White Noise"); i++;
    whiteNoiseBtn = XtCreateManagedWidget("whiteNoiseBtn", xcuButtonWidgetClass,
       buttonTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNresizable, True); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
```

```
XtSetArg(args[i], XtNlabel, "Bandpass Noise"); i++;
bpNoiseBtn = XtCreateManagedWidget("bpNoiseBtn", xcuButtonWidgetClass,
   buttonTbl, args, i);

i = 0;
XtSetArg(args[i], XtNresizable, True); i++;
XtSetArg(args[i], XtNcursor, dot); i++;
XtSetArg(args[i], XtNlabel, "Tone Combo"); i++;
toneComboBtn = XtCreateManagedWidget("toneComboBtn", xcuButtonWidgetClass,
   buttonTbl, args, i);

managed_buttons[0] = toneBtn;
managed_buttons[1] = whiteNoiseBtn;
managed_buttons[2] = bpNoiseBtn;
managed_buttons[3] = toneComboBtn;

managed_values[0] = (caddr_t)"tone";
managed_values[1] = (caddr_t)"white";
managed_values[2] = (caddr_t)"bandpass";
managed_values[3] = (caddr_t)"tonecombo";

XcuBmgrManage(buttonMgr, managed_buttons, managed_values, FOUR);
XtAddCallback(buttonMgr, XtNsetCallback, stim_changer, NULL);

stimdeck = XtCreateManagedWidget("stimdeck", xcuDeckWidgetClass,
   parent, NULL, 0);

i = 0;
XtSetArg(args[i], XtNformatString, "c."); i++;
freqTbl = XtCreateManagedWidget("freqTbl", xcuTblWidgetClass,
   stimdeck, args, i);

i = 0;
sprintf(buf, "Frequency: %d Hz", stimulus.freq);
XtSetArg(args[i], XtNlabel, buf); i++;
freqLbl = XtCreateManagedWidget("freqLbl", xcuLabelWidgetClass,
   freqTbl, args, i);

i = 0;
XtSetArg(args[i], XtNorientation, XtorientHorizontal); i++;
freqSBHund = XtCreateManagedWidget("freqSBHund", scrollbarWidgetClass,
   freqTbl, args, i);
XtAddCallback(freqSBHund, XtNjumpProc, freqJumpProcHund, (caddr_t)NULL);

i = 0;
XtSetArg(args[i], XtNorientation, XtorientHorizontal); i++;
freqSBTens = XtCreateManagedWidget("freqSBTens", scrollbarWidgetClass,
   freqTbl, args, i);
XtAddCallback(freqSBTens, XtNjumpProc, freqJumpProcTens, (caddr_t)NULL);

i = 0;
XtSetArg(args[i], XtNlabel, "pfffffffffff"); i++;
XtSetArg(args[i], XtNshadow, True); i++;
wnoiseLbl = XtCreateManagedWidget("wnoiseLbl", xcuLabelWidgetClass,
   stimdeck, args, i);

makeBandPassMenu(stimdeck);

i = 0;
XtSetArg(args[i], XtNformatString, "c\n c\n c\n c\n c\n c\n c c c."); i++;
yaTbl = XtCreateManagedWidget("yaTbl", xcuTblWidgetClass,
   parent, args, i);

i = 0;
sprintf(buf, "Record Duration: %d msec", stimulus.total_dur);
XtSetArg(args[i], XtNlabel, buf); i++;
recDurLbl = XtCreateManagedWidget("recDurLbl", xcuLabelWidgetClass,
   yaTbl, args, i);

i = 0;
XtSetArg(args[i], XtNorientation, XtorientHorizontal); i++;
recDurSB = XtCreateManagedWidget("recDurSB", scrollbarWidgetClass,
   yaTbl, args, i);
XtAddCallback(recDurSB, XtNjumpProc, recDurJumpProc, (caddr_t)NULL);
```

```
    i = 0;
    sprintf(buf, "Pre-stimulus Delay: %d msec", stimulus.prestim_delay);
    XtSetArg(args[i], XtNlabel, buf); i++;
    psDelayLbl = XtCreateManagedWidget("psDelayLbl", xcuLabelWidgetClass,
       yaTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNorientation, XtorientHorizontal); i++;
    psDelaySB = XtCreateManagedWidget("psDelaySB", scrollbarWidgetClass,
       yaTbl, args, i);
    XtAddCallback(psDelaySB, XtNjumpProc, psDelayJumpProc, (caddr_t)NULL);

    i = 0;
    sprintf(buf, "Stimulus Duration: %d msec", stimulus.stim_dur);
    XtSetArg(args[i], XtNlabel, buf); i++;
    stimDurLbl = XtCreateManagedWidget("stimDurLbl", xcuLabelWidgetClass,
       yaTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNorientation, XtorientHorizontal); i++;
    stimDurSB = XtCreateManagedWidget("stimDurSB", scrollbarWidgetClass,
       yaTbl, args, i);
    XtAddCallback(stimDurSB, XtNjumpProc, stimDurJumpProc, (caddr_t)NULL);

    i = 0;
    sprintf(buf, "Stimulus Reps: %d", stimulus.nreps);
    XtSetArg(args[i], XtNlabel, buf); i++;
    repsLbl = XtCreateManagedWidget("repsLbl", xcuLabelWidgetClass,
       yaTbl, args, i);

    i = 0;
    XtSetArg(args[i], XtNlabel, "+"); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    incrReps = XtCreateManagedWidget("incrReps", xcuCommandWidgetClass,
       yaTbl, args, i);
    XtAddCallback(incrReps, XtNcallback, addRep, (caddr_t)NULL);

    i = 0;
    XtSetArg(args[i], XtNlabel, "-"); i++;
    XtSetArg(args[i], XtNcursor, dot); i++;
    decrReps = XtCreateManagedWidget("decrReps", xcuCommandWidgetClass,
       yaTbl, args, i);
    XtAddCallback(decrReps, XtNcallback, subRep, (caddr_t)NULL);
}

static void stim_changer(w, client, call)
Widget w;
caddr_t client, call;
{
    if (strcmp("tone", (String)call) == 0) {
       XcuDeckRaiseWidget(stimdeck, freqTbl);
       stimulus.stimtype = Tone;
    }
    else if (strcmp("white", (String)call) == 0) {
       XcuDeckRaiseWidget(stimdeck, wnoiseLbl);
       stimulus.stimtype = WNoise;
    }
    else if (strcmp("bandpass", (String)call) == 0) {
       XcuDeckRaiseWidget(stimdeck, BandpassTbl);
       stimulus.stimtype = BPNoise;
    }
    else if (strcmp("tonecombo", (String)call) == 0) {
       stimulus.stimtype = ToneCombo;
    }
}

static void curve_changer(w, client, call)
Widget w;
caddr_t client, call;
{
    if (strcmp("freqBtn", (String)call) == 0) {
       XcuDeckRaiseWidget(curvedeck, freqTunTbl);
       curvetype = Freq;
    }
    else if (strcmp("itdBtn", (String)call) == 0) {
```

```
        XcuDeckRaiseWidget(curvedeck, itdTbl);
        curvetype = Itd;
    }
    else if (strcmp("iidBtn", (String)call) == 0) {
        XcuDeckRaiseWidget(curvedeck, iidTbl);
        curvetype = Iid;
    }
    else if (strcmp("fiidBtn", (String)call) == 0) {
        XcuDeckRaiseWidget(curvedeck, fiidTbl);
        curvetype = Fiid;
    }
}

int hundfreq = 500;
int tensfreq = 0;

void freqJumpProcHund(scrollbar, client, percent_ptr)
Widget scrollbar;
caddr_t client;
caddr_t percent_ptr;
{
    float percent = *(float *)percent_ptr;  /* 0.0 to 1.0 */
    Arg args[2];
    char buf[80];

    /* ok, we want the frequency to be adjustable between 500 Hz
     * and 10,000 Hz
     * This scrollbar controls the hundred of Hz
     */
    hundfreq = 100*(int)(90.0*percent) + 500;
    stimulus.freq = hundfreq + tensfreq;
    sprintf(buf, "Frequency: %d Hz", stimulus.freq);
    XtSetArg(args[0], XtNlabel, buf);
    XtSetValues(freqLbl, args, ONE);
}

void freqJumpProcTens(scrollbar, client, percent_ptr)
Widget scrollbar;
caddr_t client;
caddr_t percent_ptr;
{
    float percent = *(float *)percent_ptr;
    Arg args[2];
    char buf[80];

    /* this scrollbar controls tens of Hz */
    tensfreq = 10*(int)(10.0*percent);
    stimulus.freq = hundfreq + tensfreq;
    sprintf(buf, "Frequency: %d Hz", stimulus.freq);
    XtSetArg(args[0], XtNlabel, buf);
    XtSetValues(freqLbl, args, ONE);
}

void recDurJumpProc(scrollbar, client, percent_ptr)
Widget scrollbar;
caddr_t client, percent_ptr;
{
    Widget parent = XtParent(scrollbar);
    float percent = *(float *)percent_ptr;
    int dur;
    Arg args[2];
    char buf[80];

