

Copyright  
by  
Jeri Cameron Rodgers  
2011

**The Dissertation Committee for Jeri Cameron Rodgers Certifies that this is the approved version of the following dissertation:**

**Comparative Morphology of the Vestibular  
Semicircular Canals in Therian Mammals**

**Committee:**

---

Timothy B. Rowe, Supervisor

---

Christopher J. Bell

---

James T. Sprinkle

---

Edward C. Kirk

---

Lawrence M. Witmer

**Comparative Morphology of the Vestibular  
Semicircular Canals in Therian Mammals**

**by**

**Jeri Cameron Rodgers, B.S.; M.S.**

**Dissertation**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Doctor of Philosophy**

**The University of Texas at Austin**

**December 2011**

## **Dedication**

To Michael, Genevieve, and Alexandra Rodgers

## Acknowledgements

My sincerest thanks go to the long-suffering core of my committee: Tim Rowe, Chris Bell, and Jim Sprinkle. Each of these professors has shown patience and provided direction in so many different academic and personal areas of this doctoral trek. Tim Rowe, my advisor, accepted a student he knew nothing about and fulfilled his role as a mentor throughout many discussions and advisements. Chris Bell saw to the teaching of comparative osteology, proper usage of language, and provided an example of a great teacher. Jim Sprinkle gave me the first opportunity to be a teaching assistant, allowed me to accompany him in many student field trips, and showed how to collaborate on a scientific paper. Larry Witmer unknowingly initiated this dissertation through his offhand remark that “ears are hot.” Chris Kirk willingly stepped on to this committee in a moment of great need. These people truly deserve their designation as professors.

Numerous non-committee people made this dissertation possible. Ted Macrini volunteered the use of many marsupial CT scans for this work, and also the use of *Monodelphis domestica*  $\mu$ MRI and HRXCT scans which he and Tim Rowe originally commissioned. He provided editing of the last chapter in this dissertation. Eric Ekdale shared his knowledge of the petrosal and bony labyrinth on many occasions, as well as helping me to learn the imaging programs with which he worked. Michael Cameron Rodgers spent untold hours listening to the requirements of a semicircular canal sensitivity program that I wanted. The staff of The University of Texas High-Resolution

X-ray CT facility (UTCT), particularly Jessie Maisano and Matt Colbert, spent many hours assisting me with scans and directions for working imaging programs and machines in general. Rich Ketcham provided the technical magic necessary to translate the \*.nifti format to a format importable into VGStudioMax. Tim Hullar at Washington University in St. Louis, MO, corresponded with me at length regarding his program for canal sensitivity, and about the function of the vestibular system in general. Thanks are also due to the group known collectively as the paleo grad students for support, advice, and social opportunities. In particular, Lyn Murray, Jon Wagner, Jen Olori, Kerin Claeson, Sebastian Egberts, Heather Ahrens, Chris Jass, Christian George, Nick Smith, Michelle Stocker, and Sterling Nesbitt escorted me through these years. The specimens I scanned were paid for in part through the Jackson School of Geosciences Analytic Funds grants.

The completion of this dissertation is a multi-generational success story. My parents and siblings though far away, provided support in many ways during critical phases of the work and only rarely asked when I would be done. In an extended family context George Rodgers and Robert Berlin helped me initiate and complete graduate school with support and patient listening. Michael, Genevieve, and Alexandra Rodgers grew up with a graduate student for a mother and did a marvelous job of raising me. My greatest thanks are reserved for them.

# **Comparative Morphology of the Vestibular Semicircular Canals in Therian Mammals**

Publication No. \_\_\_\_\_

Jeri Cameron Rodgers, PhD.

The University of Texas at Austin, 2011

Supervisor: Timothy B. Rowe

The peripheral vestibular membranous ducts that detect angular motion are contained within bony semicircular canals of mammalian petrosals. I investigated morphology and function in the three membranous semicircular ducts through measurements on the bony semicircular canals of 31 skeletonized skulls from different genera.

While the prevailing theory of semicircular canal researchers is that the locomotor agility of extant and extinct mammals can be understood by measuring the size of the three bony semicircular canal arcs, I propose that there are important and quantifiable features other than the adult size of radius of curvature of the semicircular canal arc ( $R$ ) that influence angular movement detection in mammals and perhaps in their ancestors.

Initially, I sought to verify that there was no significant asymmetry of  $R$  across the study specimens. However, there was significant asymmetry in canal pair angles

between contralateral sets: ipsilateral canal pair angles differed by up to 14°, and contralateral synergistic angle pairs differed by up to 18°. Canal pair contralateral differences were lower for specimens of more agile taxa. In addition, the angle between the left and right lateral canals varied by up to 27° from parallelism, so the use of the lateral bony canal in one petrosal to represent the horizontal animal resting position could result in significant skull orientation errors.

I utilized a program to quantify the effects of canal plane non-orthogonality and to calculate a maximum rotational sensitivity axis for a given taxon. My results concur with earlier research indicating that canal orientation significantly affects the location of maximum rotational sensitivity axes in the head, and should be considered in future quantitative research.

Finally, I determined the volumes of the subarcuate fossa and the petrosal lobule in three *Monodelphis domestica* animals (76 days postnatal) by utilizing both cranial and tissue volumes in fresh specimens. The petrosal lobule fills 93-97% of the *Monodelphis domestica* fossa, a greater volume than the 50% estimated by previous researchers. These results highlight the difficulties of using histologic or preserved specimens to make quantitative determinations of brain tissue volumes, and reopen the question of whether the subarcuate fossa volume provides a record of the agility for an extinct taxon.

## Table of Contents

List of Tables .....	xii
List of Figures .....	xiv
CHAPTER 1: INTRODUCTION TO THE BONY SEMICIRCULAR CANAL SYSTEM OF MAMMALS .....	1
General Structure of the Membranous Semicircular Ducts .....	2
Purpose of This Doctoral Research.....	3
Study Considerations .....	3
Dissertation Overview .....	5
CHAPTER 2: COMPARISON OF CONTRALATERAL SEMICIRCULAR CANALS IN THERIAN MAMMALS.....	9
ABSTRACT.....	9
INTRODUCTION .....	10
Background.....	11
Questions.....	13
Symmetry of R for contralateral canals (Question 1).....	14
Symmetry of canal pair angles (Question 2).....	15
METHODS .....	16
Reference Planes.....	17
Semicircular Canal Imaging .....	18
Comparison of R Calculation Methods.....	19
Defining the Orientation of Semicircular Canals.....	21
RESULTS .....	22
Comparison of R calculation methods.....	22
Radius of curvature for contralateral canals (Question 1) .....	24
Symmetry of angles between canal pairs (Question 2).....	25
DISCUSSION.....	28
Comparison of R calculation methods.....	28
Radius of curvature for contralateral canals (Question 1) .....	29

Symmetry of angles between canal pairs (Question 2).....	30
Study limitations .....	34
CONCLUSIONS.....	35
TABLES .....	38
FIGURES.....	53
CHAPTER 3: ROTATIONAL VESTIBULAR SENSITIVITY DIRECTIONS IN THERIAN MAMMALS.....	68
ABSTRACT.....	68
INTRODUCTION .....	69
Exemplar Animal: <i>Dromiciops gliroides</i> .....	70
Semicircular Canal System .....	70
Semicircular Duct Dimensions and Function .....	72
MATERIALS AND METHODS.....	76
RESULTS .....	78
DISCUSSION .....	81
Reference Plane Selection.....	81
Canal Maxima versus Prime Direction for Sensitivity Calculations ...	83
CONCLUSIONS.....	84
TABLES .....	86
FIGURES.....	88
CHAPTER 4: THE PETROSAL LOBULE ALMOST FILLS THE SUBARCUATE FOSSA IN <i>MONODELPHIS DOMESTICA</i> (DIDELPHIDAE, MARSUPIALIA): IMPLICATIONS FOR AGILITY EVALUATIONS .....	98
ABSTRACT.....	98
INTRODUCTION .....	99
MORPHOLOGY .....	101
Petrosal.....	101
Semicircular Canals .....	102
Subarcuate Fossa.....	103
Petrosal Lobule .....	104
Relative Petrosal Lobule Volume within the Subarcuate Fossa .....	105

<i>Monodelphis domestica</i> .....	107
METHODS .....	108
High-Resolution MRI .....	109
Petrosal Lobule Volume Calculations .....	110
Subarcuate Fossa Volume Calculations.....	111
RESULTS .....	113
DISCUSSION .....	114
Discrepancy of Present Results with Those of Previous Studies.....	114
The Petrosal Lobule as Represented within the Subarcuate Fossa ....	116
Semicircular Canals versus Subarcuate Fossa as Indicators of	
Agility .....	117
Developmental Considerations of the Subarcuate Fossa and	
Semicircular Canals .....	119
FUTURE WORK.....	121
TABLES .....	122
FIGURES.....	129
Appendices.....	137
Appendix 1. Coordinates and vectors for semicircular canals.....	137
Appendix 2. Comparison of $R_r$ and $R_{hw}$ .....	144
Appendix 3. Ipsilateral Canal Radius Differences.....	148
Appendix 4. Summary of Semicircular Canal Angle Relationships.....	150
Appendix 5. Differences in angles between left and right petrosal	
canals.....	152
Appendix 6. Instructions for assembly of the isomap C++ program used	
for sensitivity calculations. ....	154
Appendix 7. Sensitivity Results for All Taxa .....	260
Appendix 8. Isomaps for all taxa in this study.....	261
References.....	292
Vita .....	312

## List of Tables

<b>Table 2.1.</b>	Taxa, museum specimen number, High Resolution X-ray Computed Tomography image slices used for skull images, spacing between image slices in image stack, Field of Reconstruction, and image slice pixel size. ....	38
<b>Table 2.2.</b>	List of abbreviations.....	40
<b>Table 2.3.</b>	<i>Petauroides volans</i> (AMNH 150055) dimensions of left petrosal semicircular canals calculated by different methods .....	41
<b>Table 2.4.</b>	Equal variance, two-tail t-test for mean difference between petrosal angle pair difference, averaged for all three ipsilateral semicircular canals.....	42
<b>Table 2.5.</b>	ANOVA Single factor comparison of ipsilateral canal pair angles. ....	43
<b>Table 2.6.</b>	Previous semicircular canal pair angle research results.....	44
<b>Table 2.7.</b>	ANOVA single factor analysis of left ipsilateral canal pair angles taken from previous studies cited in Table 2.5. ....	52
<b>Table 3.1.</b>	Abbreviations used in this chapter. ....	86
<b>Table 3.2.</b>	S calculations for all combinations of the six <i>Dromiciops gliroides</i> semicircular canals, showing the values and locations of maxima and minima.....	87
<b>Table 4.1.</b>	$\mu$ MRI scan parameters .....	122
<b>Table 4.2.</b>	HRXCT scan parameters for study specimens.....	124
<b>Table 4.3.</b>	Summary of $\mu$ MRI and HRXCT scan results from specimens of <i>Monodelphis domestica</i> used in this study.....	125

<b>Table 4.4.</b>	Percent of subarcuate fossa filled by the petrosal lobe calculated from Table 4.3A scans. ....	127
<b>Table 4.5.</b>	Summary of ontogenetic specimens of <i>Monodelphis domestica</i> , ages, and volumes of petrosal lobules.....	128

## List of Figures

<b>Figure 1.1.</b> Structures of the right human membranous and bony labyrinth, anterior view .....	7
<b>Figure 1.2.</b> Diagram of a membranous semicircular duct and associated structures.....	8
<b>Figure 2.1.</b> Bony inner ear endocast of <i>Petauroides volans</i> (AMNH 150055) showing embedded head-centered reference planes .....	53
<b>Figure 2.2.</b> <i>Petauroides volans</i> (AMNH 150055) left petrosal bony labyrinth endocast of ipsilateral semicircular canals rotated into viewing plane along regression plane of best fit.....	54
<b>Figure 2.3.</b> Difference between canal plane normal and canal plane rotational vector.....	55
<b>Figure 2.4.</b> Arc radius of curvature determination methods compared as functions of body mass, plotted as double logarithmic plots of mean canal radius against body mass (BM) for 31 specimens, data from Appendix 2 .....	56
<b>Figure 2.5.</b> Distribution of $\Delta$ LSCR compared as functions of Log <sub>10</sub> body mass (BM), with a linear regression shown.....	57
<b>Figure 2.6.</b> Comparison of ipsilateral canal pair angles from left and right petrosals .....	58
<b>Figure 2.7.</b> Difference in ipsilateral canal pair angles from left and right sides of skull .....	59
<b>Figure 2.8.</b> Distribution of differences between ipsilateral canal pair angles between left and right sides of all study specimens.....	60

<b>Figure 2.9.</b> Side-averaged ipsilateral canal pair angles compared as functions of $\text{Log}_{10}$ body mass (BM) for differing agility values .....	61
<b>Figure 2.10.</b> Contralateral canal pair angles compared as a function of $\text{Log}_{10}$ body mass for all agility levels from Spoor et al. (2007) .....	62
<b>Figure 2.11.</b> Anterior and posterior contralateral canal pair angles compared as a function of agility as defined by Spoor et al. (2007). .....	63
<b>Figure 2.12.</b> Side-averaged canal pair angle between the anterior and posterior semicircular pair compared as a function of the anterior and posterior contralateral canal pair angles .....	64
<b>Figure 2.13.</b> Synergistic angle between left posterior semicircular canal and right anterior semicircular canal compared to the synergistic contralateral canal pair of the left anterior semicircular canal and the right posterior semicircular canal .....	65
<b>Figure 2.14.</b> The synergistic lateral canal pair angles compared as a function of $\text{Log}_{10}$ body mass for all agilities as defined by Spoor et al. (2007).....	66
<b>Figure 2.15.</b> Double logarithmic plots of mean side-averaged radius of curvature as a function of $\text{Log}_{10}$ body mass for slow, medium, and fast agilities provided by Spoor et al. (2007) .....	67
<b>Figure 3.1.</b> Working from skull morphology to answer questions about the source of membranous semicircular duct size and orientation .....	88
<b>Figure 3.2.</b> Skull of <i>Dromiciops gliroides</i> (FMNH 127463) showing orientation of designated reference planes and axes with respect to skull landmarks.....	89
<b>Figure 3.3.</b> Locations of inner ear structures and primary measurements used	

in this study shown for <i>Dromiciops gliroides</i> on digitally constructed endocast of the left inner ear .....	90
<b>Figure 3.4.</b> General orientations of canal planes and directions of rotation vectors taken in positive right-hand excitatory directions, as indicated on digitally rendered bony endocasts of <i>Dromiciops gliroides</i> inner ear .....	91
<b>Figure 3.5.</b> Unit sphere and contour map compilation for single semicircular canal and paired canal combination sensitivities shown for <i>Dromiciops gliroides</i> .....	92
<b>Figure 3.6.</b> Unit sphere and contour map compilation for ipsilateral canals on left and right sides and total combined canal sensitivities shown for <i>Dromiciops gliroides</i> .....	93
<b>Figure 3.7.</b> Contour map compilation for <i>Dromiciops gliroides</i> calculated by the Yang and Hullar program (2007).....	94
<b>Figure 3.8.</b> Illustrations of rotation terms. <b>A</b> , roll; <b>B</b> , pitch; <b>C</b> , yaw.....	95
<b>Figure 3.9.</b> Isomap for all six semicircular canals combined in <i>Mus domestica</i> generated with the MCR program.....	96
<b>Figure 3.10.</b> Isomap for all six semicircular canals combined in <i>Mus domestica</i> generated with the Yang and Hullar (2007) program.....	97
<b>Figure 4.1.</b> TMM M-9039. Juvenile <i>Monodelphis domestica</i> , aged 76 days ....	129
<b>Figure 4.2.</b> Left medial view of <i>Monodelphis domestica</i> (TMM M-9039) basicranium from 3D digital HRXCT reconstruction.....	130
<b>Figure 4.3.</b> <i>Monodelphis domestica</i> TMM M-9039 cutaway view of 3D HRXCT image reconstruction showing the plane of the left anterior semicircular canal.....	131

<b>Figure 4.4.</b> Endocasts of the bony semicircular canals and subarcuate fossa from <i>Monodelphis domestica</i> TMM M-9039 HRXCT scan.....	132
<b>Figure 4.5.</b> Coronal images of <i>Monodelphis domestica</i> TMM M-9039 taken at the maximum height of the left anterior semicircular canal .....	133
<b>Figure 4.6.</b> Transverse image of <i>Monodelphis domestica</i> (TMM M-9039) $\mu$ MRI postmortem scan taken through greatest width of subarcuate fossa .....	134
<b>Figure 4.7.</b> <i>Monodelphis domestica</i> TMM M-8265 Day 27 HRXCT coronal sections.....	135
<b>Figure 4.8.</b> Table 4.4 <i>Monodelphis domestica</i> ontogenetic series subarcuate fossa volume plotted against specimen age .....	136

## **CHAPTER 1: INTRODUCTION TO THE BONY SEMICIRCULAR CANAL SYSTEM OF MAMMALS**

Tunicate larvae need to sense ‘down’ for a few hours to one day to locate a hard substrate for attachment. These urochordates accomplish the simple task with a combined photo-gravity sensor (Sorrentino et al., 2000). Developmentally, the larvae exhibit the induction of an otic placode and adult tunicates possess neural hair cells similar to those of more complex vertebrate internal ears (Streit, 2001). Mammals benefit enormously from an improved, permanent sense of motion detection known as the peripheral vestibular end organ. This sensory organ generally functions so well that it occurs on an unconscious level in humans. However, malfunctions in our vestibular system become painfully obvious in a “profound, incapacitating influence on almost every aspect of our lives” (Highstein, 2004: 2). A patient who lost the use of his vestibular system emphasized: “I’d give up my hearing if it would mean getting my balance back” (Della Santina, 2010:68).

The existence of the inner ear or ‘labyrinth,’ as named by Galen in the 2nd century CE (Hawkins and Schacht, 2005), was considered an organ of hearing early on. Antonio Scarpa first dissected and described the membranous labyrinth of the inner ear (Canalis et al., 2001), allowing later researchers to determine that mammalian detection of motion was located within that system (e.g., Mach, 1873, Breuer, 1874, 1875, Brown, 1874, McKenzie, 1912) within the paired petrosal bones (periotics, petrous portion of the temporal bone). Experiments with pigeons and rabbits by Marie-Jean-Pierre Flourens in 1874 indicated that the sensation of balance was a separate function of the semicircular ducts (Hawkins and Schacht, 2005). Therefore, the membranous labyrinth of the inner ear (Figure 1.1) has a double function: hearing, which is detected by the cochlea within the

anterior-ventral-medial *pars cochlearis* of the petrosal; and motion detection, located in the vestibular end organs of the internal caudal-dorsal-lateral *pars canalicularis* (Wible et al., 1995). The vestibular organ itself consists of three membranous motion detectors: the sacculus (which detects head translation movements), the utricle (the ancient gravity detector), and three tubes that detect angular head motion: the anterior (superior), lateral (horizontal), and posterior (inferior) membranous semicircular ducts. I chose to investigate the morphology and function of these semicircular ducts for this dissertation, through the the study of the bony semicircular canals that contained them during life.

#### **GENERAL STRUCTURE OF THE MEMBRANOUS SEMICIRCULAR DUCTS**

Each semicircular duct consists of a tube of connective tissue bent into a truncated arch, originating at the sac of the utricle (Figure 1.2). The other end of the duct enlarges to form an ampulla, which then attaches to the utricle. All parts of this connected system are filled with a watery, cation-enriched endolymph. Within the ampulla an endothelial tissue contains embedded nerve cells called hair cells because they have hair-like extensions of kinocilia and stereocilia that extend into a gelatinous, connective tissue cupula. The cupula extends across the ampulla to the other side, to form an apparent diaphragm. With rotational movement within the plane of the duct system (shown by double-arrow curve in Figure 1.2), inertial drag by the fluid impinges on the diaphragm of the cupula and excites the connected nerve cells to send either excitatory or inhibitory signals along to the brain via the vestibular nerve. In all mammals, the anterior semicircular duct and the posterior semicircular duct are connected by a primary common crus (Figure 1.1), and in some animals the posterior and lateral semicircular ducts share a secondary common crus (not shown). The membranous duct system is tethered within the

bony spaces of the semicircular canals (Figure 1.1) which are filled with a secondary fluid, the perilymph.

## **PURPOSE OF THIS DOCTORAL RESEARCH**

Study of the membranous semicircular duct system can provide paleontologists with at least two sources of information on extant and extinct species, phylogenetic (e.g., Ekdale, 2005, 2009) or functional. The most interesting aspect of the system to me was its potential in understanding the locomotor lifestyles or agilities (*sensu* Spoor et al., 2007) of mammals and their extinct ancestors.

A prevailing theory is that the agility of an extinct mammalian species can be determined by the length of its orthogonal semicircular ducts (Gray, 1907, 1908, Jones and Spells, 1963, ten Kate et al., 1970, Oman et al., 1987, Spoor et al., 1996, 2007, Rabbitt, 1999, Spoor, 2003, Hullar, 2006, Ifediba et al., 2007, Yang and Hullar, 2007, Ekdale, 2009, 2010, Kandel and Hullar, 2010, Malinzak et al., 2011). However, I propose that there are quantifiable features other than the adult size of the membranous semicircular ducts which influence angular movement detection of those ducts in therian species and perhaps in their ancestors. The following chapters detail possible features and the results from researching each hypothesized feature.

## **STUDY CONSIDERATIONS**

The semicircular ducts detect angular changes in head movement through a complex architecture, which will be detailed in more detail as applicable in each chapter. The morphological and physiological system of the membranous semicircular duct needs

to be considered, but unfortunately it cannot be studied directly in skeletonized specimens or fossils. Paleontologists, then, have a triple hindrance to research in this sensory organ: 1) the detection system is complex and is still an active area of research by medical and vestibular researchers, 2) the functional tissues and fluids of the system are absent, and 3) the system is completely encased in the petrosal, the densest bone in the body. To deal with these hindrances, I have used several sources for the necessary information.

The comparative study of the vestibular system has progressed, but few animal models have received detailed sensory experimentation. Medical researchers, however, have interests in evaluating the normal human responses to movement detection, in correcting pathological conditions, and dealing with the issues of flight and space flight for extended periods. For example research in the galvanic stimulation of the semicircular duct system provides insight into normal angular movement detection, effects of flight, and potential correction of pathology (Goldberg et al., 1984, Moore et al., 2006, Son et al., 2008). This rich source of research results can inform new comparative studies such as the present one.

The missing membranous labyrinth in skeletonized material is approximately preserved in size through the bony spaces of the petrosal. In particular, the spaces of the bony semicircular canals approximate in acceptable extent the sizes of each missing duct formerly contained in each canal (Hashimoto et al., 2005, Ifediba et al., 2007). Therefore, the reader will note that each chapter discusses the bony semicircular canals rather than the ducts they once contained.

Unfortunately, the bony semicircular canals present study difficulties, because observing and measuring the spaces of the petrosal require natural or experimental destruction and concomitant loss of possible information. That destruction is especially

discouraged for valuable fossil specimens. However, since the 1990's the use of CT scanning technology for imaging the bony canals has allowed nondestructive acquisition of all requisite dimensions for study. Therefore, each study reported here utilized the High Resolution X-ray Computed Tomography (HRXCT) scans made available through the Jackson School of Geosciences (Rowe et al., 1997).

## **DISSERTATION OVERVIEW**

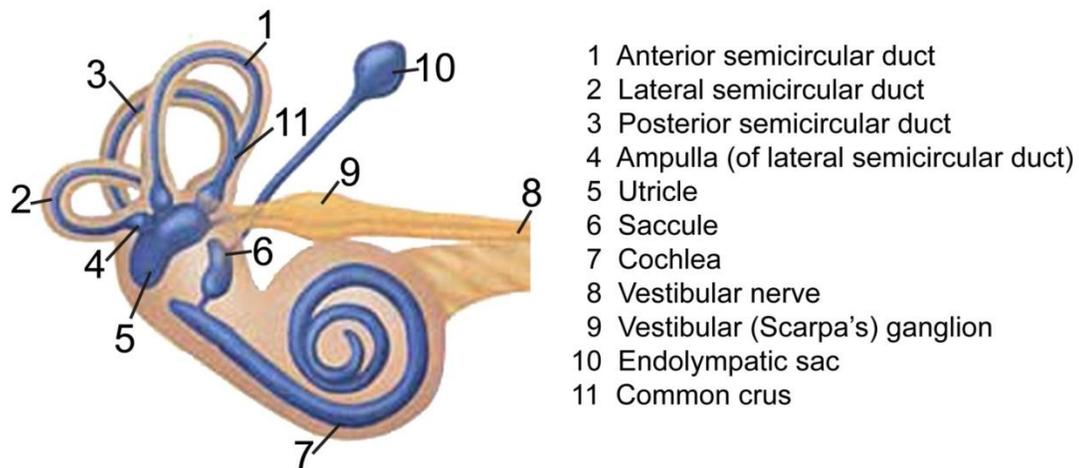
Chapter 2 details a comparative study of semicircular canal orientations for animals of known agility or locomotor style. Within each petrosal, there are two vertical canals, the anterior (superior) semicircular canal and the posterior (inferior) semicircular canal, and a lateral (horizontal) semicircular canal. Each canal has historically been considered orthogonal to the other two, the lateral canal has been thought to lie in the resting horizontal head orientation, and canals would detect rotations in mutually exclusive directions. Although measurements of semicircular canals in single species studies (usually including only domestic laboratory animals used in vestibular research and humans) had long documented lack of orthogonal relationships and non-horizontality of the lateral canal (see Table 2.5), very few of those studies looked at relationships for canals in both labyrinths and therefore side to side symmetry, and a study to investigate canal angle relationships has not been done across Theria for various locomotor methods and speeds (although see Malinzak, 2010, Malinzak et al., 2011 for a very recent study on primate canal orientation and agility). The analysis of the canal orientations for 31 study specimens of recent taxa indicates a lack of ipsilateral canal orthogonality. A second result is that the contralateral lateral canals are non-planar and non-horizontal. Finally,

there is a correlation between the magnitudes of differences in canal pair angles from contralateral sides and the agility of the specimen taxon agility.

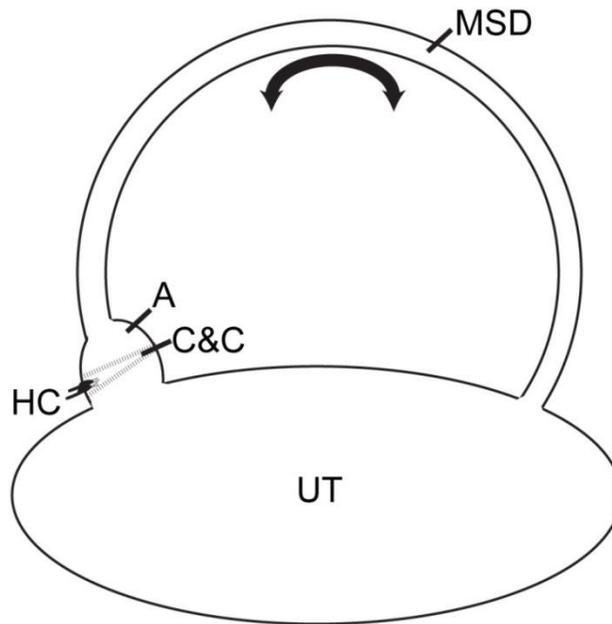
Given the results of the second chapter, how are the signals sent by semicircular ducts affected by their orientations? For example, a non-horizontally oriented lateral duct may register rotational movement with both horizontal and vertical components. Chapter 3 addresses the effects of the actual canal orientations on the firing rates of hair cell neurons through the use of a custom program that utilizes the measurements of both canal size and canal orientation for all six canals in the specimen (based on an example program detailed by Yang and Hullar, 2007). The program calculates a hair cell firing rate for each canal when rotated in any direction, does a vector summation of all six canals along any direction, and allows for additive or subtractive relationships. The results from such an analysis of the exemplar taxon *Dromiciops gliroides* are plotted and give a maximum direction of rotation to which the head is sensitive. Such results can be obtained for any specimen small enough for both petrosals to be scanned in situ.

Apart from angular relationships, numerous authors (e.g., Olson, 1944) have informally questioned whether the volume occupied by ontogenetic expansion of the petrosal lobule into the subarcuate fossa of non-hominoid petrosals either provides a comparable or improved measure of agility in extinct taxa. There have also been questions about the effects of a centrifugally expanding petrosal lobe upon the size of the developing semicircular canals (e.g., Sánchez-Villagra, 2002, Jeffery and Spoor, 2006, Jeffery et al., 2008). In order to investigate both questions, simultaneous measurements of the semicircular canal size, subarcuate fossa volume, and petrosal lobule volume are required, and this was difficult to do in the past. Through the use of inadequate methods or materials, some researchers even concluded that a consideration of the effects of the subarcuate fossa expansion upon semicircular canal size was moot, as the petrosal lobe

does not fill the expanding subarcuate fossa in ontogeny, or even the full-sized fossa in the adulthood of some taxa, such as *Monodelphis domestica*. In Chapter 4, the use of High-Resolution Magnetic Resonance Imaging ( $\mu$ MRI) and HRXCT scans on the same three specimens documents the filling of the subarcuate fossa by the petrosal lobule in *Monodelphis domestica*. It combines the volumes calculated for the studied specimens with those from another ontogenetic *Monodelphis domestica* series to document no restraint upon the expansion of the fossa other than in the restrictions formed by the canals. The question of agility and subarcuate fossa volumes can be pursued again with the assurance that cerebellar tissue does fill the subarcuate fossa in fresh specimens.



**Figure 1.1.** Structures of the right human membranous and bony labyrinth, anterior view. Blue structures, membranous labyrinth; pink structure, spaces of bony labyrinth; yellow structures, nerves (modified from <http://www.meniect.com/>).



**Figure 1.2.** Diagram of a membranous semicircular duct and associated structures. **A**, ampulla; **C&C**, crista and cupula; **HC**, hair cell nerve fibers; **MSD**, membranous semicircular ducts; **UT**, utricle. Arc with arrows show potential rotational movement sensed by this system.

## CHAPTER 2: COMPARISON OF CONTRALATERAL SEMICIRCULAR CANALS IN THERIAN MAMMALS

### ABSTRACT

Morphologists comparing the semicircular membranous duct system assume equivalence of differing arc radius calculation methods, as well as symmetry between contralateral semicircular canals. Non-equivalence of calculation methods would alert future researchers to the dangers of including previous studies without considering differences in calculation methods. Symmetry differences could have functional implications for animals with different locomotor lifestyles, and would raise questions about results obtained from measurements of only one specimen petrosal. I conducted quantitative analyses of bilateral semicircular canal endocasts from High Resolution X-ray Computed Tomography (HRXCT) scans of 31 specimens of extant species to address these assumptions. The most common analysis of the arc length of semicircular canals is the radius length ( $R$ ) defining the circle best approximating the canal arc. Best-fit linear regression calculations from numerous points along an arc radius of semicircular canal curvature ( $R_r$ ) provided better approximations of canal arc length than methods utilizing the average height and width ( $R_{hw}$ ) of the semicircular canal arc (differences of up to 26%), but neither measured true path length through canal and utricle. No significant systematic contralateral asymmetry of  $R_r$  was found for most specimens. Ipsilateral canal pair angles differed by up to  $14^\circ$  ( $4^\circ$  mean difference) between sides, and contralateral synergistic angle pairs differed by up to  $8^\circ$  ( $3^\circ$  mean difference). Highly agile species exhibited significantly lower contralateral angle pair variations. Side-averaged ipsilateral canal pair angles ranged between  $65^\circ$  and  $114^\circ$ , with orthogonality of only the lateral to posterior canal angle. Synergistic anterior and posterior canals varied up to  $20^\circ$  from

parallelism ( $10^\circ$  mean difference), and synergistic lateral canals varied by up to  $27^\circ$  from parallelism ( $12^\circ$  mean difference). Results from this study agree with previous studies of multiple specimens from single species.

## **INTRODUCTION**

Vertebrate paleontologists ascertain many functional and physiological characteristics from fossils, but interpreting sensory experiences of extinct species remains among the most challenging feature to decipher. Understanding sensory ability requires that some skeletal record of the sensory mechanics be preserved that can be interpreted through extant animal models. By teasing out the skeletal pathways of a preserved sensory pathway and comparing it to living animal models, researchers create inroads in understanding smell, hearing, vision (e.g., Cartmill, 1992, Kemp, 2009), and electroreception (Rowe et al., 2008). One of the last sensory organs to be found and traced to important balance functions is the sense of angular motion detection of the head, and that function takes place within the semicircular membranous ducts of the inner ear. Although the membranous ducts decay upon the death of an animal, they leave a record of their size and orientation within the skull.

While Urochordata has a larval otic gravity sensor (Streit, 2001), the appearance of semicircular canals occurred in fossil Craniata (Rowe, 2004) with the number of semicircular canals increasing to up to seven among fossil cyclostomes (Wever, 1975), and became stable in number at three canals in Gnathostomata (Rowe, 2004). These semicircular canals are found in skulls as spaces that once enclosed the membranous inner ear organs. Recently, through work with laboratory animals and volunteer humans,

neurologists and vestibular researchers also concluded that the inner ear also contributes to an animal's navigational abilities (e.g., Fitzpatrick et al., 2006).

Understanding the contribution of the inner ear to both balance and short-distance navigation of extinct mammals requires quantitative comparative studies (involving measurements of canal dimensions and orientation) on a variety of extant species with varying locomotor lifestyles. I present a preliminary quantitative survey with data from 31 extant marsupial and placental mammal species along with a discussion of the results from past researchers. The goal of this study was to evaluate symmetry in dimension and orientation of the membranous semicircular ducts (as recorded in their enclosing bony semicircular canals). This is the basis for interpreting the important functions of the vestibular organ in extinct mammals.

## **Background**

The connection between the membranous semicircular ducts and the sensation of angular acceleration was made two centuries ago (Hawkins and Schacht, 2005), but the challenges of studying an organ encased within dense endochondral bone hampered subsequent study of the physiological details of this complex system. Consequently, the tedious work involved in acquiring measurements of semicircular duct dimension and orientation was simplified by making two assumptions. At first it was assumed that dimensions of the bony semicircular canals containing the membranous semicircular ducts serve as reasonable proxies for the membranous duct dimensions. That untested assumption was confirmed by Hashimoto et al. (2005) and Ifediba et al. (2007). Second, contralateral (located on the opposite side) semicircular canals were, and still are, assumed to be essentially identical in dimensions and orientations. Testing this second assumption is the focus of this paper.

Three membranous ducts, along with the connected utricle, sacculus, and cochlea form each inner ear (Summers et al., 2004) (Figure 2.1). These inner ears are encased within ossifications of the otic capsule, known as the two petrosal bones of the mammalian skull (MacIntyre, 1972, Wible, 1990). In primates and a few mammals the petrosals may become part of the temporal bone to become the petrous portion (Gray, 1858).

Semicircular canals are each named by general orientation. The left petrosal contains an anterior (LASC), posterior (LPSC), and lateral (LLSC) semicircular canal with corresponding canals in sagittally-mirrored orientations on the right side (RASC, RPSC, and RLSC). Each canal traces an arc away from the vestibule, and then terminates in an ampulla that reconnects to the vestibule (Figure 2.1C). The shape of this arc, termed the arc of curvature, can be approximated by a partial circle with a center and a radius, termed the arc radius of curvature ( $R$ , Figure 2.1C).

The canals enclosed within one petrosal are termed ipsilateral canals. The combinations of left ASC with right PSC, right ASC with left PSC, and left LSC with right LSC are termed synergistic pairs, because an excitatory signal in one is complemented by an inhibitory signal from the other (an interaction also termed push-pull, Spoor, 2003).

A thorough summary of background research on the semicircular canal system was reviewed by Spoor (2003). In brief, theoretical models and practical testing on animals by Steinhausen (1933), Jones and Spells (1963), and ten Kate et al. (1970) showed that inertial drag by the endolymph filling a membranous duct acts upon hair cells in an ampulla to signal rotation of the canal. These signals combine as excitatory or inhibitory signals via the vestibular nerve and are sent directly to eye muscles for reflexive eye correction movements as the vestibulo-ocular reflex, and to spinal nerves

for reflexive body orientation corrections. They are simultaneously sent to the cerebellum and cerebrum for eye tracking movements (Jeffery et al., 2008). In early comparative research (Gray, 1907, 1908) and later quantitative research (Steinhausen, 1933, Jones and Spells, 1963, ten Kate et al., 1970, Oman et al., 1987), a larger bony canal R was associated with higher duct sensitivity. Those early researchers assumed that a larger canal would displace its ampullar hair cells to a greater degree, a conclusion supported by nerve response experiments (e.g., Yang and Hullar, 2007).

From that qualitative comparative work, researchers began histology, dissection, exposure, and radiographic study of canals to determine canal dimensions and orientations in numerous species (Blanks et al., 1972, 1975, Curthoys et al., 1975, Caix and Outrequin, 1979, Ezure and Graf, 1984, Matano et al., 1985, Lindenlaub et al., 1995, Spoor and Zonneveld, 1995, 1998, Calabrese and Hullar, 2006, Ifediba et al., 2007, Spoor et al., 2007, Silcox et al., 2009, Welker et al., 2009, Ekdale, 2010). The novel use of medical computed tomography to quantify the dimensions of the semicircular canals in fossil and modern specimens was first made by Spoor et al. (1994). Computed tomography is now the standard for most recent studies, because it is nondestructive and provides excellent resolution of internal skull spaces. I used High-Resolution X-ray Computed Tomography (HRXCT) for this study (Rowe, 1995, Rowe et al., 1997), because it provides the high resolution of internal cranial spaces. I used 3D imaging software to extract digital endocasts of semicircular canal spaces from HRXCT scan images for comparison of dimensions and orientations.

## **Questions**

I address the question of whether semicircular canals are contralaterally symmetrical in their size and orientation. The first part of this is to ask whether R is equal

for the same canal on the contralateral side. The traditional hypothesis was that R values do not differ significantly in corresponding contralateral canals for any individual animal (e.g., Rabbitt et al., 2004, Welker et al., 2009). Any systematic differences in the length of R in one of the two petrosals across numerous taxa could indicate a checking of canal responses by the vestibular processing system. A systematic difference between sides also would serve as a caution to future researchers who use only one petrosal for their canal analyses.

The second question is whether canal pairs show the traditionally assumed orthogonal angular relationships for ipsilateral and contralateral canal pairs, or parallelism (for synergistic canal pairs). The general assumption is that canal pair angles are symmetrical in contralateral petrosals and that all canal pairs are orthogonal or parallel (e.g., Romer, 1962, Anson and Donaldson, 1981). However, if ipsilateral canals are not orthogonal, each canal would detect angular acceleration within a direction also detected by another. In that case, certain directions of head rotation would induce increased neural firing than others, leading to the possibility of a preferred direction of head rotation sensitivity. That information could lead to the development of maps for the preferred head sensitivity direction for animals with different locomotor strategies.

### **Symmetry of R for contralateral canals (Question 1)**

First order afferent signals from the semicircular ducts travel through the vestibular nerve ganglia, superior vestibular nuclei and the abducens nuclei for ocular muscle coordination. The signals also travel through the vestibular nuclei and the vestibulospinal tract to spinal cord motor neurons for balance reflex corrections. Symmetrical dimensions of corresponding contralateral canals are considered necessary for eye and balance coordination. However there are other primary afferent signals

sent from the semicircular ducts to vestibular nuclei and the cerebellum which use comparison of signals from contralateral ducts in addition to inhibitory signals from synergistic canals to coordinate information from vestibular and ocular sources (Ezure and Graf, 1984). Would significant or systematic asymmetry of size in contralateral semicircular canals give evidence of those possible connections? To answer that question I compared R values determined via the regression fit method for contralateral canals in the study taxa. If those values show no significant difference in R values, then use of calculations from either petrosal can be justifiably used in locomotor agility studies. If a systematic asymmetry was found, then use of one petrosal in analyses would result in loss of information on signal matching in neural processing. My results indicate that in general there is no significant asymmetry in R values between right and left canals, but a few notable exceptions do occur. If those exceptions hold up in future measurements, bilateral studies should be conducted wherever possible.

### **Symmetry of canal pair angles (Question 2)**

Most previous researchers concentrated on the use of canal size to estimate agility (Jones and Spells, 1963, ten Kate et al., 1970, Matano et al., 1985, Spoor et al., 1994, Spoor, 2003, Spoor et al., 2007), but some recent authors included the orientations of canals in their calculation of vestibular sensitivities (Yang and Hullar, 2007, Malinzak, 2010, Malinzak et al., 2011). The classic canal sensitivity model requires three orthogonal canals interacting such that the maximum rotational signal from one canal occurs with no contribution from other rotational directions (e.g., Romer, 1962). The ASC and PSC are modeled as nearly vertical, and the LSC (also commonly referred to as the horizontal semicircular canal) lies within the horizontal plane in the resting head orientation (see Mazza and Winterson, 1984). The anterior and posterior canals are

modeled as orthogonal to their contralateral counterparts, with excitatory stimulation of one canal direction resulting in an inhibitory stimulation within the contralateral canal. Finally, synergistic canal pairs are modeled as parallel in space.

The classical model was not supported by canal measurements from humans and laboratory animals (e.g., Blanks et al., 1975, Curthoys et al., 1975, Ezure and Graf, 1984, Mazza and Winterson, 1984, Calabrese and Hullar, 2006). A theoretical consideration of canal orientation in vestibular sensitivity was provided by Rabbitt (1999), and utilized for the vestibular sensitivity of the mouse by Yang and Hullar (2007). Both sets of researchers concluded that the orientation of canals should be measured and considered in any study of vestibular sensitivity. If canal orientations are important in vestibular sensitivity, how symmetrical are canal orientations from one petrosal to the other? For this I asked whether variations in the angles on ipsilateral, contralateral, and synergistic canal pairs among different non-domestic mammal species were greater than the variations obtained for multiple individuals of a single species, many of which were domestic laboratory species. I also looked at several multiple-individual human research groups measured in the laboratory for semicircular canal orientation.

## **METHODS**

The skulls of thirty one extant therian mammals, each representing a different genus, were scanned by high resolution X-ray computed tomography at the Jackson School of Geosciences High-Resolution X-ray CT Facility. Taxon, museum specimen number, and scan parameters for each skull used in this study are listed in Table 2.1. Sex was generally unavailable for these specimens. Image stacks were imported into

VGStudioMax© (Versions 1.2 and 2.0; Volume Graphics GmbH, 2004 and 2007) for all 3D imaging and analysis. Abbreviations used in this paper are listed in Table 2.2.

## **Reference Planes**

Blanks et al. (1972:55) noted that “any description of the position of the semicircular canal planes requires a reliable external frame of reference.” For the present study, canal angle comparisons required stable head-centered reference planes, especially for angle comparison of contralateral canals. Three reference planes were determined and segmented into the 3D digital images before any other analysis was done. The terminology follows that of vestibular researchers (e.g., Rabbitt, 1999, Calabrese and Hullar, 2006, Ifediba et al., 2007) and not necessarily general morphological reference planes. Approximately eight small reference segments along the median sutures of the skull images (e.g., nasals, nasion, bregma, and medial palatine sutures) were aligned in a best-fit plane to define the vertical sagittal (XZ) plane. Numerous previous authors assumed that the LSC represents the horizontal plane of a live animal’s head orientation, thus the alternative designation of the canal as the horizontal semicircular canal (e.g., Romer, 1962, Rabbitt, 1999), but that also assumed that both the left and right lateral canals lie within the same horizontal plane. The assumption continues at present; David et al. (2010) attempted a reference system for the semicircular canal dimensions that does not utilize skull reference points and defined the horizontal plane as aligned with the lateral semicircular canals. For the horizontal or frontal plane (XY), I used Reid’s line, defined by the Taber’s Cyclopedic Medical Dictionary (Venes, 2005:1873) as the line extending from the lower edge of the orbit to the center of the aperture of the external auditory canal. Reid’s Plane is derived from those reference points on both sides of the skull. I defined this plane in 3D imaging software by selecting small image segments at

the ventral-most infraorbital margins of both orbits and small segments at the midpoint of the external auditory meatuses of both sides and including all four segments in a best-fit plane. The axial (YZ) plane contained the line connecting the two external auditory meatuses (interaural line) perpendicular to the frontal and sagittal reference planes. Thus, the center of this head-centered coordinate system occurred at the intersection of all three planes. The positive X axis passed through the skull's rostrum, the positive y axis passed through the left meatus, and the positive Z axis passed dorsally through the skull (Figure 2.1). Embedding the reference planes and center point as segments in the 3D skull volume rendering enabled all further measurements to be referred to this head-centered coordinate system. The frontal reference plane was not assumed to be in a horizontal position, and the plane was defined solely for the sake of measurement standardization.

### **Semicircular Canal Imaging**

In VGStudioMax the voids representing a bony labyrinth were outlined for each CT slice image and added to others to produce segments representing digital internal spaces or endocasts of both bony labyrinths. These endocasts were merged with the reference planes and extracted as a separate 3D file of the two inner ears. A resulting file of the reference planes and bony labyrinth endocasts for *Petauroides volans* is shown in Figure 2.1.

VGStudioMax allows a least-squares calculation of R that I designated  $R_r$  (in mm) approximating the method used by Calabrese and Hullar (2006). This method was based on earlier studies using destructive exposure of the bony canals mounted in head-referencing mechanisms similar to the original mechanism developed by Horsley and Clarke (Clark, 1939). Electrical probes were maneuvered along the canals for determination of arc radius of curvature dimensions in a three-dimensional Cartesian

coordinate along with a least-squares planar orientation of the canal (Curthoys et al., 1977). For my calculations in VGStudioMax, a measurement tool was used to select numerous points (60 to >100) representing the lumen centers of a canal from the end of the ampulla, around the canal and including the common crus (CC). I used the frontal slice orientations for selecting lumen center points, but checked with the two other orthogonal orientations to refine a lumen center-point selection, especially where the frontal slices were tangential to canal cross sections. VGStudioMax calculated a circle circumference by a linear regression best-fit of all the selected lumen points by minimizing the distance of each selected point to the final circle. The resultant magnitude of  $R_r$  was reported for each canal, along with the center coordinates of the best fit circle. The coordinates of all centers of all canals, as well as the center of the coordinate system and all  $R_r$  values were exported directly into Microsoft Excel files, and are shown in Appendix 1.

### **Comparison of R Calculation Methods**

Two separate methods exist for calculating semicircular canal radius of curvature. The original method was suggested by Gray (1907, 1908) who measured and reported the height and width of each semicircular duct. The height of the canal was given as the maximum distance from the vestibule to the zenith of the duct arc. The width was given as the maximum of the distance between two arms of the duct arc along a line parallel to the duct arc height. This method was modified for semicircular canal measurements by Spoor and Zonneveld (1994) and formalized as  $R = 0.5(\text{height} + \text{width})/2$  by (Spoor and Zonneveld, 1995). I designate the radius calculated by the height and width method as  $R_{hw}$ . The formalization of the height and width method allowed Spoor and Zonneveld (1995) to incorporate Gray's measurements into their own list of measurements.

The method of calculating  $R_r$  used by Calabrese and Hullar (2006) was updated from the earlier methods requiring exposure of each semicircular canal and electric probe recordings of points along the canal within the stereoscopic mounting. All used a least squares fit of numerous selected points along the canals to determine the arc of the canal and its radius. This method of determining  $R_r$  was never directly compared to the  $R_{hw}$  method by measurements from the same specimen, although measurements resulting from both methods were equated and listed in compilations of the arc radius of curvature from many taxa (e.g., Spoor et al., 2007). If significant and systematic differences exist between them, it would invalidate the use of both methods within a single comparative semicircular canal measurement compilation.

To determine whether such differences exist, I conducted a direct comparison on my own study species, by comparing the  $R_r$  for the LASC with a calculation of  $R_{hw}$  for the same canal in each specimen. After determining  $R_r$  for the LASC, I used VGStudioMax to rotate the plane of the LASC into alignment with one of three available orthogonal views that showed the entire anterior canal (see Figure 2.2), and the four points defining the lines of greatest height and width were selected in that view. From these points the equation  $R_{hw} = 0.5(\text{height} + \text{width})/2$  calculated the value of the Spoor and Zonneveld (1995) method, and these values were exported to an Excel file. The four points served as the basis for a regression calculation of  $R_{hw}$  to image a circle circumference, also shown in Figure 2.2.

An instrument available to VGStudioMax, the polyline, approximates the actual path of endolymph through a canal and its continuation through the associated ampulla and the utricle (Oman et al., 1987). With each canal radius defined and aligned to a 2D slice orientation, I used this instrument to select a consecutive series of points approximately 0.2 mm apart in the canals of *Petauroides volans* (paths shown in Figure

2.2). The polyline instrument sums the discrete distances between these point selections, and returns the total path length. A summary of the information obtained for all three canals of *Petauroides volans* is given in Table 2.3.

### **Defining the Orientation of Semicircular Canals**

Once each canal radius was defined, those same fitpoints were imported into another VGStudioMax best-fit calculation to obtain the plane containing that semicircular canal (Ezure and Graf, 1984), defined by coordinates of the unit normal axis perpendicular to that plane. A plane's normal line has no polarity, as shown in Figure 2.3A. However, each semicircular canal can be rotated in a direction that provides an increase in hair cell firing rate (excitatory direction), or if rotated in the opposite direction results in a decrease in hair cell firing rate (inhibitory direction). To express that additional information, the normal line, serving as an axis of rotation, was polarized to give a vector showing excitatory sensitivity direction according to the right-hand rule (Figure 2.3B) as described by Ezure and Graf (1984) and utilized by Calabrese and Hullar (2006). Mathematical calculation of angles between all canals was performed in VGStudioMax, with corrections to ensure all angles are internal (between the ipsilateral and contralateral canals directly), and not the external supplements of those canals (see Ezure and Graf, 1984, Cox and Jeffery, 2008). Naming convention of the angles closely follows that of Spoor and Zonneveld (1998). For example, the angle between the left anterior semicircular canal and the left lateral semicircular canal would be indicated by LASC<LLSC.

Agility categories for my study taxa were those assigned by Spoor et al. (2007) for the taxa also sampled by those researchers. A taxon not evaluated by those researchers was assigned an agility based upon its closest correlate from that group, along

with locomotor descriptions by Walker (Nowak, 1999). All the agility assignments are listed with taxa in Appendix 2.

For all values listed in the appendices, the values from each side measured are listed first, left side measurements before right side measurements. Where a  $\Delta$  is used (e.g.,  $\Delta$  ASC), it indicates a subtraction between the values from two sides (e.g.,  $\Delta$  ASC = LASC – RASC). Where % is used, it indicates the percentage of that subtracted value compared to the average of the two side values (e.g., %ASC =  $(\Delta$  ASC)/((LASC + RASC)/2)). Angles are indicated as LASC<LLSC (for the angle between the plane of the left anterior semicircular canal and the left lateral semicircular canal).

## **RESULTS**

### **Comparison of R calculation methods**

The direct comparison of calculations from the  $R_r$  and  $R_{hw}$  methods is shown in Appendix 2. Each calculation was based upon the left canals for specimens, with a calculation of  $(R_r - R_{hw})$  representing the difference between the measurements resulting from the two methods. Mean differences across all specimens ranged from 0.10 to 0.17 mm, with the LASC showing the largest difference, although the standard deviations for all canals were large. When the absolute values of the individual differences were indicated as percentages of averaged method calculation  $(R_r + R_{hw})/2$ , some results represented large percentages of the method-averaged R values themselves. Figure 2.4 illustrates that when plotted as a logarithmic function of R (mm) compared to BM (g), in nearly every specimen  $R_r > R_{hw}$ . Because the semicircular canals ossify early in development, the standard comparative depiction of the BM vs. R relationship indicates

negative allometry, and are therefore plotted as logarithmic relationships (Jones and Spells, 1963, Figure 2.4).

The circles calculated from both  $R_r$  and  $R_{hw}$  as imaged for the left petrosal endocast of *Petauroides volans* rotated to the best fit plane of the  $LASCR_r$ ,  $LLSCR_r$  and  $LPSCR_r$ , are illustrated in Figure 2.2. For this specimen  $LASCR_r$  was more than 20% greater than  $LASCR_{hw}$ , and the circles representing the circumferences described by each method calculation illustrate the difference for that canal, which has an oval shape. By not constraining the radius to the height above the vestibule, and by taking numerous measurement points (103) along the center of the canal lumen beginning at the vestibule around to its widening at the ampulla,  $LASCR_r$  more closely follows the true arc of the curve in those areas. However, it describes an unrealistic path below the utricle, that holds the reservoir of endolymph used within the canals.  $LASCR_{hw}$  does not follow the arc of the canal as closely, but it does approximate the path of endolymph flow into the utricle. In contrast, the circles of the  $LPSCR_r$  and  $LPSCR_{hw}$  differ by approximately 5% (Appendix 2, shown in Figure 2.5), and the lateral canal has a more circular shape, so both calculated circles pass through the utricle.

These results indicate that  $R_r$  provides a closer calculation of canal path length than  $R_{hw}$  when a canal has a nearly circular curvature, but neither method provides a close measurement of path length when a canal has an eccentric curvature, which is given from the polyline results in Table 2.3. When mean values of  $R_r$  were compared with those of  $R_{hw}$ , t-test results indicated no significant variation between the two. In addition, ANOVA comparisons of the mean percentage method differences for all three canals proved insignificant because of differences in the variance between those canals (see Appendix 1). However, 95% confidence levels for all three canals indicated that at minimum the percent differences between measurements of  $R_r$  and  $R_{hw}$  ranged from 7%

to 10%. Among the three canals percent differences between the two measurement methods ranged from 1% to 32%.

### **Radius of curvature for contralateral canals (Question 1)**

The  $R_r$  calculations for each canal (in mm) on both sides of the skull for each specimen are listed in Appendix 3.  $R_r$  values for the right canals were subtracted from their contralateral canals and listed; then, to allow for magnitude differences of  $R_r$  values between differently-sized species, the absolute value of that left-right difference was divided by the average of the two canals and reported as a percentage. The average interspecies differences between each contralateral canal pair were 3.0 - 4.5%. Of the sampled species, the arc radius of curvature difference for the PSC contralateral pair ( $PSCR_r$ ) showed the smallest variation with  $ASCR_r$  and  $LSCR_r$  having approximately equal variations. The differences between these mean interspecies percentages are strongly influenced by results from the *Chironectes minimus* and *Chrysochloris* sp. specimens, which indicate strong asymmetry in their  $ASCR_r$  and  $LSCR_r$ . Removing these specimens from calculations reduced  $ASCR_r$ ,  $LSCR_r$ , and  $PSCR_r$  to 3.2, 4.0, and 3.1%, respectively.

No asymmetry is indicated in any of the canals, because calculations of differences between left and right  $SCR_r$  resulted in a nearly equal number of positive and negative values, no significant differences from one canal orientation, and no indication of larger differences with increased BM, as shown in Figure 2.5

## **Symmetry of angles between canal pairs (Question 2)**

The calculated angles for canal pairs commonly considered in sensitivity studies are listed in Appendix 4. The angle between two canals is indicated by the example of the ‘angle between the left ASC and the left LSC’ being ‘LASC<LLSC’. Angle differences between left and right canal pairs in Appendix 5 were reported in degrees (e.g.,  $\Delta$  ASC<LSC).

A first consideration in angle relationships was whether ipsilateral canal plane pairs had angles equal to those of the corresponding pairs in the contralateral petrosal, so canal pair angles from the right petrosal were subtracted from those on the left side for the three ipsilateral pairs and the results listed as Side Comparisons in Appendix 5. A negative value indicates that the right canal pair angle is greater than that of the left side. Angle differences across all canal pairs ranged from  $-13^\circ$  to  $19^\circ$ , with mean values of  $0^\circ$  to  $1^\circ$ . Large differences were infrequent, and there was no indication that one side had consistently larger angles than the other, or that any canal pair showed larger or smaller mean differences. A graph of the ipsilateral angle pair values compared for both petrosals is depicted in Figure 2.6. The lack of angle equivalence is apparent, as indicated by the best-fit regression values. There appears to be no comparative increase or decrease in angle differences with increasing body mass, even when agility is also considered (Figure 2.7,  $R^2 < 0.2$  for all regression lines). However, a plot of the angle differences between all ipsilateral canal pairs from the two petrosals suggested less angle variation for animals with the highest agility scores (Figure 2.8). Results of a t-test based on mean of the absolute value of canal angle differences for the three ipsilateral canal pairs supported this observation (Table 2.4). The test returns a value above critical level with  $P < 0.01$ , when using a two-tail distribution of angle differences and assuming equal variance.

This result indicates that greater agility correlates with a decreased variation of angles between canal pairs from contralateral petrosals. This invalidates the null hypothesis while supporting the alternate hypothesis of agility influencing variation in symmetry of canals between petrosals for this question. Results from the side comparisons also indicated that neither petrosal showed a systematic increase in any ipsilateral canal pair angles, the actual absolute value of the difference in canal angles was not zero. The absolute value of the difference in a canal pair angle from contralateral petrosals is summarized in Appendix 5. Means of those differences for all canal pairs approximated 4°, and 95% confidence levels ranged from 3° to 5°.