    /* recording duration varies between stimulus duration and 500 msec */
    dur = 10*(int)(50.0*percent);
    if (dur < stimulus.stim_dur) dur = stimulus.stim_dur;
    stimulus.total_dur = dur;
    sprintf(buf, "Record Duration: %d msec", stimulus.total_dur);
    XtSetArg(args[0], XtNlabel, buf);
    XtSetValues(XtNameToWidget(parent, "recDurLbl"), args, ONE);
}

void stimDurJumpProc(scrollbar, client, percent_ptr)
Widget scrollbar;
caddr_t client, percent_ptr;
```

```
{
   Widget parent = XtParent(scrollbar);
   float percent = *(float *)percent_ptr;
   int dur;
   Arg args[2];
   char buf[80];

   /* stimulus duration varies between 0 and 500 msec
    * 0 basically means no stimulus
    */
   dur = 10*(int)(50.0*percent);
   stimulus.stim_dur = dur;
   sprintf(buf, "Stimulus Duration: %d msec", stimulus.stim_dur);
   XtSetArg(args[0], XtNlabel, buf);
   XtSetValues(XtNameToWidget(parent, "stimDurLbl"), args, ONE);
   /* adjust total_dur if necessary */
   if (dur > stimulus.total_dur) {
      stimulus.total_dur = dur;
      sprintf(buf, "Record Duration (msec): %d", stimulus.total_dur);
      XtSetArg(args[0], XtNlabel, buf);
      XtSetValues(XtNameToWidget(parent, "recDurLbl"), args, ONE);
   }
}

void psDelayJumpProc(scrollbar, client, percent_ptr)
Widget scrollbar;
caddr_t client, percent_ptr;
{
   Widget parent = XtParent(scrollbar);
   float percent = *(float *)percent_ptr;
   int dur;
   Arg args[2];
   char buf[80];

   /* prestim_delay + stim_dur should be <= total_dur */
   dur = 10*(int)(25.0*percent);
   if ((stimulus.stim_dur + dur) > stimulus.total_dur) {
      /* adjust it */
      dur = stimulus.total_dur - stimulus.stim_dur;
   }
   stimulus.prestim_delay = dur;
   sprintf(buf, "Pre-stimulus Delay: %d msec", stimulus.prestim_delay);
   XtSetArg(args[0], XtNlabel, buf);
   XtSetValues(XtNameToWidget(parent, "psDelayLbl"), args, ONE);
}

void addRep(w, client, call)
Widget w;       /* the incrRep command widget */
caddr_t client, call;
{
   char buf[80];
   Arg args[2];

   ++stimulus.nreps;
   sprintf(buf, "Stimulus Reps: %d", stimulus.nreps);
   XtSetArg(args[0], XtNlabel, buf);
   XtSetValues(repsLbl, args, ONE);
}

void subRep(w, client, call)
Widget w;
caddr_t client, call;
{
   char buf[80];
   Arg args[2];

   if (stimulus.nreps > 1)
      --stimulus.nreps;
   sprintf(buf, "Stimulus Reps: %d", stimulus.nreps);
   XtSetArg(args[0], XtNlabel, buf);
   XtSetValues(repsLbl, args, ONE);
}

makeBandPassMenu(parent)
Widget parent;
```

```
{
   Widget lowDialog, hiDialog, synth;
   Arg args[5];
   int i;
   char buf[16];
   void makeBandpassNoise();

   i = 0;
   XtSetArg(args[i], XtNformatString, "c c\n c."); i++;
   BandpassTbl = XtCreateManagedWidget("BandpassTbl", xcuTblWidgetClass,
      parent, args, i);

   i = 0;
   sprintf(buf, "%d", stimulus.bpNoiseStim.startfreq);
   XtSetArg(args[i], XtNvalue, buf); i++;
   XtSetArg(args[i], XtNlabel, "Low Freq (Hz):"); i++;
   lowDialog = XtCreateManagedWidget("lowDialog", dialogWidgetClass,
      BandpassTbl, args, i);
   i = 0;
   XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(lowDialog, "label")); i++;
   XtSetArg(args[i], XtNfromVert, NULL); i++;
   XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
   XtSetArg(args[i], XtNwidth, 50); i++;
   XtSetValues(XtNameToWidget(lowDialog, "value"), args, i);
   {
      String lowTrans =
         "#override\n   <Key>Return: getBPlow()\n";
      XtTranslations lowtable;
      lowtable = XtParseTranslationTable(lowTrans);
      XtOverrideTranslations(XtNameToWidget(lowDialog, "value"), lowtable);
   }
   XtSetKeyboardFocus(lowDialog, XtNameToWidget(lowDialog, "value"));

   i = 0;
   sprintf(buf, "%d", stimulus.bpNoiseStim.endfreq);
   XtSetArg(args[i], XtNvalue, buf); i++;
   XtSetArg(args[i], XtNlabel, "High Freq (Hz):"); i++;
   hiDialog = XtCreateManagedWidget("hiDialog", dialogWidgetClass,
      BandpassTbl, args, i);
   i = 0;
   XtSetArg(args[i], XtNfromHoriz, XtNameToWidget(hiDialog, "label")); i++;
   XtSetArg(args[i], XtNfromVert, NULL); i++;
   XtSetArg(args[i], XtNinsertPosition, strlen(buf)); i++;
   XtSetArg(args[i], XtNwidth, 50); i++;
   XtSetValues(XtNameToWidget(hiDialog, "value"), args, i);
   {
      String hiTrans =
         "#override\n   <Key>Return: getBPhi()\n";
      XtTranslations hitable;
      hitable = XtParseTranslationTable(hiTrans);
      XtOverrideTranslations(XtNameToWidget(hiDialog, "value"), hitable);
   }
   XtSetKeyboardFocus(hiDialog, XtNameToWidget(hiDialog, "value"));

   i = 0;
   XtSetArg(args[i], XtNlabel, "Synthesize"); i++;
   XtSetArg(args[i], XtNcursor, dot); i++;
   synth = XtCreateManagedWidget("synth", xcuCommandWidgetClass,
      BandpassTbl, args, i);
   XtAddCallback(synth, XtNcallback, makeBandpassNoise, (caddr_t)NULL);
}

void getBPlow(widget, event, params, numparams)
Widget widget;          /* the value widget of the dialog */
XEvent *event;
String *params;
Cardinal *numparams;
{
   Widget dialog = XtParent(widget);
   char *value;
   int freq;
   Arg args[2];
   char buf[80];

   value = XawDialogGetValueString(dialog);
```

```
    freq = atoi(value);
    if (freq > 1000 && freq < 10000) {
        stimulus.bpNoiseStim.startfreq = freq;
    }
    else {
        sprintf(buf, "%d", stimulus.bpNoiseStim.endfreq);
        XtSetArg(args[0], XtNvalue, buf);
        XtSetValues(dialog, args, ONE);
    }
}

void getBPhi(widget, event, params, numparams)
Widget widget;          /* the value widget of the dialog */
XEvent *event;
String *params;
Cardinal *numparams;
{
    Widget dialog = XtParent(widget);
    char *value;
    int freq;
    Arg args[2];
    char buf[80];

    value = XawDialogGetValueString(dialog);
    freq = atoi(value);
    if (freq > 1000 && freq < 10000) {
        stimulus.bpNoiseStim.endfreq = freq;
    }
    else {
        sprintf(buf, "%d", stimulus.bpNoiseStim.endfreq);
        XtSetArg(args[0], XtNvalue, buf);
        XtSetValues(dialog, args, ONE);
    }
}

void getFreq(widget, event, params, numparams)
Widget widget;
XEvent *event;
String *params;
Cardinal *numparams;
{
    Widget dialog = XtParent(widget);
    Widget parent = XtParent(dialog);
    Window window;
    char *value;
    int freq;

    value = XawDialogGetValueString(dialog);
    freq = atoi(value);
    if (dialog == (XtNameToWidget(parent, "start"))) {
        if (freq < 500) freq = 500;
        if (freq > 10000) freq = 10000;
        FreqCurve.start = freq;
        window = XtWindow(XtNameToWidget(parent, "end"));
        XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
    }
    else if (dialog == (XtNameToWidget(parent, "end"))) {
        if (freq < 500) freq = 500;
        if (freq > 10000) freq = 10000;
        FreqCurve.end = freq;
        window = XtWindow(XtNameToWidget(parent, "step"));
        XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
    }
    else if (dialog == (XtNameToWidget(parent, "step"))) {
        FreqCurve.step = freq;
        window = XtWindow(XtNameToWidget(parent, "itd"));
        XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
    }
}

void getITD(widget, event, params, numparams)
Widget widget;          /* value widget of the dialog which invoked this proc */
XEvent *event;
String *params;
Cardinal *numparams;
```