Ipsilateral canal pair angles averaged from both petrosals for each specimen (Appendix 5) showed that LSC<PSC had a mean value of 92° with a 95% confidence level that included orthogonality. This angle approximates the ‘ideal’ classic model when considered across a range of mammalian genera. The mean of the ASC<LSC averaged for both sides was 85°, and the 95% confidence interval did not include orthogonality. The mean ASC<PSC averaged for both sides was 94%, close to that of LSC<PSC, and a 95% confidence level that did not include orthogonality. ANOVA single factor analysis indicates that the variance between average angles for the different canal pairs is significant (Table 2.5), with  $P < 0.0001$ . Thus the null hypothesis is rejected, with departures of canal angle pairs from orthogonality, and significant differences between the mean angles of the three pairs. These side-averaged ipsilateral canal pair angles showed no relationships to animal mass or agility (Figure 2.9).

The angles of contralateral canal pairs LASC<RASC and LPSC<RPSC had mean values of slightly less than 90°. There was considerable variation in these angles, ranging from 62-107°, and no apparent relationship between those angles and body mass or agility of genera (Figure 2.11, no significant t-test values). These results did not

invalidate the null hypothesis that there is no systematic variation in these canals. These contralateral canal pair angles show an inverse relationship to the average ipsilateral ASC<PSC canal pair angles when plotted for all specimens (Figure 2.12), which would be logical if the sum of all of these angles approximates 360° in the XY plane (ignoring contributions along the other two axis directions).

Synergistic canal pair angles ranged from less than 180° (obtuse) to greater than 180° (reflexive). There is also no ‘internal angle’ between such pairs, because they occur on opposite sides of the skull. In visually checking initial values for the LLSC<RLSC pair, I found three incidents of reflexive angles (i.e., both canals inclined at more than 90° from the positive Z axis), so I reported all synergistic canal pair angles as differences from 0° in Appendices 4 and 5. By doing this, the coefficients of variation became meaningless in these calculations. To invalidate the presumptive hypothesis, a synergistic canal pair angle significantly greater than 0° is required. Synergistic anterior-posterior canal pairs deviated from parallel by angles of up to 24°, with mean values of 10° for both sets. The side-averaged synergistic ASC<PSC angle for each specimen gives no indication of relation to either body mass or agility (supported by non-significant t-test results), shown in Figure 2.13.

Although there are wide variations in the angles between these two synergistic canal pairs, there is no indication that one set is systematically larger than its contralateral equivalent (approximately equal positive and negative values in the subtraction of the right pair from the left pair in Appendix 5). The mean absolute value of all specimen differences was 4°, similar to that of the ipsilateral canal pairs. Again, there is no relationship between agility and angles from synergistic, contralateral canal pairs (Figure 2.14).

The mean angle of all synergistic lateral canal pairs showed the greatest departure from the classic model, with a value of  $12^\circ$ . Seven of the 31 specimens had lateral canals that departed from parallelism by  $20^\circ$  or more. In such specimens it is difficult to say that the two canals essentially respond to movements in a purely horizontal plane or are even strictly synergistic. Again, there was no relationship between these angles and body mass or agility (Figure 2.14).

## **DISCUSSION**

### **Comparison of R calculation methods**

The comparison of  $R_r$  and  $R_{hw}$  methods indicates that the  $R_r$  method more closely approximates the actual polyline path of a semicircular canal. This dimension can be valuable in comparisons of double logarithmic functions of R with body mass that were utilized in agility studies beginning with Jones and Spell (1963). However, Jones and Spell (1963) did not describe whether they based their measurement of R upon height and width or by calculation from numerous points selected around the membranous duct.

Although neither the  $R_r$  or the  $R_{hw}$  method accurately described the total length of the canal path that includes that of the utricle length (for which the membranous labyrinth must be seen), at present the most common use of R measurements is to plot them in logarithmic comparisons with the body mass of an extinct or extant species to provide a qualitative estimate of agility (Spoor et al., 1994, 2007). I plotted such a comparison in Figure 2.15 including both the  $R_r$  and  $R_{hw}$  results from the present study, as well as the data from Spoor et al. (2007). According to those authors a ‘slow’ animal would plot below the linear regression representing a linear best fit of all specimens included in the study, while a ‘fast’ animal would plot above the regression. Given the generality of the

definition, using  $R_r$  as opposed to  $R_{hw}$  in the regression plot would result in variable changes in the individual agility assignments.

A different consideration for these method differences can be found in the vestibular-nerve afferent sensitivity study conducted by Yang and Hullar (2007). Again using *Petauroides volans*, the estimated sensitivity of the left anterior semicircular canal regular afferent signal would increase from  $G_{rhw} = 0.28$  (spikes  $\cdot$  sec<sup>-1</sup>)/(degrees  $\cdot$  sec<sup>-1</sup>) to  $G_{rr} = 0.36$  (spikes  $\cdot$  sec<sup>-1</sup>)/(degrees  $\cdot$  sec<sup>-1</sup>) with essentially no change in the lateral or posterior canals. Thus, a quantitative study of vestibular signals that are sent to higher order processing systems can be affected by the method used. Additionally, when R values are combined with orientation data from the canals, overall sensitivity patterns can be affected by differences in R calculation methods (Yang and Hullar, 2007). The method chosen by researchers for future investigations involving calculations of R should be informed by whether the goals of a future study would include sensitivity calculations, or other more qualitative comparative descriptions of semicircular canals.

Other methods do exist for evaluating the size of semicircular canals, based either on the total traced length around the path of the canal and through the utricle, or on the area enclosed by that traced length (Oman et al., 1987, Lindenlaub et al., 1995, McVean, 1999, Rabbitt, 1999). Most recently Cox and Jeffery (2010) compared the traced path method with R estimations and found little difference between the two methods.

### **Radius of curvature for contralateral canals (Question 1)**

Recently Welker et al. (2009) compared R measurements from both the left and right petrosals of wild-caught *Blarina brevicauda* for symmetry. They reported R measurements for both left and right petrosals, which allows a comparison with the percent variation of their specimens with those from my study. They reported ranges of 0

to 13% in R measurements from contralateral sides of specimens, with mean differences of < 3% for all three canals. The authors found no significant asymmetry of semicircular canals in *Blarina brevicauda*, and listed similar results from previous authors. Most of the single specimens from my study gave similar percent range R differences as their measurements for an individual species, with a few significant exceptions. The specimens exhibiting more than a 10% difference between sides were *Chironectes* (ASCR = 31%, LSCR = 17%), *Chrysochloris* (ASCR = 13%) and *Thylacinus* (LSCR = 12%). These species warrant symmetry measurements on additional specimens to confirm true intraspecies asymmetry or to document warping effects in the specimens I used, but otherwise my results concur with many previous studies of multiple specimens that either averaged R values from both skull sides or measured the canals from one side of a specimen (e.g., Blanks et al., 1972, Curthoys et al., 1975, Ezure and Graf, 1984, Della Santina et al., 2005, Calabrese and Hullar, 2006, Hullar and Williams, 2006). Asymmetry of semicircular canal dimensions does not appear to play a part in balance or navigational signals to the brain, regardless of agility.

### **Symmetry of angles between canal pairs (Question 2)**

I found no significant asymmetries between ipsilateral canal pair angles measured from left and right petrosals that researchers need to consider when looking at the vestibular system from a therian specimen with unknown agility. This validates the practice of the researchers who previously measured the orientation of canals and averaged contralateral canal pair angles. Of the previous researchers that reported angles from ipsilateral canal pairs (Table 2.5) only Calabrese and Hullar (2006) and Hullar and Williams (2006) listed mean values for each side. Differences between mean values of

each side were  $< 2^\circ$  for two strains of mice and  $< 3^\circ$  for the chinchilla, while my measurements average approximately  $4^\circ$  for one specimen each of 31 genera.

However, the finding that mammals with greater agility also have a lower angle variation for left and right ipsilateral canal plane pairs was a new one. While that relationship does not affect qualitative comparative studies of the vestibular system of a mammal, a more quantitative investigation of the functional or phylogenetic implications merits further attention. In particular, research similar to that by Yang and Hullar (2007) could find differences in maximal vestibular afferent sensitivity that results from such contralateral orientation differences.

Significant differences in the distributions of ipsilateral canal pair angles and particularly the lower mean value of the ASC<LSC from single species representatives can be compared with previous research that utilized numerous specimens of single species for their calculation of angle mean. While previous authors consistently mentioned other research looking at ipsilateral canal pair angles, few authors considered the results from a large group of mammalian species. All research chosen for Table 2.6 utilized methods that looked at total angle variation, rather than variation evident from projection of canal pairs onto one reference plane. Data from 209 specimens of 43 species is summarized in Table 2.6. For each ipsilateral canal pair, most researchers reported angles averaged from both petrosals, which minimized individual specimen variation. *Felis catus* has nearly orthogonal ipsilateral canals (Blanks et al., 1972, Ezure and Graf, 1984), as does *Saimiri sciureus* (Blanks et al., 1985). Results for the human are equivocal for the ASC<LSC between two studies; the results from Blanks et al. (1975) showed an angle of  $111.8^\circ$  for measurements from 10 specimens, but Della Santina, et al. (2005) measured the same angles for 10 specimens with a mean angle of  $90.6^\circ$ . The result from Blanks et al. (1975) was apparently skewed by the planar projection techniques they

used (Della Santina et al., 2005). Laboratory mouse strains investigated by Calabrese and Hullar (2006) provided results notable for non-selected variation within a laboratory species model. The ASC<LSC was close to orthogonal for both strains, but ASC<PSC differed by 10° between the two strains, and both had LSC<PSC angles > 101°. *Macaca mulatta* had ipsilateral canal pair angles in approximate equivalence and orthogonality (Blanks et al., 1985). For *Oryctolagus cuniculus*, Ezure and Graf (1984) found angles of 76° to 80° for all canal pairs, whereas Mazza and Winterson (1984) reported a significantly higher angle for ASC<LSC because Mazza and Winterson (1984) did not correct their equations to use only the internal angles (Cox and Jeffery, 2008). This was corrected in Table 2.6. My measurements did not include specimens of the domestic rabbit, but included a specimen of *Lepus californicus* for initial comparison. That specimen had side-averaged ipsilateral canal pair angles of 87-95°, nearly orthogonal. This difference may point to phylogenetic, functional, or domestication differences, any of which merit further investigation.

Means for all angles reported by all researchers confirm the ipsilateral canal pair angle relationships summarized in Appendix 4. Side averaged means for all studies were 83° for ASC<LSC, 92° for ASC<PSC, and 91° for LSC<PSC. ANOVA single factor analysis for the means from the measurements of left ipsilateral canal pairs (50 researchers reported these angles, Table 2.6) gave nearly identical results to mine (Table 2.7) with a significant result ( $P < 0.001$ ).

Only six research groups reported the angles of contralateral anterior canal pairs or contralateral posterior canal pairs from their chosen species, so comparison with my measurements was limited. The results from previous researchers gave a mean value for contralateral ASC canal pairs of 84° and a mean value for contralateral PSC canal pairs

of 95° (Table 2.6). The contralateral PSC canal pair angle standard deviation from previous researchers was large ( $\pm 26^\circ$ ).

Few researchers reported synergistic ASC<PSC canal pair angles and most researchers averaged the angles from both sides. Of the researchers that reported separate values for each pair, the mouse strain angles reported by Calabrese and Hullar (2006) again showed noticeable synergistic angle differences for the two strains, as well as some asymmetry between the angles of the CBA/CaJ strain. The two research groups reporting these canal pairs for *Oryctolagus cuniculus* averaged the measured angles for both sides of the skull, and for unknown reasons the Mazza and Winterson (1984) angle results (27° for LASC<RPSC, 27° for RASC<LPSC, and 15° for LLSC<RLSC) are nearly twice those of Ezure and Graf (1984) (14° for LASC<RPSC, 14° for RASC<LPSC, and 9° for LLSC<RLSC).

Earlier researchers considering R started with the assumption that the main effect of that dimension was primarily upon the vestibulo-ocular and proprioception reflexes. Recent authors considered endolymph flow interactions between canals, and interaction of canal vectors in the various reference directions (Rabbitt, 1999, Rabbitt et al., 2004, Ifediba et al., 2007). A convention introduced by Rabbitt (1999) and utilized by Ifediba et al. (2007) defines a prime direction for each canal. The prime direction is defined as the rotational vector direction for a canal that acts in the null component direction of the other two ipsilateral canals. This vector component for the canal is separate from the vector representing the plane of the canal expressed in the right-hand rule for excitatory afferent signal maximum direction. Recently Cox and Jeffery (2008) investigated the relationship between canal prime directions and the vestibulo-ocular reflex, and found little correlation between ocular muscles and orientation of the prime directions. They

noted that neural activity within the vestibular nuclei of the flocculus might correct for these effects.

### **Study limitations**

Results from the present study must be viewed in light of potential limitations and sources of error. In terms of canal pair angle determinations, canal planarity should be considered. Previous researchers noted inflections and ‘torsion’ in canal planes (e.g., Curthoys et al., 1975, Cox and Jeffery, 2010, Spoor and Zonneveld, 1998). However, deviations from planarity do not significantly affect calculations based on planar regressions (Cox and Jeffery, 2010).

Skull warping during skeleton preparation and storage may affect angle measurements of canal pairs. The petrosal is the densest bone in the mammalian skull, so warping within the petrosal itself should be limited, and ipsilateral canal pair angle measurements should be the least affected by warping. Contralateral and synergistic canal pair angles may be affected by such warping, and use of single specimens for each species determination needs to be considered in this light. However, warping should affect the side differences between synergistic anterior and posterior canal pairs, but those differences were the smallest of the canal pairs measured, which would indicate that skull warping was not significant in these specimens.

My research addressed only angular relationships between semicircular canals, and not any asymmetries between corresponding canals and their orientation with regard to head-centered reference planes, although that information is recorded in Appendix 1. Such orientations need to be utilized in future quantitative analysis of semicircular canal relationships and an animal’s vestibular sensory sensitivity. In that regard, describing a complication of setting up a head-centered reference system based upon the external

acoustic meatus is relevant when dealing with specimens of typical experimental mammals. Animals that inhabit fossorial (e.g., *Talpa*) or semi-aquatic (e.g., *Enhydra*) environments possess a meatus displaced far away from the vestibular system, making an orientation that approximates the inner ear/eye reference orientation impossible using the traditional Reid's plane definition. As research increases on animals with lifestyles differing from the typical laboratory environment, a new orientation system would be beneficial. Aligning image renderings to the fenestra vestibuli and infraorbital margin (or even the maxillary/incisor contact) might allow for more standardized orientation.

Finally, bony semicircular canals serve only as proxies for the membranous semicircular ducts, and the semicircular ducts fill only a portion of the bony canals in certain species (Curthoys et al., 1977). In those species (which include rodents and primates) membranous ducts have slightly different planar relationships that are limited within the bony canals. Such effects should be minimal, however.

## **CONCLUSIONS**

The canal pair angle relationships I measured agree with a compilation of previously researched multiple specimen, single species canal pairs in showing wide variation of ipsilateral, contralateral, and synergistic canal pairs. Within each specimen there was general symmetry in contralateral petrosals, but the correlations of angle pair differences between contralateral petrosals and agility were unanticipated.

Future researchers will begin to focus more on quantitative analyses combining the dimensions and orientations of the semicircular canals. The angle variations found in the species represented here should be considered in that research.

The navigational contributions of higher order semicircular canal signal processing were previously investigated for animal models and humans that navigate primarily in a 2D fashion. Those studies showed that the horizontal projection of these signals affects navigation (Fitzpatrick et al., 2006, Angelaki and Cullen, 2008). In animals that navigate extensively in 3D fashion (e.g., arboreal, volant, or semi-aquatic species) will the navigational processing also include vertical components of the vestibular system?

Lateral canal pairs consistently vary from parallelism in the species I measured. In ten specimens (*Caluromys*, *Cercartetus*, *Chironectes*, *Dromiciops*, *Glaucomys*, *Heterocephalus*, *Monodelphis*, *Notoryctes*, *Petauroides*, and *Wallabia*), the lateral canal pair orientations deviated from parallel by two standard deviations or more. The specimens with such high deviations ranged from those with low agility (e.g., *Notoryctes*) through those with high agility (e.g., *Petauroides*). Given the wide variation of the lateral canals from parallelism, to what extent can a researcher assume all of the signals from the lateral canals should be ascribed to rotation within only the horizontal plane? The synergistic lateral canal pair angle variation should also raise questions about assuming the lateral canal represents the horizontal resting orientation of a mammal's head (e.g., Mazza and Winterson, 1984).

Vestibular research (primarily on humans) previously determined that the vertical canals (ASC and PSC) contribute most significantly to balance and vestibulo-ocular reflexes, while the lateral canal (frequently termed the horizontal canal) influences navigational abilities (see Day and Fitzpatrick, 2005; Angelaki and Cullen, 2008). Previous authors specified that it is the horizontal projection of the LSC that produces navigational information processed in higher-order neuronal pathways, but did not specify whether the canal radius of the canal, orientation of the canal, or both, influences

that signal. Additionally, if the ASC and PSC are not oriented in a precisely vertical manner, do the horizontal vectors of their signals also influence this navigational processing? Results from my research and future semicircular canal orientation research from mammals can help resolve these questions.

**TABLES**

**Table 2.1.** Taxa, museum specimen number, High Resolution X-ray Computed Tomography image slices used for skull images, spacing between image slices in image stack, Field of Reconstruction, and image slice pixel size. Museum Abbreviations: **AMNH**, American Museum of Natural History, New York; **FMNH**, Field Museum of Natural History, Chicago; **TMM**, Texas Natural Science Centers, Vertebrate Paleontology, Austin; **UCL GMZ**, University College, London Grant Museum of Zoology; **SO**, University of California Los Angeles Museum; **UCLA**, University of California Los Angeles; **UCMVZ**, University of California Museum of Vertebrate Zoology, Berkeley; **UMMZ**, University of Michigan Museum of Zoology, Ann Arbor.

<b>Taxon</b>	<b>Museum Specimen Number</b>	<b>Number of slices</b>	<b>Interslice Spacing (mm)</b>	<b>Field of Reconstruction (mm)</b>	<b>File size (pixels)</b>
<i>Acrobates pygmaeus</i>	AMNH 155057	406	0.03	28.0	1024 X 1024
<i>Allactaga major</i>	AMNH 178795	1170	0.04	37.0	1024 X 1024
<i>Anomalurus beecrofti</i>	AMNH 50483	1270	0.04	39.0	1024 X 1024
<i>Caluromys</i> sp.	AMNH 95526	746	0.08	38.0	1024 X 1024
<i>Cercartetus caudatus</i>	AMNH 155090	705	0.04	18.0	1024 X 1024
<i>Chironectes minimus</i>	AMNH 129701	1522	0.04	33.0	1024 X 1024
<i>Chrysochloris</i> sp.	AMNH 82372	513	0.05	31.0	1024 X 1024
<i>Crocota crocuta</i>	UCMVZ 184551	528	0.50	166.0	512 X 512
<i>Dactylopsila trivirgata</i>	AMNH 104040	1301	0.05	45.0	1024 X 1024
<i>Dromiciops gliroides</i>	FMNH 127463	711	0.04	16.6	1024 X 1024
<i>Enhydra lutris</i>	SO 2853-97	645	0.22	106.0	1024 X 1024
<i>Glaucomys volans</i>	TMM M-6332	474	0.08	22.9	522 X 522
<i>Hemibelideus lemuroides</i>	AMNH 154375	1207	0.05	40.0	1024 X 1024
<i>Heterocephalus glaber</i>	AMNH 113974	1050	0.02	21.0	1024 X 1024
<i>Lepus californicus</i>	TMM M-7500	660	0.14	67.0	1024 X 1024

<b>Taxon</b>	<b>Museum Specimen Number</b>	<b>Number of slices</b>	<b>Interslice Spacing (mm)</b>	<b>Field of Reconstruction (mm)</b>	<b>File size (pixels)</b>
<i>Monodelphis domestica</i>	TMM M-9039	885	0.14	21.0	1024 X 1024
<i>Notoryctes typhlops</i>	AMNH 202107	705	0.04	18.0	1024 X 1024
<i>Pedetes capensis</i>	AMNH 42016	1145	0.07	67.0	1024 X 1024
<i>Petauroides volans</i>	AMNH 150055	1251	0.05	48.0	1024 X 1024
<i>Petaurus breviceps</i>	TMM M-8226	555	0.07	33.0	1024 X 1024
<i>Petropseudes dahli</i>	AMNH 183391	1424	0.05	46.0	1024 X 1024
<i>Potorous tridactylus</i>	AMNH 65337	915	0.01	48.0	1024 X 1024
<i>Pseudocheirus peregrinus</i>	TMM M-847	795	0.09	43.0	1024 X 1024
<i>Pseudocheirops cupreus</i>	AMNH 151829	1289	0.05	49.5	1024 X 1024
<i>Pseudochirulus forbesi</i>	AMNH 104136	1339	0.03	33.0	1024 X 1024
<i>Sciurus niger</i>	UMMZ 123729	450	0.16	44.4	512 X 512
<i>Talpa europaea</i>	UCL GMZ 5437	585	0.06	18.5	1024 X 1024
<i>Tarsipes rostratus</i>	AMNH 119717	921	0.03	13.0	1024 X 1024
<i>Thylacinus cynocephalus</i>	AMNH 35244	458	0.50	140.0	1024 X 1024
<i>Vulpes vulpes</i>	UCLA 13112	825	0.17	80.0	1024 X 1024
<i>Wallabia bicolor</i>	TMM M-4169	885	0.16	74.5	1024 X 1024

**Table 2.2.** List of abbreviations

Abbreviation	Definition	Source
ASC		
BM	Body Mass	
LASC	Left anterior semicircular canal	
LASC<LLSC	Angle between the left ASC and the left LSC	
LASC <sub>r</sub>		
LLSC	Left lateral semicircular canal	
LLSC <sub>r</sub>		
LPSC	Left posterior semicircular canal	
LPSC <sub>r</sub>		
LSC		
N	Normal to semicircular canal plane	
PSC		
R	Arc radius of semicircular canal curvature	(Jones and Spells, 1963)
RASC	Right anterior semicircular canal	
R <sub>hw</sub>	Radius of semicircular canal arc defined by $R_{hw} = 0.5 \times (\text{height} + \text{width})/2$	(Spoor and Zonneveld, 1995)
RLSC	Right lateral semicircular canal	
RPSC	Right posterior semicircular canal	
R <sub>r</sub>	Radius of semicircular canal arc defined by best-fit calculation of numerous points along semicircular canal arc.	(Curthoys et al., 1977)
V	Vector perpendicular to canal plane defining axis of excitatory rotation	(Ezure and Graf, 1984)

**Table 2.3.** *Petauroides volans* (AMNH 150055) dimensions of left petrosal semicircular canals calculated by different methods. Radius  $R_r$  calculated by linear regression of at least 60 selected points around semicircular arc of curvature, Radius  $R_{hw} = 0.5 \times (\text{height} + \text{width})/2$ . Circumference values calculated from each method by Circumference =  $2\pi R$ , and Canal path length calculated by addition of numerous discrete distance lengths chosen approximately 0.2 mm apart along semicircular canal arc. Canal abbreviations: **LASC**, left anterior semicircular canal; **LLSC**, left lateral semicircular canal; **LPSC**, left posterior semicircular canal.

Canal	Radius		Circumference		Canal Path Length
	$R_r$	$R_{hw}$	$R_r$	$R_{hw}$	
LASC	1.97	1.61	12.38	10.12	10.99
LLSC	1.48	1.46	9.30	9.19	9.71
LPSC	1.67	1.59	10.49	9.97	10.68

**Table 2.4.** Equal variance, two-tail t-test for mean difference between petrosal angle pair difference, averaged for all three ipsilateral semicircular canals. Thirteen specimens from combined agility levels defined by Spoor et al. (2007) divided into two groups of slow (AG 2-4, specimens indicated by ‘-’ in Appendix 3) and fast (AG 5-6, indicated by ‘+’ in Appendix 3). Mean angle values were calculated by equation  $\bar{x} = ((|ASC<LSC| + |ASC<PSC| + |LSC<PSC|)/3)$ .

	AG 2-4	AG 5-6
Mean	5.751	3.227
Variance	8.750	1.453
n	13	13
Pooled Variance	5.102	
df	24	
t Stat	2.848	
P(T<=t) two-tail	0.009	
t Critical two-tail	2.064	

**Table 2.5.** ANOVA Single factor comparison of ipsilateral canal pair angles.

SUMMARY						
Canal pair angles	n	Sum	Average	Variance		
Ave ASC<LSC	31	2622.4	84.594	55.914		
Ave ASC<PSC	31	2911.595	93.922	50.121		
Ave LSC<PSC	31	2845.245	91.782	22.320		

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1480.602	2	740.301	17.303	4.38E-07	3.098
Within Groups	3850.675	90	42.785			
Total	5331.277	92				

**Table 2.6.** Previous semicircular canal pair angle research results. Where sides were averaged for results, those results were recorded in both the right and left sides. Each entry includes standard deviation and coefficient of variation (in italics) if available. Total specimens measured, 199; 44 different species reported. Abbreviations for notes: **[1]**, sides averaged; **[2]**, MRI study based on one side only; **[3]**, mostly left labyrinths used, but specimens with right sides indeterminate; **a**, least squares fit and eigenvector analysis; **b**, null point technique - mounting and orientation that gives no afferent response from a particular canal; **c**, visual fit; \*, **ASC<LSC** corrected for internal angles.

44

Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes
	LASC ∓LLSC	LASC ∓LPSC	LLSC ∓LPSC	RASC ∓RLSC	RASC ∓RPSC	RLSC ∓RPSC	LASC ∓RASC	LPSC ∓RPSC	LASC ∓RPSC	RASC ∓LPSC	LLSC ∓RLSC		
<i>Atelerix albiventris</i>	<b>82.20</b>	<b>91.70</b>	<b>92.10</b>									1	Ekdale (2009) [3]c
Balaenopteridae	<b>71.60</b>	<b>105.00</b>	<b>75.60</b>									1	Ekdale (2009) [3]c
<i>Bathymenys reevesi</i> †	<b>86.00</b>	<b>99.60</b>	<b>91.30</b>									1	Ekdale (2009) [3]c
<i>Canis familiaris</i>	<b>80.40</b>	<b>101.00</b>	<b>89.10</b>									1	Ekdale (2009) [3]c
<i>Cavia porcellus</i>	<b>57.85</b>	<b>76.71</b>	<b>82.36</b>	<b>57.85</b>	<b>76.71</b>	<b>82.36</b>			<b>32.17</b>	<b>36.16</b>	<b>30.82</b>	10	Curthoys et al (1975)*
	<i>± 6.13</i>	<i>± 5.49</i>	<i>± 4.74</i>	<i>± 6.13</i>	<i>± 5.49</i>	<i>± 4.74</i>			<i>± 4.42</i>	<i>± 4.86</i>	<i>10.05</i>		[1]a
	10.6	7.16	5.76	10.60	7.16	5.76			13.74	13.44	32.61		
<i>Cavia porcellus</i>				<b>91.22</b>	<b>91.20</b>	<b>85.88</b>						8	Cox and

Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes	
	LASC ∓LLSC	LASC ∓LPSC	LLSC ∓LPSC	RASC ∓RLSC	RASC ∓RPSC	RLSC ∓RPSC	LASC ∓RASC	LPSC ∓RPSC	LASC ∓RPSC	RASC ∓LPSC	LLSC ∓RLSC			
				± 6.97	± 5.78	± 5.3								
				7.64	6.34	6.17								
<i>Cavia porcellus</i>	<b>77.20</b>	<b>105.00</b>	<b>85.50</b>									1	Jeffery (2008) [2]a Ekdale (2009) [3]c	
<i>Chinchilla lanigera</i>	<b>76.30</b>	<b>90.90</b>	<b>103.00</b>	<b>78.70</b>	<b>91.00</b>	<b>101.00</b>			<b>15.60</b>	<b>14.80</b>	<b>14.60</b>	3	Hullar and Williams (2006)*a	
	± 2.7	± 1.7	± 5.7	± 5.7	± 2.3	± 6.9			± 1.3	± 0.3	± 12			
	3.54	1.87	5.53	7.24	2.53	6.83			8.33	2.03	82.19			
<i>Chrysochloris</i> sp.	<b>65.60</b>	<b>86.90</b>	<b>96.70</b>									1	Ekdale (2009) [3]c	
<i>Cynocephalus volans</i>	<b>92.20</b>	<b>90.00</b>	<b>91.80</b>									1	Ekdale (2009) [3]c	
<i>Dasypus novemcinctus</i>	<b>62.40</b>	<b>67.70</b>	<b>87.30</b>									1	Ekdale (2009) [3]c	
<i>Didelphis virginiana</i>	<b>109.00</b>	<b>102.00</b>	<b>104.00</b>									1	Ekdale (2009) [3]c	
Elephantoidea	<b>66.30</b>	<b>73.70</b>	<b>96.70</b>									1	Ekdale (2009) [3]c	
<i>Equus caballus</i>	<b>84.70</b>	<b>93.30</b>	<b>90.10</b>									1	[3]c	

Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes
	LASC ±LLSC	LASC ±LPSC	LLSC ±LPSC	RASC ±RLSC	RASC ±RPSC	RLSC ±RPSC	LASC ±RASC	LPSC ±RPSC	LASC ±RPSC	RASC ±LPSC	LLSC ±RLSC		
<i>Eumetopias jubatus</i>	<b>79.70</b>	<b>105.00</b>	<b>90.60</b>									1	Ekdale (2009) [3]c
<i>Felis catus</i>	<b>95.60</b>	<b>87.90</b>	<b>94.10</b>	<b>95.60</b>	<b>87.90</b>	<b>94.10</b>	<b>81.10</b>	<b>103.80</b>	<b>13.90</b>	<b>13.90</b>	<b>14.40</b>	3	[1]a
<i>Felis catus</i>	<b>89.62</b>	<b>90.21</b>	<b>94.23</b>	<b>89.62</b>	<b>90.21</b>	<b>94.23</b>			<b>13.92</b>	<b>14.49</b>	<b>12.49</b>	7	Blanks et al. (1972) [1]a
	± 8.71	± 4.05	± 3.84	± 8.71	± 4.05	± 3.84			± 3.99	± 4.52	± 9.21		
	9.72	4.49	4.08	9.72	4.49	4.08			28.66	31.19	73.74		
<i>Felis catus</i>				<b>78.36</b>	<b>103.34</b>	<b>89.49</b>						8	Cox and Jeffery (2008) [2]a
				±									
				10.05	± 9.7	± 6.94							
				12.83	9.39	7.76							
<i>Felis catus</i>	<b>76.80</b>	<b>91.40</b>	<b>96.70</b>									1	Ekdale (2009) [3]c
<i>Hemicentetes semispinosum</i>	<b>85.40</b>	<b>117.00</b>	<b>92.60</b>	<b>12.83</b>								1	Ekdale (2009) [3]c
<i>Homo sapiens</i>	<b>90.60</b>	<b>94.40</b>	<b>90.40</b>	<b>90.60</b>	<b>94.40</b>	<b>90.40</b>	<b>103.40</b>	<b>83.20</b>	<b>15.30</b>	<b>15.30</b>	<b>11.30</b>	10	Della Santina et al. (2005) [1]c
	± 6.2	± 1.85	± 4.9	± 6.2	± 4	± 4.9	± 9.5	± 9.7	± 7.2	± 7.2	± 6.9		
	6.84	2.02	5.42	6.84	4.24	5.42	9.19	11.66	47.06	47.06	61.06		Blanks et al. (1975)*
<i>Homo sapiens</i>	<b>68.24</b>	<b>86.16</b>	<b>95.75</b>	<b>68.24</b>	<b>86.16</b>	<b>95.75</b>			<b>24.56</b>	<b>23.73</b>	<b>19.82</b>	10	[1]a



Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes
	LASC ∓LLSC	LASC ∓LPSC	LLSC ∓LPSC	RASC ∓RLSC	RASC ∓RPSC	RLSC ∓RPSC	LASC ∓RASC	LPSC ∓RPSC	LASC ∓RPSC	RASC ∓LPSC	LLSC ∓RLSC		
<i>Macroscelides proboscideus</i>	<b>100.00</b>	<b>90.70</b>	<b>73.50</b>									1	[3]c Ekdale (2009)
<i>Manis tricuspis</i>	<b>77.00</b>	<b>84.80</b>	<b>88.60</b>									1	[3]c Ekdale (2009)
<i>Mus C57BL/6J</i>	<b>92.56</b>	<b>99.02</b>	<b>101.17</b>	<b>91.55</b>	<b>99.26</b>	<b>101.26</b>			<b>17.61</b>	<b>17.66</b>	<b>9.14</b>	4	[3]c Calabrese and Hullar (2006)a
	± 1.93	± 1.46	± 0.97	± 1.02	± 1.29	± 0.96			± 2.73	± 1.43	± 0.98		
	2.09	1.47	0.96	1.11	1.30	0.95			15.50	8.10	10.72		
<i>Mus CBA/CaJ</i>	<b>96.82</b>	<b>89.65</b>	<b>102.29</b>	<b>95.47</b>	<b>88.94</b>	<b>102.05</b>			<b>11.00</b>	<b>14.79</b>	<b>10.42</b>	4	Calabrese and Hullar (2006)a
	± 5.73	± 2.51	± 1.86	± 1.94	± 1.98	± 2.14			± 1.24	± 2.11	± 3.8		
	5.92	2.80	1.82	2.03	2.23	2.10			11.27	14.27	36.47		
<i>Mus musculus</i>				<b>76.63</b>	<b>101.54</b>	<b>96.08</b>						9	Cox and Jeffery (2008)
				± 6.02	± 6.32	± 6.06							
				7.86	6.22	6.31							[2]a Ekdale (2009)
<i>Mus musculus</i>	<b>88.80</b>	<b>94.40</b>	<b>95.60</b>									1	[3]c Ekdale (2009)
<i>Nycteris grandis</i>	<b>85.90</b>	<b>112.00</b>	<b>94.90</b>									1	[3]c Matano et al. (1985)
<i>Nycticebus coucang</i>		<b>88.60</b>			<b>88.60</b>		<b>68.20</b>	<b>114.40</b>	<b>23.10</b>	<b>23.10</b>		3	[1]
<i>Orycteropus afer</i>	<b>78.50</b>	<b>91.90</b>	<b>95.70</b>									1	Ekdale

Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes
	LASC ∓LLSC	LASC ∓LPSC	LLSC ∓LPSC	RASC ∓RLSC	RASC ∓RPSC	RLSC ∓RPSC	LASC ∓RASC	LPSC ∓RPSC	LASC ∓RPSC	RASC ∓LPSC	LLSC ∓RLSC		
<i>Oryctolagus cuniculus</i>	<b>79.80</b>	<b>76.60</b>	<b>75.50</b>	<b>79.80</b>	<b>76.60</b>	<b>75.50</b>	<b>90.50</b>	<b>116.50</b>	<b>13.60</b>	<b>13.60</b>	<b>8.60</b>	3	(2009) [3]c Ezure & Graf (1984) [1]a
<i>Oryctolagus cuniculus</i>	<b>79.36</b> ± 9.4	<b>71.36</b> ± 4.4	<b>75.85</b> ± 6.7	<b>79.36</b> ± 9.4	<b>71.36</b> ± 4.4	<b>75.85</b> ± 6.7	<b>85.76</b> ± 5.6	<b>47.54</b> ± 5.3	<b>26.78</b> ± 6.8	<b>26.78</b> ± 6.8	<b>15.32</b> ± 7.2	7	Mazza and Winterson (1984)* [1]a
<i>Oryctolagus cuniculus</i>	11.84	6.17	8.83	11.84	6.17	8.83	6.53	11.15	25.39	25.39	47.00	9	Cox and Jeffery (2008) [2]a
<i>Procavia capensis</i>	<b>87.40</b>	<b>112.00</b>	<b>87.40</b>	<b>81.73</b> ± 11	<b>97.05</b> ± 5.6	<b>97.52</b> ± 9.82	13.46	5.77	10.07			1	Ekdale (2009) [3]c
<i>Pteropus lyelli</i>	<b>84.90</b>	<b>98.30</b>	<b>90.40</b>									1	Ekdale (2009) [3]c Cox and Jeffery (2008) [2]a
<i>Rattus norvegicus</i>				<b>73.35</b> 6.37	<b>97.57</b> 4.80	<b>98.12</b> 10.89	<b>8.68</b>	<b>4.92</b>	<b>11.10</b>			8	

Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes
	LASC ∓LLSC	LASC ∓LPSC	LLSC ∓LPSC	RASC ∓RLSC	RASC ∓RPSC	RLSC ∓RPSC	LASC ∓RASC	LPSC ∓RPSC	LASC ∓RPSC	RASC ∓LPSC	LLSC ∓RLSC		
<i>Rattus norvegicus</i>	<b>97.60</b>	<b>94.20</b>	<b>93.50</b>	<b>97.60</b>	<b>94.20</b>	<b>93.50</b>			<b>9.90</b>	<b>9.90</b>	<b>8.00</b>	14	Blanks and Torigoe (1989) [1]b
<i>Rhinolophus ferrumequinum</i>	<b>79.90</b>	<b>104.00</b>	<b>87.90</b>									1	Ekdale (2009) [3]c
<i>Saimiri sciureus</i>	<b>90.43</b> ± 6.94 7.67	<b>87.02</b> ± 4.22 4.85	<b>89.95</b> ± 5.08 5.65	<b>90.43</b> ± 6.94 7.67	<b>87.02</b> ± 4.22 4.85	<b>89.95</b> ± 5.08 5.65			<b>12.53</b> ± 5.55 44.29	<b>14.80</b> ± 5.37 36.28	<b>15.45</b> ± 5.98 38.71	10	Blanks et al. (1985) [1]a
<i>Sciurus carolinensis</i>				<b>78.97</b> ± 6.61 8.37	<b>89.52</b> ± 4.4 4.92	<b>104.41</b> ± 9.64 9.23						5	Cox and Jeffery (2008) [2]a
<i>Sorex monticolus</i>	<b>75.30</b>	<b>89.60</b>	<b>89.30</b>									1	Ekdale (2009) [3]c
<i>Sus scrofa</i>	<b>82.80</b>	<b>96.00</b>	<b>87.90</b>									1	Ekdale (2009) [3]c
<i>Sylvilagus floridanus</i>	<b>92.70</b>	<b>97.50</b>	<b>77.90</b>									1	Ekdale (2009) [3]c
<i>Tadarida brasiliensis</i>	<b>74.70</b>	<b>98.40</b>	<b>98.40</b>									1	Ekdale (2009) [3]c

Taxon	Ipsilateral Canals						Contralateral Canals		Synergistic Canals			n	Sources and Notes
	LASC ∓LLSC	LASC ∓LPSC	LLSC ∓LPSC	RASC ∓RLSC	RASC ∓RPSC	RLSC ∓RPSC	LASC ∓RASC	LPSC ∓RPSC	LASC ∓RPSC	RASC ∓LPSC	LLSC ∓RLSC		
<i>Tarsius bancanus</i>		<b>91.80</b>			<b>91.80</b>		<b>73.80</b>	<b>102.80</b>	<b>14.50</b>	<b>14.50</b>		3	Matano et al. (1985)
<i>Trichechus manatus</i>	<b>52.20</b>	<b>84.90</b>	<b>86.30</b>									1	[1] Ekdale (2009)
<i>Tupaia glis</i>	<b>82.30</b>	<b>106.00</b>	<b>102.00</b>									1	[3]c Ekdale (2009)
<i>Tursiops truncatus</i>	<b>52.20</b>	<b>84.90</b>	<b>77.50</b>									1	[3]c Ekdale (2009)
<i>Ukhaatherium gobiensis</i> <sup>†</sup>	<b>88.80</b>	<b>105.00</b>	<b>88.40</b>									1	[3]c Ekdale (2009)
<i>Zalambdalestes lechei</i> <sup>†</sup>	<b>81.00</b>	<b>93.60</b>	<b>85.60</b>									4	[3]c Ekdale (2009)
Zhelestid <sup>†</sup>	<b>88.80</b>	<b>96.80</b>	<b>93.10</b>									7	[3]c
<b>Mean</b>	<b>82.46</b>	<b>93.21</b>	<b>90.37</b>	<b>84.27</b>	<b>90.42</b>	<b>92.35</b>	<b>83.79</b>	<b>94.71</b>	<b>17.01</b>	<b>17.65</b>	<b>13.28</b>		
<i>Standard Dev.</i>	$\pm 11.87$	$\pm 10.14$	$\pm 7.19$	$\pm 10.41$	$\pm 7.82$	$\pm 7.85$	$\pm 12.52$	$\pm 25.95$	$\pm 6.59$	$\pm 6.94$	$\pm 6.85$		
<i>Coeff. Variation</i>	14.39	10.88	7.96	12.35	8.64	8.50	14.94	27.40	38.77	39.35	51.63		
<i>95% Confidence Interval</i>	80.78 84.13	91.80 94.61	89.35 91.39	82.00 86.54	88.79 92.05	90.64 94.07	78.68 88.90	84.11 105.30	15.31 18.71	15.85 19.44	11.38 15.88		

**Table 2.7.** ANOVA single factor analysis of left ipsilateral canal pair angles taken from previous studies cited in Table 2.5.

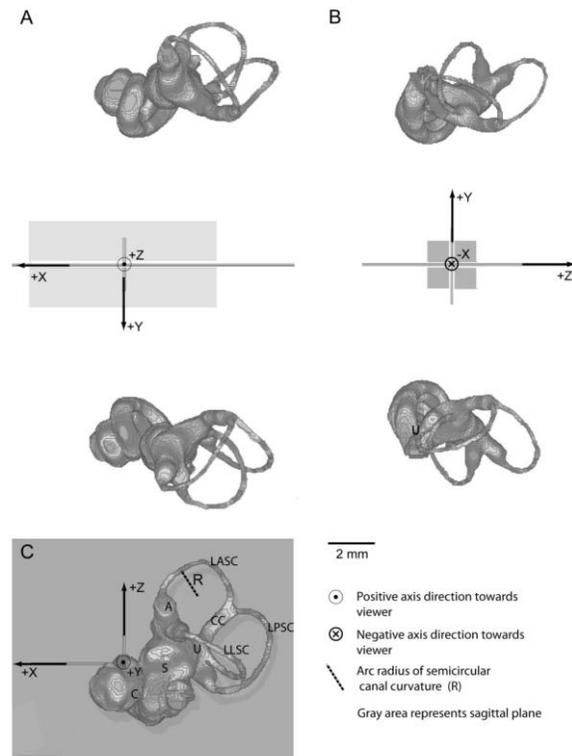
SUMMARY						
Canal pair angles	n	Sum	Average	Variance		
LASC <LLSC	50	4122.81	82.456	140.785		
LASC <LPSC	50	4666.31	93.326	106.514		
LLSC <LPSC	50	4518.61	90.372	51.762		

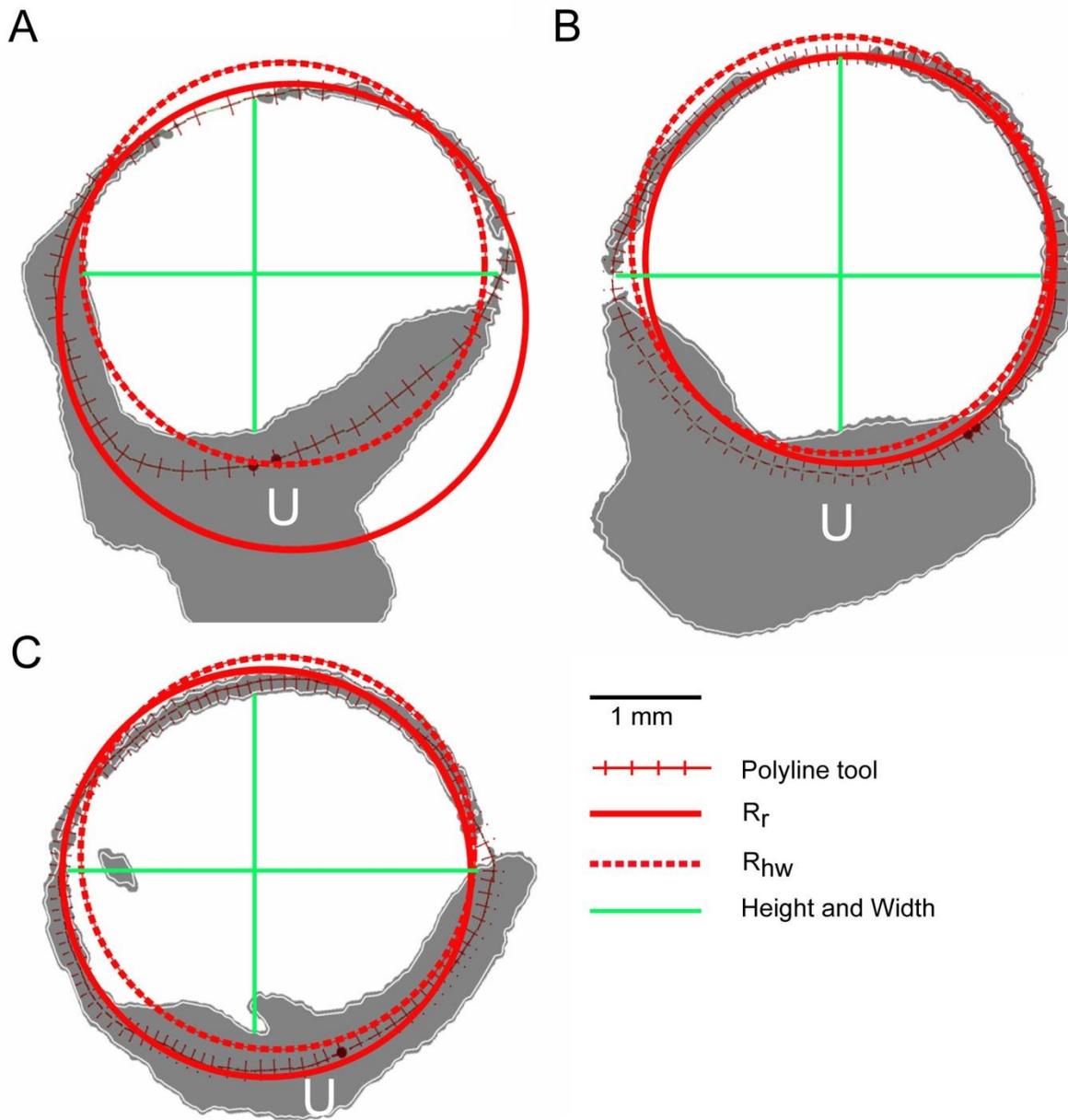
  

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3159.101	2	1579.551	15.845	5.87E-07	3.058
Within Groups	14654.050	147	99.687			
Total	17813.150	149				

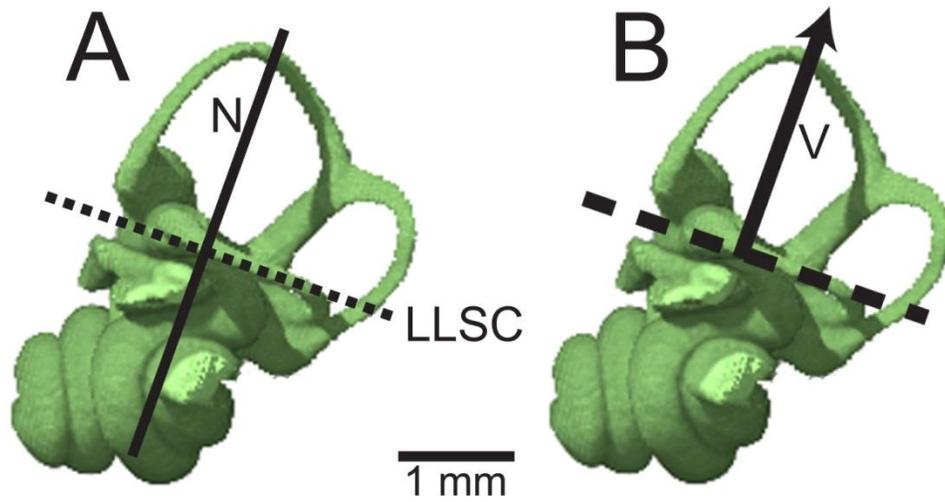
## FIGURES

**Figure 2.1.** Bony inner ear endocast of *Petauroides volans* (AMNH 150055) showing embedded head-centered reference planes. A, dorsal view showing X, Y, and Z axes. Center point occurs at intersection of all three planes. Axial plane = YZ reference plane passing through the interaural line; frontal plane = XY reference plane defined by Reid's Plane (perpendicular to viewer); sagittal plane = XZ reference plane passing through central features. B, Posterior view with axial reference plane perpendicular to viewer. C, Left lateral view with sagittal plane perpendicular to viewer. Abbreviations: A, ampulla; C, cochlea; CC, common crus; LASC, left anterior semicircular canal; LLSC, left lateral semicircular canal; LPSC, left posterior semicircular canal; R, arc radius of curvature of the left anterior semicircular canal; S, saccule; U, utricle

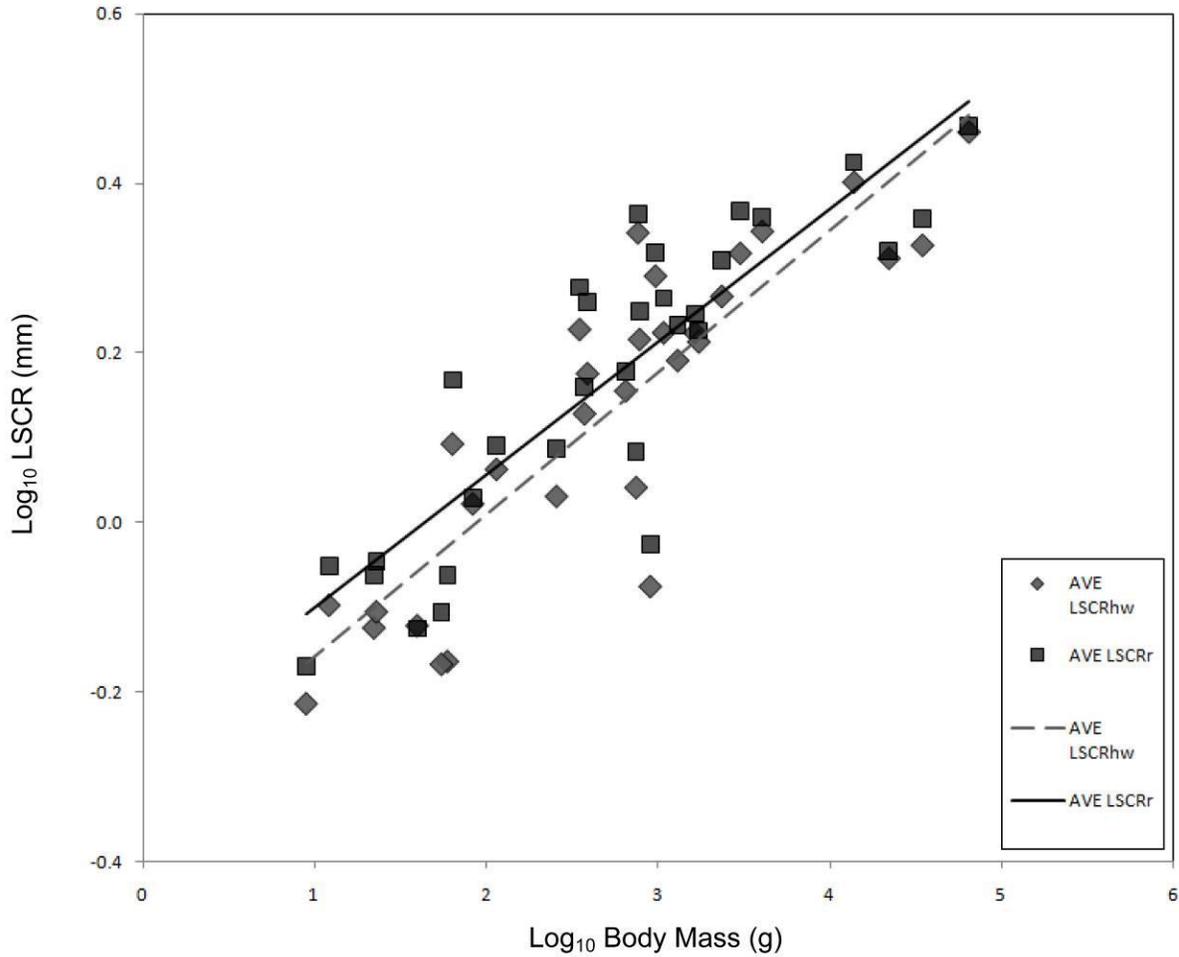




**Figure 2.2.** *Petauroides volans* (AMNH 150055) left petrosal bony labyrinth endocast of ipsilateral semicircular canals rotated into viewing plane along regression plane of best fit. Circumferences of circles described by radii of curvature calculated by  $R_r$ , best fit regression and  $R_{hw}$  method described in text. **A**, LASC; **B**, LLSC; **C**, LPSC. **U**, utricle.

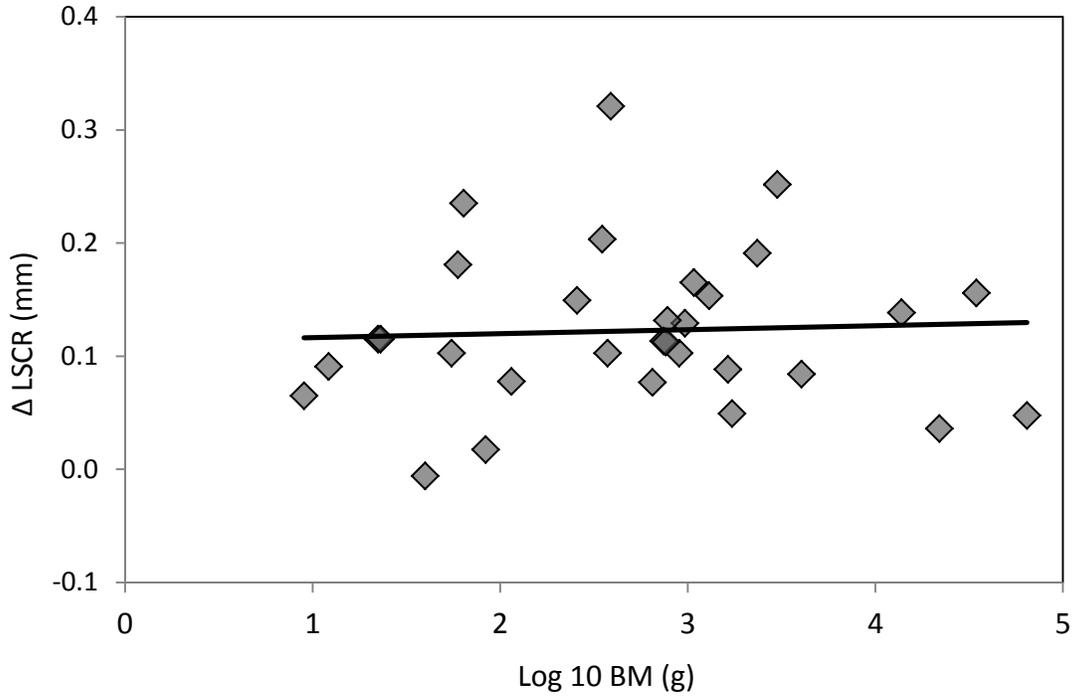


**Figure 2.3.** Difference between canal plane normal and canal plane rotational vector. **A**, Left lateral semicircular canal plane indicated by dashed line, with canal normal line perpendicular to the plane. **B**, Left lateral semicircular canal plane indicated by dashed line, with canal plane vector. Rotation around this vector axis according to the right-hand rule will result in excitatory hair cell firing increases. **LLSC**, left lateral semicircular canal plane; **N**, canal plane normal; **V**, rotational vector defining left lateral semicircular canal plane.