```
{
   Widget dialog = XtParent(widget);
   Widget parent = XtParent(dialog);
   Window window;
   char *value;
   int itd;

   value = XawDialogGetValueString(dialog);
   itd = atoi(value);
   if (dialog == (XtNameToWidget(parent, "start"))) {
      ItdCurve.start = itd;
      window = XtWindow(XtNameToWidget(parent, "end"));
      XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
   }
   else if (dialog == (XtNameToWidget(parent, "end"))) {
      ItdCurve.end = itd;
      window = XtWindow(XtNameToWidget(parent, "step"));
      XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
   }
   else if (dialog == (XtNameToWidget(parent, "step"))) {
      ItdCurve.step = itd;
      window = XtWindow(XtNameToWidget(parent, "iid"));
      XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
   }
   else if (dialog == (XtNameToWidget(parent, "itd"))) {
      /* dialog isn't from itdTbl */
      if (parent == iidTbl) {
         IidCurve.itd = itd;
         window = XtWindow(XtNameToWidget(parent, "abi"));
         XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
      }
      else if (parent == freqTunTbl) {
         FreqCurve.itd = itd;
         window = XtWindow(XtNameToWidget(parent, "iid"));
         XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
      }
      else if (parent == fiidTbl) {
         FiidCurve.itd = itd;
         window = XtWindow(XtNameToWidget(parent, "start"));
         XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
      }
   }
}

void getIID(widget, event, params, numparams)
Widget widget;          /* value widget of the dialog which invoked this proc */
XEvent *event;
String *params;
Cardinal *numparams;
{
   Widget dialog = XtParent(widget);
   Widget parent = XtParent(dialog);
   Window window;
   char *value;
   int iid;

   value = XawDialogGetValueString(dialog);
   iid = atoi(value);
   if (parent == iidTbl) {
      if (dialog == (XtNameToWidget(parent, "start"))) {
         IidCurve.start = iid;
         window = XtWindow(XtNameToWidget(parent, "end"));
         XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
      }
      else if (dialog == (XtNameToWidget(parent, "end"))) {
         IidCurve.end = iid;
         window = XtWindow(XtNameToWidget(parent, "step"));
         XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
      }
      else if (dialog == (XtNameToWidget(parent, "step"))) {
         IidCurve.step = iid;
         window = XtWindow(XtNameToWidget(parent, "itd"));
         XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
      }
   }
```

```
    else if (parent == fiidTbl) {
        if (dialog == (XtNameToWidget(parent, "start"))) {
            FiidCurve.start = iid;
            window = XtWindow(XtNameToWidget(parent, "end"));
            XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
        }
        else if (dialog == (XtNameToWidget(parent, "end"))) {
            FiidCurve.end = iid;
            window = XtWindow(XtNameToWidget(parent, "step"));
            XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
        }
        else if (dialog == (XtNameToWidget(parent, "step"))) {
            FiidCurve.step = iid;
            window = XtWindow(XtNameToWidget(parent, "fixed"));
            XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
        }
        else if (dialog == (XtNameToWidget(parent, "fixed"))) {
            FiidCurve.fixed = iid;
            window = XtWindow(XtNameToWidget(parent, "itd"));
            XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
        }
    }
    else if (dialog == (XtNameToWidget(parent, "iid"))) {
        /* dialog isn't from iidTbl or fiidTbl */
        if (parent == itdTbl) {
            ItdCurve.iid = iid;
            window = XtWindow(XtNameToWidget(parent, "abi"));
            XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
        }
        else if (parent == freqTunTbl) {
            FreqCurve.iid = iid;
            window = XtWindow(XtNameToWidget(parent, "abi"));
            XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
        }
    }

}

void getABI(widget, event, params, numparams)
Widget widget;          /* value widget of the dialog which invoked this proc */
XEvent *event;
String *params;
Cardinal *numparams;
{
    Widget dialog = XtParent(widget);
    Widget parent = XtParent(dialog);
    Window window;
    char *value;
    int abi;

    value = XawDialogGetValueString(dialog);
    abi = atoi(value);
    if (parent == freqTunTbl) {
        FreqCurve.abi = abi;
        window = XtWindow(XtNameToWidget(parent, "start"));
        XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
    }
    else if (parent == itdTbl) {
        ItdCurve.abi = abi;
        window = XtWindow(XtNameToWidget(parent, "start"));
        XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
    }
    else if (parent == iidTbl) {
        IidCurve.abi = abi;
        window = XtWindow(XtNameToWidget(parent, "start"));
        XWarpPointer(display, None, window, 0, 0, 0, 0, 50, 20);
    }
}

void zeroAtten(w, client, call)
Widget w;
caddr_t client, call;
{
    setRack(0, 0, 0);
}
```

```
void Quit(w, client, call)
Widget w;
caddr_t client, call;
{
#ifdef mc700
   mrclosall();
#endif
   setRack(0, 0, 0);
   XtUnmapWidget(toplevel);
   exit(0);
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

short left[50000];
short right[50000];
short sound[50000];

/* for freq tuning curves */
loadFreq(freq)
int freq;
{
   register int i, j;
   int numpts, delay, itdoffset;
   int l_level, r_level;
   int temp;
   void ShapeWaveform();

   /* save whatever frequency we've been using for itd, iid, etc. */
   temp = stimulus.freq;
   stimulus.freq = freq;
   bzero(stim.buf, sizeof(stim.buf));
   bzero(left, sizeof(left));
   bzero(right, sizeof(right));
   bzero(sound, sizeof(sound));
   /* convert stimulus duration to number of samples */
   numpts = (stimulus.stim_dur * DA_FREQ)/1000;
   /* convert prestimulus delay to number of samples */
   delay = (stimulus.prestim_delay * DA_FREQ)/1000;
   make_tone(freq, 5000, sound, numpts);
   ShapeWaveform(sound, numpts, 5.0, 5.0, (double)DA_FREQ);

   if (FreqCurve.itd == 0) itdoffset = 0;
   else itdoffset = (int)((float)FreqCurve.itd * DA_FREQ/(1000.0*1000.0));
   if (itdoffset < 0) {  /* negative ITD means left leads right */
      itdoffset *= -1;
      for (j=0, i=delay; j<numpts; i++, j++) left[i] = sound[j];
      for (j=0, i=delay+itdoffset; j<numpts; i++, j++) right[i] = sound[j];
   }
   else if (itdoffset > 0) {  /* right leads left */
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = sound[j];
      for (j=0, i=delay+itdoffset; j<numpts; i++, j++) left[i] = sound[j];
   }
   else if (itdoffset == 0)
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = left[i] = sound[j];
   /* stuff the D/A array */
   for (i=0; i<numpts+delay; i++) {
      stim.buf[i+i] = left[i];
      stim.buf[i+i+1] = right[i];
   }
   /* set the digital attenuators */
   if (FreqCurve.iid == 0)
      setAtten(FreqCurve.abi, FreqCurve.abi);
   else {
      if (FreqCurve.iid < 0) {  /* left is louder than right */
         l_level = ((-1*FreqCurve.iid)/2) + FreqCurve.abi;
         r_level = FreqCurve.abi - ((-1*FreqCurve.iid)/2);
      }
      else {
         r_level = (FreqCurve.iid/2) + FreqCurve.abi;
         l_level = FreqCurve.abi - (FreqCurve.iid/2);
      }
      setAtten(l_level, r_level);
   }
   /* restore previous frequency */
   stimulus.freq = temp;
}

initITD()
{
   register int i, j;
   int numpts, delay, offset;
   int l_level, r_level;
   void ShapeWaveform();
```

```
    bzero(stim.buf, sizeof(stim.buf));
    bzero(left, sizeof(left));
    bzero(right, sizeof(right));
    bzero(sound, sizeof(sound));
    numpts = (stimulus.stim_dur * DA_FREQ)/1000;
    delay = (stimulus.prestim_delay * DA_FREQ)/1000;
    offset = (int)((float)ItdCurve.start * DA_FREQ/(1000.0*1000.0));
    /* check that numpts+offset < length of sound array */
    /* this allows for no onset time difference */
    if (stimulus.stimtype == Tone)
       make_tone(stimulus.freq, 5000, sound, numpts+abs(offset));
    else if (stimulus.stimtype == WNoise)
       getWhiteNoise(sound, numpts+abs(offset));
    else if (stimulus.stimtype == BPNoise)
       getBandpassNoise(sound, numpts+abs(offset));
    ShapeWaveform(sound, numpts, 5.0, 5.0, (double)DA_FREQ);
    if (offset < 0) {  /* negative ITD means left leads right */
       offset *= -1;
       for (j=0, i=delay; j<numpts; i++, j++) {
          left[i] = sound[j];
          right[i] =  sound[j+offset];
       }
    }
    else if (offset > 0) {  /* right leads left */
       for (j=0, i=delay; j<numpts; i++, j++) {
          right[i] = sound[j];
          left[i] = sound[j+offset];
       }
    }
    else if (offset == 0)
       for (j=0, i=delay; j<numpts; i++, j++) right[i] = left[i] = sound[j];
    /* stuff the D/A array */
    for (i=0; i<numpts+delay; i++) {
       stim.buf[i+i] = left[i];
       stim.buf[i+i+1] = right[i];
    }
    /* set the digital attenuators */
    if (ItdCurve.iid == 0)
       setAtten(ItdCurve.abi, ItdCurve.abi);
    else {
       if (ItdCurve.iid < 0) {  /* left is louder than right */
          l_level = ((-1*ItdCurve.iid)/2) + ItdCurve.abi;
          r_level = ItdCurve.abi - ((-1*ItdCurve.iid)/2);
       }
       else {
          r_level = (ItdCurve.iid/2) + ItdCurve.abi;
          l_level = ItdCurve.abi - (ItdCurve.iid/2);
       }
       setAtten(l_level, r_level);
    }
}