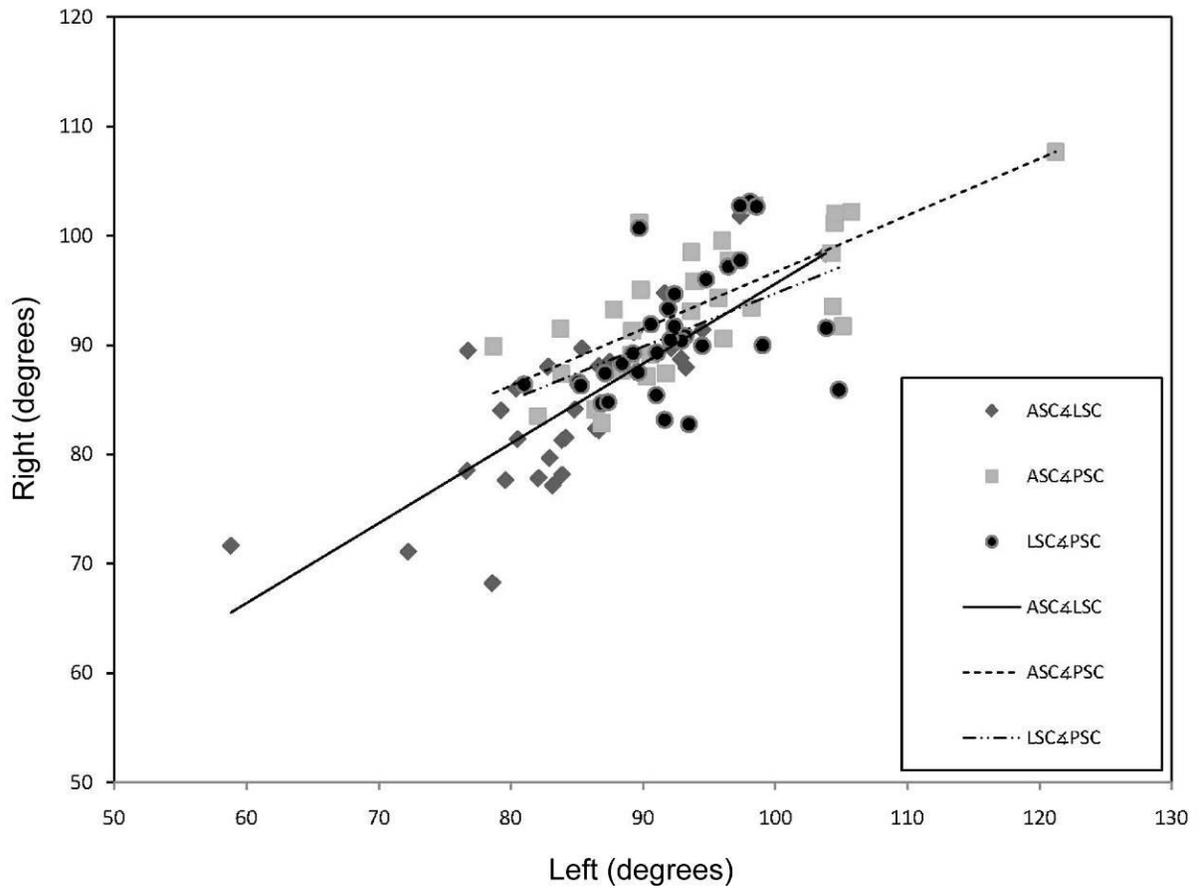


**Figure 2.4.** Arc radius of curvature determination methods compared as functions of body mass, plotted as double logarithmic plots of mean canal radius against body mass (BM) for 31 specimens, data from Appendix 2. Radius  $R_r$  calculated by linear regression,  $R_{hw} = 0.5 \times (\text{height} + \text{width})/2$ . Values for each method are averaged from left ipsilateral canals of study specimens, and values are listed in Appendix 1. Linear regressions are shown for each method,  $LSCR_r r^2 = 0.7554$ ,  $LSCR_{hw} r^2 = 0.7772$ .

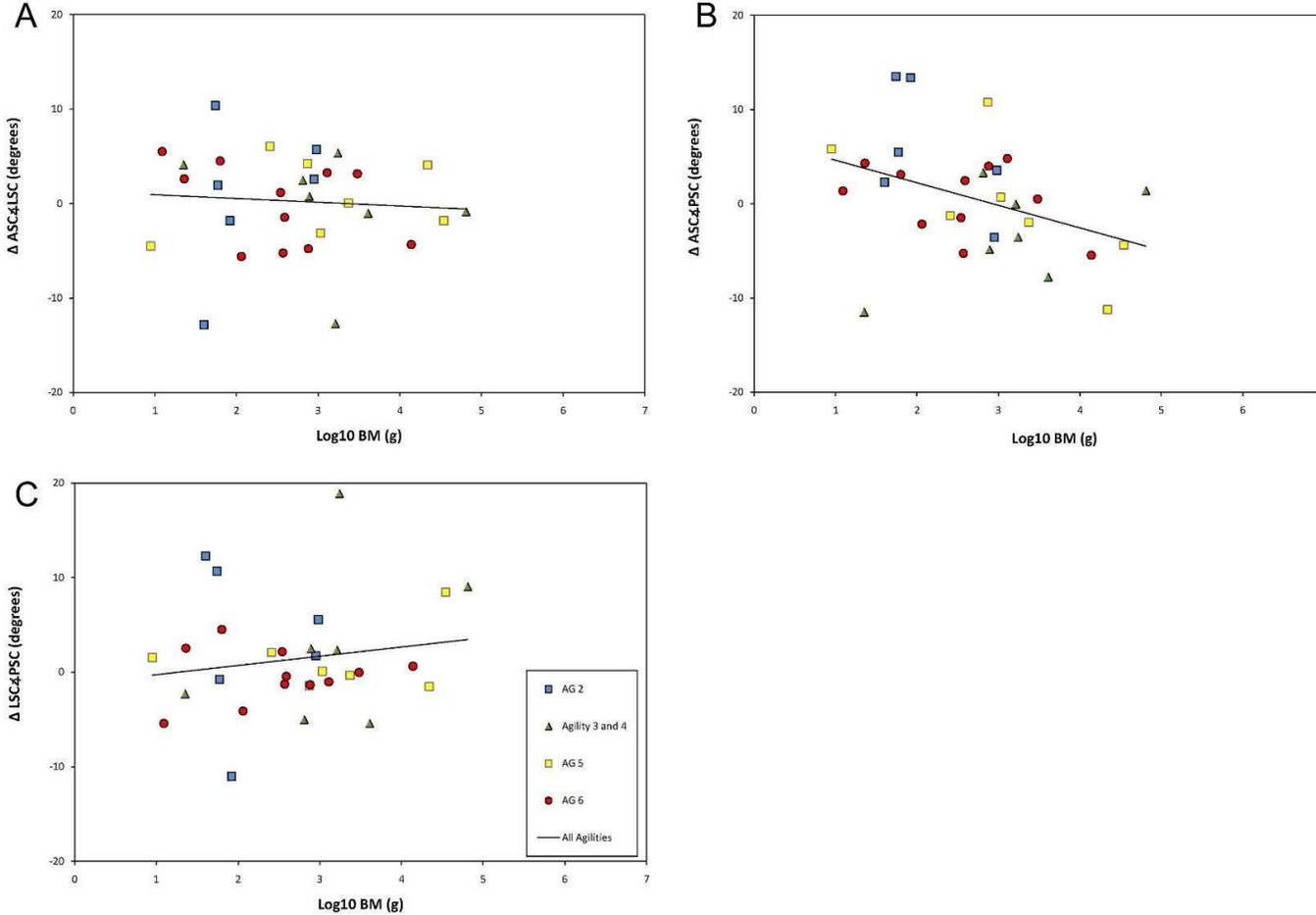
Average Semicircular Canal Difference between  $R_r$  and  $R_{hw}$  for Study Specimens



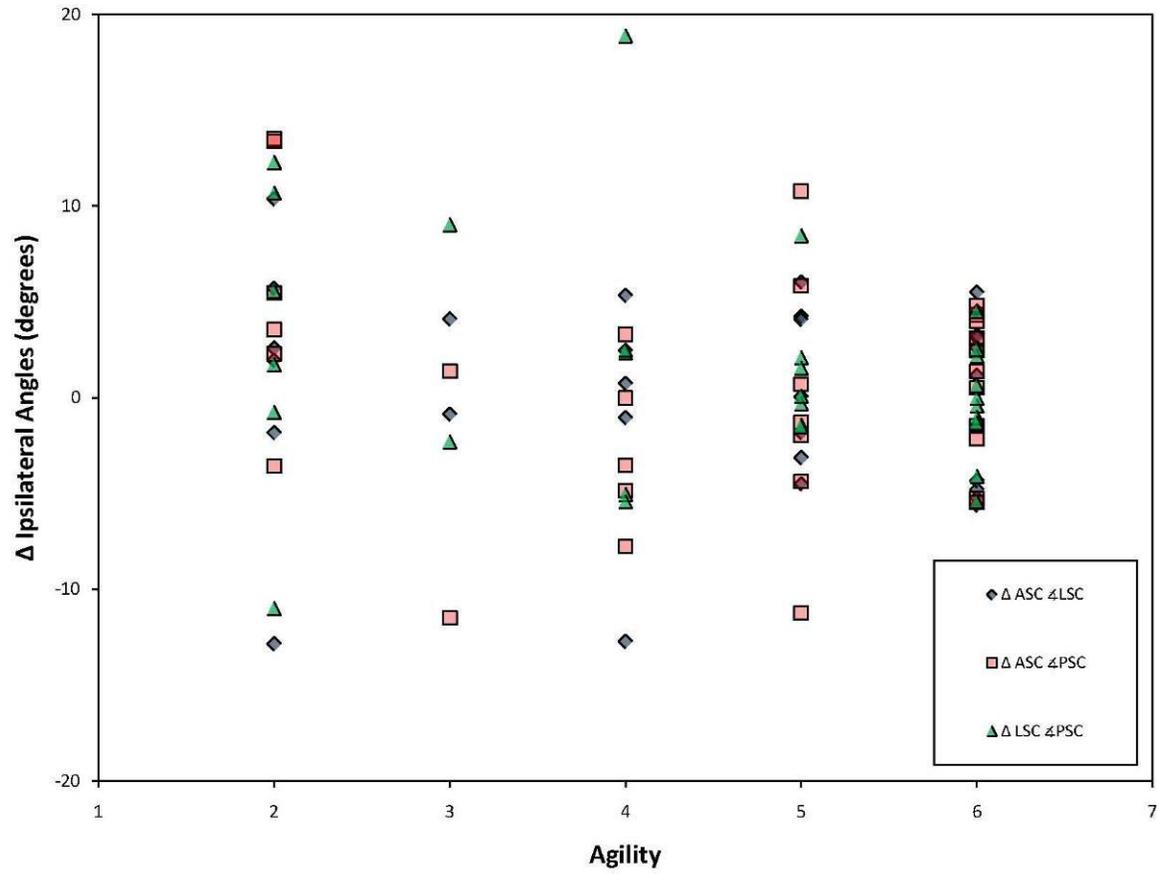
**Figure 2.5.** Distribution of  $\Delta$  LSCR compared as functions of Log10 body mass (BM), with a linear regression shown.



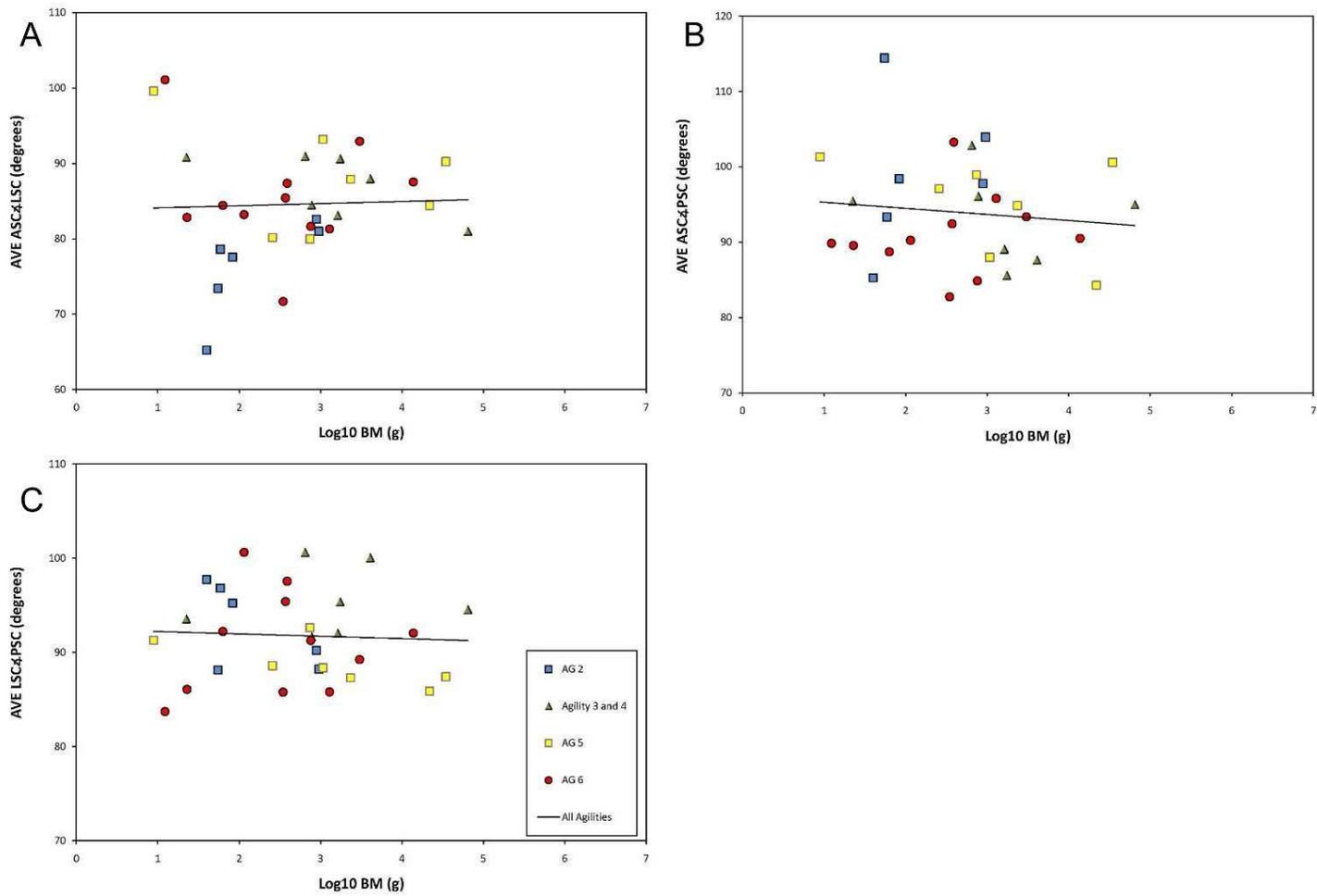
**Figure 2.6.** Comparison of ipsilateral canal pair angles from left and right petrosals. Lines represent linear regressions; ASC<math><LSC r^2 = 0.4647</math>, ASC<math><PSC r^2 = 0.5376</math>, LSC<math><PSC r^2 = 0.1975</math>



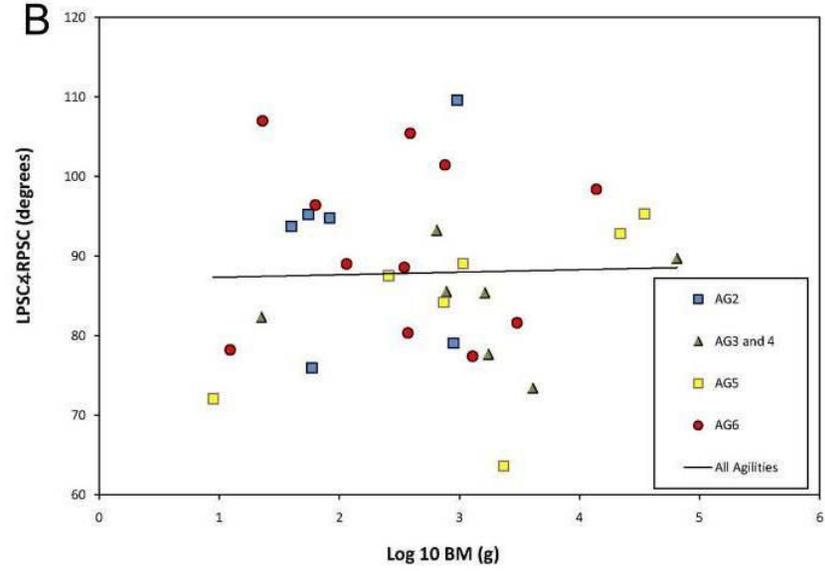
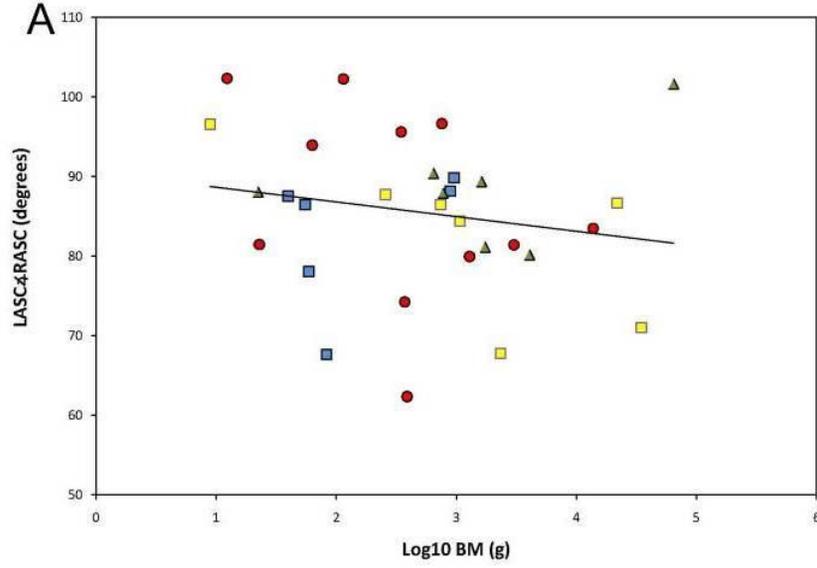
**Figure 2.7.** Difference in ipsilateral canal pair angles from left and right sides of skull. Differences are plotted in degrees for all agility levels (defined in Spoor et al., 2007 compared as functions of Log<sub>10</sub> body mass (BM)). **A**, Angle between ASC and LSC,  $r^2 = 0.0058$ . **B**, Angle between ASC and PSC,  $r^2 = 0.1556$ . **C**, Angle between LSC and PSC,  $r^2 = 0.0271$ . Differences are calculated by subtracting the angle between the canals on the right side from the angle between the canals on the left side. Linear regression taken from every specimen of all agility levels.



**Figure 2.8.** Distribution of differences between ipsilateral canal pair angles between left and right sides of all study specimens.  $\Delta$  Ipsiateral Angles compared as functions of agility scores as defined by Spoor et al. (2007).

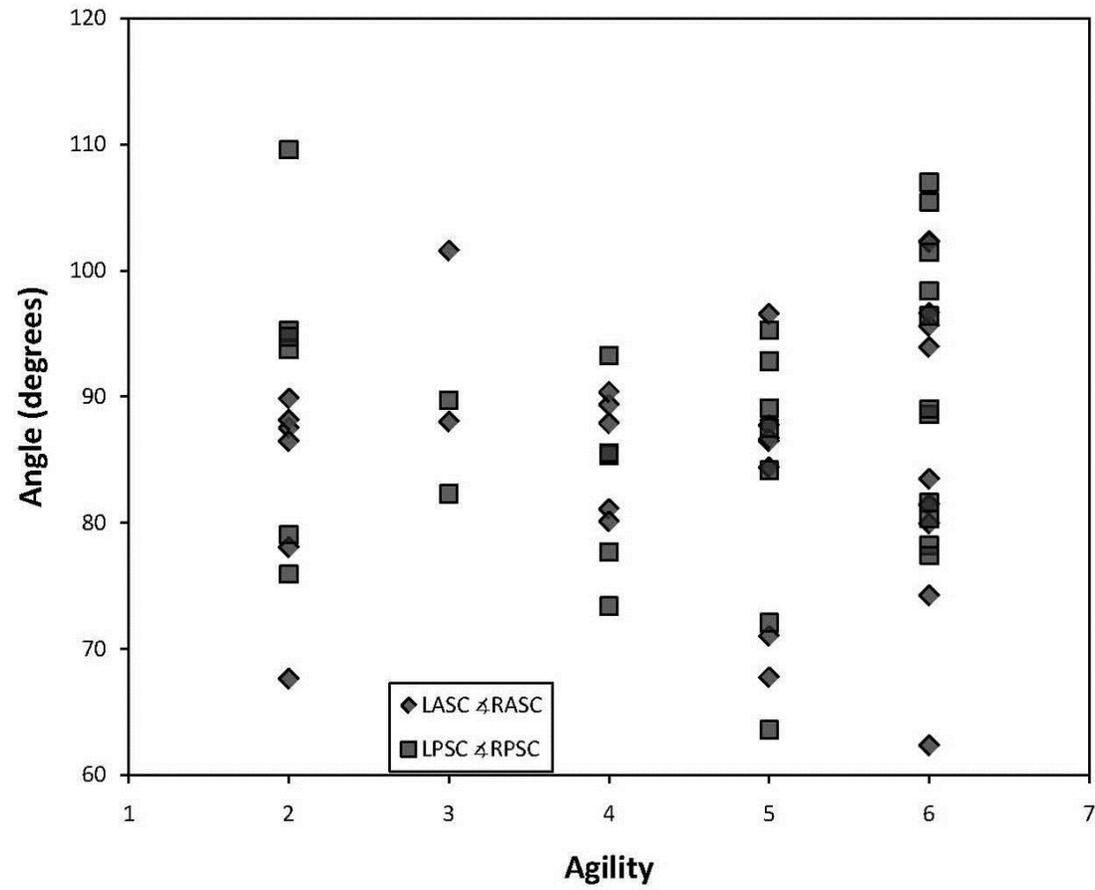


**Figure 2.9.** Side-averaged ipsilateral canal pair angles compared as functions of Log10 body mass (BM) for differing agility values. **A**, Side-averaged ASC<LSC,  $r^2 = 0.0014$ ; **B**, Side-averaged ASC<PSC,  $r^2 = 0.013$ ; **C**, Side-averaged LSC<PSC,  $r^2 = 0.0028$ .

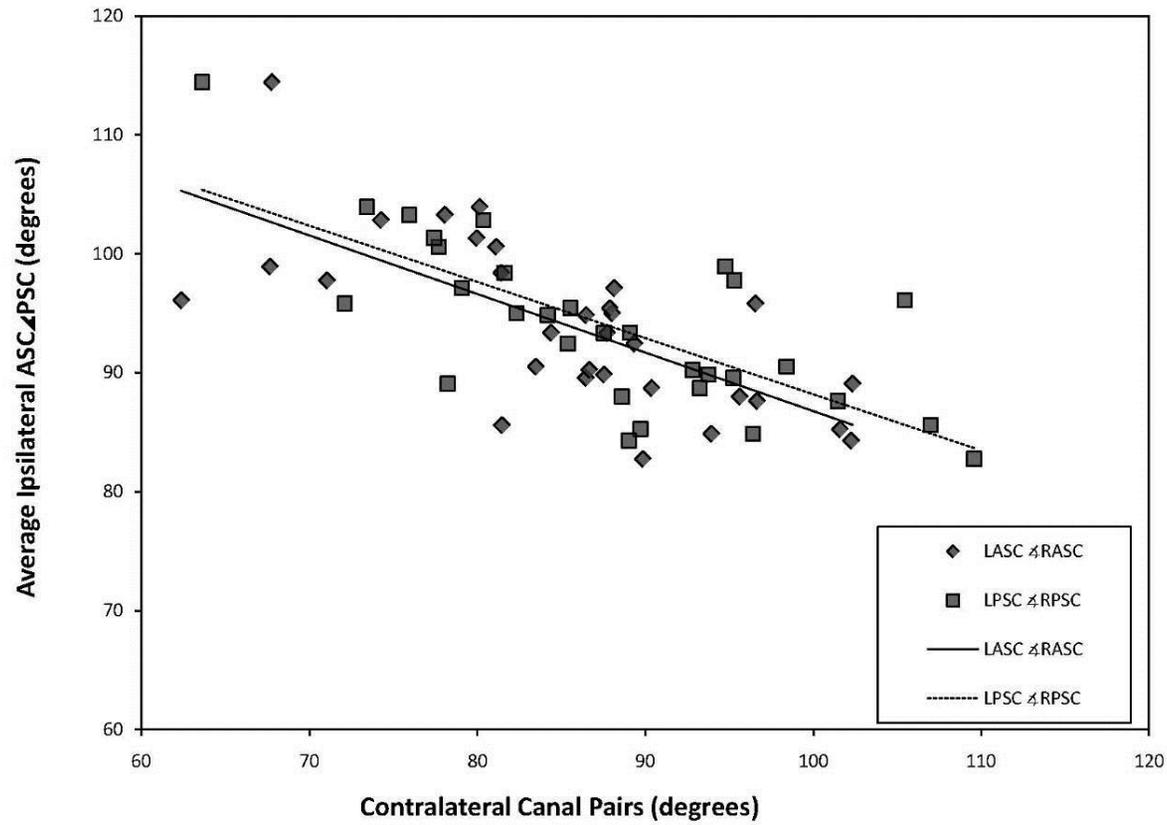


62

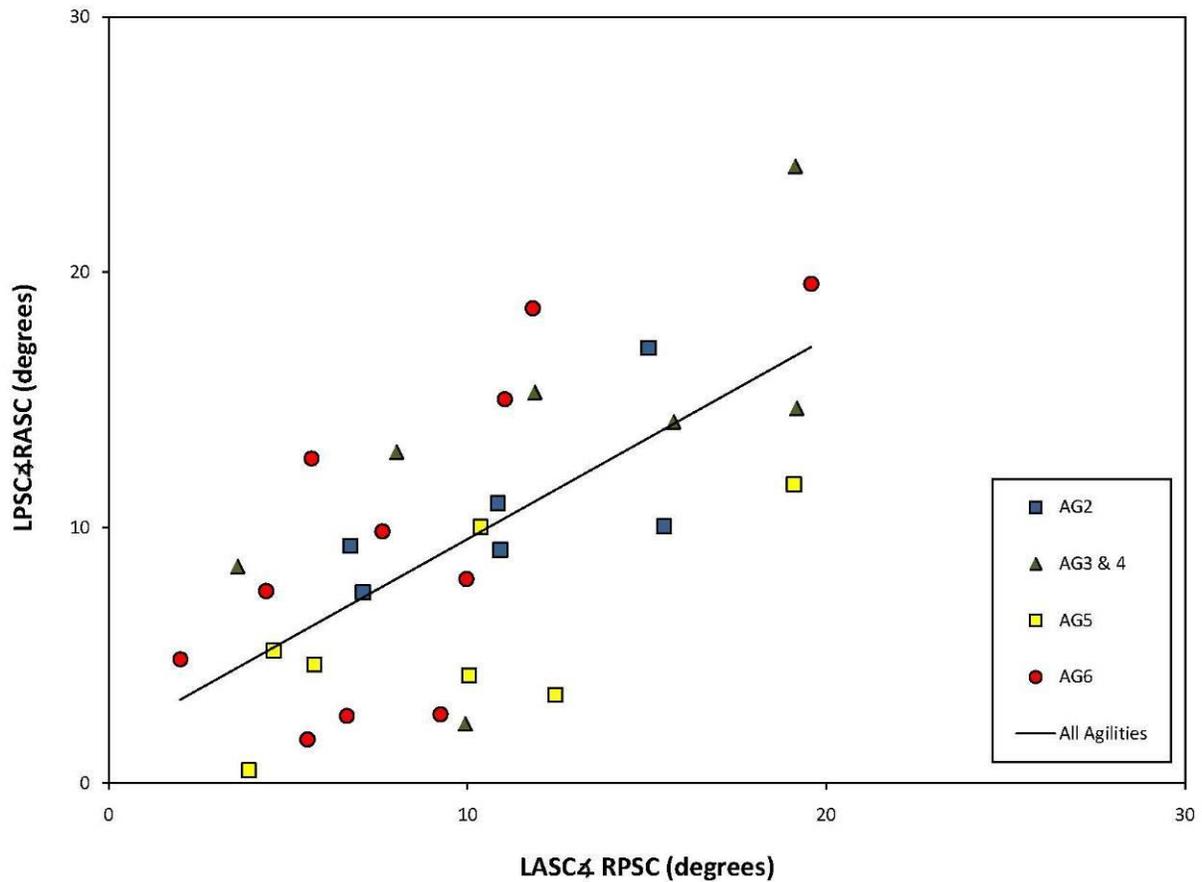
**Figure 2.10.** Contralateral canal pair angles compared as a function of Log10 body mass for all agility levels from Spoor et al. (2007). **A**, Contralateral anterior semicircular canal pair angle,  $r^2 = 0.0337$ ; **B**, Contralateral posterior semicircular canal pair angle,  $r^2 = 0.0009$ .



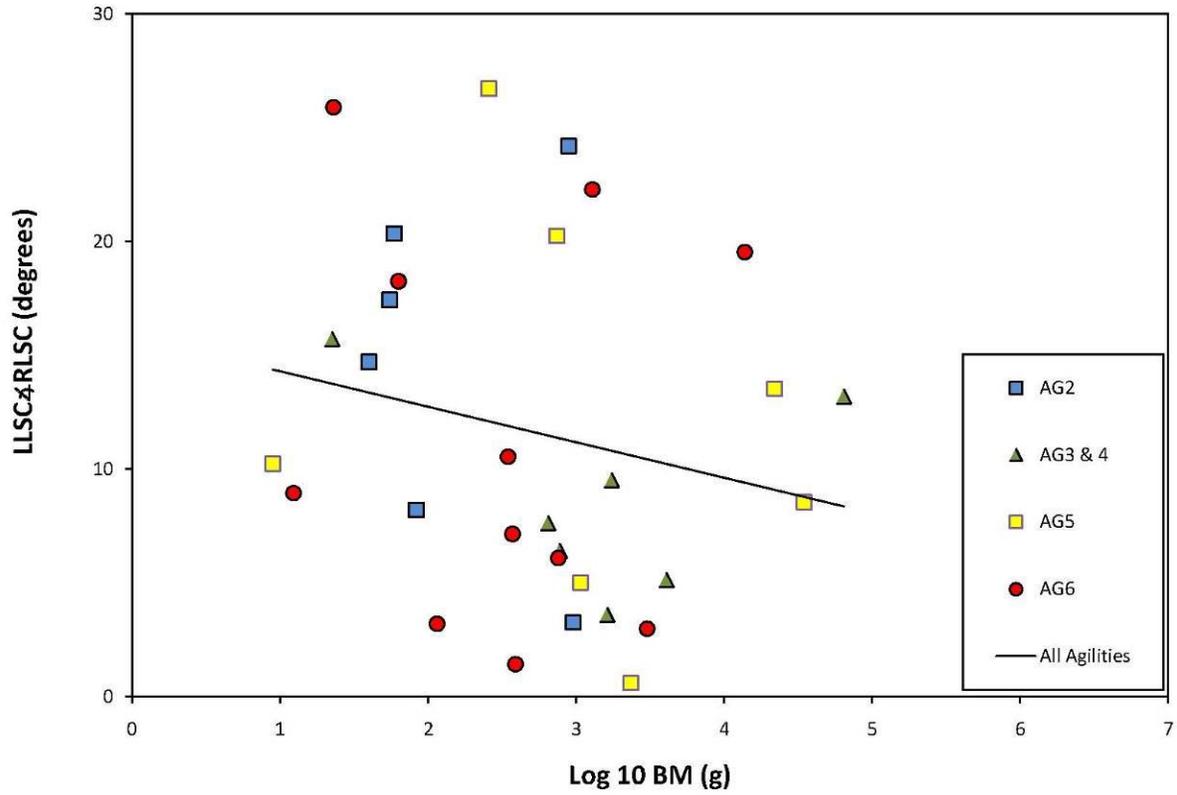
**Figure 2.11.** Anterior and posterior contralateral canal pair angles compared as a function of agility as defined by Spoor et al. (2007).



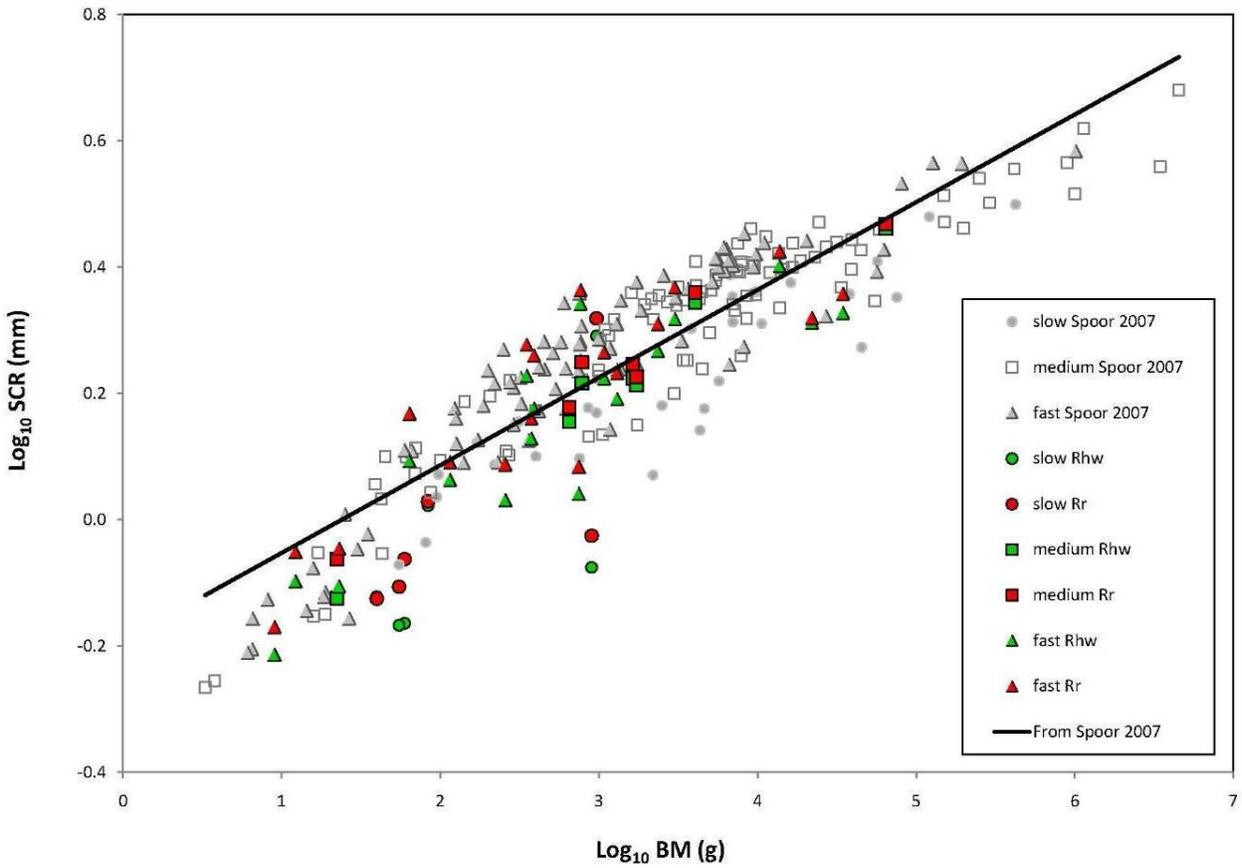
**Figure 2.12.** Side-averaged canal pair angle between the anterior and posterior semicircular pair compared as a function of the anterior and posterior contralateral canal pair angles. LASC<RASC  $r^2 = 0.4843$ , LPSC<RPSC  $r^2 = 0.5181$



**Figure 2.13.** Synergistic angle between left posterior semicircular canal and right anterior semicircular canal compared to the synergistic contralateral canal pair of the left anterior semicircular canal and the right posterior semicircular canal. The comparison is made for all agility levels as defined by Spoor et al. (2007). The linear regression is based on all agility levels combined,  $r^2 = 0.4498$ .



**Figure 2.14.** The synergistic lateral canal pair angles compared as a function of Log10 body mass for all agility groups as defined by Spoor et al. (2007). Linear regression is based on all agility groups combined,  $r^2 = 0.0414$ .



**Figure 2.15.** Double logarithmic plots of mean side-averaged radius of curvature as a function of Log<sub>10</sub> body mass for slow, medium, and fast agilities provided by Spoor et al. (2007). Sources for R values from supplemental information of Spoor et al. (2007) for 210 mammals and R<sub>r</sub> and R<sub>hw</sub> values calculated in the present study. Linear regression based on combined R values from all agilities reported by Spoor et al. (2007),  $r^2 = 0.8355$ .

## CHAPTER 3: ROTATIONAL VESTIBULAR SENSITIVITY DIRECTIONS IN THERIAN MAMMALS

### ABSTRACT

Paleontologists often wonder about the lifestyles and sensory capabilities of the extinct taxa they find as fossils. Although much of that information is lost during skeletonization, the petrosal preserves the empty spaces of the vestibular system, which detected passive and active head rotations in the semicircular canals during life. In this paper I describe a model and program to input measurements from the size and orientation of bony semicircular canals and determine the sensitivities of all six canals and a total sensitivity of the head to rotation in any direction. For demonstration, I use a high resolution x-ray computed tomographic scan of the extant marsupial *Dromiciops gliroides* and present the results in table and map form. *Dromiciops gliroides* has a head-centered sensitivity maximum of  $0.27 \text{ (spikes} \cdot \text{sec}^{-1})/(\text{degrees} \cdot \text{sec}^{-1})$  in an approximate pitch direction a maximum greater than that of any single canal or combination of canals in the vestibular system. This result indicates that the scansorial *Dromiciops gliroides* registers head-up, head-down angular changes with the greatest vestibular discrimination for sending to brain processing centers. My results support earlier research indicating that the orientations of the semicircular canals provide necessary information for quantitative vestibular analyses. Similarities and differences in head rotation sensitivity maxima should be considered in comparative evaluation of extinct therian mammal locomotor lifestyle.

## INTRODUCTION

Modern and fossil vertebrate bones hold great value to paleontologists and morphologists, because they record information on the structure and appearance of animals, as well as phylogenetic relationships, uniquely derived features, and the function of many structures found on and within them. The skull is a particularly valuable skeletal element for it preserves the greatest number of features evidencing the taxonomic relationships, posture, diet, and sensory input of the animal.

Researchers can test a living animal for sensory responses to visual stimulus, taste, odor preference (and aversions), hearing responses, electrical acuity, touch, and balance or movement sensitivity. Unfortunately, upon death and subsequent decay of soft anatomy much information about these senses is lost or hidden deeply within the skull. Until recently, researchers needed to partially or completely destroy a skull or find previously broken skulls to gain insights on some sensory attributes. Natural or intentional breakage of the petrosal bone exposes structures that reveal information on hearing, as well as balance and angular movement detection.

I focus on the ability of extant and extinct therian (= placental and marsupial) mammals to detect head rotation, and ask whether specialization in that ability is determined predominantly by phylogenetic history or locomotor lifestyles. Head rotation is detected by six membranous tubes (the membranous semicircular ducts) that are part of the inner ear, so the answer to my question requires finding detailed information on the shape and orientation of those ducts. Upon loss of soft tissues, the duct orientations and sizes are preserved as spaces within the skull. Therefore, study of the bony semicircular canals within the skull can provide the information on the soft tissues necessary to answer my question. Figure 3.1 shows the question in image form, and the terms are described in

more detail below. In this paper I describe a new method to determine head sensitivity to rotation and discuss the importance of results obtained from that method.

### **Exemplar Animal: *Dromiciops gliroides***

I used *Dromiciops gliroides* (FMNH 127463) to illustrate the method and the results than can be derived from it. *Dromiciops gliroides*, known colloquially as the monito del monte is the sole known extant member of the order Microbiotheria, which has been of contentious relation to other extant marsupial groups. *Dromiciops gliroides*, with an average weight of 29 grams (Silva and Downing, 1995), looks much like the placental mouse, and it is found only in Chile. It is scansorial, mainly insectivorous, and has a prehensile tail (Nowak, 1999).

### **Semicircular Canal System**

The petrosal bones (or petrous portion of the temporal bone when they are fused to the cranium) are located ventrolaterally to the squamosals, and are the densest bones in the skull. In life a petrosal encloses the inner ear – a membranous labyrinth with connected organs to detect hearing (cochlea), linear gravity accelerations (sacculus), lateral accelerations in translation and tilt (utricle), and angular motion (the three semicircular ducts). The location of the left inner ear in the skull is shown in Figure 3.2, its structures labeled in Figure 3.3, and all abbreviations used in this paper are listed in Table 3.1.

Semicircular ducts are named according to the approximate orientation of their planes in the head. The anterior (or superior) duct is approximately vertical and closest to the rostrum, the posterior (or inferior) is again oriented in an approximately vertical plane

that is roughly perpendicular to the anterior canal and closest to the occiput, and the lateral (or horizontal) duct is roughly perpendicular to the other two and more closely earth-parallel. With three membranous semicircular ducts in each petrosal, there are six total ducts in the head. Upon death and loss of soft tissue the exact size and orientation of the ducts is lost to researchers. However, the bony semicircular canals (SC) that once enclosed these ducts are well-preserved whenever the dense petrosal bone survives, and serve as reliable proxies for the shape and orientation of the original membranous ducts (Hashimoto et al., 2005, Ifediba et al., 2007). Bony canals are identified the same way anterior (ASC), posterior (PSC), and lateral (LSC) semicircular canals.

After Antonio Scarpa published the first dissection of the membranous inner ear in 1789 (Canalis et al., 2001), other curious anatomists began to make casts of both the membranous and the bony inner ear (Gray, 1907:17). The correspondence of inner ear shape and locomotor lifestyles (for example fossorial, scansorial, stotting) among marsupials and placentals was noted by Gray (1907, 1908) based upon casts of membranous inner labyrinths. Those studies and subsequent research required destructive methods. Recent researchers used non-destructive high resolution X-ray computed tomography (HRXCT) to view skull interiors (Spoor et al., 1994, Rowe et al., 1997); that was the instrument used for this study.

In life, membranous semicircular ducts were toroidal tubes filled with a fluid, the endolymph, that connected to the sac-shaped utricle, ran through the SCs (themselves filled by another fluid called perilymph), and ended in enlarged expansions called ampullae. An ampulla contains a ‘saddle-shaped’ crista (Hullar and Page, 2010) made of a gelatinous material topped by an epithelium of afferent nerve cells. Those cells are referred to as hair cells because they have hair-like extensions of membrane that send constant firing signals along their axons, exiting by their axons through the bony

labyrinth. The crista is anchored to the bony labyrinth where those axons exit, and to the opposite side by a nearly transparent, diaphragm-like cupula. As the head moves, endolymph within a semicircular duct responds by inertial drag, and the change in fluid pressure within the ampulla distorts the cupula (Selva et al., 2009). When the cupula is moved, it deflects the hair cells in one direction or the other. The hair cells respond with a change in their firing rate from a resting rate. If deflected in one direction, they respond with an increase in firing rate (an excitatory response), and deflection in the other direction causes a decrease in firing rate (an inhibitory response). The firing rates are transmitted along the axons of the nerves from each ampulla.

The lengthy hair cell nerve fibers merge into the vestibular nerve which exits the petrosal via the medial interior acoustic meatus to a ganglion (Scarpa's ganglion). Connecting neurons pass through the superior nuclei to abducens nucleus motor nerves for eye muscle tracking responses. Other neurons pass to the spinal cord muscle motor nerves for posture stabilization. Other areas also receive afferent vestibular signals, predominantly through connections with the vestibular nuclei of the medulla and pons. The vestibular nuclei communicate with other brain areas for further afferent and motor responses. The nerve signals sent to the vestibular nuclei encode a head-centered reference system, which is translated into various reference systems, including an earth-centered reference system.

### **Semicircular Duct Dimensions and Function**

I determined in what direction any animal experienced the greatest sensitivity to head rotation. Most researchers previously looked only at the size of the SC, but SC size gave more specific information when combined with the orientation of each SC in the head in recent models (Rabbitt, 1999, Ifediba et al., 2007). Therefore I determined both

the size and orientation for each SC, and then asked a colleague (Michael Cameron Rodgers) to develop an algorithm for determining rotational sensitivity of every canal individually and in interaction with every other canal based on those recent models. The final algorithm was written as a program to orient vestibular signal results with respect to skull reference planes.

By convention, the fit of selected points (of varying number depending on the researcher) along the canal arc to a circle is taken as an estimate of canal length, and the radius of that circle (R in mm) is used in comparisons of SC size (Curthoys et al., 1977, Spoor and Zonneveld, 1995). An example of R is shown in Figure 3.2C. Radius size appears associated with locomotor lifestyle; animals judged as more ‘agile’ (Spoor et al., 2007), ‘acrobatic’ (Sipla and Spoor, 2008), or having a greater “frequency and erraticism of head movements” (Jeffery and Cox, 2010:498), have a larger R relative to their overall body size (taken as mass) in at least one SC (reviewed by Spoor, 2003).

Canal length and head rotation rate influence the inertial drag of endolymph impinging on the cupula, and subsequently the frequency of nerve firings. In a living animal, afferent hair cell neurons within ampullae have a continuous firing rate when not stimulated by movement (resting rate). This resting rate differs for each animal and canal size. Experimental mice had a rate of 62 to 69 spikes  $\cdot$  sec<sup>-1</sup> (Yang and Hullar, 2007) where spikes are neuron discharges. Squirrel monkeys had an average resting rate of 91 spikes  $\cdot$  sec<sup>-1</sup> (Goldberg and Fernández, 1971), with that resting rate being unpredictable across taxa (Yang and Hullar, 2007). An animal with a larger SC (when corrected by negative allometry in body size) is predicted to be more agile. This hypothesis was supported by experimental results reported by Yang and Hullar (2007), who recorded an increase in hair cell neuron firing rates with different rotation rates for two differently-sized canals in experimental mice. In that case body size did not need to be considered, as

the canals came from the same animals. They also plotted a linear regression of canal sizes versus hair cell firing rates for six species from other publications. Afferent firing rate ( $\text{spikes} \cdot \text{sec}^{-1}$ ) over resting rates for regular neurons per rate of head rotations ( $\text{degrees} \cdot \text{sec}^{-1}$ ) were reported by Yang and Hullar (2007) as canal sensitivity ( $G$  in  $\text{spikes} \cdot \text{sec}^{-1}/\text{degrees} \cdot \text{sec}^{-1}$ ), where

$$G = 0.23R - 0.09$$

As an example of this calculation, *Dromiciops gliroides* has a left anterior semicircular canal (LASC) of  $R = 1.05$  mm and  $G = 0.15$  ( $\text{spikes} \cdot \text{sec}^{-1}/\text{degrees} \cdot \text{sec}^{-1}$ ). The LASC firing rate occurs when the head is rotated around an axis perpendicular to the plane of the LASC in the direction of an excitatory hair cell response. This direction is designated the canal plane normal (N); an example is shown in Figure 3.3D.

What happens if the head is rotated around an axis other than the perpendicular to the semicircular canal plane? The sensitivity and number of firings would decrease from that maximum with a cosine-dependent shape as the angle between the rotation axis and the canal normal increases, to reach the resting rate when rotated in a direction parallel to the canal plane (Rabbitt, 1999, Rabbitt et al., 2004, Ifediba et al., 2007, Yang and Hullar, 2007). This trend continues to an even lower firing rate indicating an inhibitory signal as the rotation axis approaches the direction opposing the normal. Depending on the canal resting rate and rotation rate, cell firing rates could reach  $0 \text{ spikes} \cdot \text{sec}^{-1}$ . For example, cell firings in *Saimiri sciureus* vary between an excitatory maximum of  $400 \text{ spikes} \cdot \text{sec}^{-1}$  to a resting rate of  $90 \text{ spikes} \cdot \text{sec}^{-1}$ , and from 89 to  $0 \text{ spikes} \cdot \text{sec}^{-1}$  with rotation in an inhibitory direction (Goldberg and Fernández, 1971).

Rotation of the head around a canal normal can also produce response signals from the other five canals within the head, and those responses depend on each canal orientation with respect to the axis of head rotation. In a classical system, three ipsilateral

canals would be orthogonal to one another, having orthogonal or parallel relationships to the three contralateral canals, two canals would lie parallel to an earth-oriented (or horizontal) plane, and the other four canals would be vertical in a head neutral position (e.g., the description by Romer, 1962:485). The lateral (or horizontal) canals were assumed to lie in the head neutral plane in the classic system. In that system ipsilateral canals would register rotations in mutually exclusive directions.

However, semicircular canals from numerous species deviate from orthogonality, as demonstrated through prior studies of commonly used laboratory animals (see Table 2.6 for a summary of previous research). Those non-orthogonal canal relationships imply that head rotations can simultaneously report hair cell firings above and below the resting rates in more than one SC, and the cumulative sensitivity with any rotation direction can be greater than that predicted by rotation of a single canal. That idea was modeled mathematically for the toadfish by Rabbitt (1999) and Ifediba et al. (2007), and tested on mice by Yang and Hullar (2007); those authors all concluded that the orientation of semicircular canals in the skull was important in understanding the sensitivity of an animal's vestibular system.

Although the theoretical and experimental research done previously has made a significant preliminary contribution to understanding the importance of canal orientation to the sensitivity of rotational directions in the head, it has yet to be explored for taxa other than one strain of *Mus*. I considered the applications of those preliminary results to understanding preferred rotational sensitivity for a variety of taxa with varying locomotor styles and agilities (Appendices 7 and 8). As an example of those applications I show the results for *Dromiciops gliroides*, and graphically present the total sensitivity derived from all six semicircular canals when the head was rotated along any direction in a skull-centered reference frame.

## MATERIALS AND METHODS

Measurements for this study came from digital image scans done at the Jackson School of Geosciences High-Resolution X-ray CT Facility (Rowe et al., 1997) and rendered with VGStudioMax© (Versions 2.2; Volume Graphics GmbH, 2010). Details of the scanning methods, resolutions, and measurement methods are listed in Chapter 2.

The analysis consisted of finding the sensitivity of a single canal (G) of any SC rotated within a head-centered reference system, and then the combined sensitivity from all six SCs. Results were then mapped upon a reference unit sphere. For visual purposes that sphere is projected onto a map view, with the maximum and minimum sensitivity values indicated. Three head-centered reference planes were already incorporated into 3D digital images of each rendered skull, and all orientations for semicircular canals were recorded with regard to those planes.

In standard vestibular studies, the positive X axis passes through the rostrum, the positive Y axis passes through the left meatus, and the Z axis passes dorsally through the skull (Blanks et al., 1972). The planes containing any two of these axes are the axial, frontal, and sagittal reference planes.

After importing the 2D CT images into VGStudioMax to render a 3D model of the *Dromiciops gliroides* skull, approximately eight small reference segments along median sutures (e.g. nasals, nasion, bregma, and medial palantine sutures) were aligned in a best-fit plane to define the sagittal (XZ) plane. For the ‘horizontal’ or frontal reference plane (XY) I used Reid’s line, defined by the Tabor’s Cyclopedic Medical Dictionary as the line extending from the lower edge of the orbit to the center of the aperture of the external auditory canal (Venes, 2005: 1873). A Reid’s Plane derived from

this reference line included points taken from both sides of the skull. The axial (YZ) plane contained the line connecting the left and right external auditory meatus (known as the interaural line) perpendicular to the frontal and sagittal reference planes. The origin of the head-centered coordinate system was located at the intersection of all three planes. There is no assumption in this model that the frontal reference plane represents an actual horizontal or animal head-neutral posture; the plane was defined solely for the sake of measurement standardization. These reference planes are shown in Figure 3.2.

The least-squares fit canal radii (R) are listed in Appendix 1. Fitpoints defining each arc radius of curvature also were used to calculate a linear best-fit regression for the plane containing that SC (Ezure and Graf, 1984) and the perpendicular canal normal (N). The right-hand rule was employed to define a rotation vector of excitatory hair cell response (V) along N (Ezure and Graf, 1984, Calabrese and Hullar, 2006). The vector has both direction N and magnitude G:

$$V = NG$$

Figure 3.4 shows the maximum V along each canal normal for *Dromiciops gliroides*. All data for each canal radius and coordinates for the canal plane V are reported in Table 3.2.

To calculate the varying canal sensitivity ( $S_i$ ) of each canal ( $SC_i$ ) for any point (X) on a unit sphere with the head-centered origin as its center, Michael C. Rodgers wrote a program in the C++ programming language utilizing the general parameters of Yang and Hullar (2007). Each value of X represents one of an infinite number of locations defined by an azimuth ( $\Theta$ ) and an elevation ( $\Phi$ ) where:

$$X = \{(\cos\Theta)(\cos\Phi), (\sin\Theta)(\cos\Phi), (\sin\Phi)\}$$

Sensitivity of any canal to rotation around any axis passing through a point X (in spikes  $\cdot$  sec<sup>-1</sup>/degrees  $\cdot$  sec<sup>-1</sup>) is:

$$S_i = |X \cdot V_i|$$

An example of this calculation plotted on a spherical projection is shown for all left single canals of *Dromiciops gliroides* in Figure 3.5A-C.

The total sensitivity represented by the contributions of all six canals to any axis of rotation (S) is given by:

$$S = \sum_{i=1}^6 |X \cdot V_i|$$

An example of S plotted on a spherical projection with contours of equal intervals is shown in Figure 3.5 for every pair of left-side *Dromiciops gliroides* canals; the combination of all three canals for each side, and the total S for all six canals combined are shown in Figure 3.6. In the case of a single canal, a single view of one spherical projection shows the simple maximum, but this format cannot display the complexity of maxima and minima for combinations of two or more canals. For this reason, the final graphic results of multiple canal combinations are mapped as Mercator projections -- just as the world globe is projected onto a map. These contour maps have the same distortion near poles as all Mercator maps; but they allow perusal of all information and the relationship of maxima and minima to the skull in all directions with one view. The program and documentation can be found in Appendix 6.

## RESULTS

I introduce the reader to the results of analyses through the first seven entries in Table 3 and the set of images that resulted from those analyses in Figures 3.5 and 3.6. Table 3.2 lists the calculated magnitude of both a maximum sensitivity value ( $S_{MAX}$ ) and a minimum sensitivity value ( $S_{MIN}$ ) for each left single canal as well as the azimuth and elevation location on the unit sphere. The same information is listed for the right single

canals, but will not be figured. The  $S_{MAX}$  and  $S_{MIN}$  are the result of rotating the canal in the positive (excitatory) direction around  $V$ . In the case of a single canal, if you rotated the opposite pole according to the right-hand rule you would receive the maximum inhibitory signal by slowing down the hair cell firing rate down to its slowest (possibly achieving a rate of  $0 \text{ spikes} \cdot \text{sec}^{-1}$  depending on the rotation rate of the head). Figures 3.5A, B, C show the unit sphere plot orientations with the best view of each single canal maximum (green dot) which also corresponds to the location of the canal plane normal. The unit sphere plots hide the sensitivity values on the far side of the sphere. Figure 3.5D shows the Mercator projection (2D isomap) of the LASC single canal. On that map both the maximum and its pole are shown in proper orientation with respect to all three axes and Reid's plane (heavy black line). For these and all following images, a gray scale was applied to the contours so the  $S_{MAX}$  is the whitest, and conversely,  $S_{MIN}$  the darkest. The images are also placed over unscaled red images of the *Dromiciops gliroides* skull for visualizing viewing orientations. All images show the results with respect to a head-centered reference system only, and do not include post-cranial morphology, even on the map projections. Because the skull images are not scaled, larger skull images that filled the map to the back could be used, but that use would obscure the axial view because the rostrum would protrude through the unit sphere.

The next three table entries and Figure 3.5E, F, G show reference sphere orientations for combinations of two left canals at a time.  $S_{MAX}$  and  $S_{MIN}$  shifted in each case owing to the additive values of each canal; the isomap for LASC + LPSC in Figure 3.5H shows this most dramatically. The LASC influenced the position of  $S_{MAX}$  the most because it is the largest canal in the combination, but  $S_{MAX}$  lies between both the LASC and LPSC maxima, which are not orthogonal. In all two-canal combinations  $S_{MAX}$  is also greater than that of any single canal, but the story is more complex for  $S_{MIN}$ . There also

are secondary maxima because of the complex additive values of the two canals. Visualizing all these relationships at once is what highlights the value of having these 2D projections.

Figure 3.6A and 3.6B show the results of combining all three left canals, and for this the isomap is more illustrative. This combined  $S_{MAX}$  stays between LASC and LPSC, but is now slightly above Reid's plane because of the addition of LLSC (which is only slightly smaller than LPSC).  $S_{MAX}$  for the combination of all three canals is also greater than that of any other single left canal or left canal combination ( $0.20 \text{ spikes} \cdot \text{sec}^{-1}/\text{degrees} \cdot \text{sec}^{-1}$ ).

The combination of all three right canals, are illustrated in Figure 3.6C and 3.6D. Once again,  $S_{MAX}$  for the three-canal combination is greater than the  $S_{MAX}$  for any single or right-canal pair ( $0.18 \text{ spikes} \cdot \text{sec}^{-1}/\text{degrees} \cdot \text{sec}^{-1}$ ).

Finally, all six canals are combined, with the results listed as the last entry in Table 3.2 and shown in Figure 6E and 6F. In this situation each canal contributes a paired  $S_{MAX}$  with every other one, leading to a complex set of secondary maxima. But the final  $S_{MAX}$  is greater than any other single canal or canal combination ( $0.27 \text{ spikes} \cdot \text{sec}^{-1}/\text{degrees} \cdot \text{sec}^{-1}$ ). Clearly the non-orthogonal nature of the canals affected the presence of a maximum sensitivity that would not have occurred had all canals been in orthogonal relationships and so had not interacted. In that case the two anterior canal values would be considered the major determinants of sensitivity to rotation in *Dromiciops gliroides*.

The parameters used for *Dromiciops gliroides* were analyzed by a similar program written by Yang and Hullar (2007) that set the two groups of ipsilateral canals to a  $S_{MAX}$  of  $1 \text{ spike} \cdot \text{sec}^{-1}/\text{degrees} \cdot \text{sec}^{-1}$ , and obtained a similar orientation of  $S_{MAX}$ . This additional analysis served as a check on the program written for this study, shown in Figure 3.7.

Another common method to designate head rotation is in terms of roll, pitch, and yaw. In vestibular biophysical convention roll refers to rotation about the X axis, pitch to rotation about the Y axis, and yaw to rotation about the Z axis (Figure 3.8). In these terms of rotation, *Dromiciops gliroides* is most sensitive to pitch rotation (rotation about or close to the Y axis), and least sensitive to yaw rotation. This method may allow for a simplification in interpretations of head rotation sensitivities, as discussed below.