loadITD(itd)
int itd;  /* in usec */
{
    register int i, j;
    int numpts, delay, offset;

    bzero(stim.buf, sizeof(stim.buf));
    bzero(left, sizeof(left));
    bzero(right, sizeof(right));
    /* neither zero, nor recalculate the sound array */
    numpts = (stimulus.stim_dur * DA_FREQ)/1000;
    delay = (stimulus.prestim_delay * DA_FREQ)/1000;
    offset = (int)((float)itd * DA_FREQ/(1000.0*1000.0));
    if (offset < 0) {  /* negative ITD means left leads right */
       offset *= -1;
       for (j=0, i=delay; j<numpts; i++, j++) left[i] = sound[j];
       for (j=0, i=delay+offset; j<numpts; i++, j++) right[i] = sound[j];
    }
    else if (offset > 0) {  /* right leads left */
       for (j=0, i=delay; j<numpts; i++, j++) right[i] = sound[j];
       for (j=0, i=delay+offset; j<numpts; i++, j++) left[i] = sound[j];
    }
```

```
   else if (offset == 0)
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = left[i] = sound[j];
   /* stuff the D/A array */
   for (i=0; i<numpts+delay; i++) {
      stim.buf[i+i] = left[i];
      stim.buf[i+i+1] = right[i];
   }
   /* no need to set the digital attenuators again since IID remains
    * constant for a given ITD curve.
    */
}

initIID()
{
   register int i, j;
   int numpts, delay, offset;
   int l_level, r_level;
   void ShapeWaveform();

   bzero(stim.buf, sizeof(stim.buf));
   bzero(left, sizeof(left));
   bzero(right, sizeof(right));
   bzero(sound, sizeof(sound));
   numpts = (stimulus.stim_dur * DA_FREQ)/1000;
   delay = (stimulus.prestim_delay * DA_FREQ)/1000;
   if (stimulus.stimtype == Tone)
      make_tone(stimulus.freq, 5000, sound, numpts);
   else if (stimulus.stimtype == WNoise)
      getWhiteNoise(sound, numpts);
   else if (stimulus.stimtype == BPNoise)
      getBandpassNoise(sound, numpts);
   ShapeWaveform(sound, numpts, 5.0, 5.0, (double)DA_FREQ);
   offset = (int)((float)IidCurve.itd * DA_FREQ/(1000.0*1000.0));
   if (offset < 0) {  /* negative ITD means left leads right */
      offset *= -1;
      for (j=0, i=delay; j<numpts; i++, j++) left[i] = sound[j];
      for (j=0, i=delay+offset; j<numpts; i++, j++) right[i] = sound[j];
   }
   else if (offset > 0) {  /* right leads left */
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = sound[j];
      for (j=0, i=delay+offset; j<numpts; i++, j++) left[i] = sound[j];
   }
   else if (offset == 0)
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = left[i] = sound[j];
   /* stuff the D/A array */
   for (i=0; i<numpts+delay; i++) {
      stim.buf[i+i] = left[i];
      stim.buf[i+i+1] = right[i];
   }
   /* set the digital attenuators */
   if (IidCurve.start == 0)
      setAtten(IidCurve.abi, IidCurve.abi);
   else {
      if (IidCurve.start < 0) {  /* left is louder than right */
         l_level = ((-1*IidCurve.start)/2) + IidCurve.abi;
         r_level = IidCurve.abi - ((-1*IidCurve.start)/2);
      }
      else {
         r_level = (IidCurve.start/2) + IidCurve.abi;
         l_level = IidCurve.abi - (IidCurve.start/2);
      }
      setAtten(l_level, r_level);
   }
}


loadIID(iid)
int iid;      /* in dB */
{
   int l_level, r_level;

   /* no need to rezero any buffers, just reset the digital attenuators */
   if (iid == 0)
      setAtten(IidCurve.abi, IidCurve.abi);
   else {
```

```
      if (iid < 0) {  /* left is louder than right */
         l_level = ((-1*iid)/2) + IidCurve.abi;
         r_level = IidCurve.abi - ((-1*iid)/2);
      }
      else {
         r_level = (iid/2) + IidCurve.abi;
         l_level = IidCurve.abi - (iid/2);
      }
      setAtten(l_level, r_level);
   }
}


initFIID()
{
   register int i, j;
   int numpts, delay, offset;
   int l_level, r_level;
   void ShapeWaveform();

   bzero(stim.buf, sizeof(stim.buf));
   bzero(left, sizeof(left));
   bzero(right, sizeof(right));
   bzero(sound, sizeof(sound));
   numpts = (stimulus.stim_dur * DA_FREQ)/1000;
   delay = (stimulus.prestim_delay * DA_FREQ)/1000;
   if (stimulus.stimtype == Tone)
      make_tone(stimulus.freq, 5000, sound, numpts);
   else if (stimulus.stimtype == WNoise)
      getWhiteNoise(sound, numpts);
   else if (stimulus.stimtype == BPNoise)
      getBandpassNoise(sound, numpts);
   /* rise/fall */
   ShapeWaveform(sound, numpts, 5.0, 5.0, (double)DA_FREQ);
   /* give the sound ITD */
   offset = (int)((float)FiidCurve.itd * DA_FREQ/(1000.0*1000.0));
   if (offset < 0) {  /* negative ITD means left leads right */
      offset *= -1;
      for (j=0, i=delay; j<numpts; i++, j++) left[i] = sound[j];
      for (j=0, i=delay+offset; j<numpts; i++, j++) right[i] = sound[j];
   }
   else if (offset > 0) {  /* right leads left */
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = sound[j];
      for (j=0, i=delay+offset; j<numpts; i++, j++) left[i] = sound[j];
   }
   else if (offset == 0)
      for (j=0, i=delay; j<numpts; i++, j++) right[i] = left[i] = sound[j];
   /* stuff the D/A array */
   for (i=0; i<numpts+delay; i++) {
      stim.buf[i+i] = left[i];
      stim.buf[i+i+1] = right[i];
   }
   /* set the digital attenuators */
   if (FiidCurve.fixed < 0) {  /* left ear is fixed */
      l_level = -1*FiidCurve.fixed;
      if (FiidCurve.start == 0) r_level = l_level;
      if (FiidCurve.start < 0)  /* left ear louder */
         r_level = l_level + FiidCurve.start;
      else   /* right ear louder */
         r_level = FiidCurve.start - FiidCurve.fixed;
   }
   else if (FiidCurve.fixed > 0) {  /* right ear is fixed */
      r_level = FiidCurve.fixed;
      if (FiidCurve.start == 0) l_level = r_level;
      if (FiidCurve.start < 0)   /* left ear louder */
         l_level = (-1*FiidCurve.start) + FiidCurve.fixed;
      else {  /* right ear louder */
         l_level = FiidCurve.fixed - FiidCurve.start;
      }
   }
   else if (FiidCurve.fixed == 0) {  /* what the hell is this??? */
   }
   setAtten(l_level, r_level);
}
```

```
loadFIID(iid)
int iid;
{
   int l_level, r_level;

   /* set the digital attenuators */
   if (FiidCurve.fixed < 0) {  /* left ear is fixed */
      l_level = -1*FiidCurve.fixed;
      if (iid == 0) r_level = l_level;
      if (iid < 0)  /* left ear louder */
         r_level = l_level + iid;
      else   /* right ear louder */
         r_level = iid - FiidCurve.fixed;
   }
   else if (FiidCurve.fixed > 0) {  /* right ear is fixed */
      r_level = FiidCurve.fixed;
      if (iid == 0) l_level = r_level;
      if (iid < 0)   /* left ear louder */
         l_level = (-1*iid) + FiidCurve.fixed;
      else {  /* right ear louder */
         l_level = FiidCurve.fixed - iid;
      }
   }
   else if (FiidCurve.fixed == 0) {  /* what the hell is this??? */
   }
   setAtten(l_level, r_level);
}

setAtten(Llevel, Rlevel)
int Llevel, Rlevel;
{
   register int i;
   char buf[80];
   int freq, latten, ratten;
   float lmax, rmax;

   if (stimulus.stimtype == Tone || curvetype == Freq) {
      freq = stimulus.freq;
      if (freq < Calib.freq[0]) {
         sprintf(buf, "Frequency %d Hz out of calibration range.", freq);
         latten = 80 - Llevel;
         ratten = 80 - Rlevel;
         setRack(0, latten, ratten);
         uhoh(buf);
         return;
      }
      for (i=0; Calib.freq[i] < freq; i++)
      if (i > 91) {
         sprintf(buf, "Frequency %d Hz out of calibration range.", freq);
         latten = 80 - Llevel;
         ratten = 80 - Rlevel;
         setRack(0, latten, ratten);
         uhoh(buf);
         return;
      }
      if (Calib.freq[i] == freq) {
         latten = rnd(Calib.Lmax[i] - Llevel);
         ratten = rnd(Calib.Rmax[i] - Rlevel);
      }
      else {  /* interpolate */
         float x1, y1, x2, y2;
         x1 = (float)Calib.freq[i];
         x2 = (float)Calib.freq[i-1];
         y1 = Calib.Lmax[i];
         y2 = Calib.Lmax[i-1];
         lmax = ((y2 - y1)/(x2-x1) * ((float)freq - x1)) + y1;
         y1 = Calib.Rmax[i];
         y2 = Calib.Rmax[i-1];
         rmax = ((y2 - y1)/(x2-x1) * ((float)freq - x1)) + y1;
         latten = rnd(lmax - Llevel);
         ratten = rnd(rmax - Rlevel);
      }
      if (latten < 0 || latten > 127) {
         sprintf(buf, "Unable to get %ddbSPL at %d Hz\non left earphone\n Doing best possible."
, latten, freq);
```

```
            if (latten < 0) latten = 0;
            if (latten > 127) latten = 127;
            uhoh(buf);
         }
         if (ratten < 0 || ratten > 127) {
            sprintf(buf, "Unable to get %ddbSPL at %d Hz on right earphone.",
               ratten, freq);
            if (ratten < 0) ratten = 0;
            if (ratten > 127) ratten = 127;
            uhoh(buf);
         }
      }
      else if (stimulus.stimtype == WNoise || stimulus.stimtype == BPNoise) {
         latten = 80 - Llevel;
         ratten = 80 - Rlevel;
      }
      setRack(0, latten, ratten);
}
```

```
/*   OASys:  Ovoidalis Analysis System
 *     ITD, IID, Frequency tuning and whatever else becomes necessary.
 */

#include "includes.h"
#include "defines.h"
#include "globals.h"

#define PROPORT_32kHz_AD_DELAY   82 /* 82 samples */

int event_stat, counter, whichrep;
int last, step;
int rasterline;
BOOLEAN DONE, INPROGRESS, COMPLETE;

main (argc, argv)
int argc;
char **argv;
{
   void mainloop();
   int da_tcr(), event_tcr();

   setup(&argc, argv);
   INPROGRESS = False;
   DONE = COMPLETE = True;
#ifdef mc700
   mrevtcr(da_path, da_tcr);
   mrevtcr(event_timer, event_tcr);
#endif
   mainloop();
}

void mainloop()
{
   XEvent event;

   for (;;) {
      if (XtAppPending(app_context) != 0) {  /* are there X events? */
         XtAppNextEvent(app_context, &event);
         XtDispatchEvent(&event);
      }
      if (!INPROGRESS) {
         if (!DONE)  do_trial();
         else if (!COMPLETE) do_next();
      }
   }
}

void start_run()
{
   Arg args[1];

   if (datafile == (FILE *)NULL) {
      uhoh("Datafile is NULL");
      return;
   }
   write_header();
   rasterline = counter = 0;
   setup_raster();
   /* clear the raster & stimulus windows */
   XClearWindow(display, XtWindow(raster));
   XClearWindow(display, XtWindow(stimCanvas));
   COMPLETE = False;
   loadFirst();
   do_trial();
}


loadFirst()
{
   /* load first stimulus in the tuning curve series */
   switch (curvetype) {
      case Freq: {
         previous = FreqCurve.start;
         last = FreqCurve.end;
```

```
            step = FreqCurve.step;
            loadFreq(FreqCurve.start);
            break;
         }
         case Itd: {
            previous = ItdCurve.start;
            last = ItdCurve.end;
            step = ItdCurve.step;
            initITD();
            break;
         }
         case Iid: {
            previous = IidCurve.start;
            last = IidCurve.end;
            step = IidCurve.step;
            initIID();
            break;
         }
         case Fiid: {
            previous = FiidCurve.start;
            last = FiidCurve.end;
            step = FiidCurve.step;
            initFIID();
            break;
         }
      }
}

loadNext()
{
   /* load next stimulus in the tuning curve series */
   int next;

   next = previous + step;
   if (next <= last) {
      switch (curvetype) {
         case Freq:
            loadFreq(next);
            break;
         case Itd:
            loadITD(next);
            break;
         case Iid:
            loadIID(next);
            break;
         case Fiid:
            loadFIID(next);
            break;
      }
      previous = next;
   }
   else COMPLETE = True;
}

do_next()
{
   Arg args[1];

   loadNext();
   if (!COMPLETE) do_trial();
   else {
      fclose(datafile);
      datafile = (FILE *)NULL;
      analyze_data();
   }
}

do_trial()
{
   void Digital_IO();
   unsigned int *events;  /* list of spike times */
   unsigned int numevents;
   unsigned int *getTTL();

   if (!INTERRUPT) {
```

```
      DONE = False;
      INPROGRESS = True;
      whichrep = counter+1;
      ++rasterline;
      {
      Digital_IO(stimulus.total_dur);
      events = getTTL(24000);
      if (events != NULL)
         numevents = events[0];
         plot_raster(events, events[0], rasterline, previous);
         if (fwrite((char *)&numevents, sizeof(numevents), 1, datafile) < 1) {
            uhoh("Error writing events[0] to file");
            INTERRUPT = True;
            return;
         }
         if (events[0] > 0) {
            if (fwrite((char *)events, events[0], 1, datafile) < 1) {
               uhoh("Error writing event buffer to file.");
               INTERRUPT = True;
               return;
            }
         }
      }
      INPROGRESS = FALSE;
      free(events);
      IntentionalDelay(1000);
      ++counter;
      if (counter < stimulus.nreps)
         return;
      else {
         DONE = True;
         counter = 0;
      }
   }
   else if (INTERRUPT) {
      DONE = True;
      COMPLETE = True;
      fclose(datafile);
      datafile = (FILE *)NULL;
      counter = 0;
      INTERRUPT = False;
   }
}

void Digital_IO(duration)
int duration;  /* in msec */
{
   int nframes;
   int nmissed, missrec, missplay;
   int cc;

   /* convert duration to number of samples */
   nframes = (duration * DA_FREQ)/1000;

   drp_rpm_constructor(drp, &rpm);
   drp_stream_constructor(drp, &rpm.rec_stream, nbits, Stereo, SAMPLING_RATE);
   if (drp_stream_config_buf(drp, &rpm.rec_stream, -(nframes),
      (char *)ad.buf, NULL, 0) == -1) {
      fprintf(stderr, "drp_stream_config_buf bombed for recording.\n");
      return;
   }
   rpm.rec_stream.buflen = 0;
   drp_stream_constructor(drp, &rpm.play_stream, nbits, Stereo, SAMPLING_RATE);
   if (drp_stream_config_buf(drp, &rpm.play_stream, -(nframes),
      (char *)stim.buf, NULL, 0) == -1) {
      fprintf(stderr, "drp_stream_config_buf bombed for playing.\n");
      return;
   }
   rpm.play_stream.buflen = sizeof(stim.buf);

   if (drp_start(drp, DRP_DIR_RECPLAY) == -1) {
      fprintf(stderr, "drp_start: %s\n", drp->error);
      return;
   }
   cc = drp_rec_play_mem(drp, &rpm);
```

```
   if (cc == -1) {
      fprintf(stderr, "drp_rec_play_mem: %s\n", drp->error);
      return;
   }
   if (drp_end(drp) == -1) {
      fprintf(stderr, "drp_end: %s\n", drp->error);
      return;
   }
   nmissed = drp_missed(drp, &missrec, &missplay);
   if (nmissed == -1) {
      fprintf(stderr, "drp_missed: %s\n", drp->error);
      return;
   }
   if (nmissed != 0) {
      fprintf(stderr,"WARNING: missed %d record samples and %d play samples.\n",
         missrec, missplay);
   }
   drp_stream_destructor(&rpm.rec_stream);
   drp_stream_destructor(&rpm.play_stream);
}

#define AD_MAX_VALUE 32767

unsigned int *getTTL(buflen)
     int buflen;               /* examine from j=0 to buflen */
{
  int thresh;     /* threshold (must cross) for TTL pulse */
  int in_pulse = 0;      /* state machine: in or out of TTL pulse */
  int count = 0;       /* count of spikes */
  unsigned int *spikes; /* pointer to spike info */
  int i, j;     /* index vars */
  int room = 10;  /* available room in growing spike bufer */

   /* TTL pulses from BES uA-200D amplifier are negative going */
   thresh = 0.25*(float)AD_MAX_VALUE;

   spikes = (unsigned int *)malloc((1+(2*room))*sizeof(int));

   for (j=2*PROPORT_32kHz_AD_DELAY, i = 0; j < 2*buflen; i++, j += 2) {
      if (!in_pulse && ad.buf[j] > thresh) {
         in_pulse = 1;
         if (count >= room) {
            room += 10;
            spikes = (unsigned int *)
               realloc(spikes, (1+(2*room))*sizeof(unsigned int));
         }
         /* spike time in usecs */
         spikes[++count] = (int)(1.0e6*i/SAMPLING_RATE);
      }
      else if (in_pulse && ad.buf[j] < thresh) {
         in_pulse = 0;
      }
   }
   spikes[0] = count;
   fprintf(stderr, "Number of spikes: %d\n", count);
   return(spikes);
}


int SetupEvtClocks(duration)
int duration;  /* in msec */
{
#ifdef mc700
   int evstat;     /* status info */
   unsigned long overflow;
   void ClearEventBuffer();

   mrclksetter(timer_clock,    /* path number of clock */
         11,           /* src = 6 MHz, rising edge */
          6,           /* lcnt: 6 MHz/6 = 1 MHz clock freq */
          0, 0,        /* hold register count, gating mode */
          CLK_STLOW,
          CLK_PULSE,
          CLK_REPEAT);
```