## **DISCUSSION**

### **Reference Plane Selection**

Although there are several head orientation reference planes in use, I used Reid's plane. That plane was first defined as a 'horizontal' reference by Horsley and Clarke (1908) who used it for a stabilizing device to explore the cerebellum. It worked through the insertion of stabilizing bars into the two primate acoustic meatus and fixing the device so both were level with the inferior margin of the orbits. Reid's plane has since been in common use as a reference plane system for vestibular studies in all types of subject taxa (Blanks et al., 1972, 1975, Curthoys et al., 1975, Day and Fitzpatrick, 2005, Della Santina et al., 2005). Unfortunately, the plane can be a poor choice for many taxa that have an external meatus oriented behind or above the inner ear (e.g., rabbits, otters, and moles). Additionally, animals with large orbits (e.g., *Allactaga*) skew the Z axis with respect to the dorsal surface of the skull. Other reference planes could easily be defined that would work around such problems, but one requirement of any system would be that the Y axis pass through the inner ear and sagittal plane, because that system should be

part of a head-centered reference, which has been supported as the actual system for vestibular signaling to brain centers. Although authors may change reference planes, the program written by MCR can accommodate any that have coordinates and an R value for each semicircular duct or bony canal plane.

The unfortunate naming of the human lateral semicircular canal as the ‘horizontal’ canal leads to much confusion, and probably the assumption that the plane of the lateral canal represents the true natural horizontal resting head orientation in any animal. Even in humans, the lateral (‘horizontal’) canals are inclined  $18.8^\circ$  above the earth-neutral plane from posterior to anterior with regard to resting head orientation (Blanks et al., 1975). For their experiments with rabbits, Mazza and Winterson (1984) had to mechanically pitch the heads downward  $15^\circ$  from resting position to bring the lateral canals into a horizontal position. Thus, the horizontal reference plane should not be set to the plane containing the lateral semicircular canals.

A recently published method for determining semicircular canal orientations did not use external skull information for a horizontal reference, but instead relied on the assumptions that synergistic lateral canals lie in the same plane and are essentially perpendicular to the force of gravity (David et al., 2010). That method did not include the use of inhibitory vectors for each of the six canals when determining sensitivity, and therefore oversimplified the sensitivity percentage plots.

## Canal Maxima versus Prime Direction for Sensitivity Calculations

The sensitivities I calculated were taken directly from each of the six canal vectors and radii; this is the head sensitivity initially sent by individual nerve fibers to the vestibular ganglion (Rabbitt, 1999, Ifediba et al., 2007). En route to their several primary destinations there is evidence that the signals are combined so that separate branches of the ipsilateral vestibular nerve report signals from one canal in prime direction (PD), calculated as the V which includes only the orientations for one canal that exclude sensitivity signals from the other two canals (along their canal planes). This PD method was used by other authors (Haque et al., 2004, Calabrese and Hullar, 2006, Hullar and Williams, 2006, Cox and Jeffery, 2008, Jeffery and Cox, 2010). In their study of maximum rotational sensitivity directions in mice, Yang and Hullar (2007) described using canal maxima, but mistakenly reported model results for PD (Hullar pers. communication 2010; confirmed by the present model). My method calculated the sensitivity taken directly from each of six canal vectors, because that is the head sensitivity initially sent by the nerve fibers individually (Rabbitt, 1999, Ifediba et al., 2007), although the program is developed to allow calculation and display of PD results.

I compared the results for *Dromiciops gliroides* with those for an experimental lab strain of *Mus domestica*, for which Yang and Hullar (2007) gave usable measurements in a comparable reference system. Although the wild *Mus musculus* is judged to be a “terrestrial but able climber” and a “good swimmer” (Nowak, 1999) with an agility score of 4 indicating a generalist locomotor lifestyle (Spoor et al., 2007), *Dromiciops gliroides* appears to be mostly scansorial with an agility score of 5 indicating faster 3D movement

(Spoor et al., 2007). The diagrams and data indicate that both taxa are most sensitive to pitch rotation (compare Figure 3.6, 3.7, 3.8, and 3.9), because maxima axes of rotational sensitivity for both taxa lay close to the Y axis.

Do such comparisons with other taxa show that all taxa have similar greater pitch sensitivity? Other authors previously indicated that the vestibulo-ocular relationships in frontal vs. lateral-eyed animals do not rely on alignment of eye muscles and canal orientation but have more to do with other vestibular needs (Cox and Jeffery, 2008). Other workers found contributions of the vertical canals in balance control, although lateral canals appear to contribute to navigation in addition to head orientation (Shaikh et al., 2005, Beraneck and Lambert, 2009, Muir et al., 2009) or translation of vestibular signals into earth-orientation coordinates (Day and Fitzpatrick, 2005, Yakusheva et al., 2007). Alternatively, sensitivity directions may be restricted by biomechanical constraints from skeletal structure (Moore et al., 2005).

Phylogeny may override other considerations in the orientation of semicircular canals. Finally, directions of  $S_{MAX}$  may give indications of locomotor lifestyles, as suggested by Yang and Hullar (2007). I have data from 31 species that will be analyzed for evidence supporting each of the last two possibilities and reported elsewhere.

## CONCLUSIONS

The program to evaluate changes in total semicircular canal sensitivity with head rotations around any axis in the head provided a resultant map of the total sensitivity distribution. The results for *Dromiciops gliroides* indicate maximum canal sensitivity to

rotation close to the pitch head direction. Orientation of all canals with respect to the head was an important determinant of the sensitivity. Lack of perpendicular canal relationships affected the final head sensitivity, which is used in several separate higher brain centers to determine balance, navigation, and earth-centered head reference systems. This program provided a comparison of  $S_{MAX}$  values and relative orientations for *Dromiciops gliroides* and *Mus domestica*, and can be used with other taxa and other reference planes to provide illumination of rotational sensitivities in extant and extinct taxa.

## TABLES

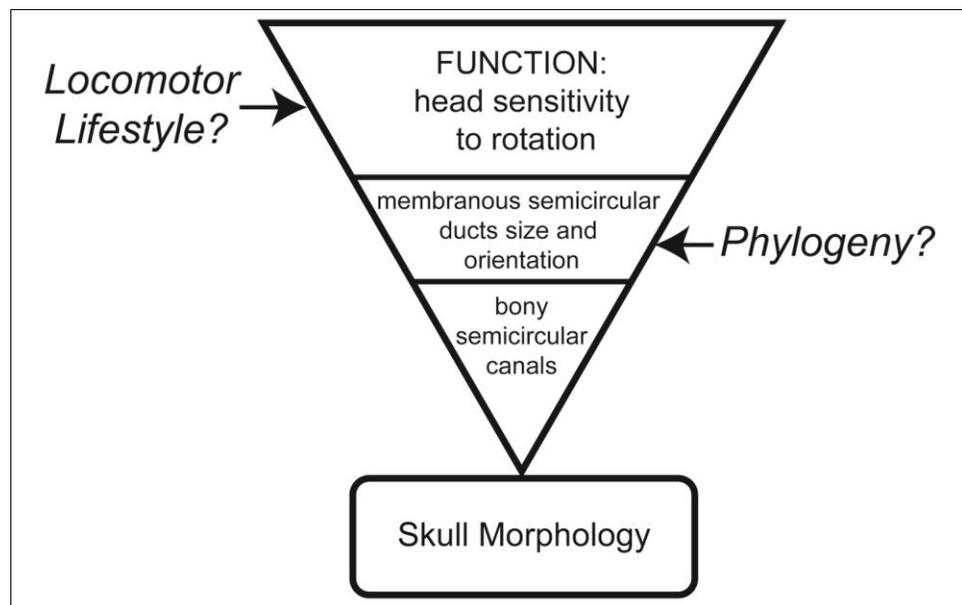
**Table 3.1.** Abbreviations used in this paper.

Abbreviation	Description	Measurement
CN VIII	Vestibular cranial nerve	
PD	Prime Direction, taken in direction where other two canals have no sensitivity above resting firing rate.	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> ) in unitless Cartesian coordinates
<b>Bony Canals:</b>		
ASC	Anterior Semicircular Canal	
LASC	Left Anterior Semicircular Canal	
LLSC	Left Lateral Semicircular Canal	
LPSC	Left Posterior Semicircular Canal	
LSC	Lateral Semicircular Canal	
PSC	Posterior Semicircular Canal	
RASC	Right Anterior Semicircular Canal	mm
RLSC	Right Lateral Semicircular Canal	mm
RPSC	Right Posterior Semicircular Canal	mm
SC	Semicircular Canal	
CC	Common Crus	
<b>Equation Variables:</b>		
G	Sensitivity of a semicircular duct	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> )
N	Canal Plane Normal perpendicular to canal plane	unitless cartesian coordinates
R	Radius of a semicircular canal	mm
S	Total sum of all canal sensitivities at any point on a Unit Sphere	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> )
Si	Sensitivity of a canal SC <sub>i</sub> mapped on the Unit Sphere	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> )
S <sub>MAX</sub>	Maximum value of S as mapped on the Unit Sphere	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> )
S <sub>MIN</sub>	Minimum value of S as mapped on the Unit Sphere	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> )
V	Vector with direction (excitatory direction of canal normal N) and magnitude G	(spikes • sec <sup>-1</sup> /degrees • sec <sup>-1</sup> ) in (x,y,z) direction
X	A single point on the unit sphere, used as an independent variable	no magnitude, points determined by azimuth and elevation

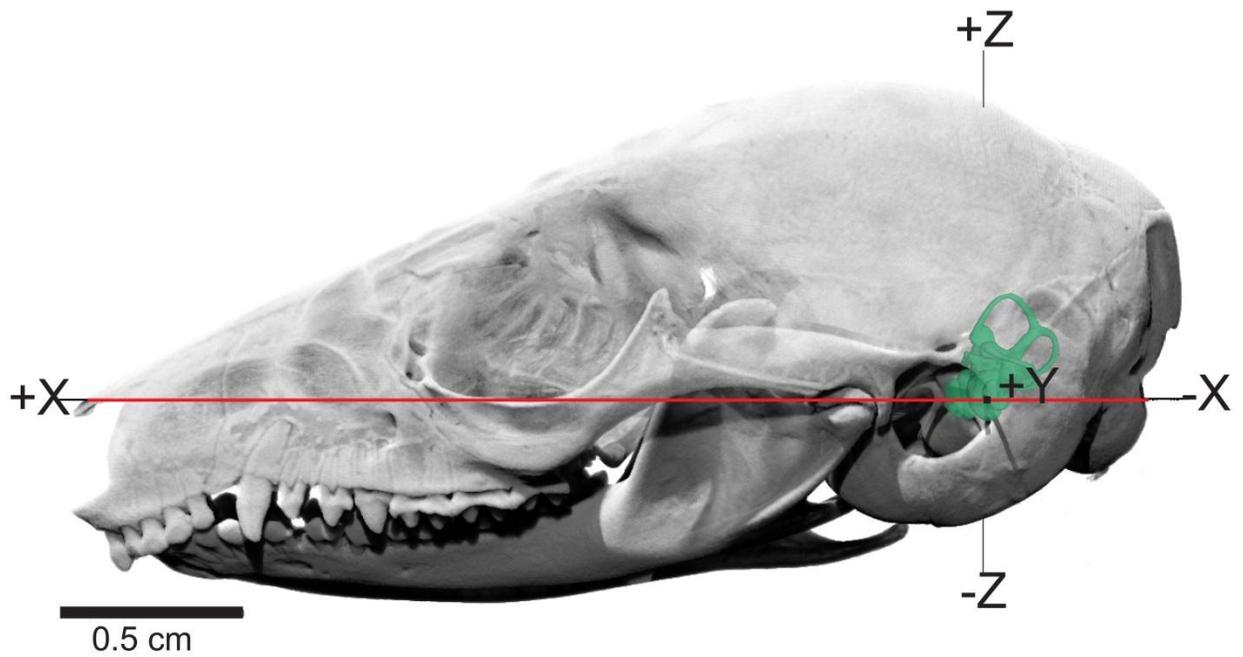
**Table 3.2.** S calculations for all combinations of the six *Dromiciops gliroides* semicircular canals, showing the values and locations of maxima and minima.

<i>Dromiciops gliroides</i> Canal Combination	Maximum			Minimum		
	S <sub>MAX</sub>	Azimuth	Elevation	S <sub>MIN</sub>	Azimuth	Elevation
LASC	0.15	137.11	-9.84	0.00	121.64	79.80
LLSC	0.09	-163.48	65.74	0.00	-44.65	12.30
LPSC	0.09	-129.02	-22.85	0.00	-129.02	-22.85
LASC + LLSC	0.18	147.30	17.58	0.00	50.63	20.39
LASC + LPSC	0.18	166.64	-20.04	0.00	-154.69	65.04
LLSC + LPSC	0.12	-139.57	21.45	0.00	135.35	-12.30
LASC + LLSC + LPSC	0.20	171.91	5.63	0.03	24.96	-65.04
RASC	0.16	50.63	8.09	0.00	-48.52	47.81
RLSC	0.07	-23.20	-66.80	0.00	-84.73	11.60
RPSC	0.08	-55.90	24.61	0.00	19.34	-29.18
RASC + RLSC	0.17	40.78	-15.47	0.00	-42.89	22.15
RASC + RPSC	0.16	57.30	4.57	0.00	156.09	61.52
RLSC + RPSC	0.10	-46.41	20.74	0.00	39.38	11.25
RASC + RLSC + RPSC	0.18	47.81	-17.93	0.02	156.09	61.52
INTERACTION OF ALL SIX CANALS	0.27	101.25	20.74	0.13	-1.41	-76.99

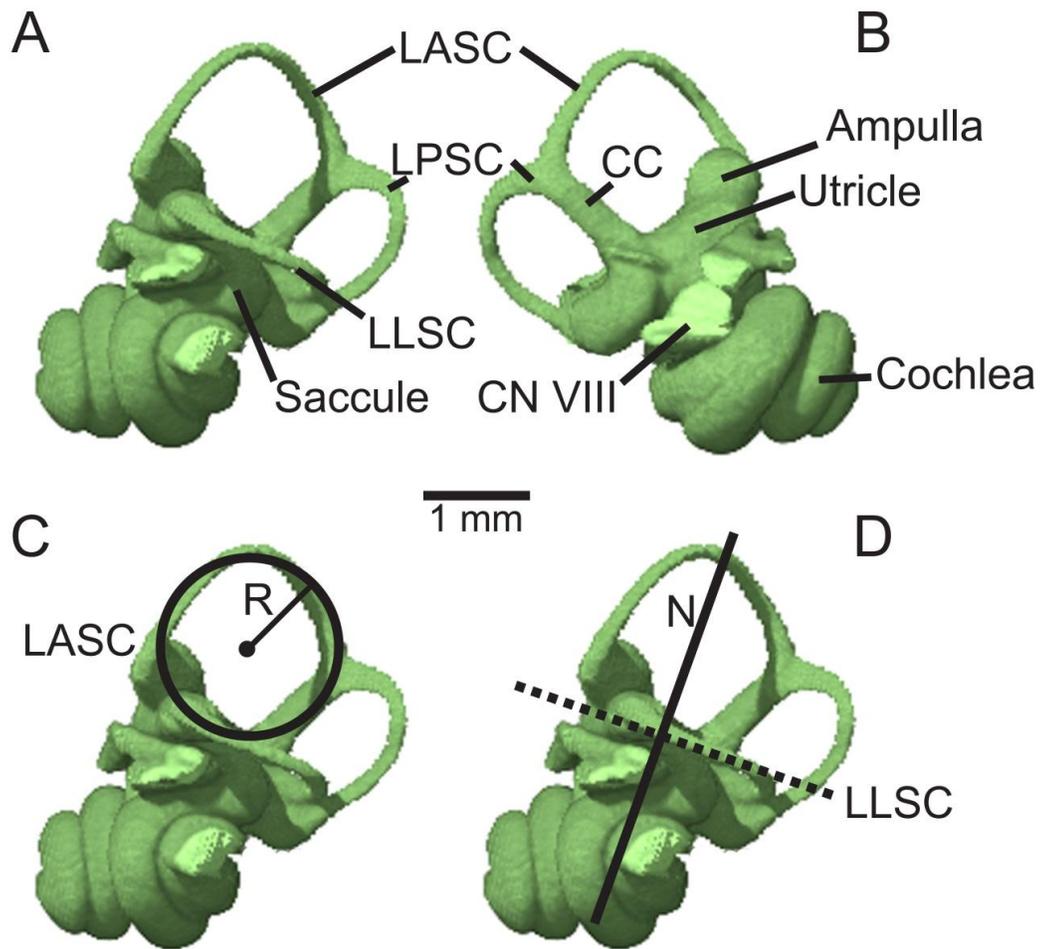
## FIGURES



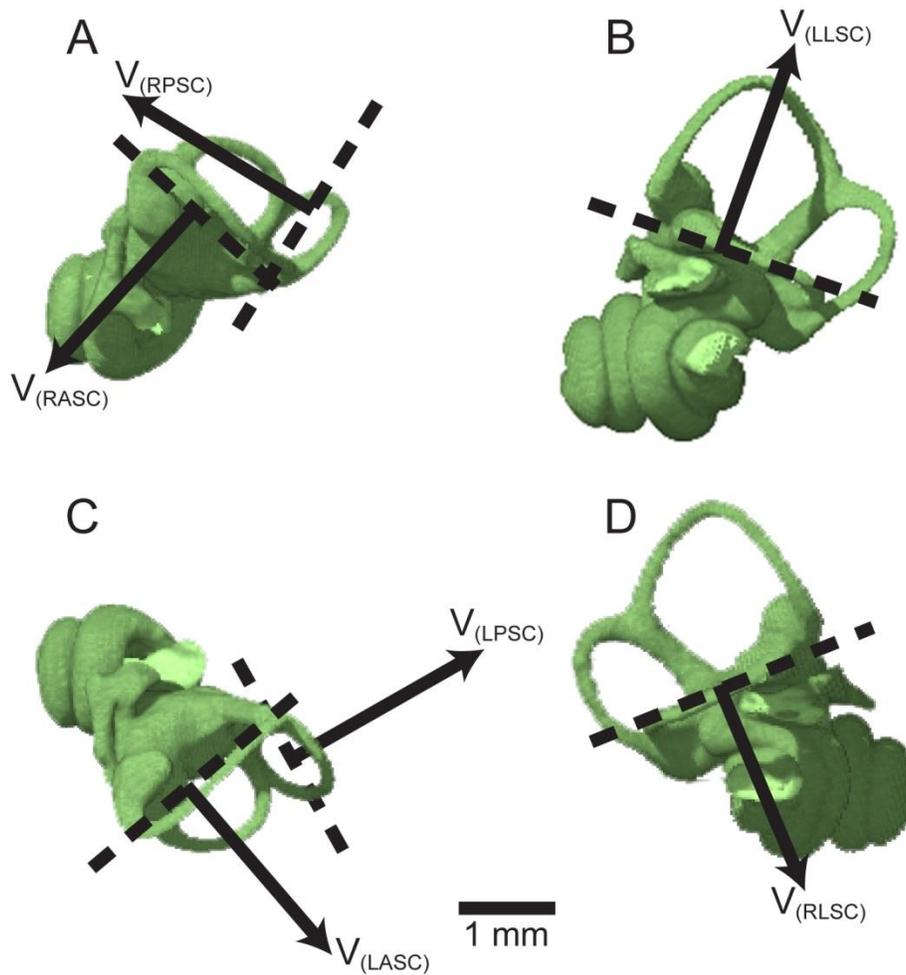
**Figure 3.1.** Working from skull morphology to answer questions about the source of membranous semicircular duct size and orientation. The skull preserves the necessary information within bony semicircular canals, from which we can infer size and orientation of membranous ducts, and therefore calculate the sensitivity of that animal to head rotation. The triangle is inverted to indicate that each step up represents yet a further inference from original morphology. Modified from Witmer (1995).



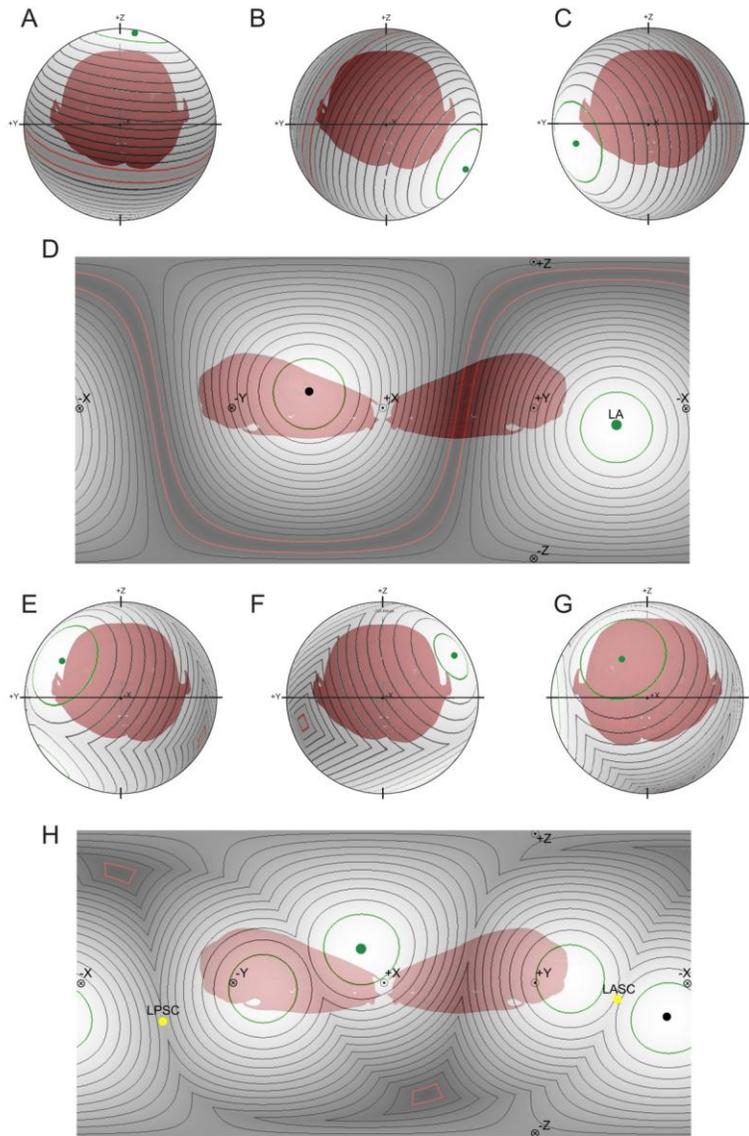
**Figure 3.2.** Skull of *Dromiciops gliroides* (FMNH 127463) showing orientation of designated reference planes and axes with respect to skull landmarks. Red line represents Reid's Plane determined by inferiormost orbits and midpoints of the two external acoustic meatus.



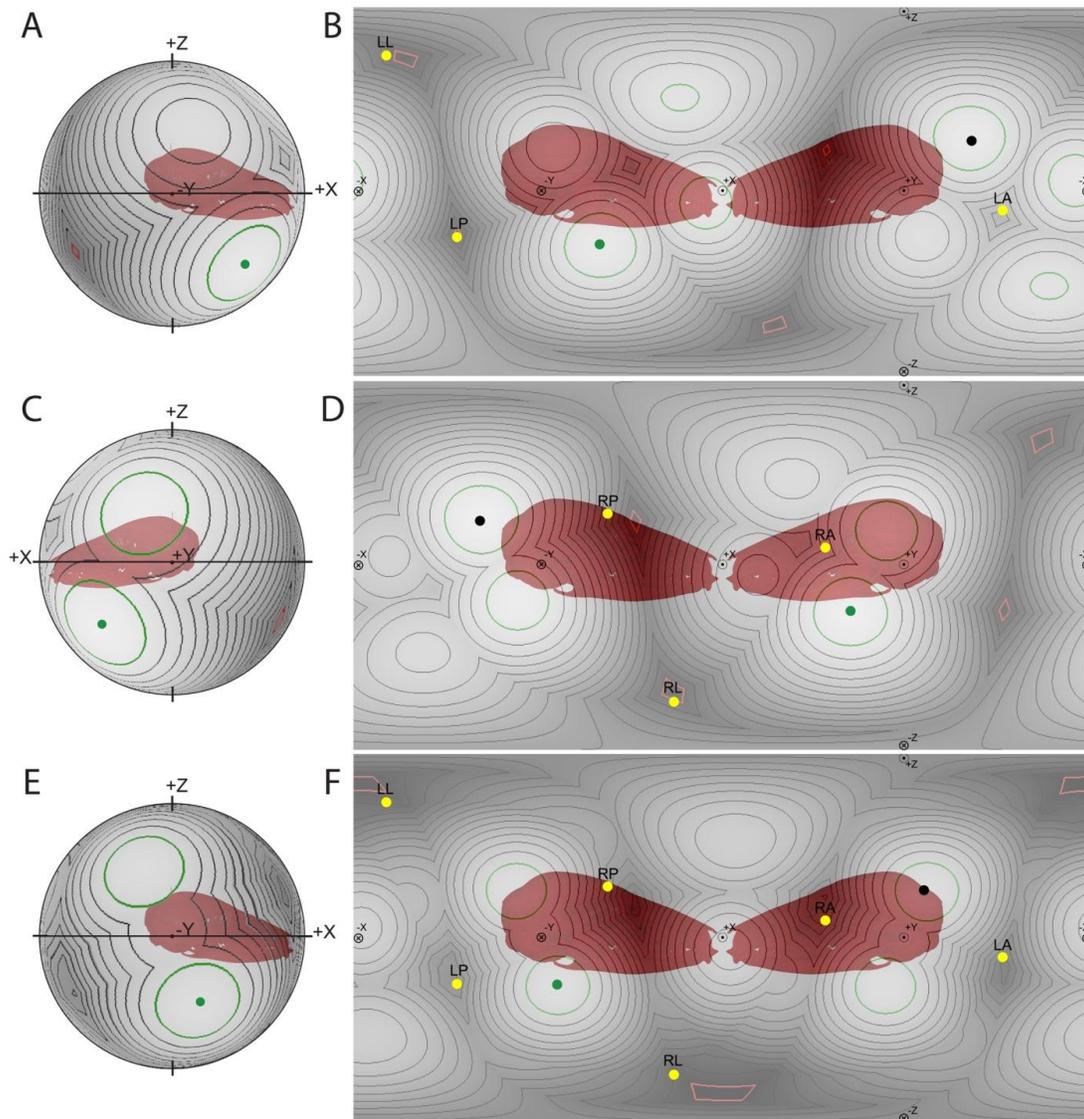
**Figure 3.3.** Locations of inner ear structures and primary measurements used in this study shown for *Dromiciops gliroides* on digitally constructed endocast of the left inner ear. **A**, left lateral view with structures labeled. **B**, medial view with structures labeled. **C**, left lateral view showing a circle fitted to the arc of LASC curvature, with the radius of curvature  $R$ . **D**, left lateral view showing the best fit plane to the LLSC and the unit normal to that plane. Note that the unit normal has a geometrically ambiguous polarity. In each view the cochlea points anteriorially. Abbreviations listed in Table 3.1.



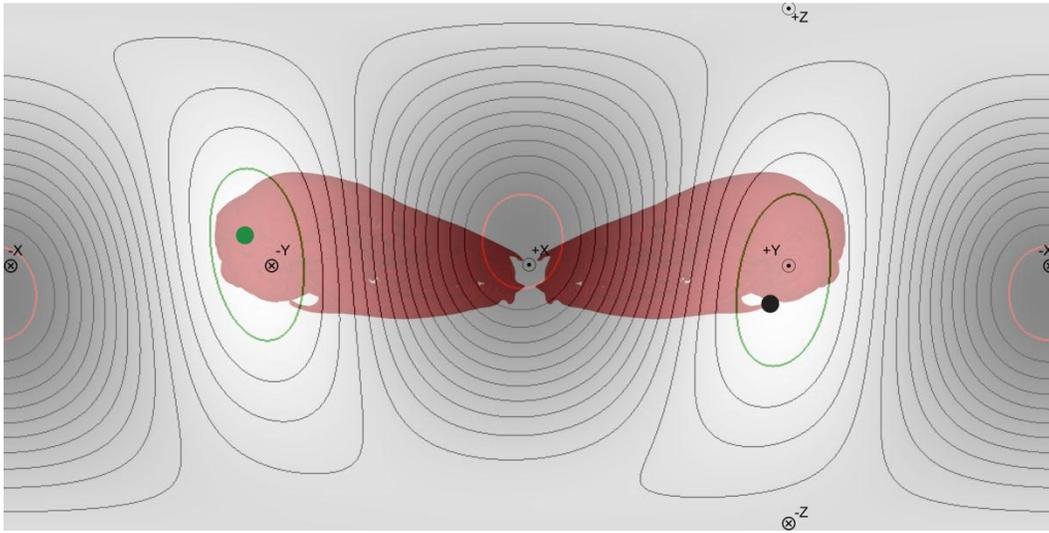
**Figure 3.4.** General orientations of canal planes and directions of rotation vectors taken in positive right-hand excitatory directions, as indicated on digitally rendered bony endocasts of *Dromiciops gliroides* inner ear. **A**, dorsal view of right inner ear showing directions of  $V$  for RASC and RPSC. **B**, left lateral view of left inner ear indicating  $V$  for LLSC (compare to  $N$  of Figure 3D,  $V$  is now polarized). **C**, dorsal view of left inner ear showing directions of  $V$  for LASC and LPSC. **D**, right lateral view of right inner ear showing direction of  $V$  for RLSC. Note direction of  $V$  is approximately opposite that of that in **B** because LLSC and RLSC are synergistic canals. In each view the cochlea points in anterior direction of the skull, and all abbreviations listed in Table 3.1.



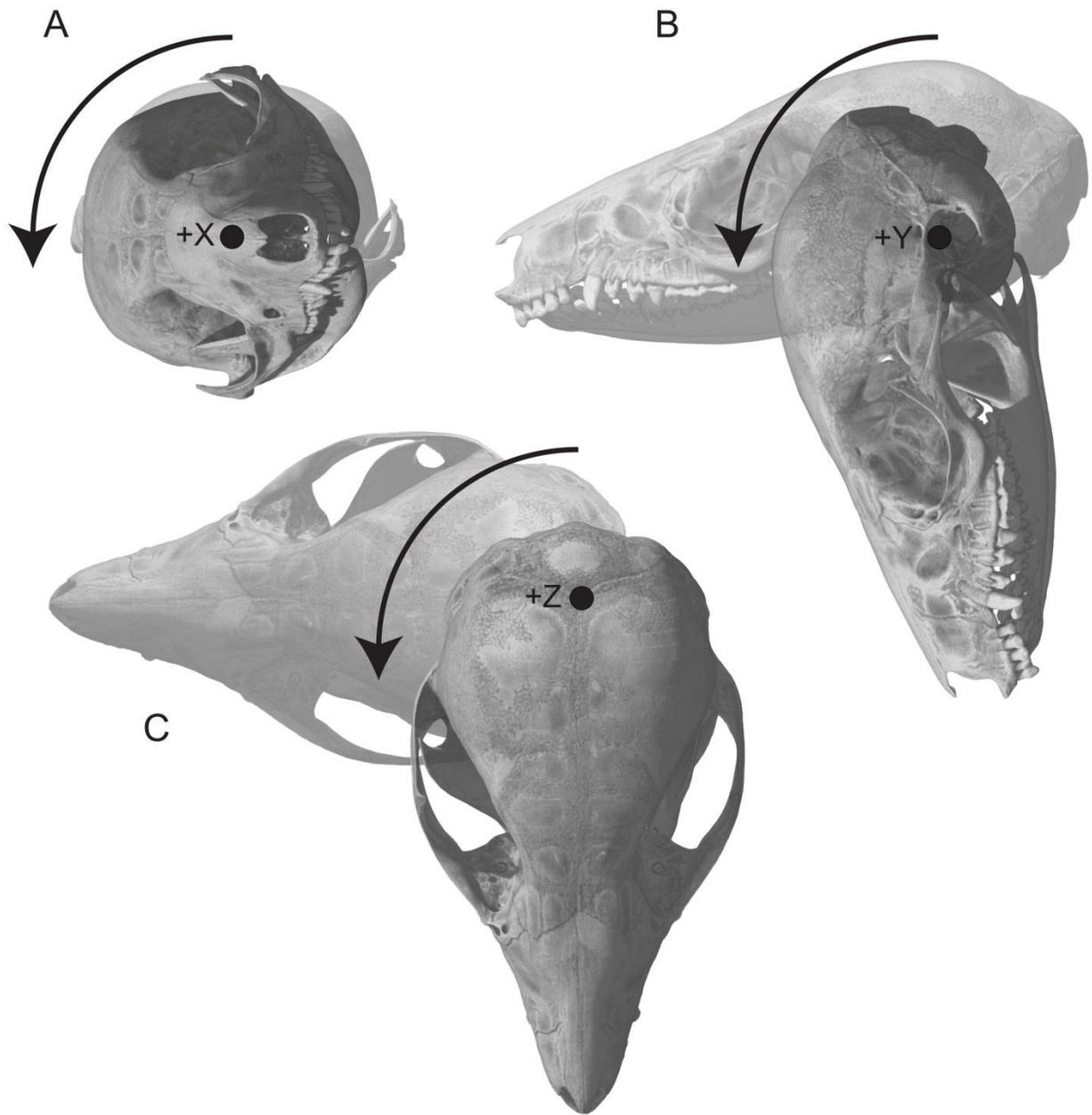
**Figure 3.5.** Unit sphere and contour map compilation for single semicircular canal and paired canal combination sensitivities shown for *Dromiciops glioides*. **A**, LLSC posterior view. **B**, LPSC posterior view. **C**, LASC posterior view. **D**, contour map of LASC. **E**, LASC + LLSC posterior view. **F**, LLSC + LPSC posterior view. **G**, LASC + LPSC anterior. **H**, contour map of LASC + LPSC. **Black line**, XY reference plane (Reid's plane); **yellow dots**, SC canal locations; **green contour and dot**,  $S_{MAX}$ ; **red contour**, zone of  $S_{MIN}$  sensitivity; **black dot**,  $180^\circ$  pole to  $S_{MAX}$  axis; maxima and minima given in Table 3. **Red skulls** are given for orientation purposes only, and do not represent scaled figures, **gray scales** indicate increasing and decreasing sensitivity with highest sensitivity in lightest tones.



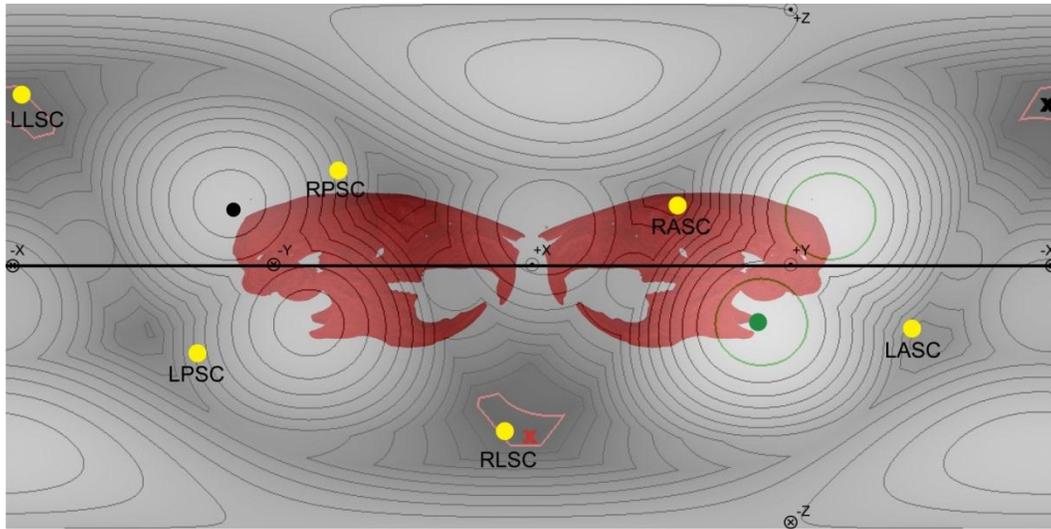
**Figure 3.6.** Unit sphere and contour map compilation for ipsilateral canals on left and right sides and total combined canal sensitivities shown for *Dromiciops gliroides*. **A**, all left canals combined and shown in right view. **B**, contour map of all left canals combined. **C**, all right canals combined shown in left lateral view. **D**, contour map of all right canals combined. **E**, all six *Dromiciops gliroides* semicircular canals combined, right lateral view. **F**, contour map of all six *Dromiciops gliroides* semicircular canals combined. Symbols as in Figure 3.5, and  $S_{MAX}$  as listed in Table 3.2. Image underlays of red skulls are given for orientation purposes only, and do not represent scaled figures. Compare the contour map for the final combined  $S_{MAX}$  with that of *Mus domestica* (Figure 3.9).



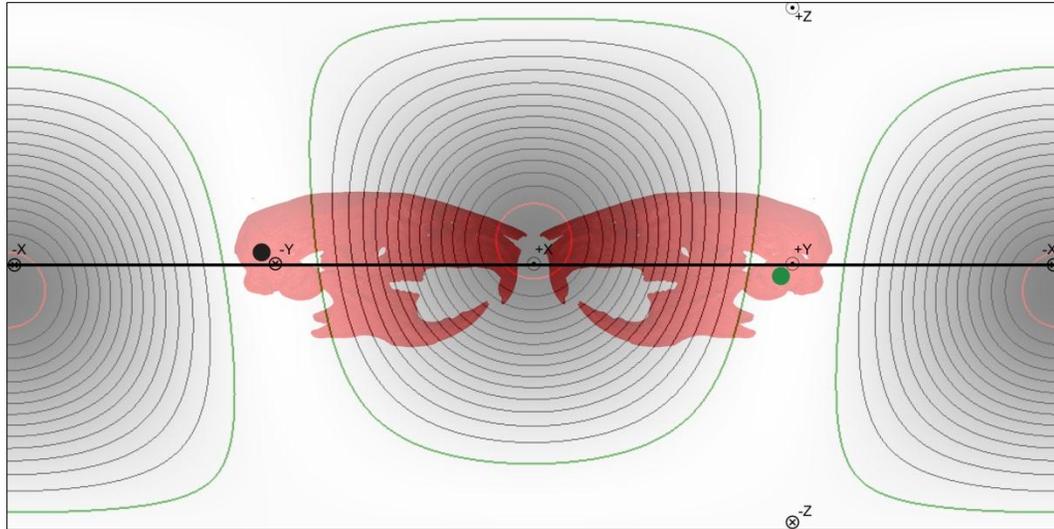
**Figure 3.7.** Contour map compilation for *Dromiciops gliroides* calculated by the Yang and Hullar program (2007).  $S_{MAX}$ , 2.15 (spikes  $\cdot$  sec $^{-1}$ /degrees  $\cdot$  sec $^{-1}$ );  $S_{MIN}$ , 1.79 (spikes  $\cdot$  sec $^{-1}$ /degrees  $\cdot$  sec $^{-1}$ ); symbols defined in Figure 3.5 and Table 3.1.



**Figure 3.8.** Illustrations of rotation terms. **A**, roll; **B**, pitch; **C**, yaw.



**Figure 3.9.** Isomap for all six semicircular canals combined in *Mus domestica* generated with the MCR program.  $S_{MAX} = 0.27$  (spikes  $\cdot$  sec $^{-1}$ )/(degrees  $\cdot$  sec $^{-1}$ ),  $S_{MIN} = 0.13$  (spikes  $\cdot$  sec $^{-1}$ )/(degrees  $\cdot$  sec $^{-1}$ ), **green contour** = 0.25 (spikes  $\cdot$  sec $^{-1}$ )/(degrees  $\cdot$  sec $^{-1}$ ), **red contour** = 0.14 (spikes  $\cdot$  sec $^{-1}$ )/(degrees  $\cdot$  sec $^{-1}$ ). Symbols defined in Figure 3.5 and Table 3.1; R and orientation values from Calabrese and Hullar (2006).



**Figure 3.10.** Isomap for all six semicircular canals combined in *Mus domestica* generated with the Yang and Hullar (2007) program.  $S_{MAX} = 2.24$  (spikes  $\cdot$  sec<sup>-1</sup>)/(degrees  $\cdot$  sec<sup>-1</sup>),  $S_{MIN} = 1.51$  (spikes  $\cdot$  sec<sup>-1</sup>)/(degrees  $\cdot$  sec<sup>-1</sup>), **green contour** = 2.15 (spikes  $\cdot$  sec<sup>-1</sup>)/(degrees  $\cdot$  sec<sup>-1</sup>), **red contour** = 1.56 (spikes  $\cdot$  sec<sup>-1</sup>)/(degrees  $\cdot$  sec<sup>-1</sup>). Symbols defined in Figure 3.5 and Table 3.1; R and orientation values from Calabrese and Hullar (2006).

## **CHAPTER 4: THE PETROSAL LOBULE ALMOST FILLS THE SUBARCUATE FOSSA IN *MONODELPHIS DOMESTICA* (DIDELPHIDAE, MARSUPIALIA): IMPLICATIONS FOR AGILITY EVALUATIONS**

### **ABSTRACT**

Previous authors questioned whether the petrosal lobule of the cerebellar paraflocculus completely fills the structurally related space within the patent subarcuate fossa, particularly in the case of *Monodelphis domestica*. Use of micro-Magnetic Resonance Imaging ( $\mu$ MRI, 7T and 11.7T magnets, 3-4 scan protocols) and High Resolution X-ray Computed Tomography (HRXCT) on the same three specimens of *Monodelphis domestica* (76 days postnatal), allowed the first quantitative volumetric documentation of both subarcuate fossa and petrosal lobule. The average volume of the subarcuate fossa occupied by the cerebellar petrosal lobule tissue was  $95\% \pm 2\%$ . This result contrasts markedly with a prior estimate of 50% based on a histologic study, and is consistent with established documentation of bone ossification response to hydrostatic neural tissue expansion. The results also indicate the problems of estimating brain volumes from histologic or preservative-fixed specimens.

A further consideration is whether the development of the subarcuate fossa determines the arc radius of curvature of the membranous semicircular ducts, or vice versa. Given the documented cessation of semicircular canal growth by 27 days postnatal in *Monodelphis domestica* versus the continued expansion of the subarcuate fossa until at least 90 days postnatal, comparison of the effects of one upon the other appears

unfounded. These findings also allow for further study of subarcuate fossa size as a measure of the agility of a taxon.

## INTRODUCTION

The relationship between fossilized bones and the soft tissues that once surrounded and filled those bones is of concern to paleontologists. That relationship determines not only the appearance of an animal, but also its physical and sensory capacities; and some extinct lineages have no extant representatives from which to extrapolate those characteristics (Witmer, 1995).

This situation is particularly pertinent in studies of the vertebrate head. The skull holds great interest for morphologists and paleontologists, because it houses numerous cranial sensory organs, as well as the brain which processes sensory signals. The relative amount of space in the skull that is devoted to housing different sensory organs in each taxon provides information regarding which sensory input dominated the acuity and feeding or locomotor strategy of the organism. An example of the possible sensory organ/processing spatial relationship in mammals concerns the semicircular membranous ducts and the petrosal lobule (also known as *lobulus petrosus*) of the cerebellum. Both of those are accommodated in the petrosal bone (or petrous pyramid) with the petrosal lobule partially constrained by a 'cage' of the bony semicircular canals, and surrounded

by the subarcuate fossa (parafloccular fossa of Hoyte, 1961). Previous terminology designating which cerebellar lobule fills the subarcuate fossa is confused; Although the petrosal lobule has also been referred to as the paraflocculus, dorsal paraflocculus, and ventral paraflocculus, only the end of the dorsal paraflocculus actually fills the actual subarcuate fossa. Previous authors questioned the extent to which the petrosal lobule fills the subarcuate fossa throughout ontogeny or adulthood (McClure and Daron, 1971, Gannon et al., 1988, Sánchez-Villagra, 2002, Jeffery and Spoor, 2006, Jeffery et al., 2008). Both features were also hypothesized to be indicators of mammalian locomotor agility. Most commonly the arc radius of curvature of the semicircular canal has been correlated to agility in numerous measurements (Jones and Spells, 1963, ten Kate et al., 1970, Spoor et al., 1994, 2007, Spoor, 2003, Hullar, 2006), but the function of the petrosal lobule in coordinating vestibular and eye interaction led to conjectures about the relationship between its size and its ability to enable agile movements (Olson, 1944). Yet given their spatial relationships the size of one system could affect the other by constraint or expansion. However, a lack of technology previously hampered our ability to work out the relationship between the petrosal and associated soft tissues in both extant and extinct species. Simultaneous quantitative evaluation of the relationships between the semicircular canals, subarcuate fossa, and petrosal lobule was nearly impossible in the past, so researchers were forced to settle for use of histologic sections and qualitative measurements of these features. Such methods, especially when utilized on preserved specimens, resulted in preparation artifacts and questionable conclusions. More recently, with the use of  $\mu$ MRI (on preserved specimens), Jeffery and Spoor (2006) found a

correlation between the anterior semicircular canal size and the subarcuate fossa in a study of three primate species that included humans.

Through the unique use of  $\mu$ MRI and HRXCT available for three freshly euthanized juvenile specimens of *Monodelphis domestica*, I obtained quantitative size and volumetric information on the bony semicircular canals, subarcuate fossa, and petrosal lobule and report the results here. The inclusion of a subarcuate fossae endocast volumes from an ontogenetic series (Macrini et al., 2007a) allows a discussion of subarcuate fossa ontogeny in *Monodelphis domestica*.

## **MORPHOLOGY**

### **Petrosal**

Figure 4.1 shows the location of the petrosal (periotic, or petrosal portion on the temporal area when fused to the skull) bone in the basicranium of *Monodelphis domestica*. Figure 2 shows the left petrosal in medial view, with articulating bone contacts. The petrosal is the densest bone in the skull (MacIntyre, 1972, Wible, 2003), and so is easily identifiable in HRXCT images. The dorso-lateral *pars canalicularis* of the petrosal surrounds a membranous utricle and three membranous semicircular ducts. Upon skeletonization of the skull, the paths of the membranous ducts are preserved as the spaces of bony semicircular canals (Figure 4.3). Another feature of the *pars canalicularis* is observable on the medial surface of the skull, the depression of the subarcuate fossa

(Figure 4.2). Although it is observable, the depth and full shape of the subarcuate fossa is difficult to measure visually. In life, the fossa is filled by the petrosal lobule of the cerebellum (LP, Larsell and Jansen, 1970). The extent to which the petrosal lobule fills the subarcuate fossa has been a matter of confusion. The semicircular canals and the petrosal each influence the morphology of the petrosal bone (Norris, 1994).

### **Semicircular Canals**

The anterior (superior), lateral (horizontal), and posterior (inferior) membranous ducts are toroids containing endolymph, surrounded by perilymph within the containing semicircular canals. Each duct terminates in an ampulla, which contains a diaphragm-like gelatinous cupula embedded in a crista of afferent ‘hair cell’ neurons. Any angular change in head movement results in inertial drag by the endolymph within at least one duct, which causes deflection of the ampullar cupula and firing of the embedded hair cells (Hullar and Page, 2010).

Semicircular ducts relay angular movement detection via primary neurons in the vestibular nerve (CN VIII) to primary processing vestibuli in the medulla-pons area. Secondary neurons carry information from these processing nuclei to other areas of the brain for various informational use. There are slight interspecific variations of these projections, but *Didelphis virginiana* and presumably for other members of Didelphidae appear to follow the general mammalian pattern (Henkel and Martin, 1977).

The skeletal spaces enclosing the semicircular canals are considered reasonable proxies for dimensions of the membranous semicircular ducts (Hashimoto et al., 2005,

Ifediba et al., 2007). In a broken, sectioned, or digitally imaged skull, the canals are easily visible (Figure 4.3), and the empty spaces can be digitally filled to provide 3D virtual casts, or endocasts of the bony semicircular canals (Figure 4.4). The path described by each semicircular canal can be approximated by a best-fit circle taken from numerous points around the arc of the canal, as reported in terms of its radius of curvature (Curthoys et al., 1977; Figure 4.3). Another method for calculating the radius of curvature entails adding the height and width of the canal and dividing by four (Spoor and Zonneveld, 1995). Previous authors related the size of the canals to the agility of species, concluding that a larger radius-of-curvature to body-mass index indicates the species of greater agility. The length of radius of curvature for the anterior canal is almost always the greatest of the three canals (Spoor, 2003).

### **Subarcuate Fossa**

The subarcuate fossa lies lateral to a margin, known as the ostium, occurring at the medial bony arc of the anterior semicircular canal (Figure 4.2). The subarcuate fossa extends to the interior mastoid exposure of the petrosal, while contacting the arcs of the lateral and posterior semicircular canals. Where not constrained by canal arcs, the subarcuate fossa expands centrifugally to a greater extent within the petrosal (Figures 4.4, 4.5, and 4.6; Hoyte, 1961, Jeffery et al., 2008). There is an important relationship between the subarcuate fossa size and body size due to negative allometry.

The subarcuate fossa is primitive in the mammalian lineage (Macrini et al., 2007b, Character #14), occurring with few exceptions in taxa as early as the Permian Therapsida (Olson, 1944: 10), and is present in most eutherians, with the primary systematic exception of the cercopithecids (Spoor and Leakey, 1996) and Hominidae. It is also found in most early basal metatherians (Sánchez-Villagra, 2002).

Authors previously raised the possibility that the volume of the subarcuate fossa indicates the agility or locomotor 'habits' of an animal (Olson, 1944:104). Early evaluations were based on visual examination of the depth of the fossa rather than quantitative volumetric measurements, and were based on the assumption that the subarcuate fossa was completely filled by the petrosal lobule (as stated for basal therapsids, Olson, 1944). That assumption often was disputed and tested, particularly in subadult specimens, in which authors contended that the subarcuate fossa is only partially filled with neural tissue and the rest is void or filled with fluid (see McClure and Daron, 1971, Gannon et al., 1988, Sánchez-Villagra, 2002, Jeffery and Spoor, 2006).

### **Petrosal Lobule**

Where present, the subarcuate fossa accommodates cerebellar tissue of the petrosal lobule, a lateral extension of the dorsal paraflocculus. The petrosal lobule passes laterally from the parafloccular lobules by a stem of white myelinated matter through the ostium of the subarcuate fossa (Figure 4.5D). Previous authors disagreed on whether the petrosal lobule is an extension of the dorsal or ventral paraflocculus (Larsell and Jansen,

1970, Glickstein et al., 1994), but the more important question is the afferent signal input for processing and the destination of its efferent signals. Information on the petrosal lobule was difficult to obtain because the dense encapsulating petrosal bone restricts access, so past authors assumed the petrosal lobule receives the same afferent fibers from vestibular nuclei that are sent to other parafloccular lobules (Brodal and Høivik, 1964, Gannon et al., 1988, Sánchez-Villagra, 2002). The petrosal lobule receives efferent fibers from the pontine and primary olivary nuclei, and controls smooth eye tracking movements (Xiong and Nagao, 2002, Hiramatsu et al., 2008, Xiong et al., 2010). Therefore, the location of the petrosal lobule has no relation to direct neural input from the membranous semicircular ducts and is not directly involved with vestibular signal processing.

### **Relative Petrosal Lobule Volume within the Subarcuate Fossa**

When evaluating the size of the petrosal lobule, most authors used visual depth of the subarcuate fossa (Olson, 1944, Sánchez-Villagra, 2002), on the assumption that brain tissue fills the mammalian cranium, as stated by Jerison (1973:50). As a test of that assumption, dissections of fresh adult mammalian specimens indicated that all patent subarcuate fossae were filled with the petrosal lobule, and that the imprint of the lobule gyri were visible on the medial subarcuate fossa when dissected in *Felis catus*, *Canis familiaris*, *Procyon lotor*, *Sus scrofa*, *Rattus norvegicus*, *Mus musculus*, *Cavia porcellus*, *Oryctolagus cuniculus*, *Didelphis virginiana*, and *Phoca vitulina* (Gannon et al., 1988).

Ontogenetic studies of the relationship between the subarcuate fossa and incipient petrosal lobule indicated the enclosure of the fossa by connective tissue before expansion of the petrosal lobule into the fossa (McClure and Daron, 1971, Sánchez-Villagra, 2002, Jeffery and Spoor, 2006). Those authors proposed that primitive dura mater contacting the petrosal lobule also contacts connective tissue in the subarcuate fossa. The connective tissue is resorbed, and the petrosal lobule is able to expand laterally into the subarcuate fossa to achieve full adult size. That growth begins prenatally in eutherians but continues postnatally (Hoyte, 1961, McClure and Daron, 1971). In adult primates without a petrosal lobule (such as Hominidae, Jeffery and Spoor, 2006), the early connective tissue ultimately ossifies.

Fresh specimens were used in the dissections of various adult animals by Gannon et al. (1988), in the ontogenetic rat series (McClure and Daron, 1971), and in an ontogenetic rabbit series (Hoyte, 1961). Another ontogenetic series of *Monodelphis domestica* with additional postnatal specimens of *Caluromys philander* were studied via archived histologic slides by Sanchez-Villagra (2002). He concluded that the petrosal lobule fills the subarcuate fossa sporadically in development but not in adulthood and therefore could not influence the volume of the subarcuate fossa. He also studied preserved adult specimens of *Marmosa* sp., *Cercartetus caudatus*, *Dromiciops gliroides*, and *Monodelphis domestica*. Interestingly, he found that in some species (*Marmosa* sp., *Cercartetus caudatus*, and *Caenolestes fuliginosus*), the petrosal lobule did fill the subarcuate fossa, whereas in other species (*Dromiciops gliroides* and *Monodelphis domestica*), the petrosal lobule only partially filled the subarcuate fossa. He reported that

the petrosal lobule filled approximately half of the subarcuate fossa in adult *Monodelphis domestica*, and that in other taxa large portions of the petrosal lobule remain outside of the subarcuate fossa with no explanation given of what filled the voids within the subarcuate fossa (and even more medial endocranial volumes). In the latter case, the relationship between the size of the subarcuate fossa and the petrosal lobule was not correlated and the subarcuate fossa volume could not be used as a proxy for the volume for the petrosal lobule.

### ***Monodelphis domestica***

*Monodelphis domestica* (Wagner, 1842), the gray short-tailed opossum, was chosen for this study because new sources of information were available to evaluate the conclusion of Sanchez-Villagra (2002) that the petrosal lobule fills only half of the subarcuate fossa in adult specimens. That conclusion contradicts statements by Jerison (1973) that brain tissues completely fill the cranial vaults of adult mammals. Additionally, the ages of laboratory specimens obtained in this study were easily determined through the breeding program records of their laboratory source, and *Monodelphis domestica* is a popular animal model (VandeBerg, 1990, 1999, Filan, 1991, Clark and Smith, 1993, Robinson et al., 1994, VandeBerg and Robinson, 1997, Sánchez-Villagra, 2001, Sánchez-Villagra and Sultan, 2002, Sánchez-Villagra and Wible, 2002, Wible, 2003, Rowe et al., 2005, Macrini et al., 2007a).

In the wild, *Monodelphis domestica* frequents rocky outcrops within its home range, and has a scansorial lifestyle (Macrini, 2004). In the wild, *Monodelphis* prefer traveling on the ground, but are able to climb readily (Nowak, 1999).

## **METHODS**

Biomedical imaging frequently relies on magnetic resonance imaging (MRI) to provide imaging of soft tissues *in vivo*. High resolution MRI (hrMRI, also  $\mu$ MRI) is used to obtain 3D image volumes of small animals, both living and sacrificed (Catana et al. 2008). Unfortunately, bone density registers in black pixels and therefore shows poor resolution on  $\mu$ MRI. High Resolution X-ray Computed Tomography (HRXCT) provides high resolution of bones, but cannot resolve tissue types. The ideal combination for studying the relationship of both brain tissue and bone would be a  $\mu$ MRI scan of an *in vivo* or freshly euthanized head, along with a HRXCT scan of the same head specimen.

The availability of both  $\mu$ MRI and HRXCT scans on four specimens of *Monodelphis domestica* allowed me to revisit the spatial relationship between the subarcuate fossa and the petrosal lobule for this taxon. I used  $\mu$ MRI scans to obtain volumetric images of brain tissue of both *in vivo* and freshly euthanized *Monodelphis domestica* specimens. The petrosal lobule volume determinations can be compared to digital endocasts of the subarcuate fossa imaged from HRXCT scans of these specimens to evaluate the percentage of the subarcuate fossa occupied by the petrosal lobule. A result showing that the petrosal lobule fills the subarcuate fossa would enable a

comparison of endocasts from the subarcuate fossa and the semicircular canals for the possible relationships between the two in *Monodelphis domestica*.

### **High-Resolution MRI**

The Texas Biomedical Research Institute (<http://txbiomed.org/>) shipped six live *Monodelphis domestica* specimens to the Caltech Brain Imaging Center in February 2006. The specimens comprised three retired breeder adults (11-12 month old) and three juveniles (two-month old). Although specific sex information was listed for each specimen when shipped, it was lost during processing at the imaging center. Of the six specimens, two adult specimens did not remain intact through all Caltech protocols.

The Caltech Brain Imaging Center performed  $\mu$ MRI scans on all six of the live *Monodelphis domestica* specimens, when the juvenile specimens were 76 days old. On March 10, 2006, each live animal received isoflurane and oxygen anesthesia through breathing tubes for immobilization during live scans with a Bruker Biospec 7T/30. That procedure followed approved Caltech Brain Imaging protocols Institutional Animal Care and Use Committee (IACUC) protocols as detailed in Procissi et al. (2007). Two method sequences were used for the scans: the T1 weighted 3D spoiled gradient recalled echo sequence (FLASH) and the strongly T2 weighted 3D fast rapid acquisition with relaxation enhancement (RARE). Table 4.1 lists all parameters for live scans of specimens used in the current study as ‘dp01.’ The animals were then euthanized by perfusion with halothane (following MacKenzie-Graham et al. 2004), decapitated, skinned and  $\mu$ MRI scanned with the higher strength 11.7T Bruker/Biospec Vertical

System (with the exception of one adult). Both FLASH and RARE scan sequences were used again, and Table 4.1 reports the scan parameters as ‘possum’ entries recorded in nifti format.

### **Petrosal Lobule Volume Calculations**

The 7T  $\mu$ MRI scans of anesthetized specimens did not provide precise resolution for quantitative measurements of petrosal lobule volume. One example of such a scan for TMM M-9039 (dp01\_3mar0.Aw1\_5) is shown in Figure 4.5C. Although the outlines of the subarcuate fossa and the petrosal lobule are apparent and the subarcuate fossa appears filled by tissue, pixelation does not allow for adequate designation of pixels to the appropriate tissues or structures in three dimensions. Multiple attempts to take volumetric measurements gave results in which the petrosal lobule volumes varied by an order of magnitude for one specimen. The inaccuracy of live scans meant that the soft-tissue scan of the remaining TMM M-9038 was unusable and that specimen was discarded from further analysis.

In contrast, postmortem  $\mu$ MRI scans utilizing the 11.7T magnet provided excellent results. One such scan image for TMM M-9039 (PossumJ3.Bv1\_4) is shown in figure 4.5D. Note the improvement in resolution compared with Figure 4.5C, even though the coronal image slices were the most pixelated of the orthogonal scan views. Meninges with cerebral spinal fluid show clearly as a light boundary to the petrosal lobule, indicating no loss of fluid pressure during specimen preparation. The central myelinated white matter of the lobular peduncle area registers as darker material in the

image slice. No shrinkage of the lobule or receding of tissue into the central cranial cavity is indicated. Volumetric results were easily obtained from such well-resolved postmortem scans.

The  $\mu$ MRI scans were recorded in nifti data format. At the University of Texas that format was translated into ANALYZE format by using LONI 2.0.2 (Neu et al., 2005), and each scan was imported into VG Studio Max 2.1© (Volume Graphics, Heidelberg, Germany) for analysis. To segment the petrosal lobule from  $\mu$ MRI volume reconstructions in VG Studio Max, each scan was imported into the program. Using coronal views of the petrosal lobule and anterior semicircular canal, the slice representing the largest distance between sections of the anterior canal was found, and the medial expression of bone was marked on both superior and inferior sections. This represented the ostium (or *limbus fossa paraflocculus* of Hoyte, 1961), defined as the plane of closure of the subarcuate fossa. In sagittal view, an enclosing boundary segment was defined to prevent selection of tissue material outside the subarcuate fossa. All non-black pixels lateral to that plane were digitally removed as the petrosal lobules for each side of the specimen, representing both the left and the right petrosal lobules. VG Studio Max provided a volume of each segment in  $\text{mm}^3$ , listed in Table 4.3 as  $\mu\text{l}$ .

### **Subarcuate Fossa Volume Calculations**

After all of the soft tissue scans were complete, the four intact specimens at the University of Texas High Resolution X-ray Computed Tomography facility (UTCT) preserved in formaldehyde. At the time of shipping, the effects of preservation for a short

time did not affect further analysis. The preserved specimens consisted of the TMM M-9038 that had not received 11.7T  $\mu$ MRI scanning and the three juveniles. They were scanned by the ACTIS Scanner (Bio-imaging Research, Inc.), with the scan parameters for each listed in Table 4.2. Scan slices were imported into VG Studio Max 2.1 for analysis.

Processed HRXCT (CT) scans indicated that the subarcuate fossae of all specimens were full of tissue and not air, but different tissue types produce similarly uniform gray-scale responses because of preservation in formaldehyde for shipping (Figure 4.5A). Images that were gray-scale maximized for bone density showed bone margins clearly (Figure 4.5A and 4.5B). With such high resolution, determining the location of an ostium for each subarcuate fossa was straightforward, and allowed segmentation of the subarcuate fossa digital cast for each petrosal and all specimens. VGStudioMax calculated subarcuate fossa volumes in  $\text{mm}^3$ . They are reported as  $\mu\text{l}$  in Table 4.3A for comparison to results from other research. Measurements were taken from both left and right sides for the subarcuate fossa of each specimen. The sizes for each subarcuate fossa relate to relative head size of each specimen, as given by the listed skull lengths and widths. TMM M-9041 J2 has the largest skull, as well as the greatest subarcuate fossa volumes. Mean subarcuate fossa volume from both sides of all three specimens is  $5.0 \pm 0.3 \mu\text{l}$ .

## RESULTS

Results of the volumetric totals are listed in Table 4.3A. Because the difference in subarcuate fossa volume between left and right petrosals differed by no more than 0.1  $\mu\text{l}$ , their average was used for the final subarcuate volume for each specimen (listed in Table 4.3A). The volume of the petrosal lobules for each specimen was calculated as the mean value from all available  $\mu\text{MRI}$  scans for each specimen. The percent of the subarcuate fossa filled by the petrosal lobule was calculated as a ratio of the petrosal lobule to that of the subarcuate fossa, and is shown for each scan in Table 4.3B.

The percentage of the subarcuate fossa which was filled by the cerebellar tissue of the petrosal lobule ranged from 87-102% with a mean volume of filling from all measurements of  $95 \pm 2$  percent. These measurements document the complete filling of the subarcuate fossa by cerebellar tissue, when the probable missing volumes of meningeal tissue and fluid are considered.

Obviously the three calculated volumes which exceeded 100% tissue filling of the bony fossa cannot be real. There are two possible reasons for these errors: 1) placement of the ostium margins presents a greater difficulty on  $\mu\text{MRI}$  scans than on CT images, because the resolutions differ significantly and the dark pixels of bony  $\mu\text{MRI}$  margins are more difficult to detect; 2) tolerance levels for tissue selection are more difficult to determine on the  $\mu\text{MRI}$  scans.

Figure 4.4C and 4.4D give the medial and lateral views of the TMM M-9039 left bony labyrinth and subarcuate fossa endocasts. Note in Figure 4.4C the sudden cutoff of the subarcuate fossa at the medial margin of the ostium. Any space between the bony

canals and the subarcuate fossa endocast indicates bone volume. In Figure 4.4D the constriction of the subarcuate fossa within the arc of the posterior semicircular canal is apparent, but the subarcuate fossa does not appear constricted by the lateral semicircular canal. Where not constrained, the subarcuate fossa increases in volume above the canal heights and through the posterior canal. A small peak in the subarcuate endocast is visible in Figure 4.4D that represents a small blood vessel foramen in the lateral petrosal wall, a common occurrence among mammals (Jeffery et al., 2008).

## **DISCUSSION**

### **Discrepancy of Present Results with Those of Previous Studies**

The calculated volume results conclusively demonstrate that cerebellar tissue of the petrosal lobule fills the space available within the subarcuate fossa of 76-day-old specimens of *Monodelphis domestica*, in contrast with the conclusions of the only previous measurements on this species by Sánchez-Villagra (2002). The results are consistent with the qualitative observations of dissections from fresh adult specimens of *Didelphis virginiana* (Gannon et al., 1988), a related didelphid. These results also demonstrate that although use of preserved specimens (especially those processed as histologic stained thin sections) are invaluable for tissue-type identification within the brain (see Sánchez-Villagra, 2001, Sánchez-Villagra and Sultan, 2002 for tissue identifications utilizing histology for *Monodelphis domestica*), their use for volume estimates of brain structures are subject to numerous drawbacks. These disadvantages are

well-known in the medical community: volumetric shrinkage of cells and organs subjected to preservatives (Loqman et al., 2010), specimen slice dehydration and rehydration during preparation, distortion of tissues during histologic preparation, loss of cerebral spinal fluids during any specific histologic preparation technique (Ross and Pawlina, 2011), and apparent tissue volume appearance due to histologic slice orientation (Ross and Pawlina, 2011). Some of these effects are apparent in the histologic images provided by Sánchez-Villagra (2002); his figures 1F and 2B definitely show shrinkage of the petrosal lobule (labeled plp) into the medial cranium, as well as missing tissue from the dura mater, arachnoid, and pia mater. Several descriptions of tissues filling the subarcuate fossa may in fact be describing dura mater, a loose connective tissue (Ross and Pawlina, 2011), with osteological precursors (see McClure and Daron, 1971, Gannon et al., 1988, Sánchez-Villagra, 2002, Jeffery and Spoor, 2006).

The inaccurate conclusions from volumetric calculations in previous studies of the petrosal lobule by use of preserved specimens are similar to a successive series of studies on the encephalization quotient of the koala (*Phascolarctos cinereus*). In an often-cited article, Haight and Nelson (1987) manually measured the cranial volume of koala skulls, and compared the results to volumes acquired from preserved koala brains. The authors concluded that the brain occupied only 61% of the endocranial volume, an unprecedentedly low percentage for a mammal. Subsequently, freshly dead specimens were studied by De Miguel and Henneberg (1997) who obtained results indicating that 74-76% of the cranial cavity was occupied by brain tissue. A recent measurements of live and frozen koala specimen brain volume versus cranial capacity utilized MRI scans, and

yielded calculations that the brain filled 87% and 80% of the cranial volume (Taylor et al., 2006). Those values that are more consistent with brain to cranium ratios of other mammals. Meninges and blood vessels filled the rest of the cavity. The authors of that study pointed out the importance of using live or fresh specimens for volumetric studies of tissues, and recommended using MRI scanning for such studies in more cases. My findings are in excellent accord with their conclusion that MRI scans allowed for rapid and more accurate estimates of cranial volumes occupied by brain tissue.

### **The Petrosal Lobule as Represented within the Subarcuate Fossa**

The petrosal lobule is difficult to dissect through the petrosal, and difficult to access for neural studies. As such, it often was designated as an indeterminate part of the floccular system, paraflocculus ('parafloccular lobe'), or as an indistinguishable extension of the dorsal paraflocculus, ventral paraflocculus, or both. For examples of the many designations of the petrosal lobule, see the cerebellar diagrams from various authors for differing species (numerous labeled illustrations were provided by Larsell and Jansen, 1970). Each of the distinct floccular and parafloccular lobules receives secondary efferent input from the vestibular nuclei, but the petrosal lobule receives input from the primary olivary nucleus and uvular nucleus, which are primarily involved with image processing. Because each of the structures mentioned has a different part to play in the processing of vestibular and visual information (e.g., the involvement of the dorsal paraflocculus in coordination of reaching coordination, Kralj-Hans et al., 2007), distinguishing which cerebellar lobule fills the subarcuate fossa helps in understanding

what part the endocast of the subarcuate fossa plays in agility processing. The most recent experiments on primates indicate that it plays a primary role in smooth eye movement control during pursuit (Hiramatsu et al., 2008).

### **Semicircular Canals versus Subarcuate Fossa as Indicators of Agility**

The size of the subarcuate fossa was previously suggested as an indicator of locomotor agility (Olson, 1944, Spoor and Leakey, 1996, Jeffery and Spoor, 2006), whereas the relationship of the semicircular canal sizes and agility is well-demonstrated (Gray, 1907, 1908, Jones and Spells, 1963, ten Kate et al., 1970, Oman and Marcus, 1980, Oman et al., 1987, Spoor and Leakey, 1996, Rabbitt, 1999, Spoor, 2003, Hullar, 2006, Ifediba et al., 2007, Ekdale, 2009, Kandel and Hullar, 2010, Malinzak et al., 2011). Given their developmental proximity, the canals of the fossa could dominate the development of agility abilities and influence the adult size of the other. Two recent papers detail attempts to address which of these affects the final size of the other in primates (through the growth of the accessory paraflocculus, the homologous structure to the petrosal lobule in primates).

Ontogenetically, an analysis of *Homo sapiens*, *Macaca nemestrina*, and *Alouatta caraya* utilized hrMRI scans of preserved specimens and maturation quotients rather than actual prenatal ages (Jeffery and Spoor, 2006). The prenatal subarcuate fossa and semicircular canals are present up to a point in *Homo sapiens*, after which the fossa fills with cartilage and then bone. *Macaca nemestrina*, and *Alouatta caraya* retain the fossa into adulthood. The prenatal subarcuate fossa and semicircular canals were identified

separately; subarcuate fossa width and height were defined by the width of the bony/cartilaginous opening parallel to the width and height of the anterior semicircular canal, whereas the depth of the subarcuate fossa was taken by a line perpendicular to the anterior semicircular canal (and lying in the same direction as the heights of the lateral semicircular and posterior semicircular canals). The semicircular measurements are the same as those defined by Spoor and Zonneveld (1995). Volumetric measurements of the subarcuate fossa also were collected. From their analysis the authors concluded that there was a correlation between the growth of the subarcuate fossa and the semicircular canals with the strongest correlation existing between the dimensions of the anterior semicircular canal and the subarcuate fossa. Further, they found that the subarcuate fossa and the semicircular canals do not grow independently, but that development of the semicircular canals and the accessory paraflocculus could not definitively be separated in *Macaca nemestrina* and *Alouatta caraya*. However, they were able to conclude that the semicircular canals are not affected by the embryological fossa growth in *Homo sapiens*, due to the cessation of fossa development before ossification of the semicircular canals.

One drawback to their study was that Jeffery and Spoor (2006) did not differentiate between cartilaginous and ossified features. This presents a difficulty in determining at which point the semicircular canals have reached their maximum size, which occurs well before the subarcuate fossa completes its development in non-primate mammals. Another difficulty is the shrinkage of preserved brain tissue, which the authors acknowledged. Such shrinkage influences the interpretation of a cerebellar feature within developing fossa. Correlations between linear dimensions of the subarcuate fossa and

semicircular canals are to be expected, given their spatial relationships, and the authors essentially confirmed those correlations without assigning causative factors to one or the other.

A follow-up study of numerous adult primate species by Jeffery et al. (2008) presented more questionable conclusions, because all linear parameters of the subarcuate fossa are directly defined by those of the semicircular canals without exclusion of bony margins (Jeffery et al., 2008). The conclusions that some anterior and posterior semicircular canal dimensions were correlated to subarcuate fossa size and that development of the features were connected appear circular.

### **Developmental Considerations of the Subarcuate Fossa and Semicircular Canals**

The osteological ontogeny of the subarcuate fossa for a species has rarely been described on a cellular level (but see the studies of humans by Bast and Anson, 1949). A series of rabbits injected intraperitoneally at birth with alizarin red and then sacrificed at various days postnatally enabled a developmental bone study of deposition and reabsorption in the *pars canalicularis* (Hoyte, 1961). The bony cavities of the entire rabbit labyrinth were fully grown at birth and did not change in size postnatally, with the exception of an enlargement of semicircular canal ampullae. The subarcuate fossa ossified postnatally and then grew circumferentially through the reabsorption of interior bone and accretion of bone on the petrosal exterior. In the process, the volume of the subarcuate fossa increased but the three static semicircular bony canals appeared as increasingly apparent ridges on the interior of the subarcuate fossa. Unfortunately, Hoyte

did not describe the volumes of the petrosal lobule or the contacts of the petrosal lobule to the subarcuate fossa. That description of the internal appearance of the fossa in rabbits is consistent with the appearance of indentations on the endocast of *Monodelphis domestica* specimens (Figure 4.4C and 4.4D).

Recent work by two research groups utilizing the same postnatal *Monodelphis domestica* growth series helps indicate the developmental petrosal growth pattern for that species. Using the specimens listed in Table 4.4, Ekdale (2010) found that the bony semicircular canals reached their full size and ossified by postnatal day 27. In a developmental study of brain endocasts, Macrini et al. (2007a) provided measurements of the subarcuate fossa endocast volumes from both petrosals added together in their Table 5. The poor ossification of the *pars canalicularis* at 27 days postnatal can be seen in Figure 4.7; only the bony labyrinth is easily identified. Figure 4.7A shows an example of the CT data with which Macrini et al. (2007a) worked; and Figure 4.7B shows the area designated as brain tissue in that scan image. Table 4.4 shows the volume measurements of subarcuate fossa endocasts from both skull sides found in their study. That table also includes specimens from my study, with the volumes summed for the right and left subarcuate, and the previously reported fossa volumes converted to  $\mu\text{l}$  for consistency. Results are plotted in Figure 4.8, and indicate that although the sizes of semicircular canal are determined by 27 days postnatally, the subarcuate fossa grows rapidly until at least 90 days postnatally. That finding coincides with the observation that brain growth as a whole for *Monodelphis domestica* extends across 90 days (Ulinsky, 1971, Rowe, 1996a, b). There is a small negative allometric relationship between the volume of the subarcuate

fossa and that of the entire brain (Macrini et al., 2007a), a finding consistent with that of Sánchez-Villagra (2002). Therefore, although there is a structural relationship between the semicircular canals and the subarcuate fossa, the subarcuate fossa grows separately and is not restricted in volume by the semicircular canals.

## **FUTURE WORK**

The question remains: is there a relationship between the size of the subarcuate fossa and the agility of the animal? Both Gannon et al. (1988) and Jeffery et al. (2008) considered the volume of the petrosal lobule versus the volume of the entire brain as an indicator for the relative importance of agility to processing within the central nervous system.

Given that the petrosal lobule was found to occupy the entire subarcuate fossa in at least two basal marsupials, a study of the volume of the subarcuate fossa vs. brain size for species of marsupials of varying locomotor abilities could elucidate the remaining question of whether the skeletal subarcuate fossa is an indicator of locomotor agility. Because marsupial brains are considered more primitive than those of placental species (Larsell and Jansen, 1970, Ashwell, 2010), a separate study of that group would be advised before combining them with placental studies.

**TABLES**

**Table 4.1.**  $\mu$ MRI scan parameters. Scan studies beginning with ‘dp01’ are from specimens of *Monodelphis domestica* while still living, and were acquired with a 7T  $\mu$ MRI Bruker Biospec. Scan studies beginning with ‘possum’ were acquired with a Bruker Biospec Vertical System. **FLASH**, fast low angle shot, T<sub>1</sub> weighted images at low flip angles; **FOV**, field of view in cm; **RARE**, Rapid Acquisition with Relaxation Enhancement, T<sub>2</sub> weighted at 90° flip angle; **TE**, Excitement time in msec; **TR**, Relaxation time in msec.

$\mu$ MRI Study	Protocol	Flip Angle	Weight	TE	TR	FOV	Acquisition Matrix	Final Matrix
<i>TMM M-9040</i> Juvenile 1								
dp01_3mar0.Au1_13	RARE 3D	90	T2	11.34	2000.00	3.5 x 2.8 x 2.4	320 x 128 x 108	256 x 128 x 128
possumj1.B51_3	RARE 3D	90	T2	22.59	2000.00	3.6 x 2.0 x 1.4	720 x 200 x 148	512 x 256 x 256
possumj1.B51_4	FLASH 3D	20	T1	6.00	80.00	3.6 x 2.0 x 1.4	720 x 200 x 148	512 x 256 x 256
possumj1.B51_5	RARE 3D	90	T2	22.59	2000.00	3.8 x 2.0 x 1.5	760 x 200 x 148	512 x 256 x 256
possumj1.B51_6	FLASH 3D	20	T1	8.00	80.00	3.8 x 2.0 x 1.5	760 x 200 x 148	380 x 200 x 148
<i>TMM M-9041</i> Juvenile 2								
dp01_3mar0.Av1_7	RARE 3D	90	T2	11.34	2000.00	2.5 x 2.4 x 2.2	132 x 128 x 116	256 x 128 x 128
possumj2.Bt1_3	RARE 3D	90	T2	12.00	1000.00	4.0 x 1.9 x 2.2	512 x 128 x 128	256 x 128 x 128
possumj2.Bt1_4	RARE 3D	90	T2	12.10	1000.00	4.0 x 1.9 x 2.2	816 x 192 x 222	408 x 192 x 222
possumj2.Bt1_5	FLASH 3D	30	T1	6.00	100.00	4.0 x 1.9 x 2.2	816 x 192 x 222	408 x 192 x 222
<i>TMM M-9039</i> Juvenile 3								
dp01_3mar0.Aw1_5	RARE 3D	90	T2	11.34	2000.00	2.3 x 2.5 x 2.1	232 x 128 x 106	128 x 128 x 128
possumj3.Bv1_3	RARE 3D	90	T2	12.00	1000.00	3.6 x 1.8 x 2.2	736 x 184 x 224	368 x 184 x 224
possumj3.Bv1_4	FLASH 3D	30	T1	6.00	100.00	3.6 x 1.8 x 2.2	816 x 192 x 222	512 x 256 x 256
<i>TMM M-9038</i> Adult								
dp01_3mar0.At1_11	FLASH 3D	25	T1	5.00	180.00	3.5 x 2.8 x 2.2	430 x 172 x 136	256 x 256 x 256

<b>μMRI Study</b>	<b>Protocol</b>	<b>Flip Angle</b>	<b>Weight</b>	<b>TE</b>	<b>TR</b>	<b>FOV</b>	<b>Acquisition Matrix</b>	<b>Final Matrix</b>
dp01_3mar0.At1_12	RARE 3D	90	T2	11.34	2000.00	3.5 x 2.8 x 2.2	312 x 128 x 100	256 x 128 x 128
dp01_3mar0.At1_14	RARE 3D	90	T2	11.34	2000.00	3.5 x 2.8 x 2.2	312 x 128 x 100	256 x 128 x 128

**Table 4.2.** HRXCT scan parameters for study specimens. **Number of slices**, final slice number after processing. The slices are usually taken in the coronal plane, so those imported images represent the original planar reconstruction. **Slice Thickness**, measured in mm; **Interslice spacing**, the distance between each final recorded slice, reported as mm; **FR**, Field of reconstruction, indicating width of an individual slice; **File Size**, size of both width and height of individual image slice, reported in pixels.

<i>M. domestica</i> Specimen #	Number of Slices	Slice Thickness	Interslice Spacing	FR	File Size
<i>TMM M-9040</i> J1	853	0.04517	0.04517	20	1024
<i>TMM M-9041</i> J2	900	0.04517	0.04517	21	1024
<i>TMM M-9039</i> J3	885	0.04517	0.04517	21	1024
<i>TMM M-9038</i> Ad	825	0.06255	0.06255	28	1024

**Table 4.3.** Summary of  $\mu$ MRI and HRXCT scan results from specimens of *Monodelphis domestica* used in this study. **LASC**, left anterior semicircular canal; **LLSC**, left lateral semicircular canal; **LPL**, left petrosal lobule; **LPSC**, left posterior semicircular canal; **LSF**, left subarcuate fossa; **R**, radius; **RASC**, right anterior semicircular canal; **RLSC**, right lateral semicircular canal; **RPL**, right petrosal lobule; **RPSC**, right posterior semicircular canal; **RSF**, right subarcuate fossa.

Specimen #	TMM M-9040 J1	TMM M-9041 J2	TMM M-9039 J3	TMM M-9038 Ad
<b>Age (days)</b>	76	76	76	350
<b>skull length (mm)</b>	33.55	39.95	34.33	42.49
<b>skull width (mm)</b>	18.40	19.16	18.62	24.05
<b>LASC R (mm)</b>	1.15	1.13	1.17	1.02
<b>LLSC R (mm)</b>	0.81	0.80	0.84	0.72
<b>LPSC R (mm)</b>	0.82	0.89	0.82	0.77
<b>RASC R (mm)</b>	1.12	1.12	1.11	1.08
<b>RLSC R (mm)</b>	0.76	0.76	0.85	0.73
<b>RPSC R (mm)</b>	0.81	0.86	0.85	0.79
<b>left ostium radius (mm)</b>	0.81	0.86	0.85	0.69
<b>right ostium radius (mm)</b>	0.79	0.79	0.76	0.73
<b>LSF (<math>\mu</math>l)</b>	4.9	5.3	4.7	3.5
<b>RSF (<math>\mu</math>l)</b>	5.0	5.4	4.7	3.8
$\mu$ MRI scan 1	PossumJ1.B51_4	PossumJ2.Bt1_3	PossumJ3.Bv1_3	
<b>LPL (<math>\mu</math>l)</b>	4.4	5.4	4.5	
<b>RPL (<math>\mu</math>l)</b>	4.3	5.5	4.5	
$\mu$ MRI scan 2	PossumJ1.B51_3	PossumJ2.Bt1_4	PossumJ3.Bv1_4	
<b>LPL (<math>\mu</math>l)</b>	4.7	5.2	4.3	

Specimen #	TMM M-9040 J1	TMM M-9041 J2	TMM M-9039 J3	TMM M-9038 Ad
<b>RPL (μl)</b>	4.5	5.1	4.8	
μMRI scan 3	PossumJ1.B51_5	PossumJ2.Bt1_5		
<b>LPL (μl)</b>	4.7	4.7		
<b>RPL (μl)</b>	4.8	4.9		
μMRI scan 4	PossumJ1.B51_6			
<b>LPL (μl)</b>	4.8			
<b>RPL (μl)</b>	4.8			
<b>Petrosal Lobules Used for Mean Volumes</b>	n = 8	n = 6	n = 4	n = 0

**Table 4.4.** Percent of subarcuate fossa filled by the petrosal lobe calculated from Table 4.3A scans.

<b>Scan Count</b>	<b>TMM M-9040 J1</b>	<b>TMM M-9041 J2</b>	<b>TMM M-9039 J3</b>	<b>Summary</b>
<b>Scan 1</b> LLP/LSF	89.93	101.38	96.35	
<b>Scan 1</b> RLP/RSF	86.81	100.81	96.07	
<b>Scan 2</b> LLP/LSF	94.65	98.26	91.82	
<b>Scan 2</b> RLP/RSF	90.37	94.18	102.29	
<b>Scan 3</b> LLP/LSF	95.30	88.00		
<b>Scan 3</b> RLP/RSF	96.22	89.72		
<b>Scan 4</b> LLP/LSF	96.62			
<b>Scan 4</b> RLP/RSF	97.04			
<b>Mean %</b>	93.37	95.39	96.63	<b>95.13</b>
<b>Standard Error %</b>	± 3.81	± 5.60	± 4.30	<b>± 1.65</b>

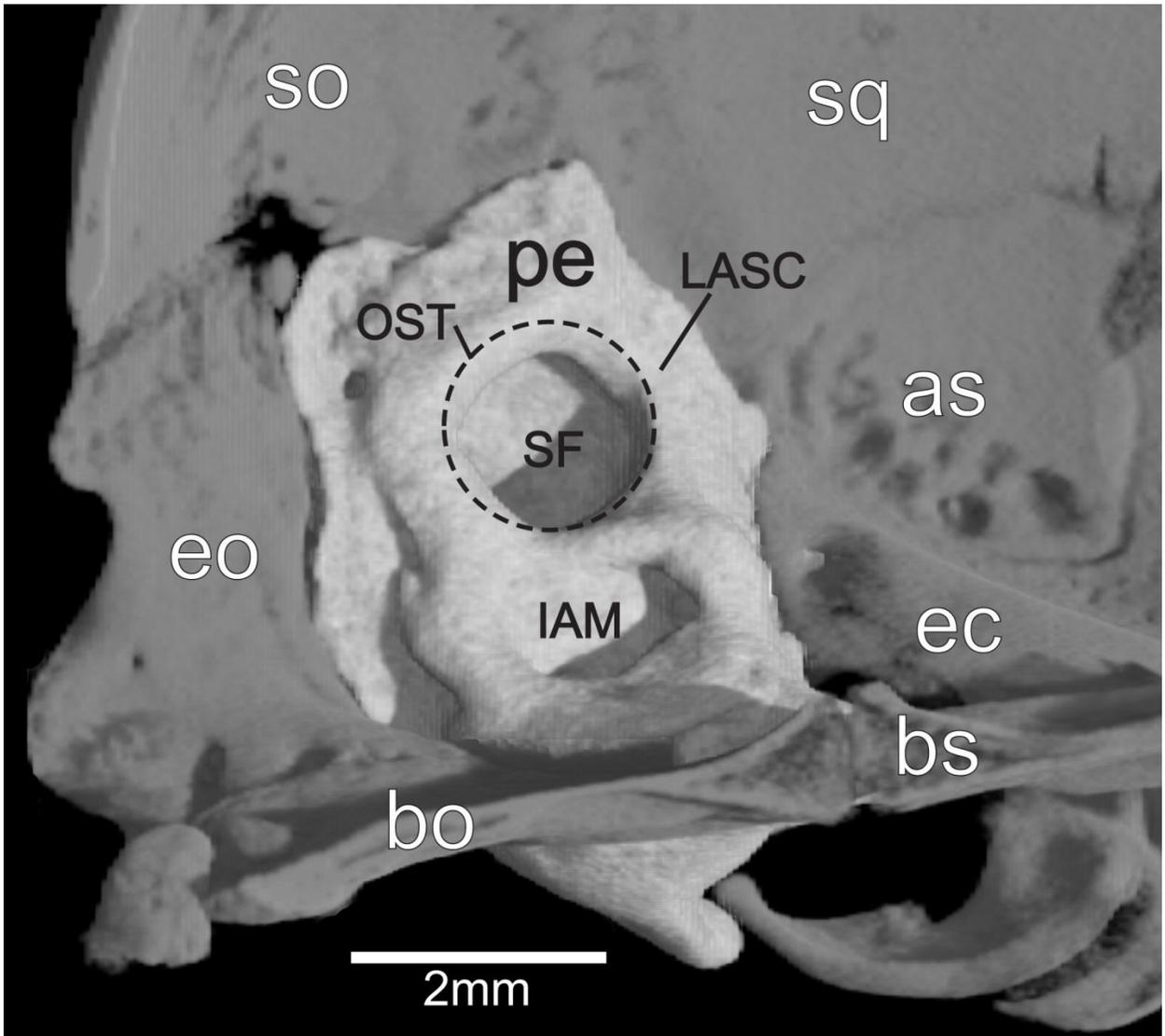
**Table 4.5.** Summary of ontogenetic specimens of *Monodelphis domestica*, ages, and volumes of petrosal lobules. Data from Macrini et al. (2007a); **a**, includes volume of both right and left petrosals; **b**, from this study.

Specimen	Sex	Age (days)	Petrosal Lobule ( $\mu$ l) <sup>a</sup>
TMM M-7595	?	27	3.4
TMM M-8265	?	27	3.2
TMM M-8261	?	27	4.1
TMM M-7536	F	48	7.3
TMM M-8269	F	48	7.8
TMM M-8266	F	56	8.8
TMM M-7539	F	57	7.4
TMM M-7542	M	75	9.7
TMM M-9040 <sup>b</sup>	?	76	9.9
TMM M-9041 <sup>b</sup>	?	76	10.7
TMM M-9039 <sup>b</sup>	?	76	9.4
TMM M-8267	M	76	10.9
TMM M-7545	F	90	9.5
TMM M-8268	M	90	12.1
TMM M-8273	M	456	11.3
TMM M-8271	F	837	11.5
TMM M-7599	F	Adult	10.5

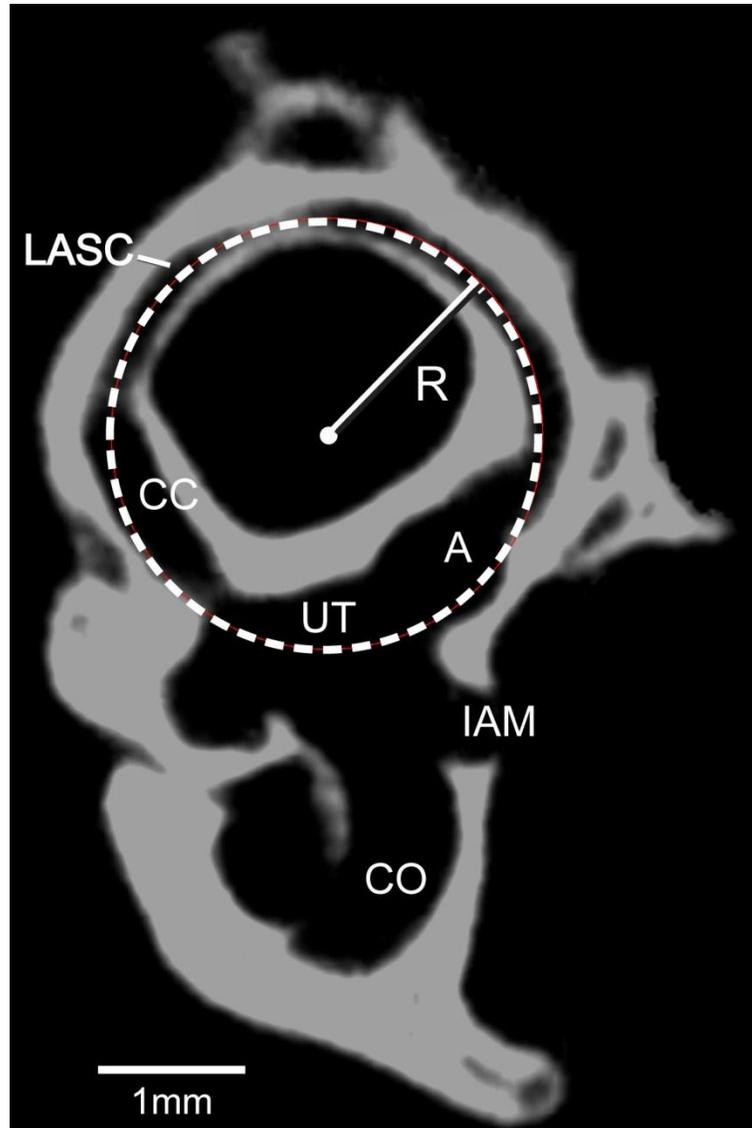
**FIGURES**



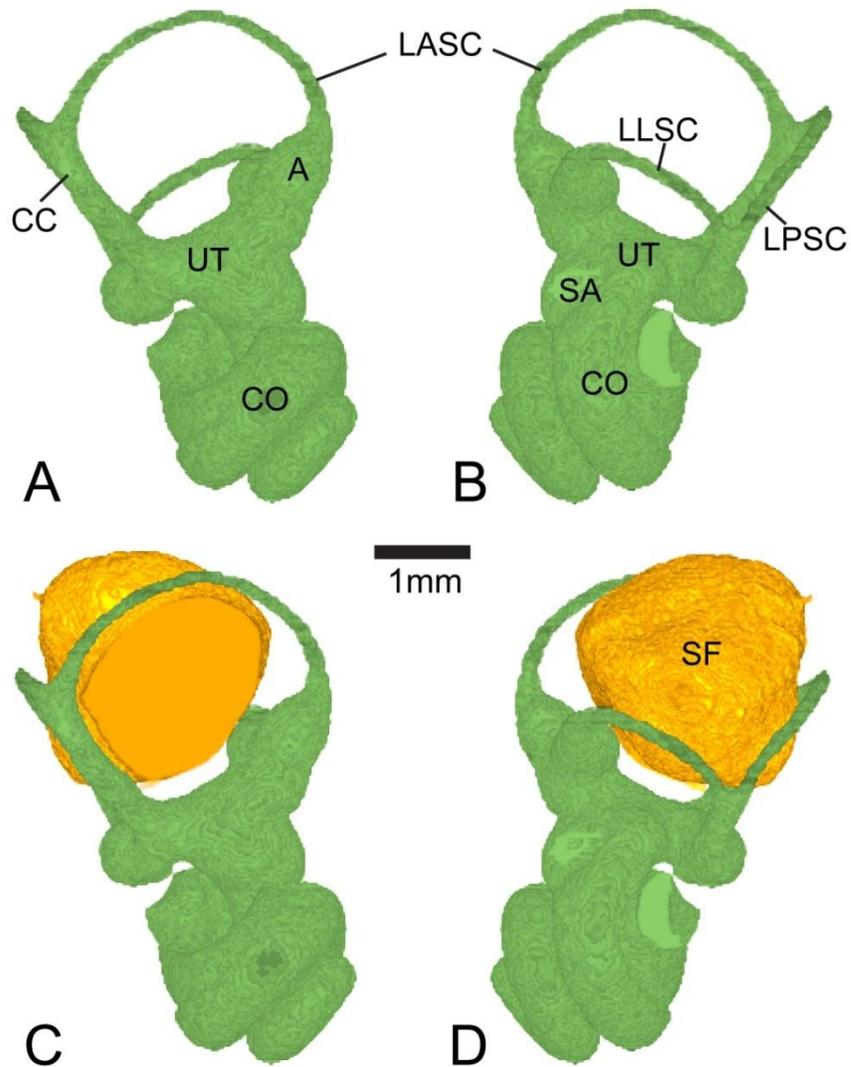
**Figure 4.1.** TMM M-9039. Juvenile *Monodelphis domestica*, aged 76 days. HRXCT digital image 3D reconstruction, left lateral view with petrosal highlighted in red.



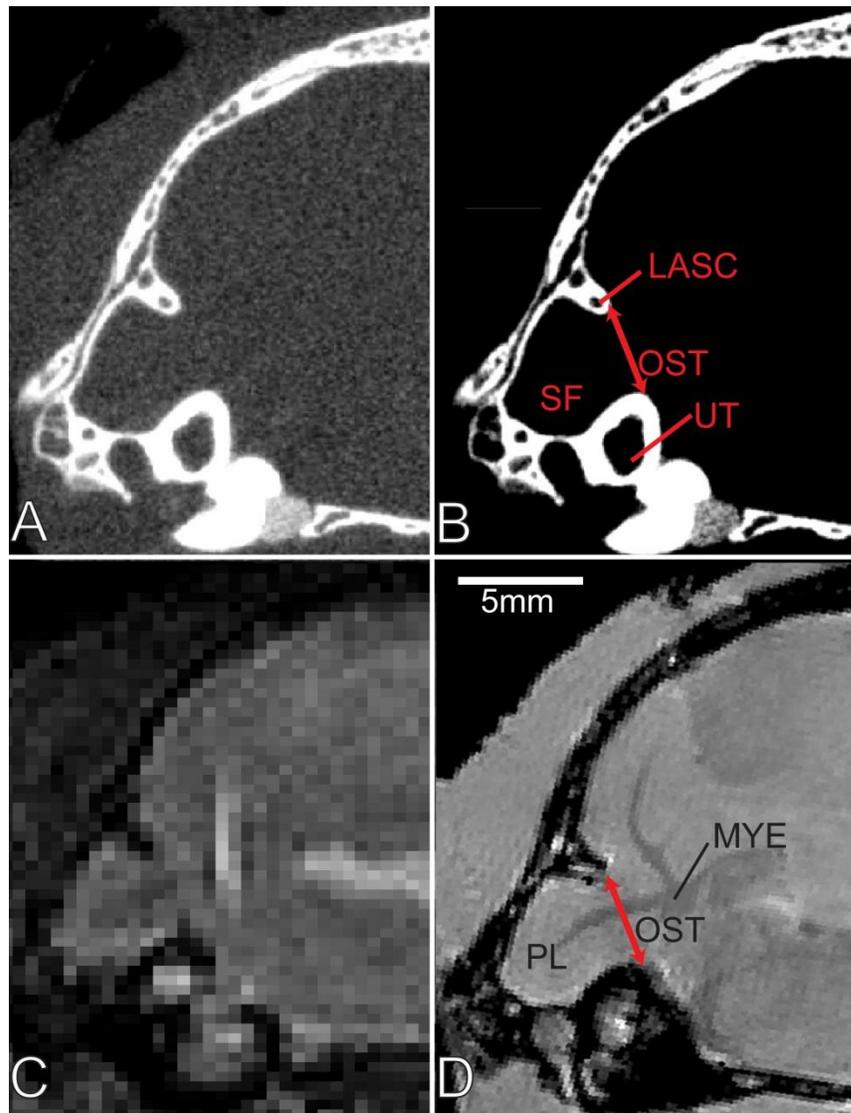
**Figure 4.2.** Left medial view of *Monodelphis domestica* (TMM M-9039) basicranium from 3D digital HRXCT reconstruction. **as**, alisphenoid; **bo**, basioccipital; **bs**, basisphenoid; **ec**, ectotympanic; **eo**, exoccipital; **IAM**, internal acoustic meatus; **LASC**, left anterior semicircular canal; **OST**, ostium; **pe**, petrosal; **SF**, subarcuate fossa; **so**, supraoccipital; **sq**, squamosal.



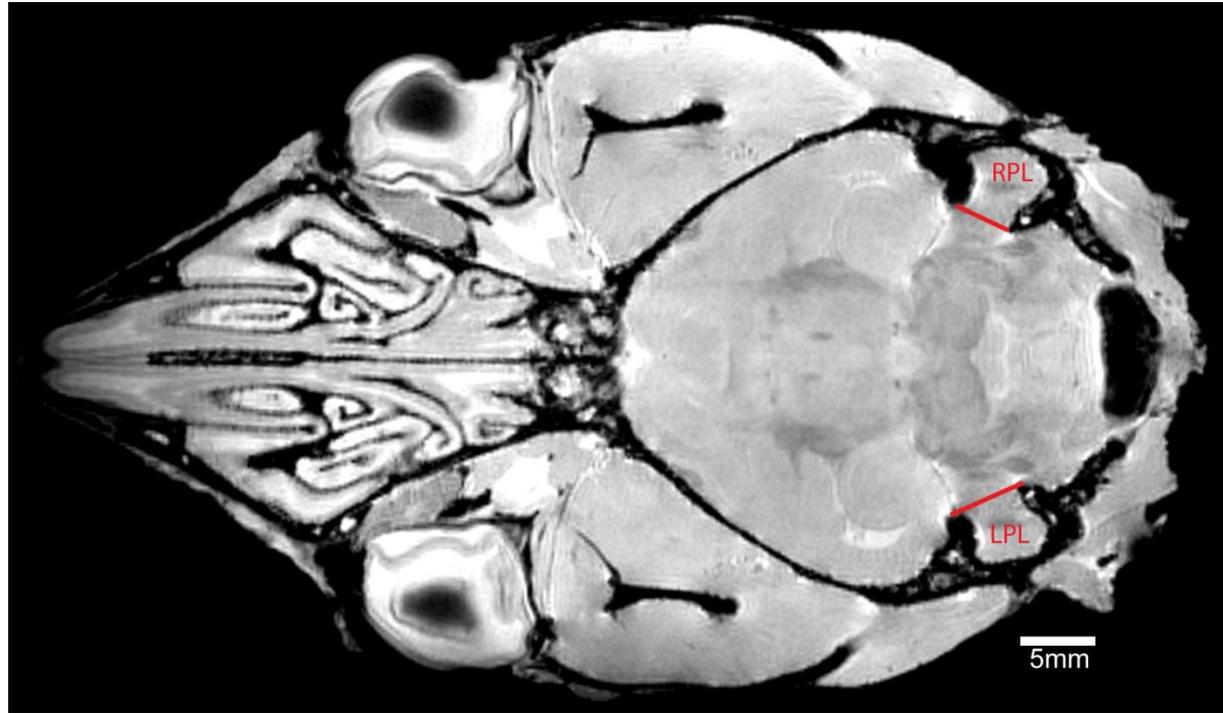
**Figure 4.3.** *Monodelphis domestica* TMM M-9039 cutaway view of 3D HRXCT image reconstruction showing the plane of the left anterior semicircular canal. **A**, ampulla; **CC**, common crus; **CO**, cochlea; **IAM**, interior acoustic meatus; **LASC**, left anterior semicircular canal; **R**, arc radius of curvature calculated for the left anterior semicircular canal; **UT**, utricle.



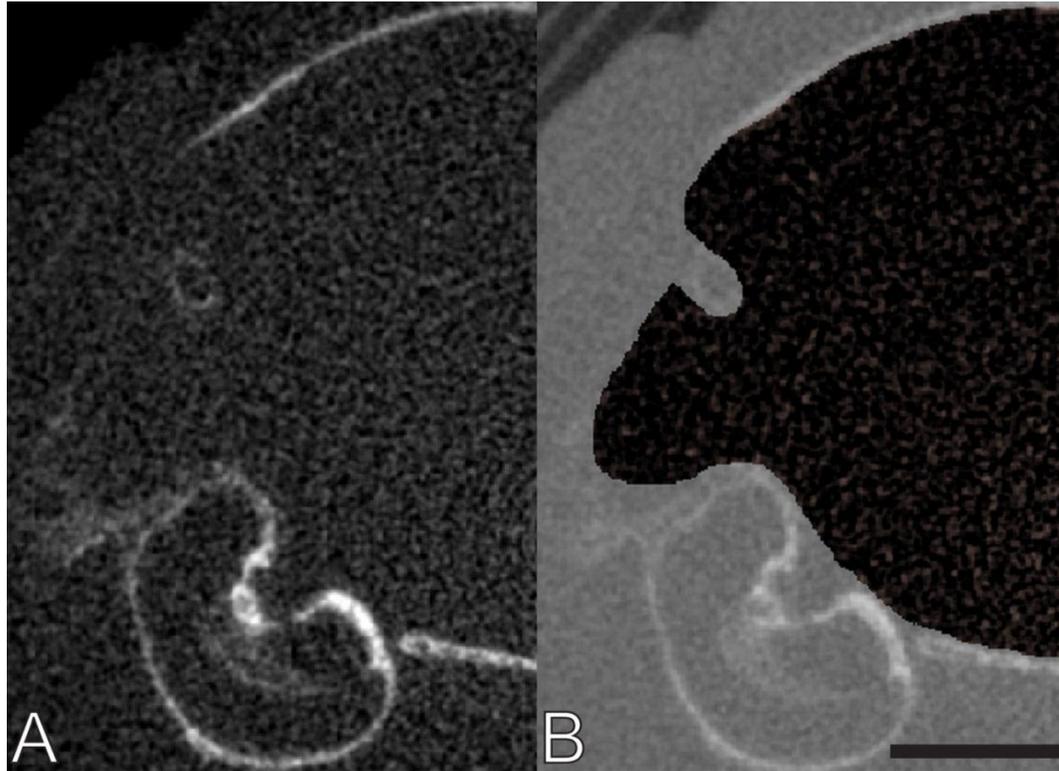
**Figure 4.4.** Endocasts of the bony semicircular canals (green) and subarcuate fossa (orange) from *Monodelphis domestica* TMM M-9039 HRXCT scan. **A**, medial view of left bony labyrinth; **B**, lateral view of left bony labyrinth; **C**, medial view of left bony labyrinth and subarcuate fossa endocast; **D**, lateral view of left bony labyrinth and subarcuate fossa endocast. **A**, ampulla; **CC**, common crus; **CO**, cochlea; **LASC**, left anterior semicircular canal; **LLSC**, left lateral semicircular canal; **LPSC**, left posterior semicircular canal; **SA**, saccule; **SF**, subarcuate fossa; **UT**, utricle.



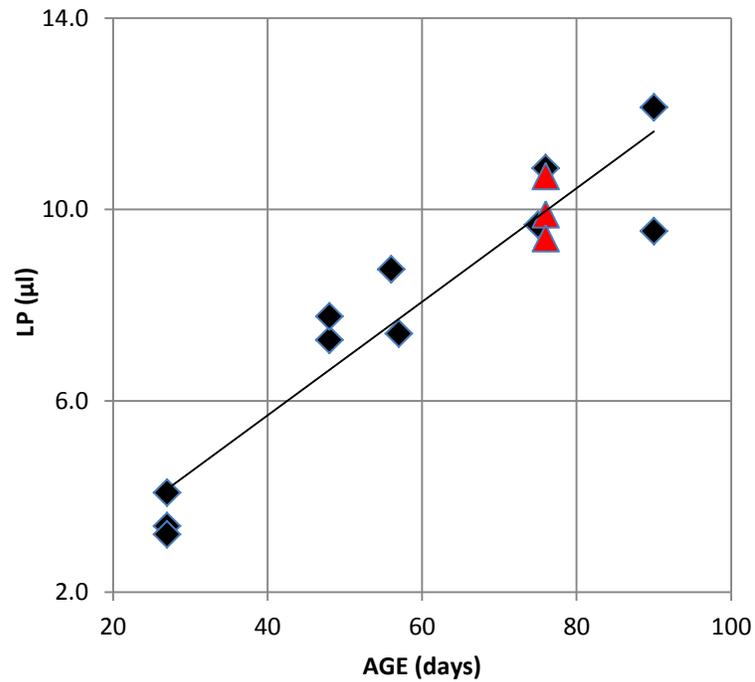
**Figure 4.5.** Coronal images of *Monodelphis domestica* TMM M-9039 taken at the maximum height of the left anterior semicircular canal. **A** and **B**, HRXCT scans showing images with **(A)** and without **(B)** soft tissue. **C**,  $\mu$ MRI scan image of live specimen with 7T magnet. **D** postmortem  $\mu$ MR scan with 11.7T magnet. **LASC**, left anterior semicircular canal; **PL**, petrosal lobule; **MYE**, myelinated tissue (darker than surrounding tissue); **OST**, ostium (with red line to show length and endpoints); **SF**, subarcuate fossa; **UT**, utricle.



**Figure 4.6.** Transverse image of *Monodelphis domestica* (TMM M-9039)  $\mu$ MRI postmortem scan taken through greatest width of subarcuate fossa. **LPL**, left petrosal lobule; **RPL**, right petrosal lobule; **red line**, ostium.



**Figure 4.7.** *Monodelphis domestica* TMM M-8265 Day 27 HRXCT coronal sections. **A**, low amount of ossification except for petrosal; and **B**, endocast area of petrosal lobule (as black area in interior scan image). Scan slices courtesy of Ted Macrini, data from these scans was summarized by Macrini et al. (2007a).



**Figure 4.8.** Table 4.4 *Monodelphis domestica* ontogenetic series subarcuate fossa volume plotted against specimen age. Regression line:  $y = 0.1187x + 0.9514$ ,  $r^2 = 0.8958$ . Adults were excluded from this graph; however the subarcuate fossa values were close to the oldest specimens plotted, showing that subarcuate fossa growth was nearly complete at about 90 days. Diamonds are taken from the data presented by Macrini et al. (2007a); triangles are the specimens used in my study.