```
   bzero(Events.list, sizeof(Events.list));
   overflow = (unsigned long)(duration * 1000.0); /*convert to usecs*/
   evstat = s_evtmod(event_timer, overflow);
   mrbufall(event_timer, Events.list, 1, EVTSIZE*2);
   mrxinq(event_timer, EVTSIZE, EVTSIZE, 0);
   return(evstat);
#endif
}


/* D/A task completion routine */
int da_tcr(rpathno)
int *rpathno;
{
#ifdef mc700
   char buf[80];

   /* this should be called on completion of d/a transfer */
   mrclkdis(1, &da_clk_path);
   if (*rpathno != da_path) {
      sprintf(buf,"BCR bad path: %d should be %d\n", *rpathno, da_path);
      uhoh(buf);
   }
#endif
}

/* event timer task completion routine */
int event_tcr(rpathno)
int *rpathno;
{
#ifdef mc700
   int cEvtBufSize, Index, status;
   int argmod = 3;     /* Index is an index to a short integer array */
   unsigned int *cEvtBuf, *CompressEventBuffer();
   char buf[80];
   Arg args[1];

   /* disarm the clock */
   mrclkdis(1, &timer_clock);
   /* stop the devices just in case */
   mrstop(event_timer, 1, &status);
   mrstop(da_path, 1, &status);
   mrstop(timer_clock, 1, &status);
   mrstop(da_clk_path, 1, &status);
   INPROGRESS = False;
   /* check the path number to make sure the BCR is servicing the
    * correct device.
    */
   if (*rpathno != event_timer) {
      sprintf(buf,"BCR bad path: %d should be %d\n", *rpathno, event_timer);
      uhoh(buf);
      return;
   }
   /* retrieve the done buffer */
   mrbufget(event_timer, argmod, &Index);
   if (Index == 0) {  /* the usual case since I'm only using one buffer */
      /* check to see if data collection has terminated */
      if (!INTERRUPT) {
         /* write the data to file */
         cEvtBuf = CompressEventBuffer(&cEvtBufSize);
         if (cEvtBuf != NULL) {
            /* update the raster */
            plot_raster(cEvtBuf, cEvtBuf[0], rasterline, previous);
            if (fwrite((char *)&cEvtBufSize, sizeof(int), 1, datafile) < 1) {
               uhoh("Error writing EvtBufSize to file");
               INTERRUPT = True;
               return;
            }
            if (cEvtBufSize > 0) {
               /* write out the event buffer */
               if (fwrite((char *)cEvtBuf, cEvtBufSize, 1, datafile) < 1) {
                  uhoh("Error writing event buffer to file.");
                  INTERRUPT = True;
                  return;
               }
```

```
            }
         }
         free(cEvtBuf);
         /* delay in msec between stimulus repititions */
         IntentionalDelay(1000);
         ++counter;
         if (counter < stimulus.nreps)
            return;
         else {
            DONE = True;
            counter = 0;
         }
      }
      else if (INTERRUPT) {
         DONE = True;
         COMPLETE = True;
         fclose(datafile);
         datafile = (FILE *)NULL;
         counter = 0;
         INTERRUPT = False;
      }
   }
#endif
}

IntentionalDelay(DelayTime)
int DelayTime;
{
  static struct timeval StartTime, NowTime;
  static struct timezone dummy;

  gettimeofday(&StartTime, &dummy);
  StartTime.tv_sec += (DelayTime / 1000);
  StartTime.tv_usec += ((DelayTime % 1000) * 1000);
  while (TRUE) {
   gettimeofday(&NowTime, &dummy);
   if (NowTime.tv_sec > StartTime.tv_sec ||
      NowTime.tv_sec == StartTime.tv_sec && NowTime.tv_usec > StartTime.tv_usec
)
     break;
  }
}

void stop_data_acq(w, client, call)
Widget w;
caddr_t client, call;
{
   INTERRUPT = True;
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

plot_raster(list, length, line, value)
unsigned int list[], length;
int line, value;
{
   register int i;
   static int previous = 0;
   XSegment rasterSeg[1];
   Drawable d;
   Window window;
   XWindowAttributes raster_info;
   int row, x, y, spiketime, v;
   float loc, rowsiz;
   char buf[80];
   void user_to_out();

   d = window = XtWindow(raster);
   XGetWindowAttributes(display, window, &raster_info);
   row = RasterRows-(line*2);
   rowsiz = 1.0;
   loc = (float)row;
   if (value != previous) {
      sprintf(buf, "%d", value);
      user_to_out(raster_wc, 0.0, loc, &x, &y,
         raster_info.width, raster_info.height);
      XDrawString(display, d, plot_gc, 0, y, buf, strlen(buf));
      previous = value;
   }
   if (length > 0) {
      for (i=0; i<length-1; i++) {  /* for each spike */
         spiketime = (int)event_time(list[i+1])/1000;  /* convert to msec */
         user_to_out(raster_wc, (float)spiketime, loc, &x, &y,
            raster_info.width, raster_info.height);
         rasterSeg[0].x1 = (short)x;
         rasterSeg[0].y1 = (short)y;
         user_to_out(raster_wc, (float)spiketime, loc+rowsiz, &x, &y,
            raster_info.width, raster_info.height);
         rasterSeg[0].x2 = (short)x;
         rasterSeg[0].y2 = (short)y;
         XDrawSegments(display, d, plot_gc, rasterSeg, 1);
      }
      XFlush(display);
   }
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

/* graph plotting utility routines */

void set_world_coords(window, xmin, ymin, xmax, ymax)
WC_window *window;  /* which window */
float xmin, ymin, xmax, ymax;
{
/* set the user coordinate system (cartesian).  This is equivalent
   to the GKS routine set window, but since this is implemented under
   X Windows, I used the name set_world_coords to avoid any confusion.
*/
   window->xmin = xmin;
   window->ymin = ymin;
   window->xmax = xmax;
   window->ymax = ymax;
   window->width = xmax - xmin;
   window->height = ymax - ymin;
}

void user_to_ndc(window, x, y, ndcx, ndcy)
WC_window window;
float x, y, *ndcx, *ndcy;
{
   *ndcx = (x - window.xmin)/(window.width);
   *ndcy = (y - window.ymin)/(window.height);
}

void ndc_to_user(window, ndcx, ndcy, x, y)
WC_window window;
float ndcx, ndcy, *x, *y;
{
   *x = (ndcx * window.width) + window.xmin;
   *y = (ndcy * window.height) + window.ymin;
}

void ndc_to_out(ndcx, ndcy, dcx, dcy, ndh, ndv)
float ndcx, ndcy; /* normalized device coordinates */
int *dcx, *dcy;   /* device coordinates */
int ndh, ndv;   /* number of dots (pixels) horizontally & vertically */
/* see Computer Graphics Software Construction, by John R. Rankin
   Prentice Hall, 1989.  pp 10 -11.
   FLOOR function substituted for rounding on advice of Paul S. Heckbert,
   "What are the coordinates of a pixel?", p.246, Graphics Gems,
   Andrew S. Glassner, ed.
*/
{
   *dcx = FLOOR(ndcx * (ndh-1));
   *dcy = ndv - FLOOR(ndcy * (ndv-1)); /* flip y values */

}

void inp_to_ndc(dcx, dcy, ndcx, ndcy, ndh, ndv)
int dcx, dcy;      /* display coordinates */
float *ndcx, *ndcy; /* normalized device coordinates */
int ndh, ndv;      /* number of dots horizontally & vertically */
/* same reference as above */
{
   *ndcx = (float)dcx/(float)(ndh-1);
   *ndcy = (float)(ndv - dcy)/(float)(ndv-1);  /* flip y values */
}

void user_to_out(window, x, y, dcx, dcy, winWidth, winHeight)
WC_window window;
float x, y;
int *dcx, *dcy;
int winWidth, winHeight;  /* width & height of window */
{
   float ndcx, ndcy;

   /* convert world coordinates to normalized device coordinates */
   user_to_ndc(window, x, y, &ndcx, &ndcy);
   /* convert normalized device coordinates to physical device coordinates */
```

```
   ndc_to_out(ndcx, ndcy, dcx, dcy, winWidth, winHeight);
}

void inp_to_user(window, dcx, dcy, x, y, winWidth, winHeight)
WC_window window;
int dcx, dcy;
float *x, *y;
int winWidth, winHeight;
{
   float ndcx, ndcy;

   /* convert physical device coordinates to normalized device coordinates */
   inp_to_ndc(dcx, dcy, &ndcx, &ndcy, winWidth, winHeight);
   /* convert normalized device coordinates to world coordinates */
   ndc_to_user(window, ndcx, ndcy, x, y);
}

/* routines for graph labeling  - from Graphics Gems, A.S. Glassner Ed.*/

double nicenum();

#define expt(a, n) pow(a, (double)(n))

loose_label(min, max, ntick, newmax, labels, numlabels)
double min, max;
int ntick;           /* desired number of tick marks */
double *newmax;      /* maximum value calculated by loose_label */
char labels[][20];   /* array of labels to return */
int *numlabels;      /* number of labels returned */
{
   register int i;
   char str[6];
   int nfrac;
   double d;                     /* tick mark spacing */
   double graphmin, graphmax;    /* graph range min and max */
   double range, x;

   /* we expect min != max */
   range = nicenum(max-min, FALSE);
   d = nicenum(range/(ntick-1), TRUE);
   graphmin = FLOOR(min/d)*d;
   graphmax = CEILING(max/d)*d;
   *newmax = graphmax;
   nfrac = MAX(-FLOOR(log10(d)), 0);   /* # of fractional digits to show */
   sprintf(str, "%%.%df", nfrac);      /* simplest axis labels */
   for (i=0, x=graphmin; x<graphmax+.5*d; x+=d, i++) {
      sprintf(labels[i], str, x);
   }
   *numlabels = i;
}

tight_label(min, max, ntick, labels, numlabels)
double min, max;
int ntick;
char labels[][20];   /* array of labels */
int *numlabels;      /* number of labels returned */
{
   register int i;
   char str[6];
   int nfrac;
   double d;                     /* tick mark spacing */
   double range, x;

   range = nicenum(max-min, 0);
   d = nicenum(range/(ntick-1), TRUE);
   nfrac = MAX(-FLOOR(log10(d)), 0);   /* # of fractional digits to show */
   sprintf(str, "%%.%df", nfrac);      /* simplest axis labels */
   for (i=0, x=min; x<max; x+=d, i++) {
      sprintf(labels[i], str, x);
   }
   *numlabels = i;
}

/*
 * nicenum: find a "nice" number approximately equal to x.
```

```
 * Round the number if round = 1, take the ceiling if round = 0
 */

static double nicenum(x, round)
double x;
int round;
{
   int exp;     /* exponent of x */
   double f;    /* fractional part of x */
   double nf;   /* nice, rounded fraction */

   exp = FLOOR(log10(x));
   f = x/expt(10.0, exp);  /* between 1 and 10 */
   if (round)
      if (f<1.5) nf = 1.0;
      else if (f<3.0) nf = 2.0;
      else if (f<7.0) nf = 5.0;
      else nf = 10.0;
   else
      if (f<=1.0) nf = 1.0;
      else if (f<=2.0) nf = 2.0;
      else if (f<=5.0) nf = 5.0;
      else nf = 10.0;
   return nf*expt(10.0, exp);
}
```

```
#include "includes.h"
#include "defines.h"
#define DEFINE_GLOBALS
#include "globals.h"

static char *dspcode = "lib/ssirp.lod";

setup(arg, argv)
int *arg;
String *argv;
{
   register int i;
   int screen;
   int error;
   long seed = -1;

   if (REALTIME) {
      /* Boot the DSP and load the application */
      drp = drp_bootdsp("qckMon.lod", dspcode);
      if (drp == NULL) {
         perror("drp_bootdsp");
         exit(1);
      }
      if (drp->error[0]) {
         fprintf(stderr, "drp_bootdsp: %s\n", drp->error);
         return;
      }
      /* set DMA mode and page size */
      if (drp_confio(drp, nbits, pagesize, readsize) == -1) {
         fprintf(stderr, "drp_confio: %s\n", drp->error);
         return;
      }
      if (drp_confperiph(drp, "proport", SAMPLING_RATE, 1, DRP_DIR_RPM) == -1) {
         fprintf(stderr, "drp_confperiph: %s\n", drp->error);
         return;
      }
   }
   /* initialize the globals structs */
   stimulus.nreps = 5;
   stimulus.total_dur = 300;
   stimulus.stim_dur = 100;
   stimulus.prestim_delay = 100;
   stimulus.stimtype = Tone;
   stimulus.freq = 1000;
   stimulus.bpNoiseStim.startfreq = 1000;
   stimulus.bpNoiseStim.endfreq = 10000;
   stimulus.tcStim.numfreqs = 0;

   curvetype = Freq;

   FreqCurve.start = 1000; FreqCurve.end = 10000; FreqCurve.step = 500;
   FreqCurve.itd = 0;  FreqCurve.iid = 0;  FreqCurve.abi = 50;

   ItdCurve.start = -300; ItdCurve.end = 300; ItdCurve.step = 30;
   ItdCurve.iid = 0;  ItdCurve.abi = 50;

   IidCurve.start = -60; IidCurve.end = 60;  IidCurve.step = 5;
   IidCurve.itd = 0;  IidCurve.abi = 50;

   FiidCurve.start = -40; FiidCurve.end = 20; FiidCurve.step = 5;
   FiidCurve.fixed = 20;  FiidCurve.itd = 0;

   fprintf(stderr, "Making white noise...\n");
   error = init_gaussian_table(2.0);
   init_random_sequence(&seed);
   error = fast_noise(wnoise, 65536, 1);
   for (i=0; i<65536; i++) bpnoise[i] = wnoise[i];
   fprintf(stderr, "...done.\n");
   readToneCalib();
   makeGUI(arg, argv);
   /* clear the pixmaps */
   screen = DefaultScreen(display);
   XSetForeground(display, plot_gc, WhitePixel(display, screen));
   XFillRectangle(display, freqPixmap, plot_gc, 0, 0, 400, 350);
   XFillRectangle(display, itdPixmap, plot_gc, 0, 0, 400, 350);
```

```
   XFillRectangle(display, iidPixmap, plot_gc, 0, 0, 400, 350);
   XFillRectangle(display, fiidPixmap, plot_gc, 0, 0, 400, 350);
   XSetForeground(display, plot_gc, BlackPixel(display, screen));
}

get_parameters()
{
   char choice;
   int i, numpts, error, frequency;

   choice = 'Z';
   while (choice != 'r') {
      show_parameters();
      fflush(stdin);
      fprintf(stderr, "\nSelect parameter to change by letter.\n\n");
      fprintf(stderr, "Type r to run.\n");
      choice = getchar();
      switch(choice) {
         case 'o':
            fprintf(stderr, "Name of data file: ");
            scanf("%s", datafilename);
            analyze_data();
            show_parameters();
            break;
         case 'q':
            setRack(0,0,0);
            XtUnmapWidget(toplevel);
#ifdef mc700
            mrclosall();
#endif
            exit(0);
         case 'r':
            break;
         case 'z':
            setRack(0, 0, 0);
            break;
      }
   }
}

show_parameters()
{
   int i;

/* system("clear"); */
   fprintf(stderr, "(o) Open and plot data file. \n");
   fprintf(stderr, "(z) Zero attenuators. \n");
   fprintf(stderr, "(q) QUIT\n");

}

/* read in the tone calibration file "earmic.cal" */
readToneCalib()
{
   FILE *calibfile;
   int i;
   float lrms, rrms, lleak, rleak;
   char buf[80];

   if ((calibfile = fopen("earmic.cal", "r")) == NULL) {
      /* set max intensity to something vaguely reasonable */
      for (i=0; i<91; i++) Calib.Lmax[i] = Calib.Rmax[i] = 80.0;
      fprintf(stderr, "Can't read earmic.cal file.\n");
      return;
   }
   /* skip the header commentary */
   for (i=0; i<5; i++)
      fgets(buf, 80, calibfile);
   /* now read the file */
   for (i=0; i<91; i++) {
      fscanf(calibfile, "%d\t%f\t%f\t%f\t%f\t%f\t%f", &Calib.freq[i],
         &Calib.Lmax[i], &Calib.Rmax[i], &lrms, &rrms, &lleak, &rleak);
   }
   fclose(calibfile);
}
```

```
setup_raster()
{
   Window window;
   XWindowAttributes info;
   float x_value_min,  y_value_min, xmin, ymin;
   float x_value_max, y_value_max, xmax, ymax;
   float x_range, y_range;
   int start, end, step;
   void set_world_coords();

   window = XtWindow(raster);
   XGetWindowAttributes(display, window, &info);
   x_value_min = 0.0;
   x_value_max = (float)stimulus.total_dur;
   y_value_min = 0.0;

   switch (curvetype) {
      case Freq:
         start = FreqCurve.start; end = FreqCurve.end;
         step = FreqCurve.step;
         break;
      case Itd:
         start = ItdCurve.start; end = ItdCurve.end;
         step = ItdCurve.step;
         break;
      case Iid:
         start = IidCurve.start; end = IidCurve.end;
         step = IidCurve.step;
         break;
      case Fiid:
         start = FiidCurve.start; end = FiidCurve.end;
         step = FiidCurve.step;
         break;
   }
   y_value_max = (fabs((float)start-(float)end)/(float)step) + 1.0;
   y_value_max *= stimulus.nreps;
   /* there is a line of space between each line of spikes, plus a line
    * of space along the bottom and top, so the number of lines is
    * 2*(# of rows)+2
    */
   y_value_max = 2.0*y_value_max + 2.0;
   RasterRows = (int)y_value_max;

   x_range = x_value_max - x_value_min;
   y_range = y_value_max - y_value_min;