## Appendices

**APPENDIX 1. COORDINATES AND VECTORS FOR SEMICIRCULAR CANALS. REFERENCE PLANES DEFINED IN CHAPTER 2 METHODS IN A HEAD-CENTERED REFERENCE SYSTEM. THESE VALUES WERE UTILIZED IN BOTH CHAPTER 2 AND CHAPTER 3 ANALYSES.**

Taxon	Canal	Canal Radius				Canal Plane		
		Radius	Center Coordinates			Vector Coordinates		
			X	Y	Z	X	Y	Z
<i>Acrobates</i>								
	LASC	0.94	-0.95	3.27	2.05	-0.69	0.72	-0.08
	LLSC	0.78	-1.35	3.62	1.11	-0.48	-0.04	0.88
	LPSC	0.95	-1.82	3.03	1.1	-0.69	-0.69	-0.23
	RASC	0.91	-0.77	-3.42	2.06	0.69	0.72	0.07
	RLSC	0.8	-1.13	-3.66	1.16	0.33	-0.02	-0.94
	RPSC	0.89	-1.67	-3.18	1.18	0.71	-0.67	0.20
<i>Allactaga</i>								
	LASC	2.33	-2.3	6.64	1.07	-0.27	0.71	-0.65
	LLSC	1.61	-2.73	8.02	-0.96	-0.72	-0.07	0.69
	LPSC	1.74	-3.85	6.52	-1.07	-0.58	-0.56	-0.59
	RASC	2.39	-2.27	-6.42	0.98	0.28	0.71	0.65
	RLSC	1.62	-2.75	-7.96	-1.05	0.72	-0.11	-0.68
	RPSC	1.75	-3.92	-6.41	-1.16	0.58	-0.59	0.57
<i>Anomalurus</i>								
	LASC	2.06	-2.74	7.43	3.9	-0.50	0.79	-0.37
	LLSC	1.83	-2.98	8.19	2.22	-0.53	-0.02	0.85
	LPSC	1.57	-4.78	6.96	2.29	-0.43	-0.79	-0.43
	RASC	2.02	-2.47	-6.81	3.99	0.53	0.77	0.36
	RLSC	1.92	-2.74	-7.39	2.29	0.51	0.00	-0.86
	RPSC	1.53	-4.59	-6.34	2.35	0.45	-0.78	0.43
<i>Caluromys</i>								
	LASC	1.43	-2.94	7.23	2.43	-0.66	0.70	-0.26
	LLSC	1.17	-3.28	7.67	0.86	-0.37	-0.19	0.91
	LPSC	1.07	-4.39	6.83	1.38	-0.51	-0.78	-0.36
	RASC	1.4	-2.95	-7.13	2.45	0.63	0.73	0.25
	RLSC	1.14	-3.31	-7.64	0.82	0.32	-0.27	-0.91
	RPSC	1.02	-4.38	-6.76	1.33	0.53	-0.76	0.36
<i>Cercartetus</i>								
	LASC	1.13	-1.87	4.41	2	-0.66	0.74	-0.15
	LLSC	0.76	-2.3	4.77	1.06	-0.28	-0.19	0.94

Taxon	Canal	Canal Radius				Canal Plane		
		Center Coordinates				Vector Coordinates		
		Radius	X	Y	Z	X	Y	Z
	LPSC	0.81	-2.99	4.17	1.37	-0.66	-0.69	-0.29
	RASC	1.08	-1.82	-4.28	2.1	0.68	0.72	0.16
	RLSC	0.76	-2.24	-4.61	1.11	0.27	-0.25	-0.93
	RPSC	0.77	-2.94	-4.01	1.38	0.70	-0.66	0.28
<i>Chironectes</i>								
	LASC	1.09	-3.74	7.45	1.8	-0.53	0.82	-0.21
	LLSC	1.4	-3.31	6.93	3.36	-0.35	-0.16	0.92
	LPSC	1.15	-5	6.82	2.43	-0.55	-0.75	-0.38
	RASC	1.49	-2.68	-6.65	3.37	0.51	0.84	0.20
	RLSC	1.18	-3.16	-7.17	1.8	0.24	-0.17	-0.96
	RPSC	1.14	-4.49	-6.54	2.48	0.72	-0.60	0.35
<i>Chrysochloris</i>								
	LASC	0.97	-1.56	3.52	3.17	-0.68	0.66	-0.32
	LLSC	0.65	-1.67	3.92	2.01	0.20	-0.11	0.97
	LPSC	0.63	-2.26	3.36	2.57	-0.68	-0.71	-0.18
	RASC	1.11	-1.59	-3.65	3.03	0.73	0.61	0.32
	RLSC	0.71	-1.8	-3.85	1.98	0.02	-0.02	-1.00
	RPSC	0.64	-2.41	-3.25	2.42	0.71	-0.71	0.05
<i>Crocota</i>								
	LASC	3.18	-5.25	21.94	10.88	-0.65	0.68	-0.34
	LLSC	2.6	-6.08	24.15	7.31	-0.35	-0.10	0.93
	LPSC	3.03	-8.94	21.8	8.49	-0.45	-0.78	-0.42
	RASC	3.34	-4.31	-24.06	11.13	0.57	0.75	0.34
	RLSC	2.52	-5.38	-26.34	7.46	0.42	-0.11	-0.90
	RPSC	3.18	-8.34	-24.22	8.72	0.60	-0.71	0.37
<i>Dactylopsila</i>								
	LASC	1.74	-2.91	7.73	0.99	-0.54	0.66	-0.53
	LLSC	1.16	-3.38	8.43	-0.64	-0.57	0.00	0.82
	LPSC	1.44	-4.3	7.39	-0.39	-0.49	-0.75	-0.44
	RASC	1.68	-3.05	-7.64	1.09	0.51	0.76	0.40
	RLSC	1.19	-3.54	-8.2	-0.6	0.49	0.08	-0.87
	RPSC	1.4	-4.58	-7.2	-0.29	0.59	-0.72	0.38
<i>Dromiciops</i>								
	LASC	1.05	-1.68	4.32	1.82	-0.72	0.67	-0.17
	LLSC	0.77	-1.77	4.52	0.86	-0.39	-0.12	0.91
	LPSC	0.78	-2.63	4.07	1.02	-0.58	-0.71	-0.39
	RASC	1.07	-1.85	-4.28	1.91	0.63	0.76	0.14

Taxon	Canal	Canal Radius				Canal Plane		
		Center Coordinates				Vector Coordinates		
		Radius	X	Y	Z	X	Y	Z
<i>Enhydra</i>	RLSC	0.71	-2.02	-4.6	0.92	0.36	-0.16	-0.92
	RPSC	0.72	-2.82	-4.02	1.08	0.51	-0.75	0.42
	LASC	2.19	-8.67	21.12	3.69	-0.66	0.74	-0.12
	LLSC	2.3	-9.5	23.17	1.19	-0.23	-0.07	0.97
	LPSC	2.35	-11.55	21.38	2.13	-0.60	-0.77	-0.23
	RASC	2.25	-7.32	-22.62	4.77	0.59	0.77	0.23
	RLSC	2.24	-8.09	-24.51	1.9	0.37	0.02	-0.93
	RPSC	2.33	-10.24	-22.88	3.05	0.61	-0.78	0.10
<i>Glaucomys</i>	LASC	1.66	-2.53	4.95	3.38	-0.63	0.69	-0.36
	LLSC	1.44	-2.8	5.8	1.94	-0.50	-0.09	0.86
	LPSC	1.32	-4.06	4.72	2.23	-0.51	-0.72	-0.47
	RASC	1.66	-2.4	-5.3	3.47	0.64	0.72	0.28
	RLSC	1.4	-2.69	-6.36	2.01	0.42	-0.21	-0.88
	RPSC	1.26	-4.02	-5.07	2.33	0.61	-0.65	0.45
	<i>Hemibelideus</i>	LASC	2.35	-4.28	7.05	3.76	-0.43	0.79
LLSC		1.68	-4.81	7.79	1.14	-0.53	0.05	0.85
LPSC		2.22	-6.48	6.89	1.55	-0.53	-0.80	-0.30
RASC		2.29	-4.28	-6.93	4.17	0.50	0.74	0.45
RLSC		1.64	-4.87	-7.56	1.34	0.49	-0.08	-0.87
RPSC		2.33	-6.41	-6.59	1.79	0.52	-0.81	0.28
<i>Heterocephalus</i>		LASC	1.01	-0.44	3.6	0.86	-0.62	0.73
	LLSC	0.79	-0.88	4.01	0.23	-0.32	-0.16	0.93
	LPSC	0.8	-1.46	3.36	0.09	-0.51	-0.75	-0.42
	RASC	1	-0.54	-3.57	0.82	0.64	0.71	0.29
	RLSC	0.78	-0.92	-4.01	0.1	0.30	-0.19	-0.93
	RPSC	0.85	-1.47	-3.31	-0.01	0.55	-0.70	0.46
<i>Lepus</i>	LASC	2.61	-0.25	7.77	-0.2	-0.60	0.70	-0.39
	LLSC	1.8	-0.68	8.78	-2.87	-0.49	0.01	0.87
	LPSC	1.71	-2.08	7.24	-2.69	-0.60	-0.75	-0.28
	RASC	2.61	0.01	-8.08	0.08	0.55	0.75	0.36
	RLSC	1.79	-0.35	-9.07	-2.56	0.50	-0.01	-0.86
	RPSC	1.66	-1.8	-7.55	-2.36	0.61	-0.73	0.31

Taxon	Canal	Canal Radius				Canal Plane		
		Center Coordinates				Vector Coordinates		
		Radius	X	Y	Z	X	Y	Z
<i>Monodelphis</i>								
	LASC	1.17	-0.45	4.43	1.52	-0.57	0.81	-0.13
	LLSC	0.84	-0.74	4.81	0.43	-0.28	-0.17	0.95
	LPSC	0.82	-1.59	4.34	0.98	-0.67	-0.66	-0.33
	RASC	1.11	-0.29	-4.34	1.6	0.56	0.82	0.11
	RLSC	0.8	-0.64	-4.73	0.41	0.27	-0.25	-0.93
	RPSC	0.85	-1.45	-4.24	1	0.63	-0.69	0.36
<i>Notoryctes</i>								
	LASC	0.8	-2.04	3.81	2.48	-0.47	0.84	-0.28
	LLSC	0.76	-2.33	4.23	1.46	-0.12	0.03	0.99
	LPSC	0.79	-3.02	3.66	1.9	-0.44	-0.90	-0.08
	RASC	0.85	-2.02	-3.97	2.64	0.54	0.82	0.18
	RLSC	0.7	-2.29	-4.56	1.43	0.13	-0.33	-0.93
	RPSC	0.82	-2.95	-3.81	1.99	0.57	-0.79	0.23
<i>Pedetes</i>								
	LASC	2.79	-2.98	9.46	2.66	-0.56	0.74	-0.38
	LLSC	2.19	-3.46	10.82	0.41	-0.66	0.00	0.75
	LPSC	2.01	-5.41	8.83	0.04	-0.53	-0.72	-0.45
	RASC	2.71	-3.27	-9.59	2.53	0.53	0.74	0.41
	RLSC	2.13	-3.58	-11.06	0.15	0.67	-0.04	-0.74
	RPSC	1.99	-5.46	-9.09	-0.15	0.51	-0.71	0.49
<i>Petauroides</i>								
	LASC	1.97	-2.64	7.53	2.27	-0.62	0.66	-0.42
	LLSC	1.48	-3.53	8.2	0.36	-0.55	-0.19	0.82
	LPSC	1.67	-4.44	7.15	0.5	-0.41	-0.84	-0.36
	RASC	1.94	-2.91	-7.5	1.95	0.60	0.67	0.44
	RLSC	1.6	-3.4	-8.08	0.02	0.53	-0.20	-0.83
	RPSC	1.73	-4.65	-7.17	0.15	0.47	-0.78	0.41
<i>Petaurus</i>								
	LASC	1.37	-2.16	5.81	1.57	-0.49	0.71	-0.50
	LLSC	1.1	-2.53	6.32	0.31	-0.49	0.04	0.87
	LPSC	1.23	-3.3	5.63	0.16	-0.56	-0.69	-0.46
	RASC	1.42	-2.09	-5.7	1.55	0.55	0.74	0.39
	RLSC	1.1	-2.49	-6.17	0.38	0.47	0.02	-0.88
	RPSC	1.15	-3.19	-5.45	0.21	0.51	-0.69	0.51
<i>Petropseudes</i>								
	LASC	1.91	-3.44	8.83	2.78	-0.37	0.70	-0.61

Taxon	Canal	Canal Radius				Canal Plane		
		Center Coordinates				Vector Coordinates		
		Radius	X	Y	Z	X	Y	Z
	LLSC	1.66	-3.93	9.82	0.91	-0.66	-0.02	0.75
	LPSC	1.72	-5.02	8.65	0.54	-0.50	-0.69	-0.53
	RASC	1.89	-3.55	-8.13	2.19	0.55	0.73	0.41
	RLSC	1.54	-4.19	-9	0.15	0.64	-0.04	-0.76
	RPSC	1.72	-5.23	-7.83	0.08	0.37	-0.85	0.38
<i>Potorous</i>								
	LASC	2.2	-0.88	9.92	3.36	-0.68	0.67	-0.31
	LLSC	1.5	-1.59	10.62	1.09	-0.46	-0.01	0.89
	LPSC	1.82	-2.79	9.64	1.55	-0.62	-0.73	-0.30
	RASC	2.02	-0.81	-9.59	3.65	0.68	0.68	0.28
	RLSC	1.53	-1.49	-10.1	1.34	0.52	-0.04	-0.86
	RPSC	1.85	-2.74	-9.2	1.71	0.61	-0.70	0.37
<i>Pseudocheirus</i>								
	LASC	2.08	-3.97	7.51	4.01	-0.47	0.85	-0.23
	LLSC	1.55	-4.71	8.59	2.12	-0.35	-0.04	0.94
	LPSC	1.7	-6.11	7.63	2.4	-0.74	-0.58	-0.35
	RASC	2.12	-3.73	-7.55	3.93	0.44	0.86	0.26
	RLSC	1.48	-4.29	-8.66	1.83	0.41	-0.05	-0.91
	RPSC	1.69	-5.87	-7.68	2.35	0.69	-0.63	0.35
<i>Pseudochirops</i>								
	LASC	2.01	-3.19	8.5	2.25	-0.52	0.75	-0.40
	LLSC	1.34	-3.5	9.68	0.12	-0.51	0.17	0.84
	LPSC	1.7	-4.83	8.41	0.32	-0.60	-0.57	-0.56
	RASC	2.05	-3.16	-8.34	1.99	0.51	0.76	0.40
	RLSC	1.48	-3.54	-9.3	0.04	0.60	-0.03	-0.80
	RPSC	1.74	-4.9	-8.29	0.08	0.67	-0.61	0.43
<i>Pseudochirulus</i>								
	LASC	1.82	-3.67	5.95	2.87	-0.53	0.80	-0.29
	LLSC	1.23	-4.35	6.61	1.14	-0.47	0.05	0.88
	LPSC	1.47	-5.43	5.85	1.28	-0.49	-0.78	-0.38
	RASC	1.88	-3.32	-6.12	2.96	0.50	0.80	0.34
	RLSC	1.3	-3.91	-6.78	1.17	0.47	0.08	-0.88
	RPSC	1.48	-5.08	-6.02	1.31	0.49	-0.74	0.45
<i>Sciurus</i>								
	LASC	2.55	-2.42	9.22	4.85	-0.60	0.70	-0.39
	LLSC	2.24	-3.02	10.26	1.81	-0.37	-0.07	0.92
	LPSC	2.14	-5.2	8.65	2.9	-0.66	-0.68	-0.33

Taxon	Canal	Canal Radius				Canal Plane		
		Center Coordinates				Vector Coordinates		
		Radius	X	Y	Z	X	Y	Z
<i>Talpa</i>	RASC	2.6	-2.38	-9.09	4.98	0.65	0.67	0.36
	RLSC	2.22	-3.04	-10.13	1.89	0.38	-0.03	-0.92
	RPSC	2.11	-5.05	-8.52	3.06	0.67	-0.66	0.34
	LASC	1.2	-5.27	3.94	1.73	-0.57	0.76	-0.33
	LLSC	1.01	-4.99	4.44	0.36	-0.27	-0.09	0.96
	LPSC	1	-6.03	3.72	0.63	-0.52	-0.83	-0.22
	RASC	1.29	-5.1	-4.16	1.72	0.57	0.76	0.32
	RLSC	1.08	-4.82	-4.5	0.38	0.26	-0.05	-0.96
	RPSC	0.94	-5.97	-3.85	0.7	0.62	-0.67	0.40
<i>Tarsipes</i>	LASC	0.83	-0.66	2.82	1.91	-0.65	0.76	0.00
	LLSC	0.59	-1.17	2.81	1.34	-0.19	0.00	0.98
	LPSC	0.61	-1.78	2.46	1.72	-0.57	-0.81	-0.15
	RASC	0.8	-0.54	-2.96	1.9	0.63	0.76	0.15
	RLSC	0.62	-0.99	-3.01	1.24	0.09	-0.14	-0.99
	RPSC	0.61	-1.68	-2.71	1.7	0.64	-0.75	0.17
	<i>Thylacinus</i>	LASC	2.57	-0.77	17.16	6.43	-0.78	0.56
LLSC	1.73	-1.39	18.13	3.58	-0.33	-0.11	0.94	
LPSC	1.96	-3.19	16.61	4.68	-0.67	-0.70	-0.23	
RASC	2.39	-0.42	-15.85	6.79	0.66	0.69	0.29	
RLSC	1.53	-1.03	-17.26	3.76	0.33	-0.12	-0.94	
RPSC	2.04	-2.78	-15.44	4.81	0.64	-0.72	0.26	
<i>Vulpes</i>	LASC	2.46	-0.74	13.85	9.5	-0.63	0.75	-0.20
	LLSC	2.23	-2	15.54	6.98	-0.22	0.02	0.98
	LPSC	2.18	-3.72	13.97	8.36	-0.76	-0.58	-0.29
	RASC	2.45	-0.92	-13.4	9.6	0.63	0.74	0.22
	RLSC	2.17	-2.14	-15.25	6.97	0.22	0.07	-0.97
	RPSC	2.19	-3.93	-13.36	8.25	0.65	-0.68	0.32
	<i>Wallabia</i>	LASC	3.12	-0.31	13.82	2.63	-0.58	0.73
LLSC	2.15	-1.18	15.04	-0.38	-0.56	-0.15	0.82	
LPSC	2.71	-2.98	13.7	-0.32	-0.54	-0.65	-0.54	
RASC	2.93	-0.36	-14.65	3.18	0.60	0.76	0.24	
RLSC	2.15	-1	-15.71	-0.11	0.55	-0.18	-0.81	

Taxon	Canal	Canal Radius				Canal Plane		
		Radius	X	Y	Z	X	Y	Z
	RPSC	2.73	-3.02	-14.45	0.09	0.52	-0.66	0.54

**APPENDIX 2. COMPARISON OF  $R_R$  AND  $R_{HW}$ .**

144

Taxon				Height Width Method (mm)									Regression Method (mm)		
	BM (g)	log <sub>10</sub> (BM)	AG	LASC		LLSC		LPSC		ASCR <sub>hw</sub>	LSCR <sub>hw</sub>	PSCR <sub>h</sub>	LASC R <sub>r</sub>	LLSC R <sub>r</sub>	LPSC R <sub>r</sub>
				h	w	h	w	h	w						
<i>Acrobates</i>	12.2	1.09	6	1.25	1.79	1.50	1.54	1.50	2.01	0.76	0.76	0.88	0.94	0.78	0.95
<i>Allactaga</i>	350	2.54	6	3.07	4.62	3.33	2.90	3.02	3.34	1.92	1.56	1.59	2.33	1.61	1.74
<i>Anomalurus</i>	389	2.59	6	3.05	4.04	2.31	3.36	2.41	2.82	1.77	1.42	1.31	2.06	1.83	1.57
<i>Caluromys</i>	256.7	2.41	5	2.38	2.92	1.60	2.16	1.84	1.99	1.33	0.94	0.96	1.43	1.17	1.07
<i>Cercartetus</i>	23	1.36	6	1.84	2.07	1.44	1.42	1.34	1.31	0.98	0.72	0.66	1.13	0.76	0.81
<i>Chironectes</i>	745	2.87	5	1.58	1.99	2.46	2.71	2.24	2.22	0.89	1.29	1.12	1.09	1.4	1.15
<i>Chrysochloris</i>	39.8	1.60	2	2.20	1.71	1.37	1.32	1.32	1.15	0.98	0.67	0.62	0.97	0.65	0.63
<i>Crocota</i>	64230	4.81	3	6.17	6.57	5.34	4.67	6.18	5.74	3.19	2.50	2.98	3.18	2.6	3.03
<i>Dactylopsila</i>	374	2.57	6	2.90	3.15	2.40	2.31	2.71	2.66	1.51	1.18	1.34	1.74	1.16	1.44
<i>Dromiciops</i>	22.3	1.35	3	1.86	2.05	1.26	1.24	1.31	1.30	0.98	0.63	0.65	1.05	0.77	0.78
<i>Enhydra</i>	34500	4.54	5	4.46	3.94	4.82	4.00	3.90	4.37	2.10	2.21	2.07	2.19	2.3	2.35
<i>Glaucomys</i>	63.8	1.80	6	2.35	3.11	2.51	2.10	2.35	2.44	1.37	1.15	1.20	1.66	1.44	1.32
<i>Hemibelideus</i>	966	2.98	2	3.96	4.56	3.12	3.14	4.27	4.40	2.13	1.57	2.17	2.35	1.68	2.22
<i>Heterocephalus</i>	59.5	1.77	2	1.11	1.81	1.26	1.30	1.28	1.47	0.73	0.64	0.69	1.01	0.79	0.8
<i>Lepus</i>	2350	3.37	5	4.27	4.99	3.14	3.49	2.82	3.48	2.32	1.66	1.58	2.61	1.8	1.71
<i>Monodelphis</i>	900	2.95	2	1.98	2.20	1.40	1.43	1.63	1.45	1.05	0.71	0.77	1.17	0.84	0.82
<i>Notoryctes</i>	55	1.74	2	1.62	1.49	1.32	1.30	1.26	1.18	0.78	0.66	0.61	0.8	0.76	0.79
<i>Pedetes</i>	3007	3.48	6	4.23	5.49	3.90	4.13	3.40	3.79	2.43	2.01	1.80	2.79	2.19	2.01
<i>Petauroides</i>	1300	3.11	6	2.70	3.74	2.70	3.15	2.90	3.45	1.61	1.46	1.59	1.97	1.48	1.67
<i>Petaurus</i>	115	2.06	6	2.18	2.85	2.06	2.21	2.31	2.26	1.26	1.07	1.14	1.37	1.1	1.23



Taxon	BM (g)	$\Delta$ ASCR	$\Delta$ LSCR	$\Delta$ PSCR	% ASC	% LSC	% PSC	Mean $\Delta$ SCR	Log10 (R)					
									ASC R <sub>hw</sub>	LSC R <sub>hw</sub>	PSC R <sub>hw</sub>	LASC R <sub>r</sub>	LLSC R <sub>r</sub>	LPSC R <sub>r</sub>
<i>Acrobates</i>	12.2	0.18	0.02	0.07	21.18	2.60	7.93	0.09	-0.12	-0.12	-0.06	-0.03	-0.11	-0.02
<i>Allactaga</i>	350	0.41	0.05	0.15	19.17	3.31	9.01	0.20	0.28	0.19	0.20	0.37	0.21	0.24
<i>Anomalurus</i>	389	0.29	0.41	0.26	15.00	25.40	18.25	0.32	0.25	0.15	0.12	0.31	0.26	0.20
<i>Caluromys</i>	256.7	0.11	0.23	0.11	7.62	21.80	11.10	0.15	0.12	-0.03	-0.02	0.16	0.07	0.03
<i>Cercartetus</i>	23	0.15	0.05	0.15	14.47	6.10	20.03	0.12	-0.01	-0.15	-0.18	0.05	-0.12	-0.09
<i>Chironectes</i>	745	0.20	0.11	0.03	19.92	7.99	3.09	0.11	-0.05	0.11	0.05	0.04	0.15	0.06
<i>Chrysochloris</i>	39.8	-0.01	-0.02	0.01	0.77	3.40	2.00	-0.01	-0.01	-0.17	-0.21	-0.01	-0.19	-0.20
<i>Crocota</i>	64230	0.00	0.10	0.05	0.16	3.82	1.66	0.05	0.50	0.40	0.47	0.50	0.41	0.48
<i>Dactylopsila</i>	374	0.23	-0.02	0.10	13.99	1.50	7.01	0.10	0.18	0.07	0.13	0.24	0.06	0.16
<i>Dromiciops</i>	22.3	0.07	0.15	0.13	7.15	20.79	17.80	0.12	-0.01	-0.20	-0.19	0.02	-0.11	-0.11
<i>Enhydra</i>	34500	0.09	0.09	0.28	4.20	4.22	12.79	0.16	0.32	0.34	0.32	0.34	0.36	0.37
<i>Glaucomys</i>	63.8	0.30	0.29	0.12	19.50	22.18	9.73	0.24	0.14	0.06	0.08	0.22	0.16	0.12
<i>Hemibelideus</i>	966	0.22	0.12	0.05	9.82	7.09	2.39	0.13	0.33	0.19	0.34	0.37	0.23	0.35
<i>Heterocephalus</i>	59.5	0.28	0.15	0.11	32.18	20.98	15.13	0.18	-0.14	-0.19	-0.16	0.00	-0.10	-0.10
<i>Lepus</i>	2350	0.30	0.14	0.14	11.98	8.24	8.22	0.19	0.36	0.22	0.20	0.42	0.26	0.23
<i>Monodelphis</i>	900	0.13	0.13	0.05	11.29	17.12	6.29	0.10	0.02	-0.15	-0.11	0.07	-0.08	-0.09
<i>Notoryctes</i>	55	0.02	0.11	0.18	2.85	14.84	25.71	0.10	-0.11	-0.18	-0.21	-0.10	-0.12	-0.10
<i>Pedetes</i>	3007	0.36	0.18	0.21	13.79	8.70	11.16	0.25	0.39	0.30	0.25	0.45	0.34	0.30
<i>Petauroides</i>	1300	0.36	0.02	0.08	20.11	1.19	5.07	0.15	0.21	0.17	0.20	0.29	0.17	0.22
<i>Petaurus</i>	115	0.11	0.03	0.09	8.56	3.00	7.38	0.08	0.10	0.03	0.06	0.14	0.04	0.09
<i>Petropseudes</i>	1640	0.09	0.06	0.11	4.96	3.99	6.45	0.09	0.26	0.20	0.21	0.28	0.22	0.24
<i>Potorous</i>	1078	0.26	0.11	0.13	12.56	7.25	7.41	0.17	0.29	0.14	0.23	0.34	0.18	0.26
<i>Pseudocheirus</i>	781	0.25	0.05	0.09	12.65	3.61	5.59	0.13	0.26	0.17	0.21	0.32	0.19	0.23

Taxon	BM (g)	$\Delta$ ASCR	$\Delta$ LSCR	$\Delta$ PSCR	% ASC	% LSC	% PSC	Mean $\Delta$ SCR	Log10 (R)					
									ASC R <sub>hw</sub>	LSC R <sub>hw</sub>	PSC R <sub>hw</sub>	LASC R <sub>r</sub>	LLSC R <sub>r</sub>	LPSC R <sub>r</sub>
<i>Pseudochirops</i>	1725	0.15	-0.05	0.04	7.89	3.48	2.53	0.05	0.27	0.14	0.22	0.30	0.13	0.23
<i>Pseudochirulus</i>	649	0.12	0.04	0.07	6.82	3.10	5.06	0.08	0.23	0.08	0.15	0.26	0.09	0.17
<i>Sciurus</i>	764	0.11	0.24	-0.01	4.41	11.32	0.58	0.11	0.39	0.30	0.33	0.41	0.35	0.33
<i>Talpa</i>	83.7	-0.15	-0.03	0.23	11.39	2.93	25.67	0.02	0.13	0.02	-0.11	0.08	0.00	0.00
<i>Tarsipes</i>	9	0.12	0.04	0.04	15.58	7.02	5.91	0.07	-0.15	-0.26	-0.24	-0.08	-0.23	-0.21
<i>Thylacinus</i>	22000	0.17	-0.02	-0.05	6.95	1.01	2.39	0.04	0.38	0.24	0.30	0.41	0.24	0.29
<i>Vulpes</i>	4041	0.13	0.17	-0.05	5.43	8.05	2.27	0.08	0.37	0.31	0.35	0.39	0.35	0.34
<i>Wallabia</i>	13767	0.20	0.07	0.15	6.71	3.19	5.50	0.14	0.47	0.32	0.41	0.49	0.33	0.43
Mean					11.26	8.36	8.75							
Standard Deviation					6.99	7.22	6.77							
Coefficient of Variation					62.07	86.38	77.37							
95% Confidence Interval					10.01	7.06	7.53							
					12.52	9.66	9.96							

**APPENDIX 3. IPSILATERAL CANAL RADIUS DIFFERENCES.**

Taxon	log <sub>10</sub> BM	AG	Radius (mm)						Side differences (mm)			Side difference percent ((Δ SCR / (LSCR + RSCR))*100)		
			LASCR <sub>r</sub>	LLSCR <sub>r</sub>	LPSCR <sub>r</sub>	RASCR <sub>r</sub>	RLSCR <sub>r</sub>	RPSCR <sub>r</sub>	Δ ASCR <sub>r</sub>	Δ LSCR <sub>r</sub>	Δ PSCR <sub>r</sub>	Δ ASCR <sub>r</sub>	Δ LSCR <sub>r</sub>	Δ PSCR <sub>r</sub>
<i>Acrobates</i>	1.09	6	0.94	0.78	0.95	0.91	0.8	0.89	0.03	-0.02	0.06	3.24	2.53	6.52
<i>Allactaga</i>	2.54	6	2.33	1.61	1.74	2.39	1.62	1.75	-0.06	-0.01	-0.01	2.54	0.62	0.57
<i>Anomalurus</i>	2.59	6	2.06	1.83	1.57	2.02	1.92	1.53	0.04	-0.09	0.04	1.96	4.80	2.58
<i>Caluromys</i>	2.41	5	1.43	1.17	1.07	1.4	1.14	1.02	0.03	0.03	0.05	2.12	2.60	4.78
<i>Cercartetus</i>	1.36	6	1.13	0.76	0.81	1.08	0.76	0.77	0.05	0.00	0.04	4.52	0.00	5.06
<i>Chironectes</i>	2.87	5	1.09	1.4	1.15	1.49	1.18	1.14	-0.40	0.22	0.01	31.01	17.05	0.87
<i>Chrysochloris</i>	1.60	2	0.97	0.65	0.63	1.11	0.71	0.64	-0.14	-0.06	-0.01	13.46	8.82	1.57
<i>Crocota</i>	4.81	3	3.18	2.6	3.03	3.34	2.52	3.18	-0.16	0.08	-0.15	4.91	3.13	4.83
<i>Dactylopsila</i>	2.57	6	1.74	1.16	1.44	1.68	1.19	1.4	0.06	-0.03	0.04	3.51	2.55	2.82
<i>Dromiciops</i>	1.35	3	1.05	0.77	0.78	1.07	0.71	0.72	-0.02	0.06	0.06	1.89	8.11	8.00
<i>Enhydra</i>	4.54	5	2.19	2.3	2.35	2.25	2.24	2.33	-0.06	0.06	0.02	2.70	2.64	0.85
<i>Glaucomys</i>	1.80	6	1.66	1.44	1.32	1.66	1.4	1.26	0.00	0.04	0.06	0.00	2.82	4.65
<i>Hemibelideus</i>	2.98	2	2.35	1.68	2.22	2.29	1.64	2.33	0.06	0.04	-0.11	2.59	2.41	4.84
<i>Heterocephalus</i>	1.77	2	1.01	0.79	0.8	1	0.78	0.85	0.01	0.01	-0.05	1.00	1.27	6.06
<i>Lepus</i>	3.37	5	2.61	1.8	1.71	2.61	1.79	1.66	0.00	0.01	0.05	0.00	0.56	2.97
<i>Monodelphis</i>	2.95	2	1.17	0.84	0.82	1.11	0.8	0.85	0.06	0.04	-0.03	5.26	4.88	3.59
<i>Notoryctes</i>	1.74	2	0.8	0.76	0.79	0.85	0.7	0.82	-0.05	0.06	-0.03	6.06	8.22	3.73
<i>Pedetes</i>	3.48	6	2.79	2.19	2.01	2.71	2.13	1.99	0.08	0.06	0.02	2.91	2.78	1.00
<i>Petauroides</i>	3.11	6	1.97	1.48	1.67	1.94	1.6	1.73	0.03	-0.12	-0.06	1.53	7.79	3.53
<i>Petaurus</i>	2.06	6	1.37	1.1	1.23	1.42	1.1	1.15	-0.05	0.00	0.08	3.58	0.00	6.72
<i>Petropseudes</i>	3.21	4	1.91	1.66	1.72	1.89	1.54	1.72	0.02	0.12	0.00	1.05	7.50	0.00

Taxon	log <sub>10</sub> BM	AG	Radius (mm)						Side differences (mm)			Side difference percent ((Δ SCR /(LSCR + RSCR))*100)		
			LASCR <sub>r</sub>	LLSCR <sub>r</sub>	LPSCR <sub>r</sub>	RASCR <sub>r</sub>	RLSCR <sub>r</sub>	RPSCR <sub>r</sub>	Δ ASCR <sub>r</sub>	Δ LSCR <sub>r</sub>	Δ PSCR <sub>r</sub>	Δ ASCR <sub>r</sub>	Δ LSCR <sub>r</sub>	Δ PSCR <sub>r</sub>
<i>Potorous</i>	3.03	5	2.2	1.5	1.82	2.02	1.53	1.85	0.18	-0.03	-0.03	8.53	1.98	1.63
<i>Pseudocheirus</i>	2.89	4	2.08	1.55	1.7	2.12	1.48	1.69	-0.04	0.07	0.01	1.90	4.62	0.59
<i>Pseudochirops</i>	3.24	4	2.01	1.34	1.7	2.05	1.48	1.74	-0.04	-0.14	-0.04	1.97	9.93	2.33
<i>Pseudochirulus</i>	2.81	4	1.82	1.23	1.47	1.88	1.3	1.48	-0.06	-0.07	-0.01	3.24	5.53	0.68
<i>Sciurus</i>	2.88	6	2.55	2.24	2.14	2.6	2.22	2.11	-0.05	0.02	0.03	1.94	0.90	1.41
<i>Talpa</i>	1.92	2	1.2	1.01	1	1.29	1.08	0.94	-0.09	-0.07	0.06	7.23	6.70	6.19
<i>Tarsipes</i>	0.95	5	0.83	0.59	0.61	0.8	0.62	0.61	0.03	-0.03	0.00	3.68	4.96	0.00
<i>Thylacinus</i>	4.34	5	2.57	1.73	1.96	2.39	1.53	2.04	0.18	0.20	-0.08	7.26	12.27	4.00
<i>Vulpes</i>	3.61	4	2.46	2.23	2.18	2.45	2.17	2.19	0.01	0.06	-0.01	0.41	2.73	0.46
<i>Wallabia</i>	4.14	6	3.12	2.15	2.71	2.93	2.15	2.73	0.19	0.00	-0.02	6.28	0.00	0.74
Mean			1.83	1.43	1.52	1.83	1.41	1.52	-0.01	0.02	0.00	4.46	4.54	3.02
Standard Deviation			0.69	0.55	0.61	0.66	0.54	0.64	0.11	0.08	0.05	5.59	3.89	2.25
95% Confidence Interval		Lower	1.70	1.33	1.41	1.71	1.32	1.40	-0.02	0.00	-0.01	3.46	3.84	2.61
		Upper	1.95	1.53	1.63	1.95	1.51	1.63	0.01	0.03	0.01	5.47	5.24	3.42

**APPENDIX 4. SUMMARY OF SEMICIRCULAR CANAL ANGLE RELATIONSHIPS. INCLUDES ANGLES BETWEEN ALL THREE PAIRS OF IPSILATERAL FOR LEFT AND RIGHT SIDES, ANGLES BETWEEN LEFT AND RIGHT CONTRALATERAL ANGLE PAIRS, AND ANGLES BETWEEN THREE SYNERGISTIC CANAL PAIRS. SPECIES USED FOR ANOVA ANALYSES IN CHAPTER 2 ARE DESIGNATED WITH (+) IF HAVING AN AGILITY OF 5 OR 6 ACCORDING TO EVALUATIONS BY SPOOR ET AL. (2007), SPECIES WITH AGILITIES OF 2 TO 4 ACCORDING TO EVALUATIONS BY SPOOR ET AL. AND USED IN CHAPTER 2 ANOVA ARE DESIGNATED WITH (-). SPECIES WITH NO DESIGNATION WERE NOT USED FOR THE CHAPTER 2 ANALYSES.**

150

Taxon and 'slow' (-) or 'fast' (+) for t-tests	Log10 BM (g)	AG	Ipsilateral Canals (degrees)						Contralateral Canals (degrees)		Synergistic Canals (degrees)		
			LASC <LLSC	LASC <LPSC	LLSC <LPSC	RASC <RLSC	RASC <RPSC	RLSC <RPSC	LASC <RASC	LPSC <RPSC	LASC <RPSC	LPSC <RASC	LLSC <RLSC
<i>Acrobates+</i>	1.09	6.00	103.86	90.54	81.01	98.35	89.17	86.43	87.53	93.75	7.63	9.86	8.95
<i>Allactaga+</i>	2.54	6.00	72.26	82.03	86.86	71.09	83.50	84.69	89.85	109.58	19.58	19.55	10.55
<i>Anomalurus+</i>	2.59	6.00	86.65	104.52	97.34	88.10	102.05	97.77	78.06	75.94	4.39	7.52	1.44
<i>Caluromys+</i>	2.41	5.00	83.19	96.48	89.63	77.14	97.75	87.54	88.14	79.07	10.36	10.03	26.72
<i>Cercartetus+</i>	1.36	6.00	84.15	91.73	87.35	81.53	87.41	84.80	86.46	95.23	9.97	7.99	25.90
<i>Chironectes+</i>	2.87	5.00	82.09	104.33	91.91	77.84	93.55	93.34	67.65	94.76	19.10	11.69	20.25
<i>Chrysochloris-</i>	1.60	2.00	58.82	86.41	103.87	71.65	84.12	91.58	101.59	89.71	15.48	10.06	14.71
<i>Crocuta-</i>	4.81	3.00	80.54	95.71	99.05	81.39	94.32	90.01	88.02	82.32	3.59	8.48	13.20
<i>Dactylopsila+</i>	2.57	6.00	82.82	89.83	94.77	88.04	95.08	96.03	89.35	85.38	9.25	2.69	7.16
<i>Dromiciops-</i>	1.35	3.00	92.87	89.71	92.39	88.75	101.19	94.69	87.88	85.54	19.17	14.68	15.72
<i>Enhydra</i>	4.54	5.00	89.35	98.42	91.64	91.16	102.79	83.18	81.11	77.68	3.91	0.51	8.55
<i>Glaucomys+</i>	1.80	6.00	86.71	90.28	94.48	82.19	87.17	89.96	90.37	93.25	5.66	12.71	18.26
<i>Hemibelideus-</i>	2.98	2.00	83.86	105.74	91.00	78.14	102.18	85.44	80.13	73.41	10.91	9.13	3.27
<i>Heterocephalus-</i>	1.77	2.00	79.61	96.08	96.45	77.65	90.61	97.21	87.72	87.53	10.84	10.96	20.34
<i>Lepus</i>	3.37	5.00	87.92	93.87	87.13	87.87	95.84	87.45	86.47	84.18	4.60	5.19	0.61
<i>Monodelphis-</i>	2.95	2.00	83.90	95.99	91.06	81.31	99.56	89.32	71.03	95.29	15.05	17.04	24.19

Taxon and 'slow' (-) or 'fast' (+) for t-tests	Log10 BM (g)	AG	Ipsilateral Canals (degrees)						Contralateral Canals (degrees)		Synergistic Canals (degrees)		
			LASC <LLSC	LASC <LPSC	LLSC <LPSC	RASC <RLSC	RASC <RPSC	RLSC <RPSC	LASC <RASC	LPSC <RPSC	LASC <RPSC	LPSC <RASC	LLSC <RLSC
<i>Notoryctes-</i>	1.74	2.00	78.61	121.21	93.47	68.23	107.69	82.78	67.76	63.61	6.74	9.29	17.44
<i>Pedetes+</i>	3.48	6.00	94.52	93.63	89.23	91.36	93.12	89.25	84.39	89.08	6.64	2.63	3.00
<i>Petauroides+</i>	3.11	6.00	82.95	98.23	85.29	79.67	93.42	86.31	96.56	72.10	11.04	15.03	22.29
<i>Petaurus+</i>	2.06	6.00	80.42	89.18	98.58	86.01	91.33	102.68	86.66	92.82	2.00	4.85	3.21
<i>Petropseudes-</i>	3.21	4.00	76.77	89.07	93.20	89.48	89.09	90.86	102.34	78.22	15.74	14.14	3.61
<i>Potorous</i>	3.03	5.00	91.64	88.35	88.41	94.76	87.65	88.32	95.61	88.60	5.74	4.63	5.01
<i>Pseudocheirus</i>	2.89	4.00	84.86	93.67	92.90	84.10	98.52	90.41	62.36	105.44	19.13	24.15	6.42
<i>Pseudochirops-</i>	3.24	4.00	93.28	83.83	104.81	87.93	87.37	85.92	81.45	106.99	11.87	15.29	9.53
<i>Pseudochirulus-</i>	2.81	4.00	92.18	104.50	98.10	89.71	101.19	103.14	74.25	80.35	9.93	2.32	7.64
<i>Sciurus+</i>	2.88	6.00	79.26	86.88	90.60	84.04	82.88	91.93	93.94	96.42	5.54	1.71	6.10
<i>Talpa-</i>	1.92	2.00	76.67	105.11	89.71	78.48	91.73	100.71	81.42	81.62	7.09	7.47	8.21
<i>Tarsipes</i>	0.95	5.00	97.36	104.25	92.06	101.85	98.42	90.51	79.95	77.42	10.04	4.21	10.24
<i>Thylacinus</i>	4.34	5.00	86.45	78.68	85.10	82.36	89.91	86.62	102.26	89.02	12.45	3.46	13.54
<i>Vulpes-</i>	3.61	4.00	87.46	83.76	97.34	88.49	91.52	102.77	96.65	101.45	8.02	12.97	5.14
<i>Wallabia+</i>	4.14	6.00	85.39	87.79	92.37	89.71	93.25	91.73	83.48	98.40	11.82	18.59	19.54
Mean			84.72	94.19	92.49	84.46	93.66	91.08	85.50	87.88	10.11	9.64	11.64
Standard Deviation			8.20	8.88	5.30	7.63	6.29	5.81	10.02	10.78	4.99	5.84	7.63
Coefficient of Variation			9.68	9.43	5.73	9.04	6.72	6.38	11.71	12.27	49.34	60.59	65.55
95% Confidence Interval	Lower		83.25	92.59	91.53	83.09	92.53	90.03	83.70	85.94	9.21	8.59	10.27
	Upper		86.20	95.78	93.44	85.83	94.79	92.12	87.30	89.81	11.00	10.69	13.01

**APPENDIX 5. DIFFERENCES IN ANGLES BETWEEN LEFT AND RIGHT PETROSAL CANALS. INCLUDES THE ABSOLUTE VALUES OF THE ANGLE DIFFERENCES, AND THE AVERAGE ANGLE PAIR VALUES FROM BOTH SIDES. MEASUREMENTS AND TERMS FROM APPENDIX 4.**

152

Taxon and 'slow' (-) or 'fast' (+) for t-tests	Log10 BM (g)	AG	Side Comparisons (degrees)				Absolute Side Comparisons (degrees)				Canal Pair Angles Averaged (degrees)			
			Δ ASC		Syn. Δ ASC		Δ ASC		Syn. Δ ASC		ASC		Syn. ASC	
			<LSC	<PSC	<PSC	<PSC	<LSC	<PSC	<PSC	<PSC	<LSC	<PSC	<PSC	<PSC
<i>Acrobates+</i>	1.09	6	5.51	1.37	-5.42	-2.23	5.51	1.37	5.42	2.23	101.11	89.86	83.72	8.75
<i>Allactaga+</i>	2.54	6	1.17	-1.47	2.17	0.03	1.17	1.47	2.17	0.03	71.68	82.77	85.78	19.57
<i>Anomalurus+</i>	2.59	6	-1.45	2.47	-0.43	-3.13	1.45	2.47	0.43	3.13	87.38	103.29	97.56	5.96
<i>Caluromys+</i>	2.41	5	6.05	-1.27	2.09	0.33	6.05	1.27	2.09	0.33	80.17	97.12	88.59	10.20
<i>Cercartetus+</i>	1.36	6	2.62	4.32	2.55	1.98	2.62	4.32	2.55	1.98	82.84	89.57	86.08	8.98
<i>Chironectes+</i>	2.87	5	4.25	10.78	-1.43	7.41	4.25	10.78	1.43	7.41	79.97	98.94	92.63	15.40
<i>Chrysochloris-</i>	1.6	2	-12.83	2.29	12.29	5.42	12.83	2.29	12.29	5.42	65.24	85.27	97.73	12.77
<i>Crocuta-</i>	4.81	3	-0.85	1.39	9.04	-4.89	0.85	1.39	9.04	4.89	80.97	95.02	94.53	6.04
<i>Dactylopsila+</i>	2.57	6	-5.22	-5.25	-1.26	6.56	5.22	5.25	1.26	6.56	85.43	92.46	95.40	5.97
<i>Dromiciops-</i>	1.35	3	4.12	-11.48	-2.30	4.49	4.12	11.48	2.30	4.49	90.81	95.45	93.54	16.93
<i>Enhydra</i>	4.54	5	-1.81	-4.37	8.46	3.40	1.81	4.37	8.46	3.40	90.26	100.61	87.41	2.21
<i>Glaucomys+</i>	1.8	6	4.52	3.11	4.52	-7.05	4.52	3.11	4.52	7.05	84.45	88.73	92.22	9.19
<i>Hemibelideus-</i>	2.98	2	5.72	3.56	5.56	1.78	5.72	3.56	5.56	1.78	81.00	103.96	88.22	10.02
<i>Heterocephalus-</i>	1.77	2	1.96	5.47	-0.76	-0.12	1.96	5.47	0.76	0.12	78.63	93.35	96.83	10.90
<i>Lepus</i>	3.37	5	0.05	-1.97	-0.32	-0.59	0.05	1.97	0.32	0.59	87.90	94.86	87.29	4.90
<i>Monodelphis-</i>	2.95	2	2.59	-3.57	1.74	-1.99	2.59	3.57	1.74	1.99	82.61	97.78	90.19	16.05
<i>Notoryctes-</i>	1.74	2	10.38	13.52	10.69	-2.55	10.38	13.52	10.69	2.55	73.42	114.45	88.13	8.02
<i>Pedetes+</i>	3.48	6	3.16	0.51	-0.02	4.01	3.16	0.51	0.02	4.01	92.94	93.38	89.24	4.64
<i>Petauroides+</i>	3.11	6	3.28	4.81	-1.02	-3.99	3.28	4.81	1.02	3.99	81.31	95.83	85.80	13.04

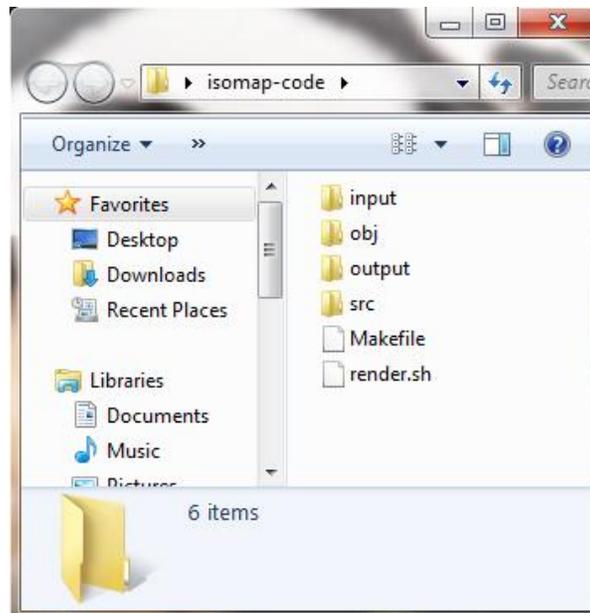
Taxon and 'slow' (-) or 'fast' (+) for t- tests	Log10 BM (g)	AG	Side Comparisons (degrees)				Absolute Side Comparisons (degrees)				Canal Pair Angles Averaged (degrees)			
			Δ ASC		Syn. Δ ASC		Δ ASC		Syn. Δ ASC		ASC		Syn. ASC	
			<LSC	<PSC	<PSC	<PSC	<LSC	<PSC	<PSC	<PSC	<LSC	<PSC	<PSC	<PSC
<i>Petaurus+</i>	2.06	6	-5.59	-2.15	-4.10	-2.85	5.59	2.15	4.10	2.85	83.22	90.26	100.63	3.43
<i>Petropseudes-</i>	3.21	4	-12.71	-0.02	2.34	1.60	12.71	0.02	2.34	1.60	83.13	89.08	92.03	14.94
<i>Potorous</i>	3.03	5	-3.12	0.70	0.09	1.11	3.12	0.70	0.09	1.11	93.20	88.00	88.37	5.19
<i>Pseudocheirus</i>	2.89	4	0.76	-4.85	2.49	-5.02	0.76	4.85	2.49	5.02	84.48	96.10	91.66	21.64
<i>Pseudochirops-</i>	3.24	4	5.35	-3.54	18.89	-3.42	5.35	3.54	18.89	3.42	90.61	85.60	95.37	13.58
<i>Pseudochirulus-</i>	2.81	4	2.47	3.31	-5.04	7.61	2.47	3.31	5.04	7.61	90.95	102.85	100.62	6.13
<i>Sciurus+</i>	2.88	6	-4.78	4.00	-1.33	3.83	4.78	4.00	1.33	3.83	81.65	84.88	91.27	3.63
<i>Talpa-</i>	1.92	2	-1.81	13.38	-11.00	-0.38	1.81	13.38	11.00	0.38	77.58	98.42	95.21	7.28
<i>Tarsipes</i>	0.95	5	-4.49	5.83	1.55	5.83	4.49	5.83	1.55	5.83	99.61	101.34	91.29	7.13
<i>Thylacinus</i>	4.34	5	4.09	-11.23	-1.52	8.99	4.09	11.23	1.52	8.99	84.41	84.30	85.86	7.96
<i>Vulpes-</i>	3.61	4	-1.03	-7.76	-5.43	-4.95	1.03	7.76	5.43	4.95	87.98	87.64	100.06	10.50
<i>Wallabia+</i>	4.14	6	-4.32	-5.46	0.64	-6.77	4.32	5.46	0.64	6.77	87.55	90.52	92.05	15.21
Mean			0.26	0.53	1.41	0.47	4.13	4.68	4.08	3.69	84.59	93.92	91.78	9.87
Standard Deviation			5.22	6.04	5.87	4.46	3.12	3.76	4.40	2.47	7.48	7.08	4.72	4.95
Coefficient of Variation							75.51	80.30	107.83	66.91	8.84	7.54	5.15	50.15
95% Confidence Level	Lower						3.57	4.01	3.29	3.24	83.25	92.65	90.93	8.98
	Upper						4.69	5.36	4.87	4.13	85.94	95.19	92.63	10.76

## APPENDIX 6. INSTRUCTIONS FOR ASSEMBLY OF THE ISOMAP C++ PROGRAM USED FOR SENSITIVITY CALCULATIONS.

Before the program is compiled, a licensed DLL file must be downloaded. The file is at this URL: <http://www.xmission.com/~nate/glut.html>. Nate Robins formatted this DLL program, originally written by Mark Kilgard. Both should be credited for the final product. The file necessary is glut-3.7.6-bin.zip, and the full URL to that file is: <http://www.xmission.com/~nate/glut/glut-3.7.6-bin.zip>. The file inside that link is named glut32.dll. Copy that file into the same directory as venice.exe (described below). The other files within the URL link can be ignored.

### **isomap-code**

This is the root file folder that contains all necessary information to run the sensitivity program. It must be completely assembled before the program can be run. The screenshot below shows the file contents of the isomap-code folder:



The first three entries listed for folders (input, obj, and output) must be set up to contain your data before, during, and after building and/or running the isomap program. Before entering your data, they will all be empty.

The files listed in this appendix are only the source code of the program. The program must be compiled before it can be executed. It may be compiled on many platforms, including UNIX, Linux, and Windows within the Cygwin environment. To compile it,

navigate into the isomap-code directory with a command-line shell, then invoke the 'make' command without any other command-line options. The program's executable will be produced in the isomap-gen directory. It will be named isomap-gen on Linux and UNIX, or isomap-gen.exe on Windows.

Execute the program from the shell with no command-line options, and it will output the command-line options it accepts to the terminal.

render.sh is a shell script which was used to generate most of the contour plots. It requires a text file with a list of specimens to process, named input-list.dat in the isomap-code directory. The sample below will process Dromiciops.txt and Monodelphis.txt from the input directory, and place the output files for each in the output directory. Including dataset names which include spaces, tabs, back-slashes, forward-slashes, or special characters will make render.sh malfunction.

### □ **input-list.dat**

Dromiciops  
Monodelphis

To execute completely, render.sh requires the GNU Image Manipulation Program (GIMP) to be available on the computer. It is only necessary for convenience functionality, like cropping and converting the PPM images into BMP images. However, even if render.sh aborts due to GIMP being unavailable, the histograms, PPM images, and summary file will still be available in the output directory.

There is an optional program, named venice on Linux and UNIX or venice.exe on Windows. This program inputs the full PPM files, such as output/Dromiciops.gabs2.ppm, and displays it in 3D projected on a sphere. It allows navigation around the sphere, to view it from any angle. The navigation keystrokes are listed on terminal output for convenience. To run it on Linux or UNIX, execute venice with a single command-line option specifying the name of the input file. To run it on Windows, click and drag the icon of the input file onto venice.exe's icon.

Compiling venice requires additional code libraries, some of which are less commonly installed on most computers. Those libraries are libX11, libXi, libXmu, libglut, libGl, and libGLU. To build venice or venice.exe, use the command-line shell to navigate into the isomap-code directory, and type the command 'make venice' (without the quotes). This will create venice or venice.exe in the isomap-code directory.

## 1. **input**

The input directory is where all dataset input files are deposited for analysis. The entry format for these files is as a text file. It contains the six semicircular canal names, the radius length for each (in mm), and the coordinates for the vectors normal to the canal best-fit planes that render an excitatory vestibular signal from each canal in a head-centered coordinate system. The general directions for the vector coordinates are shown in Figure 3.4. All of this data for my studies can be taken from Appendix 1. A table-formatted version of the data is shown below for *Dromiciops gliroides*:

Canal	Radius	Vector Coordinates		
		X	Y	Z
LASC	1.05	-0.72	0.67	-0.17
LLSC	0.77	-0.39	-0.12	0.91
LPSC	0.78	-0.58	-0.71	-0.39
RASC	1.07	0.63	0.76	0.14
RLSC	0.71	0.36	-0.16	-0.92
RPSC	0.72	0.51	-0.75	0.42

The text file itself excludes the headings shown in the table above, and the example for *Dromiciops gliroides* and *Monodelphis domestica* are given below:



Dromiciops.txt

```
LASC 1.05 -0.7202 0.6724 -0.1709
LLSC 0.77 -0.3937 -0.1155 0.9119
LPSC 0.78 -0.5822 -0.7147 -0.3877
RASC 1.07 0.6289 0.7645 0.1415
RLSC 0.71 0.3617 -0.1561 -0.9191
RPSC 0.72 0.5108 -0.7514 0.4176
```



Monodelphis.txt

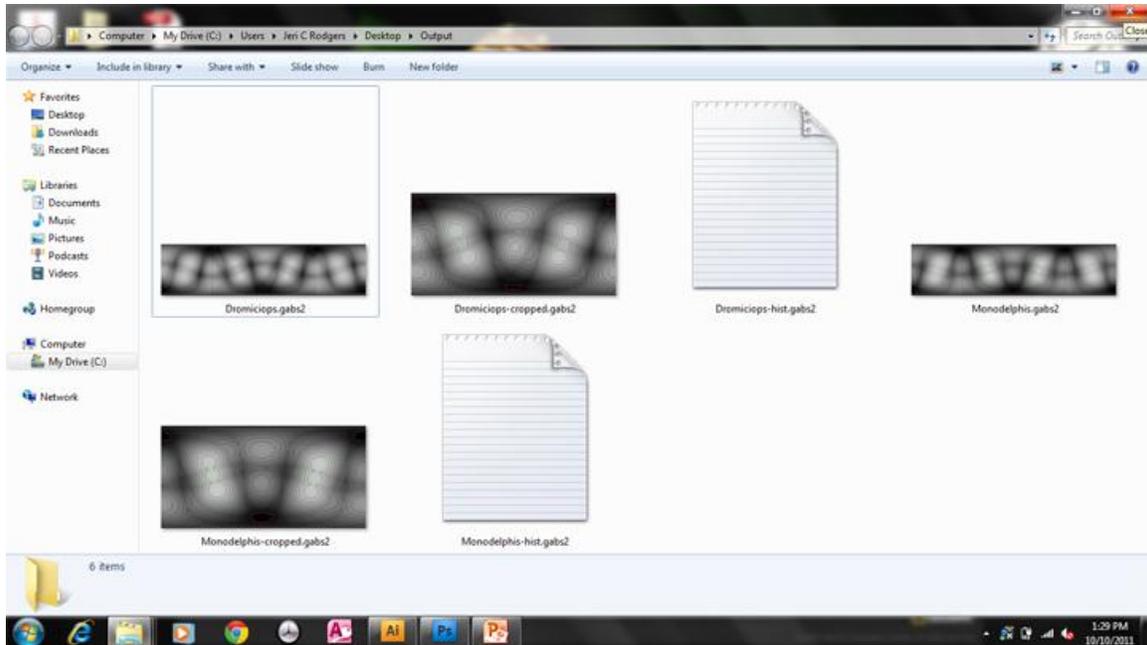
```
LASC 1.17 -0.5729 0.8084 -0.1349
LLSC 0.84 -0.2753 -0.1685 0.9465
LPSC 0.82 -0.6724 -0.6611 -0.3328
RASC 1.11 0.5633 0.8193 0.1073
RLSC 0.8 0.2725 -0.2502 -0.929
RPSC 0.85 0.634 -0.6855 0.3579
```

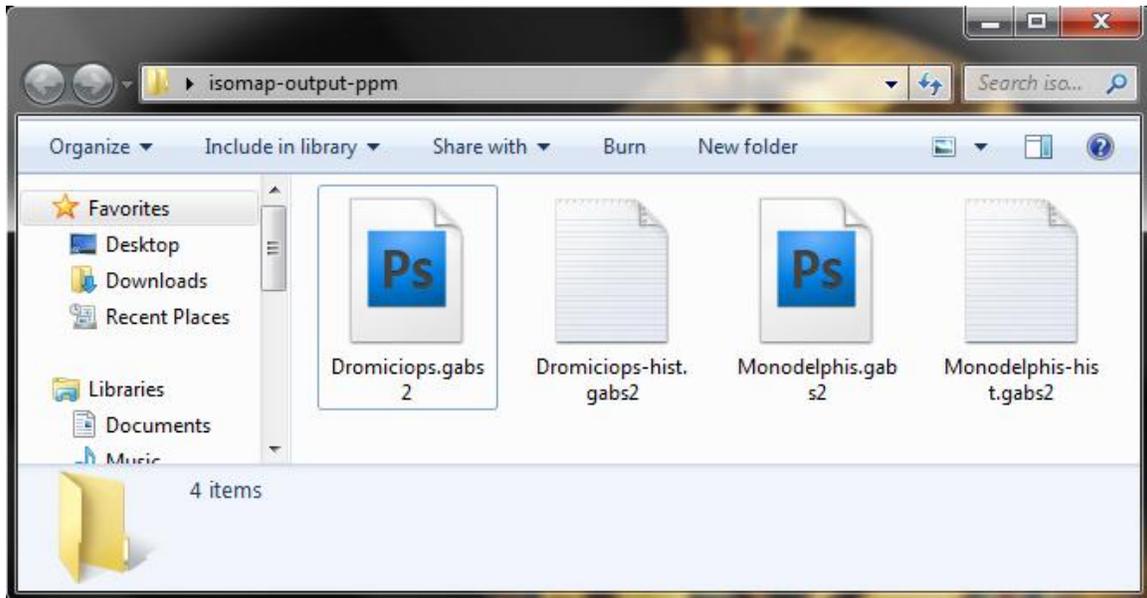
## 2. obj

The obj directory is filled with intermediate files by the compiler, and the compiler also requires it to exist. It will remain empty until the program is built. After the program's executables are built, the files inside the obj directory may be discarded.

### 3. output

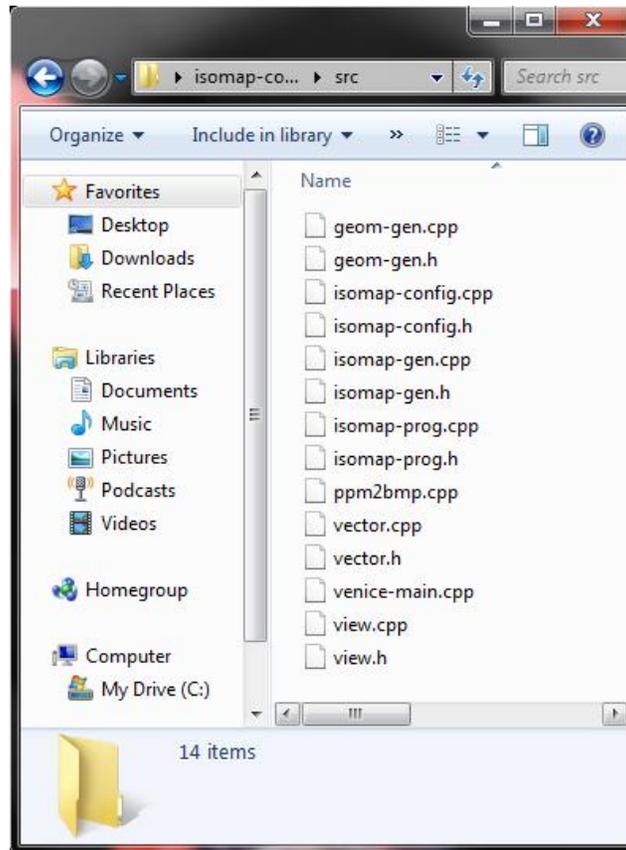
This is where the files of images and information will be found after the program has run.