   /* scale window to the data */
   xmin = x_value_min - 0.1*x_range;  /* 10% space */
   xmax = x_value_max + 0.1*x_range;
   ymin = y_value_min - 0.05*y_range; /* 5% space */
   ymax = y_value_max + 0.05*y_range;
   set_world_coords(&raster_wc, xmin, ymin, xmax, ymax);
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

void ShapeWaveform(buffer, sizeofbuffer, RiseTime, FallTime, SamplingFreq)
short buffer[];          /* data to be shaped */
int sizeofbuffer;
double RiseTime,      /* rise time in ms */
FallTime,             /* fall time in ms */
SamplingFreq;         /* hardward sampling frequency in Hz */
{
   int time, index, last_member;
   double slope, x;

   last_member=sizeofbuffer-1;

   if (RiseTime != 0.0) {
      time = (int) (slope = RiseTime/1000.0 * SamplingFreq);
      slope = 1.0 / slope;
      for (index = 0; index < time; index++) {
         x = (double) (buffer[index]);
         x *= slope * index;
         buffer[index] = (short) (x);
      }
   }

   if (FallTime != 0.0) {
      time = (int) (slope = FallTime / 1000.0 * SamplingFreq);
      slope = 1.0 / slope;
      for (index = 0; index < time; index++) {
         x = (double) (buffer[last_member - index]);
         x *= slope * index;
         buffer[last_member - index] = (short) (x);
      }
   }
}



#define TWOPI 6.283185307

make_tone(freq, Amp, array, len)
int freq;       /* requested frequency in Hz */
int Amp;        /* amplitude in mV */
int len;        /* length of array */
short array[]; /* array to store the sine wave */

{
   int i, j, num, gcd;
   int cycles, samples;
   double frTs, arg, phi;
   double phase = 0.0;
   double A;

   /* find the greatest common denominator of the sampling frequency
      and the tone frequency.
   */
   A = ((double)Amp/0.24) -24.0; /* mV/DAC slope - DAC offset */
   gcd = GCD((int)DA_FREQ, freq);
   samples = (int)DA_FREQ/gcd;
   cycles = freq/gcd;

   num = len/samples;  /* number of complete cycle segments in the array */
   if (num == 0) samples = len;  /* do as many as will fit */

   frTs = (double)freq/(double)DA_FREQ;
   phi = (TWOPI/360.0)*phase;     /* convert to radians */
   /* first segment */
   for (j=0; j<samples; j++) {
      arg = (double)j*(TWOPI*frTs + phi);
      array[j] = (short)(A*sin(arg));
   }
   /* remaining whole cycle segments */
   for (i=1; i<num; i++) {
      for (j=0; j<samples; j++)
```

```
            array[samples*i+j] = array[j];
        }
        /* leftover cycle segments */
        for (j = 0, i=num*samples; i<len; i++, j++) {
            array[i] = array[j];
        }
        /* return the actual frequency */
        return((int)(((float)cycles/(float)samples)*(int)DA_FREQ));
}


GCD(x, y)
int x, y;
{
    if (y == 0) return(x);
    else GCD(y, x%y);
}

getWhiteNoise(array, len)
short array[];      /* array to store the noise in */
int len;
{
    register int i;
    for (i=0; i<len; i++) array[i] = wnoise[i];
}

getBandpassNoise(array, len)
short array[];
int len;
{
    register int i;
    for (i=0; i<len; i++) array[i] = bpnoise[i];
}

/* callback of the Synthesize bandpass noise command widget */
void makeBandpassNoise(w, client, call)
Widget w;
caddr_t client, call;
{
    /* filter the wnoise array with the values stored in
     * stimulus.bpNoiseStim and stash the results in array.
     */
    int i, j;
    int n = 16384;
    int length;
    float a, b, *filter, *fft, *vector();
    Arg args[1];

    length = 2*n;
    filter = vector(1, length);
    fft    = vector(1, length);
    fprintf(stderr, "Defining filter...\n");
    define_filter(filter, length, (int)DA_FREQ,
        stimulus.bpNoiseStim.startfreq, stimulus.bpNoiseStim.endfreq);
    /* bpnoise array is 65536 long, so we filter it in two passes
     * of 32768 each
     */
    for (j=0, i=1; i<=length; j++, i+=2) {
        fft[i] = (float)wnoise[j];    /* real part */
        fft[i+1] = 0.0;               /* imaginary part */
    }
    fprintf(stderr, "Forward FFT, 1st pass...\n");
    four1(fft, n, 1);
    /* multiply FFT of noise by frequency response of filter */
    for (i=1; i<=length-1; i+=2) {
        a = (fft[i]*filter[i]) - (fft[i+1]*filter[i+1]);
        b = (fft[i]*filter[i+1]) + (fft[i+1]*filter[i]);
        fft[i] = a;
        fft[i+1] = b;
    }
    /* inverse FFT result */
    fprintf(stderr, "Inverse FFT, 1st pass...\n");
    four1(fft, n, -1);
    /* leave out the imaginary part (even indices)
     * scale the inverse fft to correct value.
```

```
    */
   for (j=0, i=1; i<=length-1; j++, i+=2)
      bpnoise[j] = (short)(fft[i]/n);

   /* do the same thing on the next 32k samples */
/* for (j=n, i=1; i<=length; j++, i+=2) {
      fft[i] = (float)wnoise[j];
      fft[i+1] = 0.0;
   }
   fprintf(stderr, "Forward FFT, 2nd pass...\n");
   four1(fft, n, 1);
   for (i=1; i<=length-1; i+=2) {
      a = (fft[i]*filter[i]) - (fft[i+1]*filter[i+1]);
      b = (fft[i]*filter[i+1]) + (fft[i+1]*filter[i]);
      fft[i] = a;
      fft[i+1] = b;
   }
   fprintf(stderr, "Inverse FFT, 2nd pass...\n");
   four1(fft, n, -1);
   for (j=n, i=1; i<=length-1; j++, i+=2)
      bpnoise[j] = (short)(fft[i]/n);
   fprintf(stderr, "...done\n");
*/

   free_vector(filter, 1, length);
   free_vector(fft, 1, length);

   {
      int wmax = 0, wmin = 0;
      int bpmax = 0, bpmin = 0;
      for (i=0; i<length; i++) {
         if (wmax < wnoise[i]) wmax = (int)wnoise[i];
         if (wmin > wnoise[i]) wmin = (int)wnoise[i];
         if (bpmax < bpnoise[i]) bpmax = (int)bpnoise[i];
         if (bpmin > bpnoise[i]) bpmin = (int)bpnoise[i];
      }
      fprintf(stderr, "Wnoise max: %d\tmin: %d\n", wmax, wmin);
      fprintf(stderr, "BPnoise max: %d\tmin: %d\n", bpmax, bpmin);
   }
}

define_filter(filter, length, Fs, low_freq, high_freq)
float filter[];     /* filter array */
int length;          /* length of complex frequency response to be defined */
int Fs;              /* sampling frequency */
int low_freq, high_freq;  /* low & high frequency cutoff in Hz */
{
   float freq_res;  /* frequency resolution */
   int lowcut_index, highcut_index;
   int i;

   freq_res = (float)Fs/(float)length;
   lowcut_index = rnd((float)low_freq/freq_res);
   /* make sure lowcut_index is even */
   if (lowcut_index % 2 != 0) --lowcut_index;
   highcut_index = rnd((float)high_freq/freq_res);

   fprintf(stderr, "Freq_res: %0.2f\n", freq_res);
   fprintf(stderr, "lowcut_index: %d\n", lowcut_index);
   fprintf(stderr, "highcut_index: %d\n", highcut_index);

   /* define the frequency response of the digital filter */
   /* positive frequencies */
   for (i=1; i<=lowcut_index; i++) filter[i] = 0.0;
   for (i=lowcut_index+1; i<=highcut_index-1; i+=2) {
      filter[i] = 1.0;    /* real part */
      filter[i+1] = 0.0; /* imaginary part */
   }
   for (i=highcut_index+1; i<=length/2; i++) filter[i] = 0.0;

   /* negative frequencies */
   for (i=length; i>=length-(lowcut_index-1); i--) filter[i] = 0.0;
   for (i=length-lowcut_index; i>=length-highcut_index; i-=2) {
      filter[i] = 0.0;     /* imaginary part */
      filter[i-1] = 1.0;  /* real part */
```

```
   }
   for (i=length-(highcut_index-2); i>=length-((length/2)+1); i--)
      filter[i] = 0.0;
}
```

```
#include "includes.h"
#include "defines.h"
#include "globals.h"

uhoh(string)
char *string;
{
   fprintf(stderr, string);
   fprintf(stderr, "\n");
}
```