#### 4. src

The source code itself is inside src. Below is a screen shot of the files that should be contained within this folder:



### **geom-gen.cpp**

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
#include "vector.h"
#include "view.h"
#include "geom-gen.h"
#include "isomap-gen.h"
```

```
/* The number of rows and columns of facets of the sphere.
```

```
 * The number of facets will be SPHERE_RESOLUTION * SPHERE_RESOLUTION,
creating
 * SPHERE_RESOLUTION * SPHERE_RESOLUTION * 2 triangles. Increasing the
number shrinks the
 * facets, making it more sphere-like. However, increasing it too much
will hurt system performance
 * and possibly exhaust the graphics card's resources. */
#define SPHERE_RESOLUTION 30
```

```
/* Storage buffer for the contour plot.
 * Its format is directly compatible with OpenGL texture buffers. */
IsomapPixel *texture_buf = NULL;
```

```
/* Handle for the OpenGL texture of the contour plot. */
```

```

GLuint g_texture_handle = 0;
/* Handle to the triangle list of of the sphere. */
GLuint g_sphere_list = 0;
/* Handle to the triangle list of the lobed model. */
GLuint g_lobed_list = 0;
/* Handle to the triangle list of the axis indicators. */
GLuint g_arrow_list = 0;

/* Set to true when OpenGL initialization is complete.
 * It the model is only rendered after its geometry and texture has
 * been constructed. */
bool isomap_initialized = false;

/*****
 * NAME: load_isomap
 *
 * DESCRIPTION: Initialize the isomap program. It
 * reads the contour plot image file, and converts
 * it into a texture. It also generates all of
 * the necessary triangle lists. After calling
 * this function, the program is ready to render
 * the model.
 *
 * RETURNS: N/A
 *****/
void load_isomap(string infile_name)
{
    /* Open the PPM file. */
    ifstream inpic(infile_name.c_str());

    string dummy_string;
    IsomapPixel *texture_iter;
    int horiz_res;
    int vert_res;
    int square_res;
    int color_res;
    int color_overprec;

    /* Read and discard the P3 magic number and human-readable
    comments. */
    inpic >> dummy_string; // Eat the P3
    inpic >> dummy_string; // Eat the comment
    inpic >> dummy_string; // Eat the comment
    inpic >> dummy_string; // Eat the comment

    /* Read the pixel and color resolutions from the file. */
    inpic >> horiz_res >> vert_res >> color_res;

    /* If the file uses more than 8 bits per color channel, clamp it
    down to just
    * 8 bits per channel. That's the most that OpenGL can handle.
    */
    if (color_res <= 256)
    {
        /* Make it divide by 1 to keep the color as-is. */
        color_overprec = 1;
    }
    else
    {

```

```

        /* Make it divide to eliminate enough bits so it uses 8
bits per channel. */
        color_overprec = color_res / 256;
    }

    /* The image file isn't square, but the texture needs to be. It
will end up only using
    * half of the texture, so make it big enough to fit the largest
dimension. */
    square_res = horiz_res / 2;

    /* Allocate the texture buffer. */
    texture_buf = new IsomapPixel[square_res * square_res];
    /* Prepare an iterator to fill the texture buffer. */
    texture_iter = texture_buf;

    /* Allocate a buffer to hold the "intensity" at each pixel. This
intensity will be used
    * to grow and shrink the sphere to generate the lobed model. */
    GLfloat *texture_scale = new GLfloat[square_res * vert_res];

    /* The loading process takes a while, so let the user know it's
not dead. */
    cout << "Loading image...\n";

    /* The sides of the image are actually a different representation
of the same data.
    * Compute the bounds for looping over each pixel in a row so
that detecting the
    * center and edge regions is easier. */
    int imin = -horiz_res / 4;
    int imax = 3 * horiz_res / 4;

    /* Iterate over the output picture to process all of its
information.
    * This outer loop loops over the rows of pixels. */
    for (int j = 0; j < vert_res; ++j)
    {
        /* On every 10th row, output a character as a rudimentary
progress bar.
        * Alternate the character to make it easier to notice the
progress. */
        if (j % 20 == 0)
        {
            cout << ".";
        }
        else if (j % 10 == 0)
        {
            cout << "+";
        }
    }

    /* Loop over the pixels in the current row. */
    for (int i = imin; i < imax; ++i)
    {
        /* Read in the color from the file. */
        int red, green, blue;
        inpic >> red >> green >> blue;

        /* Now, figure out what to do with it. */

```

```

        if (i >= 0 && i < square_res)
        {
            /* The pixel is in the center region, so use it
as the texture. */
            /* Discard enough precision from each color
channel to make it fit in 8 bits per channel. */
            red /= color_overprec;
            green /= color_overprec;
            blue /= color_overprec;

            /* Store the pixel into the texture buffer. */
            texture_iter->red = (unsigned char)red;
            texture_iter->blue = (unsigned char)blue;
            texture_iter->green = (unsigned char)green;
            ++texture_iter;
        }
        else if (i < 0)
        {
            /* This pixel is on the left edge, so it
represents a pixel intensity of a pixel on the right side of the center
region.
            * The pixel is monochrome, so divide any color
channel by the color resolution to get the intensity, from 0.0 to
1.0.*/
            texture_scale[j * square_res + square_res + i]
= (GLfloat)red / (GLfloat)color_res;
        }
        else
        {
            /* This pixel is on the right edge, so it
represents a pixel intensity of a pixel on the left side of the center
region.
            * The pixel is monochrome, so divide any color
channel by the color resolution to get the intensity, from 0.0 to
1.0.*/
            texture_scale[j * square_res + i - square_res]
= (GLfloat)red / (GLfloat)color_res;
        }
    }
}

/* Zero out the rest of the texture.
* If it is accidentally displayed, this prevents it from
rendering trash. */
for (int j = 0; j < vert_res; ++j)
{
    for (int i = 0; i < square_res; ++i)
    {
        texture_iter->red = 0;
        texture_iter->blue = 0;
        texture_iter->green = 0;
        ++texture_iter;
    }
}

/* Done reading from the file, so close it. */
inpic.close();

```

```

    /* Create the texture for openGL. openGL will take the color
    data, convert it into a texture,
    * and give it to the graphics card. */
    glGenTextures(1, &g_texture_handle);
    glBindTexture(GL_TEXTURE_2D, g_texture_handle);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, square_res, square_res, 0,
    GL_RGB, GL_UNSIGNED_BYTE, texture_buf);

    /* The texture data is now on the card, so we don't need our copy
    anymore. */
    delete [] texture_buf;

    /* Set some texture options to make the texture look nice. */
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    /* Generate the triangle lists, so they can be displayed at a
    moment's notice. */
    g_arrow_list = create_arrow_list();
    g_sphere_list = create_sphere_list(NULL, square_res);
    g_lobed_list = create_sphere_list(texture_scale, square_res);

    /* The texture scaling is now held in openGL in the form of the
    vertices themselves.
    * Delete this intermediate representation. */
    delete [] texture_scale;

    /* Everything is ready to display now. */
    isomap_initialized = true;
}

/*****
* NAME: create_sphere_list
*
* DESCRIPTION: Generate the triangle list for a
* sphere. It can generate either a pure sphere,
* or a lobed one. When texture_scale is NULL, it
* generates a true sphere. When texture_scale
* points to scaling data, it creates a lobed model.
*
* RETURNS: openGL handle to the new vertex list.
*****/
GLuint create_sphere_list(GLfloat *texture_scale, int square_res)
{
    /* Get the vertical resolution. */
    int vert_res = square_res / 2;

    /* Compute how much bearing and/or pitch to increment between each
    vertex. */
    double delta_bearing = 360.0 / SPHERE_RESOLUTION;
    double delta_pitch = -180.0 / SPHERE_RESOLUTION;

    /* Compute how much texture space to increment between each
    vertex.
    * U and V are 2D coordinates on the texture. */
    GLfloat delta_u = 1.0 / SPHERE_RESOLUTION;
    GLfloat delta_v = 0.5 / SPHERE_RESOLUTION;

    /* Compute the bearing and pitch of the leading edge of vertices.
    */

```

```

double curr_bering = -180.0 + delta_bering;
double curr_pitch = 90.0 + delta_pitch;

/* Compute initial values for the pitch and bering. These values
make
* each row completely wrap around and finish the ring. */
double prev_pitch = 90.0;
double prev_bering = -180.0;

/* Compute the initial current and previous U and V to match the
current and previous
* bering and pitch. */
GLfloat curr_u = delta_u;
GLfloat prev_u = 0.0;
GLfloat curr_v = delta_v;
GLfloat prev_v = 0.0;

/* Allocate a new handle for generating a triangle list. */
GLuint new_list = glGenLists(1);

/* Tell OpenGL that the new list is a triangle list. */
glNewList(new_list, GL_COMPILE);
glBegin(GL_TRIANGLES);

/* Iterate top (pitch=90) to bottom (pitch=-90).
* At each pitch increment, generate a ring of triangles. */
for (int j = 0; j < SPHERE_RESOLUTION; ++j)
{
    curr_u = delta_u;
    curr_bering = -180.0 + delta_bering;
    prev_u = 0.0;
    prev_bering = -180.0;

    /* Ensure the texture's previous and current V coordinates
stay in bounds.
* Anything outside vert_res is garbage. */
    int prev_v_ztex_i = prev_v * (GLfloat)square_res;
    if (prev_v_ztex_i >= vert_res)
    {
        prev_v_ztex_i = vert_res - 1;
    }

    int curr_v_ztex_i = curr_v * (GLfloat)square_res;
    if (curr_v_ztex_i >= vert_res)
    {
        curr_v_ztex_i = vert_res - 1;
    }

    /* Convert the integer pixel location into a floating-point
v coordinate. */
    prev_v_ztex_i *= square_res;
    curr_v_ztex_i *= square_res;

    /* Iterate left (bering=-180) to right (bering=180) as
viewed from the outside. */
    for (int i = 0; i < SPHERE_RESOLUTION; ++i)
    {
        /* Create the four corners of the spherical bering-
pitch coordinates.

```

```

    * The corners are named after their position
relative to a viewer outside
    * the sphere looking inward. */
    IsoVector top_right(curr_bering, prev_pitch);
    IsoVector top_left(prev_bering, prev_pitch);
    IsoVector bottom_left(prev_bering, curr_pitch);
    IsoVector bottom_right(curr_bering, curr_pitch);

    /* Ensure the integer U coordinates are in bounds and
are not on the black edges
    * delimiting the center region. If they are, then
loop them around to the other side.
    * Otherwise, black lines would obscure the sphere's
features. */

    int prev_u_ztex_i = prev_u * square_res;
    if (prev_u_ztex_i >= square_res - 1)
    {
        prev_u_ztex_i = square_res - 2;
    }
    else if (prev_u_ztex_i == 0)
    {
        prev_u_ztex_i = 1;
    }

    int curr_u_ztex_i = curr_u * square_res;
    if (curr_u_ztex_i >= square_res - 1)
    {
        curr_u_ztex_i = square_res - 2;
    }
    else if (curr_u_ztex_i == 0)
    {
        curr_u_ztex_i = 1;
    }

    if (texture_scale != NULL)
    {
        /* The scaling map has been specified, so it is
a lobed sphere.
        * Retrieve the scale value from the buffer for
each of the four corners.
        * Then, use that to scale the radius of the
spherical coordinates. */

        top_right =
top_right.scale(texture_scale[curr_u_ztex_i + prev_v_ztex_i]);
        top_left =
top_left.scale(texture_scale[prev_u_ztex_i + prev_v_ztex_i]);
        bottom_left =
bottom_left.scale(texture_scale[prev_u_ztex_i + curr_v_ztex_i]);
        bottom_right =
bottom_right.scale(texture_scale[curr_u_ztex_i + curr_v_ztex_i]);
    }

    /* Now, scale the sphere outward by a constant
amount. Scaling by 5 makes the axis arrows a reasonable
    * size relative to the sphere. */
    top_right = top_right.scale(5.0).correctAxes();
    top_left = top_left.scale(5.0).correctAxes();
    bottom_left = bottom_left.scale(5.0).correctAxes();

```

```

        bottom_right = bottom_right.scale(5.0).correctAxes();
        /* There are four corners, but break it up into 2
triangles. The 4 vertices are
        * not co-planar, so triangles will produce a better-
looking result. */
        glVertex3f(top_right.x,
top_right.y, top_right.z);
        glVertex3f(top_left.x,
top_left.y, top_left.z);
        glVertex3f(bottom_left.x, bottom_left.y, bottom_left.z);
        glVertex3f(top_right.x,
top_right.y, top_right.z);
        glVertex3f(bottom_left.x, bottom_left.y, bottom_left.z);
        glVertex3f(bottom_right.x, bottom_right.y, bottom_right.z);
        /* Increment the previous and current U coordinates.
*/
        prev_u = curr_u;
        curr_u += delta_u;
        /* Increment the previous and current bearing
coordinates. */
        prev_bearing = curr_bearing;
        curr_bearing += delta_bearing;
    }
    /* Increment the previous and current U coordinates. */
    prev_v = curr_v;
    curr_v += delta_v;
    /* Increment the previous and current pitch coordinates. */
    prev_pitch = curr_pitch;
    curr_pitch += delta_pitch;
}
/* Tell OpenGL that the sphere's geometry is complete, and let
OpenGL
    * finalize its internal representation. */
    glEnd();
    glEndList();
    /* Return the handle to the sphere's triangle list. That is all
the caller really needs to use it. */
    return new_list;
}
/*****
* NAME: display_isomap
*
* DESCRIPTION: Select the geometry to render to
* the screen based on the user's selections, then
* render it.
*
* RETURNS: N/A
*****/

```

```

void display_isomap()
{
    /* Do not attempt to use any handles if the program isn't
     * initialized yet. The window operates in its own thread,
     * so it can happen without this check. */
    if (isomap_initialized)
    {
        /* This is to make sure we have up-to-date matrices.
         * [Re-]Compute the matrices required to put the sphere
         * and/or axis arrows in the correct place relative to the
viewer.
         * The matrices change whenever the user physically
navigates. */
        recalc_matrices();

        /* If the user enabled the axis arrows, then render them now.
*/
        if (enable_axis_arrows)
        {
            /* Unbind the isomap texture so the arrows won't be
textured. */
            glBindTexture(GL_TEXTURE_2D, 0);
            /* Command OpenGL to render the axis arrow triangle
list. */
            glCallList(g_arrow_list);
        }

        /* Enable the isomap texture for the sphere, and specify a
color (white) to make the
         * texture appear at full brightness. */
        glColor3f(1.0f, 1.0f, 1.0f);
        glBindTexture(GL_TEXTURE_2D, g_texture_handle);

        /* Select which sphere list with the user's selection. */
        if (enable_lobed_sphere)
        {
            /* Render the lobed sphere to the screen. */
            glCallList(g_lobed_list);
        }
        else
        {
            /* Render the pure sphere to the screen. */
            glCallList(g_sphere_list);
        }
    }
}

```

```

/*****
 * NAME: destroy_isomap
 *
 * DESCRIPTION: Release all the graphics resources
 * that isomap may have allocated. This is
 * especially important because OpenGL and the
 * operating system do not automatically free these
 * resources when the program exits, like it does
 * for malloc'ed memory on the heap. Failure to
 * free it will require a reboot of the computer.
 *
 * It only attempts to free resources if they are

```

```

* allocated. A non-zero handle implies the handle
* needs to be freed.
*
* RETURNS: N/A
*****/
void destroy_isomap()
{
    if (g_sphere_list != 0)
    {
        glDeleteLists(g_sphere_list,1);
        g_sphere_list = 0;
    }

    if (g_lobed_list != 0)
    {
        glDeleteLists(g_lobed_list,1);
        g_lobed_list = 0;
    }

    if (g_arrow_list != 0)
    {
        glDeleteLists(g_arrow_list,1);
        g_arrow_list = 0;
    }

    if (g_texture_handle != 0)
    {
        glDeleteTextures(1, &g_texture_handle);
    }
}

/*****
* NAME: create_arrow_list
*
* DESCRIPTION: Allocate and generate a triangle
* list of the axis arrows.
*
* RETURNS: An OpenGL handle to the new axis arrow
* triangle list.
*****/
GLuint create_arrow_list()
{
    /* Allocate the axis list. */
    GLuint axis_handle = glGenLists(1);

    /* Tell OpenGL what type of list we are creating. */
    glNewList(axis_handle, GL_COMPILE);
    glBegin(GL_TRIANGLES);

    /* The axis arrows are tetrahedrons. Declare the vertices of
each one.
* We will wire the vertices together later to create the
triangles. */

    GLfloat x_arrow_verts[4][3] = {
        {10.0f, 0.0f, 0.0f}, /* Front */
        {8.0f, 0.25f, 0.0f}, /* Left */
        {8.0f, -0.25f, 0.0f}, /* Right */
        {8.0f, 0.0f, 0.25f} /* Top */
    };
}

```

```

GLfloat y_arrow_verts[4][3] = {
    {0.0f, 10.0f, 0.0f}, /* Front */
    {-0.25f, 8.0f, 0.0f}, /* Left */
    {0.25f, 8.0f, 0.0f}, /* Right */
    {0.0f, 8.0f, 0.25f} /* Top */
};

GLfloat z_arrow_verts[4][3] = {
    {0.0f, 0.0f, 10.0f}, /* Front */
    {0.0f, -0.25f, 8.0f}, /* Left */
    {0.0f, 0.25f, 8.0f}, /* Right */
    {0.25f, 0.0f, 8.0f} /* Top */
};

/* Declare the pattern for wiring the vertices together.
 * Each number is an index specifying the vertex. Each row
represents a full triangle. */
int axis_vert_seq[] = {
    0, 1, 2, /* bottom */
    0, 1, 3, /* left-top */
    2, 0, 3, /* right-top */
    1, 2, 3 /* back */
};

/* Let the compiler figure out how big the sequence is, in case
it is changed later. */
int vert_seq_len = sizeof(axis_vert_seq) /
sizeof(axis_vert_seq[0]);

/* Pass the X axis arrow to OpenGL.
 *
 * NOTE: OpenGL takes each triplet of vertices and makes a
triangle out of them. */
for (int i = 0; i < vert_seq_len; ++i)
{
    /* Retrieve the index of the vertex to use next. */
    int idx = axis_vert_seq[i];
    /* Create a vector object out of the vertex data. */
    IsoVector axis_corr_vect(x_arrow_verts[idx][0],
x_arrow_verts[idx][1], x_arrow_verts[idx][2]);
    /* Use the vector object to convert the coordinate system
to match OpenGL's. */
    axis_corr_vect = axis_corr_vect.correctAxes();

    /* Give OpenGL the color of the vertex first. This one is
green. */
    glColor3f(0.1, 0.9, 0.1);
    /* Give OpenGL the vertex data. */
    glVertex3f(axis_corr_vect.x, axis_corr_vect.y,
axis_corr_vect.z);
}

/* Pass the Y axis arrow to OpenGL. */
for (int i = 0; i < vert_seq_len; ++i)
{
    /* Retrieve the index of the vertex to use next. */
    int idx = axis_vert_seq[i];
    /* Create a vector object out of the vertex data. */

```

```

        IsoVector axis_corr_vect(y_arrow_verts[idx][0],
y_arrow_verts[idx][1], y_arrow_verts[idx][2]);
        /* Use the vector object to convert the coordinate system
to match OpenGL's. */
        axis_corr_vect = axis_corr_vect.correctAxes();

        /* Give OpenGL the color of the vertex first. This one is
red. */
        glColor3f(0.9, 0.1, 0.1);
        /* Give OpenGL the vertex data. */
        glVertex3f(axis_corr_vect.x, axis_corr_vect.y,
axis_corr_vect.z);
    }

    /* Pass the Z axis arrow to OpenGL. */
    for (int i = 0; i < vert_seq_len; ++i)
    {
        /* Retrieve the index of the vertex to use next. */
        int idx = axis_vert_seq[i];
        /* Create a vector object out of the vertex data. */
        IsoVector axis_corr_vect(z_arrow_verts[idx][0],
z_arrow_verts[idx][1], z_arrow_verts[idx][2]);
        /* Use the vector object to convert the coordinate system
to match OpenGL's. */
        axis_corr_vect = axis_corr_vect.correctAxes();

        /* Give OpenGL the color of the vertex first. This one is
blue. */
        glColor3f(0.1, 0.1, 0.9);
        /* Give OpenGL the vertex data. */
        glVertex3f(axis_corr_vect.x, axis_corr_vect.y,
axis_corr_vect.z);
    }

    /* Tell OpenGL that this list is complete, and let it finalize
things internally. */
    glEnd();
    glEndList();

    /* Return the handle to the sphere's triangle list. That is all
the caller really needs to use it. */
    return axis_handle;
}

```

## geom-gen.h

```

#ifndef GEOM_GEN_H
#define GEOM_GEN_H

#include <string>
using namespace std;

GLuint create_sphere_list(GLfloat *texture_scale, int square_res);
void load_isomap(string infile);
void display_isomap();
void destroy_isomap();
GLuint create_arrow_list();

```

```
#endif
```



isomap-config.cpp Here is the code for this file:

```
#include "isomap-config.h"

#include <iostream>
#include <sstream>

/*****
 * NAME: IsoConfig
 *
 * DESCRIPTION: Default constructor. Initialize
 *   all member values to their defaults.
 *
 * RETURNS: N/A
 *****/
IsoConfig::IsoConfig()
{
    output_root = "output";
    input_root = "input";

    color_scale = 1.0;
    color_offset = 0.0;
    use_prime_dir = false;
    use_auto_color = false;
    histogram_res = 16;
    img_res = 1024;
}

/*****
 * NAME: getImageFileName
 *
 * DESCRIPTION: Get the path+filename of the output
 *   PPM image file.
 *
 * RETURNS: The complete image output path.
 *****/
string IsoConfig::getImageFileName() const
{
    string filename = dataset_name + "." + suffix_name + ".ppm";
    if (output_root != "")
    {
        return output_root + "/" + filename;
    }
    else
    {
        return filename;
    }
}

/*****
 * NAME: getHistogramFileName
 *
 * DESCRIPTION: Get the path+filename of the
```

```

*   histogram output file.
*
* RETURNS: The complete histogram output path.
*****/
string IsoConfig::getHistogramFileName() const
{
    string filename = dataset_name + "-hist." + suffix_name + ".txt";
    if (output_root != "")
    {
        return output_root + "/" + filename;
    }
    else
    {
        return filename;
    }
}

/*****
* NAME: getSummaryFileName
*
* DESCRIPTION: Get the path+filename of the
*   summary output file.
*
* RETURNS: The complete summary output path.
*****/
string IsoConfig::getSummaryFileName() const
{
    string filename = summary_name + ".csv";
    if (output_root != "")
    {
        return output_root + "/" + filename;
    }
    else
    {
        return filename;
    }
}

/*****
* NAME: getInputFileName
*
* DESCRIPTION: Get the path+filename of the input
*   dataset.
*
* RETURNS: The complete input dataset path.
*****/
string IsoConfig::getInputFileName() const
{
    string filename = dataset_name + ".txt";
    if (input_root != "")
    {
        return input_root + "/" + filename;
    }
    else
    {
        return filename;
    }
}

/*****

```

```

* NAME: getDefaultVars
*
* DESCRIPTION: Insert all default variable
* mappings into dest_map.
*
* RETURNS: N/A
*****/
void IsoConfig::getDefaultVars(map<string, double> &dest_map)
{
    for (map<string,double>::iterator iter = default_vars.begin();
iter != default_vars.end(); ++iter)
    {
        dest_map[iter->first] = iter->second;
    }
}

/*****
* NAME: getInputVars
*
* DESCRIPTION: Insert all variable mappings from
* the input file into dest_map.
*
* RETURNS: N/A
*****/
void IsoConfig::getInputVars(map<string, double> &dest_map)
{
    for (map<string,double>::iterator iter = input_vars.begin(); iter
!= input_vars.end(); ++iter)
    {
        dest_map[iter->first] = iter->second;
    }
}

/*****
* NAME: getOverrideVars
*
* DESCRIPTION: Insert all override variable
* mappings into dest_map.
*
* RETURNS: N/A
*****/
void IsoConfig::getOverrideVars(map<string, double> &dest_map)
{
    for (map<string,double>::iterator iter = override_vars.begin();
iter != override_vars.end(); ++iter)
    {
        dest_map[iter->first] = iter->second;
    }
}

/*****
* NAME: getVars
*
* DESCRIPTION: Insert all variable mappings into
* dest_map. The relative precedences are taken
* into account by this function.
*
* RETURNS: N/A
*****/
void IsoConfig::getVars(map<string, double> &dest_map)

```

```

{
    getDefaultVars(dest_map);
    getInputVars(dest_map);
    getOverrideVars(dest_map);
}

/*****
 * NAME: parseDouble
 *
 * DESCRIPTION: This helper function parses a string
 *              representation of a double into a real double.
 *
 * RETURNS: The double equivalent to str.
 *****/
double IsoConfig::parseDouble(string str) const
{
    double result = 0.0;

    stringstream val_stream(str);
    val_stream >> result;

    char dummy_char;
    if (val_stream.get(dummy_char))
    {
        cerr << "Bad number format. Value was \'' << str <<
"\'\n";
        exit(1);
    }

    return result;
}

```

## ▣ isomap-configure.h

```

#ifndef ISOMAP_CONFIG_H
#define ISOMAP_CONFIG_H

#include <stdlib.h>
#include <string>
#include <map>

using namespace std;

/* Isomap Configuration Repository
 * This class assists in parsing the configuration values from the
 * command-line arguments.
 * When the class is first instantiated (through the default
 * constructor), it initializes itself
 * with the program's defaults. Then, the command-line parser in geom-
 * gen.cpp consumes the
 * command-line arguments and adds their values to the class instance.
 *
 * Once the command-line argument parsing is complete, this class
 * provides functions to retrieve
 * values that are derived from those arguments. */
class IsoConfig
{

```

```

private:
    double parseDouble(string str) const;

    /* Name of the dataset, which excludes the parent directory and
file extensions. */
    string dataset_name;
    /* Suffix for tracking the settings used to generate the output
files.
    * This suffix goes right before the file extension. */
    string suffix_name;
    /* Name of the summary file, excluding the parent directory and
file extension. */
    string summary_name;
    /* Implementation of the formula. */
    string formula;
    /* Parent directory for all output files. */
    string output_root;
    /* Parent directory for all input files. */
    string input_root;

    /* Scales a sensitivity value into a brightness value. */
    double color_scale;
    /* Offsets the brightness of all pixels. */
    double color_offset;

    /* When true, use the prime direction of each vector instead the
vectors' literal values. */
    bool use_prime_dir;
    /* When true, choose the color scale and offset automatically.
Automatic coloring chooses the best
    * scale and offset by looking at the minimum and maximum
sensitivities, then finding the best scale and
    * offset to force the image into an aesthetically-pleasing
range. */
    bool use_auto_color;

    /* Resolution of the histogram. The higher the resolution, the
more topo lines are drawn. */
    int histogram_res;
    /* Base resolution of the image. This value is the horizontal
resolution. The vertical resolution is half as big. */
    int img_res;

    /* These next three members are maps of names-to-values, so
formulas can use symbolic named values.
    * These values allow various benefits.
    * * They make the formulas easier to read and modify.
    * * They allow all datasets to use the same formulas, while
still allowing a dataset to have different numerical attributes. */

    /* Default values are passed in from the command line, and have
the lowest precedence. */
    map<string, double> default_vars;
    /* Variables defined in the dataset file. These are numerical
properties unique to the specimen or species. */
    map<string, double> input_vars;
    /* Override values are passed in from the command line with a
special flag, and have the highest precedence. They are intended for
temporarily

```

```

    * overriding the values set by the input files, enabling more
forms of analysis. */
    map<string, double> override_vars;

public:
    IsoConfig();

    /* Basic name and path functions. */

    void setDatasetName(string dataset_in) { dataset_name =
dataset_in; }
    string getDatasetName() const { return dataset_name; };

    void setSuffixName(string suffix_in) { suffix_name = suffix_in; }
    string getSuffixName() const { return suffix_name; };

    void setSummaryName(string summary_in) { summary_name =
summary_in; }
    string getSummaryName() const { return summary_name; };

    void setFormula(string formula_in) { formula = formula_in; }
    string getFormula() const { return formula; };

    void setOutputRoot(string root_dir_in) { output_root =
root_dir_in; }
    string getOutputRoot() const { return output_root; };

    void setInputRoot(string root_dir_in) { input_root = root_dir_in;
}
    string getInputRoot() const { return input_root; };

    /* Color scale functions. */

    void setUseAutoColor(bool use_auto_color_in) { use_auto_color =
use_auto_color_in; }
    bool getUseAutoColor() const { return use_auto_color; };

    void setColorScale(double scale_in) { color_scale = scale_in; }
    void setColorScale(string scale_in) { color_scale =
parseDouble(scale_in); }
    double getColorScale() const { return color_scale; };

    void setColorOffset(double offset_in) { color_offset = offset_in;
}
    void setColorOffset(string offset_in) { color_offset =
parseDouble(offset_in); }
    double getColorOffset() const { return color_offset; };

    /* Prime direction functions. */

    void setUsePrimeDir(bool use_prime_dir_in) { use_prime_dir =
use_prime_dir_in; }
    bool getUsePrimeDir() const { return use_prime_dir; };

    /* Histogram resolution functions. */

    void setHistogramRes(string res_in) { histogram_res =
atoi(res_in.c_str()); }
    int getHistogramRes() const { return histogram_res; }

```

```

    /* Image resolution functions. */
    void setImageRes(string res_in) { img_res = atoi(res_in.c_str());
}
    int getImageRes() const { return img_res; }
    int getImageVertRes() const { return img_res / 2; }
    int getImageHorizRes() const { return img_res; }

    /* Functions for adding and retrieving named variables for use by
the formulas. */

    void setDefaultVar(string var_name, double var_val)
{default_vars[var_name] = var_val; }
    void setDefaultVar(string var_name, string var_val) {
setDefaultVar(var_name, parseDouble(var_val)); }
    void getDefaultVars(map<string, double> &dest_map);

    void setInputVar(string var_name, double var_val) {
input_vars[var_name] = var_val; }
    void setInputVar(string var_name, string var_val) {
setInputVar(var_name, parseDouble(var_val)); }
    void getInputVars(map<string, double> &dest_map);

    void setOverrideVar(string var_name, double var_val) {
override_vars[var_name] = var_val; }
    void setOverrideVar(string var_name, string var_val) {
setOverrideVar(var_name, parseDouble(var_val)); }
    void getOverrideVars(map<string, double> &dest_map);

    void getVars(map<string, double> &dest_map);

    /* Filename functions. */

    string getImageFileName() const;
    string getHistogramFileName() const;
    string getSummaryFileName() const;
    string getInputFileName() const;

    bool getUseSummaryFile() const { return (summary_name != ""); }
};

#endif /* ISOMAP_CONFIG_H */

```

## ▣ isomap-gen.cpp

```

#include <iostream>
#include <iomanip>
#include <fstream>
#include <stdlib.h>
#include <sstream>
#include <math.h>
using namespace std;

#include "isomap-gen.h"
#include "isomap-prog.h"
#include "isomap-config.h"
#include "vector.h"

```

```

#define COLOR_RESOLUTION 1024

/* Sensitivity Value
 * This sensitivity value represents the output of the formula. The
 * units and meaning are not
 * strictly defined, but are instead defined by the formula. */
typedef double sens_t;

void print_usage(char *prog_name);
void generate_isomap_img(sens_t *topo_buf, int *hist_buf, IsoConfig
&cfg);
void get_weighted_canal_norms(vector<IsoVector> &canal_norms,
vector<string> &canal_names, IsoConfig &cfg);
void generate_prime_directions(vector<IsoVector> &canal_norms,
vector<string> &canal_names);
void generate_mono_isomap(IsoOperation *formula_prog, const
vector<IsoVector> &canal_norms, const vector<string> &canal_names,
sens_t *topo_buf, IsoConfig &cfg);
void generate_histogram(int *histogram_out, int *hist_samples_out,
sens_t &mono_min_out, sens_t &mono_max_out, sens_t &mono_step_out,
sens_t *topo_buf, IsoVector &min_loc_out, IsoVector &max_loc_out,
IsoConfig &cfg);
void output_histogram(int *histogram_stats, sens_t mono_min, sens_t
mono_max, sens_t mono_step, vector<IsoVector> &canal_norms,
vector<string> &canal_names, IsoVector min_loc, IsoVector max_loc,
IsoConfig &cfg);
int get_histogram_idx(sens_t mono_val, sens_t mono_min, sens_t
mono_step, IsoConfig &cfg);
int is_topo_line(int *hist_buf, int col_i, int row_i, IsoConfig &cfg);
bool is_thick_topo_line(sens_t *topo_buf, int col_i, int row_i, sens_t
mono_min, sens_t mono_step, IsoConfig &cfg);
int get_mono_raw_pix_val(sens_t pix_val, IsoConfig &cfg);
void output_mono_pix(ofstream &outpic, sens_t pix_val, IsoConfig &cfg);
void output_vector_loc(ofstream &outhist, IsoVector vect, IsoConfig
&cfg);

/*****
 * NAME: main
 *
 * DESCRIPTION: This is the main routine for the
 * isomap-gen program. The operating system
 * passes in the command-line arguments into
 * argv[], and uses argc to indicate how many
 * arguments were provided.
 *
 * RETURNS: 0 if the dataset was processed
 * successfully. Otherwise, it returns the line
 * number where processing failed. When it fails,
 * this program's output is invalid and must be
 * ignored.
 *****/
int main(
    int argc, /* Number of strings in argv[]. */
    char **argv) /* An array of strings holding the command
line arguments. */
{
    /* The command line arguments define what it must do, and if
there is only one argument (the program name

```

```

    * that's always passed in by the OS), the user hasn't specified
enough about the job. */
    if (argc < 2)
    {
        cout << "Too few arguments\n";
        print_usage(argv[0]);
        exit(__LINE__);
    }

    /* Gather all the configuration information into this IsoConfig
object. */
    IsoConfig cfg;

    /* Iterate over every command line argument except the 0th one
(the program name).
    * Each argument consumes at least one element (the argument
name) from argv[], but
    * most take 1 or 2 extra elements from argv[]. */
    int curr_arg = 1;
    while (curr_arg < argc)
    {
        /* Every option begins with the argument name, so consume
it here and use it
        * to decide what to do next. */
        string arg_type = argv[curr_arg];

        if (arg_type == "--auto-color")
        {
            /* --auto-color has no other arguments. */
            cfg.setUseAutoColor(true);

            /* Go on to parse another option after --auto-color.
*/
            ++curr_arg;
        }
        else if (arg_type == "--prime-dir")
        {
            /* --prime-dir has no other arguments. */
            cfg.setUsePrimeDir(true);

            /* Go on to parse another option after --prime-dir.
*/
            ++curr_arg;
        }
        else if (arg_type == "--default-arg")
        {
            /* --default-arg takes two arguments, excluding the -
default-arg name itself.
            * Verify that it will not run out of argv[] elements
before this option is complete. */
            if (curr_arg + 2 >= argc)
            {
                cerr << "No argument name and/or value
specified for argument type \'' << argv[curr_arg] << "'\n";
                print_usage(argv[0]);
                exit(__LINE__);
            }

            /* Extract the two pieces of information that define
the --default-arg option. */

```

```

        string arg_name = argv[curr_arg+1];
        string arg_val = argv[curr_arg+2];
        /* Increment the counter past this option. */
        curr_arg += 3;

        /* Finally, add the information to the configuration.
*/
        cfg.setDefaultVar(arg_name, arg_val);
    }
    else if (arg_type == "--override-arg")
    {
        /* The algorithm for --override-arg is almost
identical to --default-arg, except cfg.setOverrideVar() is used
* instead of cfg.setDefaultVar(). */

        if (curr_arg + 2 >= argc)
        {
            cerr << "No argument name and/or value
specified for argument type \'' << argv[curr_arg] << "'\n";
            print_usage(argv[0]);
            exit(__LINE__);
        }

        string arg_name = argv[curr_arg+1];
        string arg_val = argv[curr_arg+2];
        curr_arg += 3;

        cfg.setOverrideVar(arg_name, arg_val);
    }
    else
    {
        /* All options that take 0 or 2 extra pieces of
information have been considered, and by this point
* they are ruled out. That only leaves options
taking 1 extra piece of information.
*
* Verify that there is enough elements left in
argv[] to contain at least one extra piece of information. */
        if (curr_arg + 1 >= argc)
        {
            cerr << "No value for argument type \'' <<
argv[curr_arg] << "'\n";
            print_usage(argv[0]);
            exit(__LINE__);
        }

        /* Extract the argument and increment on to the next
option. */
        string arg_value = argv[curr_arg+1];
        curr_arg += 2;

        /* Detect the option name so the extra argument can
be deposited in the right place.
* NOTE: If arg_value is actually supposed to be a
real number or integer, the cfg.set___
* function will take care of that conversion. */
        if (arg_type == "--dataset")
        {
            cfg.setDatasetName(arg_value);
        }
    }
}

```

```

else if(arg_type == "--color-scale")
{
    cfg.setColorScale(arg_value);
}
else if(arg_type == "--color-offset")
{
    cfg.setColorOffset(arg_value);
}
else if(arg_type == "--out-suffix")
{
    cfg.setSuffixName(arg_value);
}
else if (arg_type == "--preset")
{
    /* Detect which preset was chosen. The preset
name must match one
recognized, set its equivalent
options into cfg. */
    string calc_method_name = arg_value;
    if (calc_method_name == "HULLAR")
    {
        cfg.setFormula("(mag (sumv ?i (*v (norm
=i ) (dot =axis (norm =i )))))");
        cfg.setColorScale(1.0 / 3.0);
        cfg.setColorOffset(0.0);
        cfg.setSuffixName("h");
    }
    else if (calc_method_name == "RODGERS_ORIG")
    {
        cfg.setFormula("/ (sum ?i (pos (dot
=axis =i ))) (sum ?i (mag =i )))");
        cfg.setColorScale(1.0 / 0.37);
        cfg.setColorOffset(0.0);
        cfg.setSuffixName("ro");
    }
    else if (calc_method_name == "RODGERS_NEG")
    {
        cfg.setFormula("/ (sum ?i (neg (dot
=axis =i ))) (sum ?i (mag =i )))");
        cfg.setColorScale(0.5 / 0.15);
        cfg.setColorOffset(0.5);
        cfg.setSuffixName("rn");
    }
    else if (calc_method_name == "RODGERS_ABS")
    {
        cfg.setFormula("/ (sum ?i (abs (dot
=axis =i ))) (sum ?i (mag =i )))");
        cfg.setColorScale(1.0 / 0.67);
        cfg.setColorOffset(0.0);
        cfg.setSuffixName("ra");
    }
    else if (calc_method_name ==
"RODGERS_ABS_GAIN")
    {
        cfg.setFormula("(sum ?i (abs (* (- (*
@gain_slope (mag =i )) @gain_ofs ) (dot =axis (norm =i )))))");
        cfg.setColorScale(1.0 / 0.67);
        cfg.setColorOffset(0.0);

```

```

        cfg.setSuffixName("ra");
    }
    else
    {
        /* The preset name might have a typo, and
the image this run generates will not be what the user
        * expected. Abort and give the user a
chance to revise the options. */
        cout << "Unrecognized preset name \"" <<
calc_method_name << "\"\n";
        print_usage(argv[0]);
        exit(__LINE__);
    }
}
else if (arg_type == "--summary-file")
{
    cfg.setSummaryName(arg_value);
}
else if (arg_type == "--formula")
{
    cfg.setFormula(arg_value);
}
else if (arg_type == "--hist-res")
{
    cfg.setHistogramRes(arg_value);
}
else if (arg_type == "--img-res")
{
    cfg.setImageRes(arg_value);
}
else
{
    /* All possibilities were considered, and the
option was not recognized.
        * Abort to let the user know about problem. */
    cerr << "Unknown parameter type \"" << arg_type
<< "\"\n";
    print_usage(argv[0]);
    exit(__LINE__);
}
}

/* Go back to the start of the loop to process the next
option (if there are any
    * more elements left in argv[]. */
}

/* By this point, the configuration is complete.
    * Verify that the mandatory fields are actually present. */
if (cfg.getDatasetName() == "")
{
    cerr << "No dataset name was provided! You must provide
the --dataset option.\n";
    print_usage(argv[0]);
    exit(__LINE__);
}

if (cfg.getFormula() == "")
{

```

```

    cerr << "No formula specified! You must use the --formula
or --preset option.\n";
    print_usage(argv[0]);
    exit(__LINE__);
}

/* Every other option takes reasonable defaults. */
IsoOperation *formula_prog = parse_formula(cfg);

vector<IsoVector> canal_norms;
vector<IsoVector> canal_prime_dir_norms;
vector<string> canal_names;

/* Allocate all the sensitivity samples of the sensitivity plot.
 * Even though the plot is 2D (bearing and pitch) allocating it as
a
 * 1D array makes it much easier and more efficient to iterate
over. */
sens_t *topo_buf = new sens_t[cfg.getImageHorizRes() *
cfg.getImageVertRes()];
int *hist_buf = new int[cfg.getImageHorizRes() *
cfg.getImageVertRes()];

sens_t mono_min;
sens_t mono_max;
sens_t mono_step;

/* Allocate the histogram interval counters.
 * This will count how many samples occur within each histogram
range. */
int *histogram_stats = new int[cfg.getHistogramRes()];

/* Print the progress to the user. The process takes a long
time, and the user might
 * give up after seeing too much silence. */

cout << "Processing Dataset: " << cfg.getDatasetName() << "\n";
cout << "Generating sensitivities...\n";

/* Read the canal names, normals, radii, and input variables
from the input file.
 * The input variables are deposited into cfg, and the names and
normal*radius vectors
 * are passed up through the other parameters. */
get_weighted_canal_norms(canal_norms, canal_names, cfg);

/* Compute the prime directions. */
canal_prime_dir_norms = canal_norms;
generate_prime_directions(canal_prime_dir_norms, canal_names);

if (cfg.getUseSummaryFile())
{
    /* The user specified a summary file, so attempt to open it
up to write to the end. */
    ofstream summary_file;
    string summary_filename = cfg.getSummaryFileName();
    /* Open it, but specify to append more lines to the file
(instead of erasing the current contents first). */

```

```

        summary_file.open(summary_filename.c_str(), ios_base::app |
ios_base::out);
        if (summary_file.fail())
        {
            /* The summary file couldn't be opened. Don't waste
the user's time by doing the rest of the computations. */
            cerr << "Failed to open summary file!\n";
            print_usage(argv[0]);
            exit(__LINE__);
        }
        else
        {
            cout << "Outputting summary...\n";
            /* Configure the output text to only include 4 digits
past the decimal point.
* Too many digits are hard to read, and would imply
more significant figures
* than the input data probably has. */
            summary_file << setprecision(4);
            summary_file << fixed;

            /* Append one line per canal. */
            for (int i = 0; i < (int)canal_norms.size(); ++i)
            {
                IsoVector temp_norm;

                summary_file << cfg.getDatasetName() << "," <<
cfg.getSuffixName() << "," << canal_names[i] << "," <<
                << canal_norms[i].magnitude() << ",";

                temp_norm = canal_norms[i].normalize();
                summary_file << temp_norm.x << "," <<
temp_norm.y << "," << temp_norm.z << ",";

                temp_norm =
canal_prime_dir_norms[i].normalize();
                summary_file << temp_norm.x << "," <<
temp_norm.y << "," << temp_norm.z << ",";

                summary_file <<
acos(canal_norms[i].normalize().dotProduct(temp_norm)) *
DEGREES_PER_RADIAN << "\n";
            }

            /* The summary output is complete now. */
            summary_file.close();
        }
    }

    if (cfg.getUsePrimeDir())
    {
        /* The user selected to use the prime directions, so
replace the true canal vectors with
* their prime directions before inputting them into the
formula. */
        canal_norms = canal_prime_dir_norms;
    }

    /* Generate the sensitivity samples into topo_buf. */

```

```

    generate_mono_isomap(formula_prog, canal_norms, canal_names,
topo_buf, cfg);

    cout << "Generating histogram...\n";

    IsoVector min_loc;
    IsoVector max_loc;
    /* Count how many samples occur within each histogram range. */
    generate_histogram(histogram_stats, hist_buf, mono_min, mono_max,
mono_step, topo_buf, min_loc, max_loc, cfg);
    /* Use the histogram range statistics to output the histogram
text file. */
    output_histogram(histogram_stats, mono_min, mono_max, mono_step,
canal_norms, canal_names, min_loc, max_loc, cfg);

    if (cfg.getUseAutoColor())
    {
        /* The user selected auto-coloring. Use the minimum and
maximum sample values to compute
        * how to scale the brightness and contrast. */

        /* The color range must be between 0.0 and 1.0, or else it
will saturate and be useless.
        * If the minimum were set to 0.0, figure out how to end up
with a perfect 1.0 (white) only
        * at the maximum sample value. */
        double color_scale = 1.0 / (mono_max - mono_min);
        /* Now, figure out how to make it a perfect 0.0 black only
at the minimum sample value. */
        double color_offset = -mono_min * color_scale;

        /* As it is now, the graph would be 0.0 to 1.0, but it
would still be hard to read.
        * Black topo lines don't show up at all in very dark
regions. Adjust the 0.0-1.0 range
        * into a 0.2-1.0 range. */
        color_scale *= 0.8;
        color_offset += 0.2;

        /* Set the computed color offset and scale into the
configuration. */
        cfg.setColorScale(color_scale);
        cfg.setColorOffset(color_offset);
    }

    cout << "Generating image...\n";
    /* Iterate through the topo_buf samples, and generate a complete
PPM image file. */
    generate_isomap_img(topo_buf, hist_buf, cfg);
    cout << "Done!\n\n";

    delete [] topo_buf;
    delete [] hist_buf;
    delete [] histogram_stats;
}

/*****
* NAME: get_weighted_canal_norms
*
* DESCRIPTION: Read the dataset input file

```

```

*   (specified by cfg) and read in the canal names
*   and weighted normals.
*
*   After it reads the canal data, it reads in any
*   input variables, if any are present in the file.
*   The input variable mappings are deposited into
*   cfg.
*
* RETURNS: N/A
*****/
void get_weighted_canal_norms(vector<IsoVector> &canal_norms,
vector<string> &canal_names, IsoConfig &cfg)
{
    /* The file lists the normals and radii separately. Use
    canal_radii to hold the radii
    * until they are used to scale the normals into the weighted
    normals. */
    double canal_radii[6];

    /* Get the input dataset filename, and attempt to open it. */
    string infile_name = cfg.getInputFileName();
    ifstream indata(infile_name.c_str());
    if (indata.fail())
    {
        cout << "Failed to open " << infile_name << "\n";
        exit(__LINE__);
    }

    /* Read in the canal records. There must be 6 of them. */
    for (int i = 0; i < 6; ++i)
    {
        double x, y, z;
        string curr_name;

        indata >> curr_name;
        canal_names.push_back(curr_name);

        indata >> canal_radii[i];

        indata >> x >> y >> z;
        canal_norms.push_back(IsoVector(x, y, z));

        canal_norms[i] =
canal_norms[i].normalize().scale(canal_radii[i]);
    }

    /* Add the free-form parameters from the input file. */
    string arg_name;
    string arg_val;
    indata >> arg_name >> arg_val;
    while (!indata.eof())
    {
        /* Keep attempting to read in input variable mappings until
        the file indicates
        * there is no more to read. */
        cfg.setInputVar(arg_name, arg_val);
        indata >> arg_name >> arg_val;
    }

    indata.close();

```

```

}

/*****
* NAME: generate_prime_directions
*
* DESCRIPTION: Convert original weighted normals in
*   canal_norms into their prime directions.
*
*   Converting a canal into its prime direction
*   depends on its identity, and it it requires
*   knowing which canals are ipsilateral and which
*   order to do the cross product. Therefore, it
*   relies on the normals being named LASC, LPSC,
*   LLSC, RASC, RPSC, Or RLSC.
*
* RETURNS: N/A
*****/
void generate_prime_directions(vector<IsoVector> &canal_norms,
vector<string> &canal_names)
{
    map<string,double> radius_map;
    map<string,IsoVector> norm_map;
    /* Specify the names of the two input canals (prime_dir_ops[?][0]
and prime_dir_ops[?][1]) for each
    * output prime direction name (prime_dir_ops[?][2]). */
    string prime_dir_ops[6][3] =
    {
        {"LASC","LPSC","LLSC"},
        {"LLSC","LASC","LPSC"},
        {"LPSC","LLSC","LASC"},
        {"RASC","RPSC","RLSC"},
        {"RLSC","RASC","RPSC"},
        {"RPSC","RLSC","RASC"}
    };

    /* Convert the canal_norms and canal_names arrays into maps, to
make it
    * easier to look up specific canals as they're needed.
    *
    * It also keeps the input data safe as canal_norms is being
updated. */
    for (int i = 0; i < (int)canal_norms.size(); ++i)
    {
        radius_map[canal_names[i]] = canal_norms[i].magnitude();
        norm_map[canal_names[i]] = canal_norms[i].normalize();
    }

    /* For each canal, compute its normal. */
    for (int i = 0; i < (int)canal_norms.size(); ++i)
    {
        /* Look up the input operands. */
        IsoVector lhs_norm = norm_map[prime_dir_ops[i][0]];
        IsoVector rhs_norm = norm_map[prime_dir_ops[i][1]];
        /* Retrieve the canal's non-prime-direction weighted
normal.
this, or
        * The prime direction result must be within 90 degrees of
        * else it needs to be inverted 180 degrees. */
        IsoVector approx_norm = norm_map[prime_dir_ops[i][2]];

```

```

        /* Compute the prime direction itself. */
        IsoVector prime_norm = lhs_norm.crossProduct(rhs_norm);
        if (prime_norm.dotProduct(approx_norm) < 0.0)
        {
            /* The result was more than 90 degrees away from the
original vector, so
            * flip it 180 degrees. */
            prime_norm = prime_norm.scale(-1.0);
        }

        /* The cross product includes a trigonometric term that is
undesireable in this context.
        * Re-normalize it (to magnitude 1.0) and then further
scale it to have the radius of the original vector. */
        prime_norm =
prime_norm.normalize().scale(radius_map[prime_dir_ops[i][2]]);

        /* Find the original destination for this canal. */
        for (int j = 0; j < (int)canal_norms.size(); ++j)
        {
            if (canal_names[j] == prime_dir_ops[i][2])
            {
                /* Found the correct index j to deposit the new
weighted normal. */
                canal_norms[j] = prime_norm;
            }
        }
    }
}

/*****
* NAME: generate_mono_isomap
*
* DESCRIPTION: Use formula_prog and the canals in
* canal_norms to generate all the sensitivity
* samples.
*
* RETURNS: N/A
*****/
void generate_mono_isomap(
    IsoOperation *formula_prog, /* The parsed formula to use to
generate the samples. */
    const vector<IsoVector> &canal_norms, /* The array of the
weighted canal normals. */
    const vector<string> &canal_names, /* A parallel array with
the name of each canal vector in canal_norms. */
    sens_t *topo_buf, /* Destination for the sample data. */
    IsoConfig &cfg) /* Configuration values, such as variable
mappings and resolution. */
{
    map<string, IsoVector> bound_vars;
    for (int i = 0; i < (int)canal_norms.size(); ++i)
    {
        /* Create a variable mapping out of each canal.
        * Some formulas may refer to specific canals. */
        string curr_name = "canal";
        char canal_idx_char = '0' + i;
        curr_name.append(&canal_idx_char, 1);
        /* Create one mapping with a generic "canalN" name, where

```

```

        * N is the canal's index as it appears in the input file.
*/
    bound_vars[curr_name] = canal_norms[i];
    /* Create another mapping using the canal's name from the
input file. */
    bound_vars[canal_names[i]] = canal_norms[i];
}

    /* Obtain the scalar variable mappings from cfg. */
    map<string, double> bound_scalars;
    cfg.getVars(bound_scalars);

    /* Compute how many degrees to increment when moving right by one
sample. */
    double vert_step = 180.0 / (double)(cfg.getImageVertRes());
    /* Compute how many degrees to increment when moving down by one
sample. */
    double horiz_step = 360.0 / (double)cfg.getImageHorizRes();

    /* Generate the sensitivity values. The outermost loop loops
over each row. */
    for (int row_i = 0; row_i < cfg.getImageVertRes(); ++row_i)
    {
        /* Compute what the pitch is at this row of samples. */
        double pitch = 90.0 - (double)row_i * vert_step;

        /* Loop over each column within this row. */
        for (int col_i = 0; col_i < cfg.getImageHorizRes();
++col_i)
        {
            /* Compute what the bearing is at this sample column.
*/
            double bearing = -180.0 + (double)col_i * horiz_step;

            /* Convert the bearing and pitch into a Cartesian
normal vector, defining the axis of rotation. */
            IsoVector rot_axis(bearing, pitch);

            /* Add the rotation axis as a bound variable, so the
formula can refer to it. */
            bound_vars["axis"] = rot_axis;
            /* Pass all this information to the formula, to
generate the sensitivity value for this sample.
* This line also increments topo_buf to the next
sample afterward. */
            *(topo_buf++) = formula_prog->scalarValue(bound_vars,
canal_norms, bound_scalars);
        }
    }
}

/*****
* NAME: generate_histogram
*
* DESCRIPTION: Compute the histogram statistics
* from the sample buffer.
*
* RETURNS: N/A
*****/
void generate_histogram(

```

```

counts. */    int *histogram_out, /* Destination array for the histogram
histogram index of each sample. */
value. */    int *hist_samples_out, /* Destination buffer for the
value. */    sens_t &mono_min_out, /* Destination for the minimum sample
value. */    sens_t &mono_max_out, /* Destination for the maximum sample
histogram range. */    sens_t &mono_step_out, /* Destination for the width of each
histogram. */    sens_t *topo_buf, /* Sample buffer for which to compute the
IsoVector &min_loc_out, /* Destination for the normal
vector pointing at the minimum sample's location. */
IsoVector &max_loc_out, /* Destination for the normal
vector pointing at the maximum sample's location. */
IsoConfig &cfg) /* Extra configuration data for the
histogram. */
{
    /* Initialize all histogram counters to 0. */
    for (int i = 0; i < cfg.getHistogramRes(); ++i)
    {
        histogram_out[i] = 0;
    }

    /* Search for the minimum and maximum sample values and
locations. */

    /* Setup the iterators */
    sens_t *topo_buf_iter = topo_buf;
    sens_t *topo_buf_end = topo_buf + cfg.getImageHorizRes() *
cfg.getImageVertRes();

    /* Start at some sane values. */
    mono_min_out = topo_buf[0];
    mono_max_out = topo_buf[0];
    min_loc_out = IsoVector(0, 0, cfg.getImageRes());
    max_loc_out = IsoVector(0, 0, cfg.getImageRes());

    /* Iterate over all samples. */
    while (topo_buf_iter < topo_buf_end)
    {
        if (*topo_buf_iter < mono_min_out)
        {
            /* This value is smaller than the currently-known
minimum.
            * Replace the obsolete minimum with this one. */
            mono_min_out = *topo_buf_iter;
            int pixel_y = (topo_buf_iter - topo_buf) /
cfg.getImageHorizRes();
            int pixel_x = (topo_buf_iter - topo_buf) %
cfg.getImageHorizRes();
            min_loc_out = IsoVector(pixel_x, pixel_y,
cfg.getImageRes());
        }
        else if (*topo_buf_iter > mono_max_out)
        {
            /* This value is smaller than the currently-known
maximum.
            * Replace the obsolete maximum with this one. */

```

```

        mono_max_out = *topo_buf_iter;
        int pixel_y = (topo_buf_iter - topo_buf) /
cfg.getImageHorizRes();
        int pixel_x = (topo_buf_iter - topo_buf) %
cfg.getImageHorizRes();
        max_loc_out = IsoVector(pixel_x, pixel_y,
cfg.getImageRes());
    }
    /* ELSE: The currently-known minimum and maximum still hold
their titles. */

    /* Increment to the next sample. */
    ++topo_buf_iter;
}

/* Compute how wide each histogram range is, to split the range
between the minimum and maximum into
* histogram_resolution equal ranges. */
mono_step_out = (mono_max_out - mono_min_out) /
(sens_t)cfg.getHistogramRes();

/* Iterate over all the samples again to fill in the histogram */
topo_buf_iter = topo_buf;
while (topo_buf_iter < topo_buf_end)
{
    *hist_samples_out = get_histogram_idx>(*topo_buf_iter++),
mono_min_out, mono_step_out, cfg);
    ++histogram_out[*hist_samples_out++];
}
}

/*****
* NAME: output_histogram
*
* DESCRIPTION: Use the canal data and the
* information computed from generate_histogram()
* to output the histogram text file.
*
* RETURNS: N/A
*****/
void output_histogram(
    int *histogram_stats, /* Array of histogram counters,
generated by generate_histogram(). */
    sens_t mono_min, /* The maximum sample value, found by
generate_histogram(). */
    sens_t mono_max, /* The minimum sample value, found by
generate_histogram(). */
    sens_t mono_step, /* The width of each histogram region,
generated by generate_histogram() */
    vector<IsoVector> &canal_norms, /* Array of the weighted
canal normals. */
    vector<string> &canal_names, /* Array of the canal names,
parallel to the canal_norms array. */
    IsoVector min_loc, /* Location of the minimum sample, found
by generate_histogram(). */
    IsoVector max_loc, /* Location of the maximum sample, found
by generate_histogram(). */
    IsoConfig &cfg) /* Histogram output configuration
parameters. */
{

```

```

    string hist_filename = cfg.getHistogramFileName();
    ofstream outhist(hist_filename.c_str());
    if (outhist.fail())
    {
        cerr << "Failed to open histogram output file " <<
hist_filename << "\n";
        exit(__LINE__);
    }

    /* Configure the output file to only use fixed-point and 4 digits
past the decimal point.
    * Otherwise, it is really hard to read, and can produce a
misleading number of significant
    * figures. */
    outhist << setprecision(4);
    outhist << fixed;

    /* Output the basic information. */
    outhist << "**** HISTOGRAM ****\n";
    outhist << "Dataset Name: " << cfg.getDatasetName() <<
"\n";
    outhist << "Maximum: " << mono_max << "\n";
    output_vector_loc(outhist, max_loc, cfg);
    outhist << "Minimum: " << mono_min << "\n";
    output_vector_loc(outhist, min_loc, cfg);
    outhist << "Maximum/Minimum: " << (mono_max / mono_min) <<
"\n";
    outhist << "Contour step size: " << mono_step << "\n";
    outhist << "Green Contour Line: " << (mono_max - 2 * mono_step)
<< "\n";
    outhist << "Red Countour Line: " << (mono_min + mono_step) <<
"\n";

    /* Output the canal data. */
    for (int i = 0; i < (int)canal_norms.size(); ++i)
    {
        outhist << "Canal " << canal_names[i] << ":\n";
        output_vector_loc(outhist, canal_norms[i].normalize(),
cfg);
        outhist << "\tRadius: " <<
canal_norms[i].magnitude() << "\n";
    }

    /* Output the histogram regions. */
    for (int i = 0; i < cfg.getHistogramRes(); ++i)
    {
        outhist << (mono_min + (sens_t)i * mono_step)
        << " <= x < "
        << (mono_min + (sens_t)(i + 1) * mono_step)
        << ":\n";
        << histogram_stats[i]
        << "\n";
    }

    /* The histogram file is now complete. */
    outhist.close();
}

/*****
* NAME: generate_isomap_img

```

```

*
* DESCRIPTION: Use the samples in topo_buf to
* generate the plot image file.
*
* RETURNS: N/A
*****/
void generate_isomap_img(
    sens_t *topo_buf, /* The array of samples generated by
generate_mono_isomap(). */
    int *hist_buf, /* Buffer, parallel to topo_buf, with the
histogram index of each sample. */
    IsoConfig &cfg) /* Configuration values. */
{
    string outpic_name = cfg.getImageFileName();
    ofstream outpic(outpic_name.c_str());

    /* Output the picture headers.
    * These tell what type of file it is, how big it is, and how
detailed the color is. */
    outpic << "P3\n"
        << "# unused comment\n"
        << (2*cfg.getImageHorizRes()) << " " <<
cfg.getImageVertRes() << '\n'
        << COLOR_RESOLUTION << '\n';

    /* Retrieve the image's dimensions from the configuration. */
    int row_max = cfg.getImageVertRes();
    int col_max = cfg.getImageHorizRes();
    int img_res = cfg.getImageRes();

    /* Output the picture, row by row. */
    for (int row_i = 0; row_i < row_max; ++row_i)
    {
        /* Determine at which index this row of samples begins
within topo_buf. */
        int topo_row_start = row_i * img_res;

        /* Output the right half of the topo_buf so that it wraps
around
        * to the "real" start of this row. */
        for (int col_i = img_res / 2; col_i < img_res - 1; ++col_i)
        {
            output_mono_pix(outpic, topo_buf[topo_row_start +
col_i], cfg);
        }

        /* Output a vertical black bar to show where the row really
begins. */
        outpic << "0 0 0 ";

        /* Output the real row. */
        for (int col_i = 0; col_i < col_max; ++col_i)
        {
            int pix_idx = row_i * col_max + col_i;
            int topo_line_type = is_topo_line(hist_buf, col_i,
row_i, cfg);

            if (topo_line_type != 0)
            {

```

```

- 2) if (hist_buf[pix_idx] == cfg.getHistogramRes()
{
    /* Green maximum line. */
    outpic << "128 560 96 ";
}
else if (hist_buf[pix_idx] == 0)
{
    /* Red minimum line. */
    outpic << "772 212 212 ";
}
else if (topo_line_type == 1)
{
    /* Output a black pixel for all other
topo lines. */
    outpic << "0 0 0 ";
}
else
{
    /* False alarm. This pixel is on a wide
topo line, but only the minimum and maximum topo lines should
pixel's grayscale pixel. */
    * be displayed as wide. Output the
    output_mono_pix(outpic,
topo_buf[topo_row_start + col_i], cfg);
}
}
else
{
    /* The pixel is definitely not a topo line, so
output its grayscale pixel. */
    output_mono_pix(outpic, topo_buf[topo_row_start
+ col_i], cfg);
}
}
}

/* Output a vertical black bar to show where the row really
ends. */
outpic << "0 0 0 ";

/* Output the left half of the topo_buf so that it wraps
around
* past the "real" end of this row. */
for (int col_i = 1; col_i < img_res / 2; ++col_i)
{
    output_mono_pix(outpic, topo_buf[topo_row_start +
col_i], cfg);
}

/* Begin the next row. */
outpic << '\n';
}

/* The image output is complete. */
outpic.close();
}

/*****
* NAME: get_histogram_idx
*

```

```

* DESCRIPTION: Compute the histogram index for a
* sample. The index it generates is bounds-
* checked and safe to use directly in arrays.
*
* RETURNS: The index of the sample's histogram
* region.
*****/
int get_histogram_idx(sens_t mono_val, sens_t mono_min, sens_t
mono_step, IsoConfig &cfg)
{
    int result = (int)((mono_val - mono_min) / mono_step);

    /* These histogram indices are used as array indices, so bounds
must be enforced.
    * Rounding errors may cause minimum or maximum pixels to go out
of bounds and crash the program. */
    if (result >= cfg.getHistogramRes())
    {
        result = cfg.getHistogramRes() - 1;
    }
    else if (result < 0)
    {
        result = 0;
    }

    return result;
}

/*****
* NAME: is_topo_line
*
* DESCRIPTION: Decide whether this pixel lies on
* a histogram boundary, and should be displayed
* as a topo line.
*
* A sample is on a topo line if any of its
* neighbors (above, below, left, right, or
* diagonal) belong to a different histogram
* region.
*
* RETURNS: 1 if the sample is directly on a thin
* topo line (directly on it); 2 if the sample is
* on a thick topo line (adjacent to the topo line),
* but not on thi thin topo line, or 0 if the
* sample is not on a thin or thick topo line.
*****/
int is_topo_line(int *hist_buf, int col_i, int row_i, IsoConfig &cfg)
{
    int pix_idx = row_i * cfg.getImageRes() + col_i;
    int cur_hist_idx = hist_buf[pix_idx];

    /* If this pixel is not already on the top row, check if the
sample above it belongs to a different histogram region. */
    if (row_i > 0 &&
        cur_hist_idx < hist_buf[pix_idx - cfg.getImageHorizRes()])
    {
        return 1;
    }
}

```

```

    /* If this pixel is not already on the leftmost column, check if
the sample to the
    * left of it belongs to a different histogram region. */
    if (col_i > 0 &&
        cur_hist_idx < hist_buf[pix_idx - 1])
    {
        return 1;
    }

    /* If this pixel is not already on the bottom row, check if the
sample below it belongs to a different histogram region. */
    if (row_i < cfg.getImageVertRes() - 1 &&
        cur_hist_idx < hist_buf[pix_idx + cfg.getImageHorizRes()])
    {
        return 1;
    }

    /* If this pixel is not already on the rightmost column, check if
the sample to the
    * right of it belongs to a different histogram region. */
    if (col_i < cfg.getImageHorizRes() - 1 &&
        cur_hist_idx < hist_buf[pix_idx + 1])
    {
        return 1;
    }

    /* Test if the pixel is a distance of 1 away from a topo line. */
    if (row_i > 1 &&
        cur_hist_idx < hist_buf[pix_idx - 2 *
cfg.getImageHorizRes()])
    {
        return 2;
    }

    if (col_i > 1 &&
        cur_hist_idx < hist_buf[pix_idx - 2])
    {
        return 2;
    }

    if (row_i < cfg.getImageVertRes() - 2 &&
        cur_hist_idx < hist_buf[pix_idx + 2 *
cfg.getImageHorizRes()])
    {
        return 2;
    }

    if (col_i < cfg.getImageHorizRes() - 2 &&
        cur_hist_idx < hist_buf[pix_idx + 2])
    {
        return 2;
    }

    /* The sample is not on the thin or thick topo line. */
    return 0;
}

/*****
* NAME: get_mono_raw_pix_val

```

```

*
* DESCRIPTION: Convert the sample into a grayscale
*             color value.  When the pixel is output, this
*             grayscale value should be repeated 3 times,
*             once for each color channel.
*
* RETURNS: The grayscale brightness value of this
*          pixel.
*****/
int get_mono_raw_pix_val(
    sens_t pix_val, /* The sample value for which to compute
the grayscale brightness. */
    IsoConfig &cfg) /* Image configuration.  Provides the color
scale, offset, and color resolution. */
{
    /* Use the color scale and offset to compute the unclamped value.
*/
    int raw_pix_val = (int)(cfg.getColorScale() * pix_val *
COLOR_RESOLUTION + cfg.getColorOffset() * COLOR_RESOLUTION);

    if (raw_pix_val < 0)
    {
        /* The color settings tried to make this pixel blacker than
black.  Clamp it to 0, to make sure
* that the image viewer will not display it funny, or
possibly even crash. */
        raw_pix_val = 0;
    }
    else if (raw_pix_val >= COLOR_RESOLUTION)
    {
        /* The color settings made this pixel whiter than white, so
clamp it to the maximum value. */
        raw_pix_val = COLOR_RESOLUTION - 1;
    }

    return raw_pix_val;
}

/*****
* NAME: output_mono_pix
*
* DESCRIPTION: Take one sample, and output a
*             complete pixel representing it as a grayscale
*             value.
*
* RETURNS: N/A
*****/
void output_mono_pix(
    ofstream &outpic, /* The destination output file. */
    sens_t pix_val, /* The sample value to represent in
grayscale. */
    IsoConfig &cfg) /* The image configuration. */
{
    /* Get the brightness value. */
    int raw_pix_val = get_mono_raw_pix_val(pix_val, cfg);

    /* Output the brightness value for all 3 color channels (red,
green, and blue) to
* create a grayscale pixel. */

```

```

    outpic << raw_pix_val << " " << raw_pix_val << " " << raw_pix_val
<< " ";
}

/*****
* NAME: output_vector_loc
*
* DESCRIPTION: Output a normal vector in several
*   formats.
*
* RETURNS: N/A
*****/
void output_vector_loc(
    ofstream &outhist, /* The destination output file. */
    IsoVector vect, /* The vector to output. */
    IsoConfig &cfg) /* The configuration, to provide the image
resolution. */
{
    /* Output its Cartesian values. */
    outhist << "\t{X,Y,Z}: {" << vect.x << "," << vect.y <<
    "," << vect.z << "}\n";
    /* Output it as a spherical coordinate with bearing and pitch. */
    outhist << "\t{bearing,pitch}: {" << vect.getBearing() << "," <<
    vect.getPitch() << "}\n";
    /* Output the pixel coordinates where it appears in the cropped
output image. */
    outhist << "\tpixel {X,Y}: {" <<
    vect.getPixelX(cfg.getImageRes()) << "," <<
    vect.getPixelY(cfg.getImageRes()) << "}\n";
}

/*****
* NAME: print_usage
*
* DESCRIPTION: Print out instructions on how to
*   use this program, including all the available
*   options.
*
* RETURNS: N/A
*****/
void print_usage(
    char *prog_name) /* The name of the program, as provided by
the OS through argv[0]. */
{
    cout << "\n\n";
    cout << "***** Usage *****\n";
    cout << prog_name << " {options_list}\n";
    cout << "Available options for option_list: \n";
    cout << "--auto-color\n";
    cout << "    Automatically chose the color scale and offset to
show the best contrast.\n";
    cout << "    Disabled by default.\n";
    cout << "--color-scale {VALUE}\n";
    cout << "    Scale the sensitivity value to a brightness (0.0 is
black, 1.0 is white) with {VALUE}. Controls the plot's contrast.\n";
    cout << "    Default is 1.0\n";
    cout << "--color-offset {VALUE}\n";
    cout << "    Add ${VALUE} to every brightness value after
scaling. Controls the plot's overall brightness.\n";
    cout << "    Default is 0.0\n";
}

```

```

    cout << "--default-arg {NAME} {VALUE}\n";
    cout << "    Map a default variable named {NAME} to value
{VALUE}.\n";
    cout << "    By default, no default variables are defined.\n";
    cout << "--dataset {NAME}\n";
    cout << "    Use {NAME} as the dataset\'s base filename,
excluding the parent directory and file extension.\n";
    cout << "    This option has no default, and is always
required.\n";
    cout << "--formula {FORMULA} \n";
    cout << "    Use {FORMULA} as the sensitivity formula.\n";
    cout << "    This option has no default, and is always
required.\n";
    cout << "--hist-res {INTEGER}\n";
    cout << "    Split the histogram across {INTEGER} separate equal-
sized ranges.\n";
    cout << "    Default is 16.\n";
    cout << "--img-res {INTEGER}\n";
    cout << "    Use {INTEGER} as the base horizontal resolution.\n";
    cout << "    Default is 1024.\n";
    cout << "--out-suffix {NAME}\n";
    cout << "    For the image and histogram output files, append
{NAME} after the dataset name, and before the file extension.\n";
    cout << "    By default, no suffix is appended to the output
files.\n";
    cout << "--override-arg {NAME} {VALUE}\n";
    cout << "    Map an override variable named {NAME} to
{VALUE}.\n";
    cout << "    By default, no override variables are defined.\n";
    cout << "--preset {PRESET_NAME}\n";
    cout << "    Use the preset selected by {PRESET_NAME}. This sets
the formula, out-suffix, color-scale, color-offset.\n";
    cout << "    By default, no preset is used. Each affected value
uses its own default.\n";
    cout << "--prime-dir\n";
    cout << "    Use the prime direction of each vector as input to
the sensitivity calculation, instead of the original vector.\n";
    cout << "    Disabled by default.\n";
    cout << "--summary-file {NAME}\n";
    cout << "    Append a vector summary to {NAME}.txt in the output
directory.\n";
    cout << "    By default, the summary file is disabled.\n";
    cout << "\n";
    cout << "    {VALUE} is a real number.\n";
    cout << "    {NAME} is a descriptive name. It can be any
string.\n";
    cout << "    NOTE: If it contains a space, you should
surround the name with quotes.\n";
    cout << "    {FORMULA} is a string specifying the sensitivity
formula. It must follow the formula syntax.\n";
    cout << "    NOTE: You should always surround the formula
with single-quotes.\n";
    cout << "    {INTEGER} is a whole number that is greater than
0.\n";
    cout << "    {PRESET_NAME} is a string that is equal to one of
the following options:\n";
    cout << "        HULLAR\n";
    cout << "        Use the to render Fig 7.B of Yang and Hullar
2007.\n";
    cout << "        RODGERS_ORIG\n";

```

```

        cout << "                When the dot product between the rotation
axis and the canal is greater than 0, divide it by the sum of all
canal\n"
        << "                radii, and sum it into the sensitivity.
Otherwise, the canal contributes nothing to the sensitivity at this
pixel.\n";
        cout << "                RODGERS_NEG\n";
        cout << "                When the dot product between the rotation
axis and canal is less than 0, multiply it by -1, divide it by the sum
of all\n"
        << "                canal radii, and sum it into the
sensitivity. Otherwise, the canal contributes nothing to the
sensitivity at this pixel.\n";
        cout << "                RODGERS_ABS\n";
        cout << "                Divide the absolute value of each dot
product between the rotation axis and canal by the sum of all canal
radii, and sum \n"
        << "                them all into the total sensitivity.\n";
        cout << "                RODGERS_ABS_GAIN\n";
        cout << "                Similar to RODGERS_ABS, but compute the gain
from the dot product absolute value before adding it to the total
sensitivity.\n";
    }
}

```

## ▣ isomap-gen.h

```

#ifndef ISOMAP_GEN_H
#define ISOMAP_GEN_H

#include <string>
using namespace std;

/* Isomap Pixel
 * Store the color components of a pixel in a format that is compatible
with OpenGL.
 * This format requires one byte per color channel, and the color
channels are listed
 * in the order red-green-blue. This is required for passing the
isomap to OpenGL
 * as a texture. */
struct IsomapPixel
{
    unsigned char red;
    unsigned char green;
    unsigned char blue;
};

void generate_isomap(string infile, string outfile);

#endif

```

## ▣ isomap-prog.cpp

```

#include "isomap-prog.h"

```

```

#include <stdlib.h>
#include <sstream>
#include <string.h>
#include <math.h>

/*****
* NAME: createOperation
*
* DESCRIPTION: Decode a value from a formula,
*   identify the operation it represents, and
*   instantiate an operation from it.  If there are
*   any operations nested inside of it, it recurses
*   to create those first.
*
* RETURNS: The operation object.
*****/
IsoOperation *createOperation(
    char type_ind, /* The character that determines the
    identifier type. */
    string func_name, /* The function name, if type_ind is '('.
    The variable if it is '=' or '@'. The literal number if it is '#'. */
    istream &instream) /* The input stream from which to read
    the operations input values. */
{
    if (type_ind == '=')
    {
        return new IOVariable(func_name);
    }
    else if (type_ind == '#')
    {
        double lit;
        stringstream lit_stream(func_name);
        lit_stream >> lit;
        return new IOLiteral(lit);
    }
    else if (type_ind == '@')
    {
        return new IOLiteral(func_name);
    }
    else if (type_ind == '(')
    {
        if (func_name == "pos")
        {
            return new IOPos(instream);
        }
        else if (func_name == "neg")
        {
            return new IONeg(instream);
        }
        else if (func_name == "abs")
        {
            return new IOAbs(instream);
        }
        else if (func_name == "norm")
        {
            return new IONormalize(instream);
        }
        else if (func_name == "mag")
        {
            return new IOMagnitude(instream);
        }
    }
}

```

```

}
else if (func_name == "vect")
{
    return new IOVectorize(instream);
}
else if (func_name == "comp")
{
    return new IOComponent(instream);
}
else if (func_name == "+")
{
    return new IOAddScalar(instream);
}
else if (func_name == "+v")
{
    return new IOAddVector(instream);
}
else if (func_name == "-")
{
    return new IOSubtractScalar(instream);
}
else if (func_name == "-v")
{
    return new IOSubtractVector(instream);
}
else if (func_name == "*")
{
    return new IOMultiplyScalar(instream);
}
else if (func_name == "*v")
{
    return new IOMultiplyVector(instream);
}
else if (func_name == "/")
{
    return new IODivideScalar(instream);
}
else if (func_name == "/v")
{
    return new IODivideVector(instream);
}
else if (func_name == "dot")
{
    return new IODotProduct(instream);
}
else if (func_name == "avg")
{
    return new IOAverageScalar(instream);
}
else if (func_name == "avgv")
{
    return new IOAverageVector(instream);
}
else if (func_name == "sum")
{
    return new IOSumScalar(instream);
}
else if (func_name == "sumv")
{
    return new IOSumVector(instream);
}

```

```

    }
    else if (func_name == "prod")
    {
        return new IOProduct(instream);
    }
    else if (func_name == "max")
    {
        return new IOMaximum(instream);
    }
    else if (func_name == "min")
    {
        return new IOMinimum(instream);
    }
    else if (func_name == "pow")
    {
        return new IOPow(instream);
    }
    else
    {
        cerr << "Unknown function name \'' << func_name <<
"\'\n";
        exit(1);
        return NULL;
    }
}
else
{
    cerr << "Unknown identifier character \'' << type_ind <<
"\'\n";
    exit(1);
    return NULL;
}
}

```

```

/*****
* NAME: preprocess_formula
*
* DESCRIPTION: Sanitize the raw_formula and
* perform basic error checking. This function is
* organized to make it easier to report to the
* user what is wrong with the formula. It also
* adjusts whitespaces to meet the IsoOperation's
* parsing assumptions, and indents it to make it
* easier for the user to recognize what is wrong
* with it.
*
* RETURNS: A new C string containing the
* preprocessed formula.
*****/

```

```

char * preprocess_formula(string raw_formula)
{
    enum token_type_t
    {
        TT_NULL,
        TT_FUNC_OPEN,
        TT_CLOSE_PAREN,
        TT_FLOAT_LITERAL,
        TT_INT_LITERAL,
        TT_NAMED_VAR
    };
}

```

```

token_type_t curr_type = TT_NULL;
int curr_indent_level = 0;
char next_char;
bool token_complete = false;
bool preprocess_failed = false;
bool identifier_started = false;
bool decimal_point_read = false;
bool char_accepted = false;
bool indentation_done = false;

stringstream formula_stream(raw_formula);
const int max_result_len = 1024*1024;
char * result = new char[max_result_len];
if (result == NULL)
{
    cerr << "Unable to allocate a buffer for the preprocessed
string.\n";
    exit(1);
}
int result_i = 0;

formula_stream.get(next_char);
while (!formula_stream.fail() && !formula_stream.eof() &&
!preprocess_failed)
{
    char_accepted = false;
    if (isspace(next_char))
    {
        /* The character is a whitespace. It will never be
read into any token, and will always be discarded.
* Decide whether it terminates the current token. */

        if (curr_type == TT_FUNC_OPEN || curr_type ==
TT_FLOAT_LITERAL || curr_type == TT_INT_LITERAL || curr_type ==
TT_NAMED_VAR)
        {
            if (identifier_started)
            {
                /* whitespace automatically terminates
any type of token if the identifier or literal was started. */
                token_complete = true;
            }
            else
            {
                /* There was a space between the sigil
and the identifier or literal, which is legal.
* Just discard the whitespace, but
continue the token. */
            }
        }
        /* ELSE: Ignore the whitespace. */
    }
    else if (curr_type == TT_NULL)
    {
        /* The current token type is not defined yet. The
next non-whitespace character will define it. */

        if (isspace(next_char))
        {

```

```

        /* Discard this whitespace character. */
    }
else
{
    switch (next_char)
    {
    case '(':
        curr_type = TT_FUNC_OPEN;
        token_complete = false;
        indentation_done = false;
        char_accepted = true;
        break;

    case ')':
        curr_type = TT_CLOSE_PAREN;
        token_complete = true;
        indentation_done = false;
        char_accepted = true;
        break;

    case '#':
        curr_type = TT_FLOAT_LITERAL;
        token_complete = false;
        indentation_done = false;
        char_accepted = true;
        break;

    case ':':
        curr_type = TT_INT_LITERAL;
        token_complete = false;
        indentation_done = false;
        char_accepted = true;
        break;

    case '@':
    case '=':
    case '?':
        curr_type = TT_NAMED_VAR;
        token_complete = false;
        indentation_done = false;
        char_accepted = true;
        break;

    default:
        cerr << "Unrecognized character \'' <<
next_char << "\\'\n";
        preprocess_failed = true;
    }

    if (curr_type == TT_CLOSE_PAREN &&
curr_indent_level == 0)
    {
        cerr << "Parenthesis mismatch.
Encountered unexpected close parenthesis.\n";
        preprocess_failed = true;
    }
    }
}
else if (curr_type == TT_FUNC_OPEN)
{

```

```

/* The open parenthesis was already read into the
token. The only remaining characters can be
* ones that can legally appear in a function name.
*/
if (next_char >= 'A' && next_char <= 'Z')
{
    char_accepted = true;
    identifier_started = true;
}
else if (next_char >= 'a' && next_char <= 'z')
{
    char_accepted = true;
    identifier_started = true;
}
else if (next_char == '+' || next_char == '-' ||
next_char == '*' || next_char == '/' || next_char == '_')
{
    char_accepted = true;
    identifier_started = true;
}
else
{
    cerr << "Invalid character \"" << next_char <<
"\\" in function name.\n";
    preprocess_failed = true;
}
}
else if (curr_type == TT_FLOAT_LITERAL)
{
    if (next_char == '.' && !decimal_point_read)
    {
        /* Only accept the decimal point if it hasn't
already appeared. */
        char_accepted = true;
        decimal_point_read = true;
        identifier_started = true;
    }
    else if (next_char >= '0' && next_char <= '9')
    {
        /* Accept the regular digit. */
        char_accepted = true;
        identifier_started = true;
    }
    else
    {
        cerr << "Invalid character \"" << next_char <<
"\\" in real number. Expected a digit";
        if (!decimal_point_read)
        {
            cerr << " or a decimal point.\n";
        }
        else
        {
            cerr << ".\n";
        }
        preprocess_failed = true;
    }
}
}
else if (curr_type == TT_INT_LITERAL)
{

```

```

        if (next_char >= '0' && next_char <= '9')
        {
            /* Accept the regular digit. */
            char_accepted = true;
            identifier_started = true;
        }
        else
        {
            cerr << "Invalid character '\"" << next_char <<
"\' in integer number. Expected a digit.\n";
            preprocess_failed = true;
        }
    }
    else if (curr_type == TT_NAMED_VAR)
    {
        if (next_char >= 'A' && next_char <= 'Z')
        {
            /* Accept the upper-case character. */
            char_accepted = true;
            identifier_started = true;
        }
        else if (next_char >= 'a' && next_char <= 'z')
        {
            /* Accept the lower-case character. */
            char_accepted = true;
            identifier_started = true;
        }
        else if (next_char >= '0' && next_char <= '9')
        {
            /* Accept the regular digit. */
            char_accepted = true;
            identifier_started = true;
        }
        else if (next_char == '_')
        {
            char_accepted = true;
            identifier_started = true;
        }
        else
        {
            cerr << "Invalid character '\"" << next_char <<
"\' in variable name. Expected an alphanumeric character.\n";
            preprocess_failed = true;
        }
    }
    else
    {
        /* This should never happen, so the program is either
crashing somehow (e.g. stack smash), or the algorithm is incorrect.
* Exit immediately instead of allowing it to do
damage. */
        cerr << "Unexpected program state.\n";
        exit(1);
    }
    /* NOTE: TT_CLOSE_PAREN is always terminated as soon as it
is detected, so it is never possible to encounter
* it while continuing a token. */

    if (!preprocess_failed && curr_type != TT_NULL &&
!indentation_done)

```

```

    {
        if (curr_type == TT_CLOSE_PAREN)
        {
            --curr_indent_level;
        }

        /* Indent to the proper level. */
        for (int i = 0; i < curr_indent_level * 3; ++i)
        {
            if (result_i >= max_result_len)
            {
                cerr << "The preprocessed formula is too
long! Max length is " << max_result_len << "\n";
                exit(1);
            }
            result[result_i++] = ' ';
        }

        /* Choose the next indentation level. */
        if (curr_type == TT_FUNC_OPEN)
        {
            ++curr_indent_level;
        }

        indentation_done = true;
    }

    if (char_accepted)
    {
        if (result_i >= max_result_len)
        {
            cerr << "The preprocessed formula is too long!
Max length is " << max_result_len << "\n";
            exit(1);
        }
        result[result_i++] = next_char;
    }

    if (token_complete && !preprocess_failed && curr_type !=
TT_NULL)
    {
        if (result_i >= max_result_len)
        {
            cerr << "The preprocessed formula is too long!
Max length is " << max_result_len << "\n";
            exit(1);
        }
        result[result_i++] = '\n';
        /* Reset everything for the next token. */
        curr_type = TT_NULL;
        identifier_started = false;
        decimal_point_read = false;
    }

    if (!preprocess_failed)
    {
        formula_stream.get(next_char);
    }
}

```

```

    /* Verify that the formula didn't end too soon. */
    if (!preprocess_failed)
    {
        if (!token_complete)
        {
            cerr << "Formula ended without completing the last
token.\n";
            preprocess_failed = true;
        }

        if (curr_indent_level != 0)
        {
            cerr << "Parenthesis mismatch. Expected " <<
curr_indent_level << " more close parentheses.\n";
            preprocess_failed = true;
        }
    }

    if (result_i >= max_result_len)
    {
        cerr << "The preprocessed formula is too long! Max length
is " << max_result_len << "\n";
        exit(1);
    }
    /* Terminate the C string. */
    result[result_i++] = '\0';

    if (preprocess_failed)
    {
        cerr << "CURRENT PARSE STATE:\n";
        cerr << result;
        cerr << "\n";
        cerr << "INVALID NEXT CHARACTER: \'' << next_char <<
"\'\n";
        exit(1);
    }

    return result;
}

/*****
* NAME: parse_formula
*
* DESCRIPTION: Preprocess and parse the formula,
* and use it to construct the IsoOperation.
*
* RETURNS: The new IsoOperation representing the
* formula.
*****/
IsoOperation *parse_formula(const IsoConfig &cfg)
{
    /* Preprocess the formula, then load the formula string into a
stringstream, to make it easier to traverse and process. */
    string formula = preprocess_formula(cfg.getFormula());
    stringstream formula_stream(formula);

    /* Read the first character and ensure it's an open parenthesis.
* All calls to createOperation() expect that to be done by the
caller. */
    char first_parenth;

```

```

        formula_stream >> first_parenth;
        if (first_parenth != '(')
        {
            cerr << "Malformed formula; no initial parenthesis.
Formula = \" << cfg.getFormula() << \"'\n\";
            cerr << "FORMULA:\n\" << formula;
            exit(1);
        }

        /* Get the name of the root function, and create an operation out
of it.
        * Operations are recursive, so this effectively parses the
entire formula. */
        string root_func;
        formula_stream >> root_func;
        IsoOperation *formula_prog = createOperation(first_parenth,
root_func, formula_stream);
        if (!formula_prog->validate())
        {
            cerr << "Validation of root formula failed!\n\";
            cerr << "FORMULA:\n\" << formula;
            exit(1);
        }

        /* Double-check that the ultimate result of the root operation is
a scalar.
        * Otherwise, it is impossible to plot. */
        if (formula_prog->isVector())
        {
            cerr << "Formula must not yield a vector! Root function =
\" << root_func << \"'\n\";
            cerr << "FORMULA:\n\" << formula;
            exit(1);
        }

        return formula_prog;
    }

/*****
* NAME: IsoOperation
*
* DESCRIPTION: Constructor for the abstract
* portion of the class instance when the
* operation is a function. It automatically
* parses the inputs.
*
* RETURNS: N/A
*****/
IsoOperation::IsoOperation(
    istream &instream, /* The input stream from which to read
the input values. */
    string func_name_in, /* The name of this function. */
    bool is_vector_in) /* true if the result of this function
is a vector. false if the result is a scalar. */
{
    bool op_terminated = false;

    idx_val = 0;
    func_name = func_name_in;
    is_vector = is_vector_in;

```

```

while (!op_terminated)
{
    char next_token_ind;
    instream >> next_token_ind;

    if (instream.fail())
    {
        cerr << func_name << ".IsoOperation(instream) input
stream failed!\n";
        exit(1);
    }
    else if (next_token_ind == ')')
    {
        op_terminated = true;
    }
    else if (next_token_ind == '?')
    {
        string loop_varname_in;
        instream >> loop_varname_in;

        setLoopVar(loop_varname_in);
    }
    else if (next_token_ind == ':')
    {
        instream >> idx_val;
    }
    else
    {
        string child_func_name;
        instream >> child_func_name;

        operand_list.push_back(createOperation(next_token_ind,
child_func_name, instream));
    }

    countOperands();
}

/*****
* NAME: IsoOperation
*
* DESCRIPTION: Construct an operation specifying
* only its name and whether it is a vector. It
* does not parse the function's inputs.
*
* RETURNS: N/A
*****/
IsoOperation::IsoOperation(
    bool is_vector_in, /* true if the result of this function
is a vector. false if the result is a scalar. */
    string func_name_in) /* The name of this function. */
{
    is_vector = is_vector_in;
    func_name = func_name_in;

    is_constant = false;
    is_loop = false;

```

```

        is_var = false;
    }

/*****
 * NAME: IsoOperation
 *
 * DESCRIPTION: Construct an operation representing
 *   a bound variable. The result is a vector.
 *
 * RETURNS: N/A
 *****/
IsoOperation::IsoOperation(
    string varname_in) /* The name of the bound variable. */
{
    setVar(varname_in);
}

/*****
 * NAME: IsoOperation
 *
 * DESCRIPTION: Construct an operation representing
 *   a constant literal. The result is a scalar.
 *
 * RETURNS: N/A
 *****/
IsoOperation::IsoOperation(
    double literal_in) /* The value of the constant this
operation represents. */
{
    setConstant(literal_in);
}

/*****
 * NAME: ~IsoOperation
 *
 * DESCRIPTION: Destroy this operation and any
 *   input operations it contains.
 *
 * RETURNS: N/A
 *****/
IsoOperation::~~IsoOperation()
{
    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        delete operand_list[i];
    }
}

/*****
 * NAME: isVector
 *
 * DESCRIPTION: Detect whether the result of this
 *   operation is a vector.
 *
 * RETURNS: true if the result is a vector. false
 *   if the result is a scalar.
 *****/
bool IsoOperation::isVector() const
{
    return is_vector;
}

```

```

}

/*****
* NAME: isConstant
*
* DESCRIPTION: Detect whether this operation evaluates to a constant.
*
* RETURNS: true if the result is constant. false
* otherwise.
*****/
bool IsoOperation::isConstant() const
{
    return is_constant;
}

/*****
* NAME: vectorValue
*
* DESCRIPTION: Evaluate the operation's value,
* resulting in a vector. The input operations
* are all evaluated, then given to the non-
* abstract operation itself so it can produce a
* result.
*
* RETURNS: The vector result of this operation.
*****/
IsoVector IsoOperation::vectorValue(
    const map<string,IsoVector> &bound_vars, /* The mapping of
symbolic variable names to the vector values they represent. */
    const vector<IsoVector> &canal_arr, /* The array of canals,
for loops to iterate over. */
    map<string,double> &bound_scalars) /* The mapping of
symbolic variable names to the scalar values they represent. */
{
    if (!is_vector)
    {
        cerr << func_name << ".vectorValue: Not a vector!\n";
        exit(1);
    }
    else if (is_constant)
    {
        /* No inputs to iterate over, the result is already known.
*/
        return vect_constant;
    }
    else if (is_loop)
    {
        /* Allocate a place to store the result of each iteration,
so it can be passed to the operation all at once. */
        vector<IsoVector> operands;
        /* The bound variables will be manipulated, so keep the
caller's map safe by
* allocating a new one and cloning it. */
        map<string,IsoVector> child_bound_vars = bound_vars;
        /* Iterate over the canals. */
        for (int i = 0; i < (int)canal_arr.size(); ++i)
        {
            /* Bind the variable to the currently-selected canal.
*/
            child_bound_vars[varname] = canal_arr[i];

```

```

        /* Evaluate the result of the input operation, and
add it to the result list. */
        operands.push_back(operand_list[0]-
>vectorValue(child_bound_vars, canal_arr, bound_scalars));
    }

    /* Let the non-abstract portion decide how to combine the
inputs, and return its result. */
    return this->evaluateLoop(operands);
}
else
{
    /* Let the non-abstract portion decide how to combine the
inputs, and return its result. */
    return this->evaluateVector(bound_vars, canal_arr,
bound_scalars);
}
}

/*****
* NAME: scalarValue
*
* DESCRIPTION: Evaluate the operation's value,
* resulting in a scalar. The input operations
* are all evaluated, then given to the non-
* abstract operation itself so it can produce a
* result.
*
* RETURNS: The scalar result of this operation.
*****/
double IsoOperation::scalarValue(
    const map<string,IsoVector> &bound_vars, /* The mapping of
symbolic variable names to the vector values they represent. */
    const vector<IsoVector> &canal_arr, /* The array of canals,
for loops to iterate over. */
    map<string,double> &bound_scalars) /* The mapping of
symbolic variable names to the scalar values they represent. */
{
    if (is_vector)
    {
        cerr << func_name << ".scalarValue: Not a scalar!\n";
        exit(1);
    }
    else if (is_constant)
    {
        /* No inputs to iterate over, the result is already known.
*/
        return scalar_constant;
    }
    else if (is_loop)
    {
        /* Allocate a place to store the result of each iteration,
so it can be passed to the operation all at once. */
        vector<double> operands;
        /* The bound variables will be manipulated, so keep the
caller's map safe by
* allocating a new one and cloning it. */
        map<string,IsoVector> child_bound_vars = bound_vars;
        /* Iterate over the canals. */
        for (int i = 0; i < (int)canal_arr.size(); ++i)

```

```

        {
            /* Bind the variable to the currently-selected canal.
*/
            child_bound_vars[varname] = canal_arr[i];
            /* Evaluate the result of the input operation, and
add it to the result list. */
            operands.push_back(operand_list[0]-
>scalarValue(child_bound_vars, canal_arr, bound_scalars));
        }

        /* Let the non-abstract portion decide how to combine the
inputs, and return its result. */
        return this->evaluateLoop(operands);
    }
    else
    {
        /* Let the non-abstract portion decide how to combine the
inputs, and return its result. */
        return this->evaluateScalar(bound_vars, canal_arr,
bound_scalars);
    }
}

/*****
* NAME: setConstant
*
* DESCRIPTION: Set this operation's value to be a
*   constant vector value.
*
* RETURNS: N/A
*****/
void IsoOperation::setConstant(
    IsoVector vect_constant_in) /* This operation's constant
vector value. */
{
    is_loop = false;
    is_constant = true;
    is_var = false;
    is_vector = true;
    vect_constant = vect_constant_in;
}

/*****
* NAME: setConstant
*
* DESCRIPTION: Set this operation's value to be a
*   constant scalar value.
*
* RETURNS: N/A
*****/
void IsoOperation::setConstant(
    double scalar_constant_in) /* This operation's constant
vector value. */
{
    is_loop = false;
    is_constant = true;
    is_var = false;
    is_vector = false;
    scalar_constant = scalar_constant_in;
}

```

```

/*****
* NAME: setLoopVar
*
* DESCRIPTION: Set this operation's loop variable
*             name, and initialize it as a loop.
*
* RETURNS: N/A
*****/
void IsoOperation::setLoopVar(
    string loop_varname_in) /* This operation's loop variable
name. */
{
    is_loop = true;
    is_constant = false;
    is_var = false;
    varname = loop_varname_in;
}

/*****
* NAME: setVar
*
* DESCRIPTION: Set this operation's variable name,
*             and initialize it as a vector variable.
*
* RETURNS: N/A
*****/
void IsoOperation::setVar(
    string varname_in) /* The name of the variable this
operation represents. */
{
    is_loop = false;
    is_constant = false;
    is_var = true;
    is_vector = true;
    varname = varname_in;
    func_name = "?var";
}

/*****
* NAME: evaluateVector
*
* DESCRIPTION: This is the default implementation
*             for subclasses which do not implement it
*             themselves. Subclasses that do not override
*             this function presumably do so because they do
*             not produce a vector result, so this function
*             always aborts the program.
*
* RETURNS: Never returns
*****/
IsoVector IsoOperation::evaluateVector(
    const map<string,IsoVector> &bound_vars, /* The mapping of
symbolic variable names to the vector values they represent. */
    const vector<IsoVector> &canal_arr, /* The array of canals,
for loops to iterate over. */
    map<string,double> &bound_scalars) /* The mapping of
symbolic variable names to the scalar values they represent. */
{
    cerr << func_name << ".evaluateVector not implemented!\n";
}

```

```

        exit(1);

        /* This code is unreachable, but keep the compiler happy by
returning something anyway. */
        return IsoVector();
    }

/*****
* NAME: evaluateScalar
*
* DESCRIPTION: This is the default implementation
*   for subclasses which do not implement it
*   themselves. Subclasses that do not override
*   this function presumably do so because they do
*   not produce a scalar result, so this function
*   always aborts the program.
*
* RETURNS: Never returns
*****/
double IsoOperation::evaluateScalar(
    const map<string,IsoVector> &bound_vars, /* The mapping of
symbolic variable names to the vector values they represent. */
    const vector<IsoVector> &canal_arr, /* The array of canals,
for loops to iterate over. */
    map<string,double> &bound_scalars) /* The mapping of
symbolic variable names to the scalar values they represent. */
{
    cerr << func_name << ".evaluateScalar not Implemented!\n";
    exit(1);

    /* This code is unreachable, but keep the compiler happy by
returning something anyway. */
    return 0.0;
}

/*****
* NAME: evaluateLoop
*
* DESCRIPTION: This is the default implementation
*   for subclasses which do not implement it
*   themselves. Subclasses that do not override
*   this function presumably do so because they are
*   not a loop, so this function always aborts the
*   program.
*
* RETURNS: Never returns
*****/
IsoVector IsoOperation::evaluateLoop(const vector<IsoVector> &operands)
{
    cerr << func_name << ".evaluateLoop (vector) not Implemented!\n";
    exit(1);

    /* This code is unreachable, but keep the compiler happy by
returning something anyway. */
    return IsoVector();
}

/*****
* NAME: evaluateLoop
*

```

```

* DESCRIPTION: This is the default implementation
*   for subclasses which do not implement it
*   themselves. Subclasses that do not override
*   this function presumably do so because they are
*   not a loop, so this function always aborts the
*   program.
*
* RETURNS: Never returns
*****/
double IsoOperation::evaluateLoop(const vector<double> &operands)
{
    cerr << func_name << ".evaluateLoop (scalar) not Implemented!\n";
    exit(1);

    /* This code is unreachable, but keep the compiler happy by
returning something anyway. */
    return 0.0;
}

/*****
* NAME: countOperands
*
* DESCRIPTION: Count how many function inputs
*   result in scalars, and how many result in
*   vectors. These counts allow the non-abstract
*   operation to error-check its inputs.
*
* RETURNS: N/A
*****/
void IsoOperation::countOperands()
{
    /* Start the counters at 0. */
    vect_op_cnt = 0;
    scal_op_cnt = 0;

    /* Iterate over each input. */
    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        /* Detect which type of result it creates, and increment
the corresponding counter. */
        if (operand_list[i]->isVector())
        {
            ++vect_op_cnt;
        }
        else
        {
            ++scal_op_cnt;
        }
    }
}

/*****
* NAME: validate
*
* DESCRIPTION: Command each input operation to
*   validate itself (recursively), and validate
*   this operation. It checks that every
*   operation's input requirements are met.
*
* RETURNS: true if this operation and all

```

```

*   operations it contains are OK.  false otherwise.
*****/
bool IsoOperation::validate()
{
    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        if (!operand_list[i]->validate())
        {
            cerr << "Validation failed on " << func_name << "\'s
child " << operand_list[i]->func_name << "\n";
            return false;
        }
        if (!this->validatePriv())
        {
            cerr << "Validation failed on " << func_name << "\'s member
data\n";
            return false;
        }
    }
    return true;
}

/*****
* VECTOR VARIABLE
*****/

IOVariable::IOVariable(string varname_in) : IsoOperation(varname_in)
{
}

IsoVector IOVariable::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    return bound_vars.find(varname)->second;
}

bool IOVariable::validatePriv()
{
    return true;
}

/*****
* SCALAR LITERAL
*****/

IOLiteral::IOLiteral(double literal_in) : IsoOperation(literal_in)
{
    my_literal = literal_in;
    is_symbol = false;
}

IOLiteral::IOLiteral(string symbol_name_in) : IsoOperation(false,
symbol_name_in)
{
    my_literal = 0.0;
    is_symbol = true;
    symbol_name = symbol_name_in;
}

```

```

double IOLiteral::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    if (!is_symbol)
    {
        /* The symbol_name is not used, just return the literal
value. */
        return my_literal;
    }
    else
    {
        /* Make sure this variable's name was actually assigned.
* If not, the result is completely undefined. */
        if (bound_scalars.count(symbol_name) == 0)
        {
            cerr << "Symbol \'' << symbol_name << "' is not
defined!\n";
            exit(1);
        }
        /* The variable is bound. Look it up and return it. */
        return bound_scalars[symbol_name];
    }
}

bool IOLiteral::validatePriv()
{
    return true;
}

/*****
* POSITIVE
*****/

double IOPos::evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars)
{
    double opval = operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars);
    if (opval < 0.0)
    {
        opval = 0.0;
    }
    return opval;
}

bool IOPos::validatePriv()
{
    return (scal_op_cnt == 1 && vect_op_cnt == 0);
}

/*****
* NEGATIVE
*****/

double IONeg::evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars)
{
    double opval = operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars);

```

```

        if (opval > 0.0)
        {
            opval = 0.0;
        }
        return opval;
    }

bool IONeg::validatePriv()
{
    return (scal_op_cnt == 1 && vect_op_cnt == 0);
}

/*****
 * ABSOLUTE VALUE
 *****/

double IOAbs::evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars)
{
    double opval = operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars);
    if (opval < 0.0)
    {
        opval = -opval;
    }
    return opval;
}

bool IOAbs::validatePriv()
{
    return (scal_op_cnt == 1 && vect_op_cnt == 0);
}

/*****
 * NORMALIZE
 *****/

IsoVector IONormalize::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    return operand_list[0]->vectorValue(bound_vars, canal_arr,
bound_scalars).normalize();
}

bool IONormalize::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt == 1);
}

/*****
 * MAGNITUDE
 *****/

double IOMagnitude::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    return operand_list[0]->vectorValue(bound_vars, canal_arr,
bound_scalars).magnitude();
}

```

```

}

bool IOMagnitude::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt == 1);
}

/*****
 * VECTORIZE
 *****/

IsoVector IOVectorize::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    /* Evaluate all three scalar inputs, and construct a vector from
their results. */
    IsoVector retval(operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars),
operand_list[1]->scalarValue(bound_vars,
canal_arr, bound_scalars),
operand_list[2]->scalarValue(bound_vars,
canal_arr, bound_scalars));
    return retval;
}

bool IOVectorize::validatePriv()
{
    return (scal_op_cnt == 3 && vect_op_cnt == 0);
}

/*****
 * COMPONENT OF VECTOR
 *****/

double IOComponent::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    IsoVector retvect = operand_list[0]->vectorValue(bound_vars,
canal_arr, bound_scalars);
    /* Select the Cartesian component of the single vector input. */
    switch (idx_val)
    {
    case 0:
        return retvect.x;

    case 1:
        return retvect.y;

    case 2:
        return retvect.z;

    default:
        return -1.0;
    }
}

bool IOComponent::validatePriv()
{

```

```

        return (scal_op_cnt == 0 && vect_op_cnt == 1 && idx_val >= 0 &&
idx_val < 3);
}

/*****
* SCALAR ADD
*****/

double IOAddScalar::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    double retval = 0.0;
    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        retval += operand_list[i]->scalarValue(bound_vars,
canal_arr, bound_scalars);
    }
    return retval;
}

bool IOAddScalar::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 0);
}

/*****
* VECTOR ADD
*****/

IsoVector IOAddVector::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    IsoVector retval(0.0, 0.0, 0.0);
    IsoVector rhs;
    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        rhs = operand_list[i]->vectorValue(bound_vars, canal_arr,
bound_scalars);
        retval = retval.add(rhs);
    }
    return retval;
}

bool IOAddVector::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt > 1);
}

/*****
* SCALAR SUBTRACT
*****/

double IOSubtractScalar::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    /* Start with the first input. */

```

```

        double retval = operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars);
        /* And subtract every other input from it. */
        for (int i = 1; i < (int)operand_list.size(); ++i)
        {
            retval -= operand_list[i]->scalarValue(bound_vars,
canal_arr, bound_scalars);
        }
        return retval;
    }

bool IOSubtractScalar::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 0);
}

/*****
 * VECTOR SUBTRACT
 *****/

IsoVector IOSubtractVector::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    /* Start with the first input vector. */
    IsoVector retval = operand_list[0]->vectorValue(bound_vars,
canal_arr, bound_scalars);
    IsoVector rhs;
    for (int i = 1; i < (int)operand_list.size(); ++i)
    {
        /* Compute the next input to subtract from the intermediate
result. */
        rhs = operand_list[i]->vectorValue(bound_vars, canal_arr,
bound_scalars);
        /* Subtract it from the result. */
        retval = retval.subtract(rhs);
    }
    return retval;
}

bool IOSubtractVector::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt > 1);
}

/*****
 * SCALAR MULTIPLY
 *****/

double IOMultiplyScalar::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    /* Start with 1.0 so the first multiply will result in the first
scalar value. */
    double retval = 1.0;
    /* Multiply every scalar input together. */
    for (int i = 0; i < (int)operand_list.size(); ++i)
    {

```

```

        retval *= operand_list[i]->scalarValue(bound_vars,
        canal_arr, bound_scalars);
    }
    return retval;
}

bool IOMultiplyScalar::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 0);
}

/*****
 * VECTOR MULTIPLY
 *****/

IsoVector IOMultiplyVector::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    double scalar_accum = 1.0;
    IsoVector vect;

    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        if (operand_list[i]->isVector())
        {
            vect = operand_list[i]->vectorValue(bound_vars,
            canal_arr, bound_scalars);
        }
        else
        {
            scalar_accum *= operand_list[i]-
            >scalarValue(bound_vars, canal_arr, bound_scalars);
        }
    }
    return vect.scale(scalar_accum);
}

bool IOMultiplyVector::validatePriv()
{
    return (scal_op_cnt > 0 && vect_op_cnt == 1);
}

/*****
 * SCALAR DIVIDE
 *****/

double IODivideScalar::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    double retval = operand_list[0]->scalarValue(bound_vars,
    canal_arr, bound_scalars);
    for (int i = 1; i < (int)operand_list.size(); ++i)
    {
        retval /= operand_list[i]->scalarValue(bound_vars,
        canal_arr, bound_scalars);
    }
    return retval;
}

```

```

bool IODivideScalar::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 0);
}

/*****
 * VECTOR DIVIDE
 *****/

IsoVector IODivideVector::evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    /* Gather the scalar product and vector separately. */
    double scalar_accum = 1.0;
    IsoVector vect;

    for (int i = 0; i < (int)operand_list.size(); ++i)
    {
        if (operand_list[i]->isVector())
        {
            /* Found the vector input. Store it for later. */
            vect = operand_list[i]->vectorValue(bound_vars,
canal_arr, bound_scalars);
        }
        else
        {
            /* Found another scalar value. Accumulate it into
the scalar product. */
            scalar_accum *= operand_list[i]-
>scalarValue(bound_vars, canal_arr, bound_scalars);
        }
    }

    /* The vector has been found and all scalars have been gathered.
    * Divide the vector by the product of all the scalars. */
    return vect.scale(1.0 / scalar_accum);
}

bool IODivideVector::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 1 && operand_list[0]-
>isVector());
}

/*****
 * DOT PRODUCT
 *****/

double IODotProduct::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    IsoVector rhs = operand_list[1]->vectorValue(bound_vars,
canal_arr, bound_scalars);
    return operand_list[0]->vectorValue(bound_vars, canal_arr,
bound_scalars).dotProduct(rhs);
}

```

```

bool IODotProduct::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt == 2);
}

/*****
 * SCALAR AVERAGE
 *****/

double IOAverageScalar::evaluateLoop(const vector<double> &operands)
{
    /* Start with the trivial sum of 0. */
    double retval = 0.0;
    for (int i = 0; i < (int)operands.size(); ++i)
    {
        /* Add each operand to the sum. */
        retval += operands[i];
    }

    /* Divide the sum by the number of inputs, and return that
result. */
    return retval / (double)operands.size();
}

bool IOAverageScalar::validatePriv()
{
    return (scal_op_cnt > 0 && vect_op_cnt == 0 && is_loop);
}

/*****
 * VECTOR AVERAGE
 *****/

IsoVector IOAverageVector::evaluateLoop(const vector<IsoVector>
&operands)
{
    /* Start with the trivial sum of {0,0,0}. */
    IsoVector retval(0.0, 0.0, 0.0);
    IsoVector rhs;
    for (int i = 0; i < (int)operands.size(); ++i)
    {
        rhs = operands[i];
        /* Add each operand to the sum. */
        retval = retval.add(rhs);
    }

    /* Divide the sum by the number of inputs, and return that
result. */
    return retval.scale(1.0 / (double)operands.size());
}

bool IOAverageVector::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt > 0 && is_loop);
}

/*****
 * SCALAR SUM
 *****/

```

```

double IOSumScalar::evaluateLoop(const vector<double> &operands)
{
    double retval = 0.0;
    for (int i = 0; i < (int)operands.size(); ++i)
    {
        retval += operands[i];
    }
    return retval;
}

bool IOSumScalar::validatePriv()
{
    return (scal_op_cnt > 0 && vect_op_cnt == 0 && is_loop);
}

/*****
 * VECTOR SUM
 *****/

IsoVector IOSumVector::evaluateLoop(const vector<IsoVector> &operands)
{
    IsoVector retval(0.0, 0.0, 0.0);
    IsoVector rhs;
    for (int i = 0; i < (int)operands.size(); ++i)
    {
        rhs = operands[i];
        retval = retval.add(rhs);
    }
    return retval;
}

bool IOSumVector::validatePriv()
{
    return (scal_op_cnt == 0 && vect_op_cnt > 0 && is_loop);
}

/*****
 * SCALAR PRODUCT
 *****/

double IOProduct::evaluateLoop(const vector<double> &operands)
{
    double retval = 1.0;
    for (int i = 0; i < (int)operands.size(); ++i)
    {
        retval *= operands[i];
    }
    return retval;
}

bool IOProduct::validatePriv()
{
    return (scal_op_cnt > 0 && vect_op_cnt == 0 && is_loop);
}

/*****
 * MAXIMUM
 *****/

```

```

double IOMaximum::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    /* Start with the maximum value of the first 1 input. */
    double retval = operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars);

    for (int i = 1; i < (int)operand_list.size(); ++i)
    {
        /* Update the maximum to be the maximum of the first i
inputs, which is a choice between
        * either the current input, or the maximum of the first i-
1 inputs. */
        double curr_val = operand_list[i]->scalarValue(bound_vars,
canal_arr, bound_scalars);
        if (curr_val > retval)
        {
            /* The current input is the new maximum .*/
            retval = curr_val;
        }
    }
    return retval;
}

bool IOMaximum::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 0);
}

/*****
* MINIMUM
*****/

double IOMinimum::evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars)
{
    double retval = operand_list[0]->scalarValue(bound_vars,
canal_arr, bound_scalars);

    for (int i = 1; i < (int)operand_list.size(); ++i)
    {
        double curr_val = operand_list[i]->scalarValue(bound_vars,
canal_arr, bound_scalars);
        if (curr_val < retval)
        {
            retval = curr_val;
        }
    }
    return retval;
}

bool IOMinimum::validatePriv()
{
    return (scal_op_cnt > 1 && vect_op_cnt == 0);
}

/*****
* EXPONENT

```

```

*****/
double IOPow::evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars)
{
    return pow(operand_list[0]->scalarValue(bound_vars, canal_arr,
bound_scalars), operand_list[1]->scalarValue(bound_vars, canal_arr,
bound_scalars));
}

bool IOPow::validatePriv()
{
    return (scal_op_cnt == 2 && vect_op_cnt == 0);
}

```

## isomap-prog.h

```

#ifndef ISOMAP_PROG_H
#define ISOMAP_PROG_H

#include "vector.h"
#include "isomap-config.h"

#include <iostream>
#include <map>
#include <vector>
#include <string>

using namespace std;

/* Language classification: Lisp-like
 * Format          Out Type   Description
 * (pos ...)      scalar     If the value is >0, use the
value. Otherwise, use 0.
 * (neg ...)      scalar     If the value is <0, use the
value. Otherwise, use 0.
 * (abs ...)      scalar     Take the absolute value of
the value.
 * (norm ...)     vector     Normalize a vector value
 * (mag ...)      scalar     Magnitude of a vector
 * (vect ...)     vector     Converts 3 scalars into a
vector
 * (comp :# ...) scalar     Takes just the # component of one
vector.
 * =canal#       vector     Take the canal index by
integer literal. (special case of ?var)
 * =var          vector     The canal referred by a
bound variable "var".
 * =axis        vector     The current axis normal.
(special case of ?var)
 * #scalar      scalar     where "scalar" is actually
any floating-point number.
 * @var         scalar     The scalar value
referred by a bound variable "var".
 * (+ ...)      scalar     Add scalars
 * (+v ...)     vector     Add vectors
 * (- ...)      scalar     Subtract scalars

```

```

* (-v ...)      vector      Subtract vectors
* (* ...)       scalar      Multiply scalars
* (*v ...)      vector      Multiply one vector by
scalars
* (/ ...)       scalar      Divide exactly two scalars
* (/v ...)      vector      Divide one vector by one
scalar
* (dot ...)     scalar      Dot product between exactly
two vectors
* (avg ?var ... ) scalar      Average of scalars, with a
bound iteration variable var
* (avgv ?var ... ) vector     Average of vectors, with a
bound iteration variable var
* (sum ?var ... ) scalar      Sum of scalars, with a bound
iteration variable var
* (sumv ?var ... ) vector     Sum of vectors, with a bound
iteration variable var
* (prod ?var ... ) scalar     Product of scalars, with
bound iteration variable var
* (max ...)     scalar      Take the maximum of all the
input scalars
* (min ...)     scalar      Take the minimum of all the
input scalars
* (pow ...)     scalar      Raise the first scalar to the
power of the second scalar.
*/

```

```

/* IsoMap Operation Base Class
* This is an abstract class, representing any vector or scalar
operation that can be used in
* the formula program. The concept of an operation is recursive. The
operands of an
* operation may themselves be the result of operations. This class
provides methods to
* automate this recursion, and overhead common to many or all
operations. It also provides
* mechanisms for a specific operation to perform its unique operation
and produce its result. */
class IsoOperation
{
public:
    IsoOperation(istream &instream, string func_name_in, bool
is_vector_in);
    ~IsoOperation();

    bool isVector() const;
    bool isConstant() const;
    bool isLoop() const;

    IsoOperation *baseParse(istream &instream, vector<string>
bound_vars);

    IsoVector vectorValue(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    double scalarValue(const map<string,IsoVector> &bound_vars, const
vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    void countOperands();
    bool validate();

protected:

```

```

vector<IsoOperation*> operand_list;
int vect_op_cnt;
int scal_op_cnt;
int idx_val;
string varname;
bool is_loop;

IsoOperation(bool is_vector_in, string func_name_in);
IsoOperation(string varname_in);
IsoOperation(double literal_in);

void setConstant(IsoVector vect_constant_in);
void setConstant(double scalar_constant_in);
void setLoopVar(string loop_varname_in);
void setVar(string varname_in);
virtual IsoVector evaluateVector(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars);
virtual double evaluateScalar(const map<string,IsoVector>
&bound_vars, const vector<IsoVector> &canal_arr, map<string,double>
&bound_scalars);
virtual IsoVector evaluateLoop(const vector<IsoVector>
&operands);
virtual double evaluateLoop(const vector<double> &operands);

virtual bool validatePriv(void) = 0;

private:
bool is_vector;
bool is_constant;
bool is_var;
double scalar_constant;
IsoVector vect_constant;
string func_name;
};

IsoOperation *createOperation(char type_ind, string func_name, istream
&instream);
char *preprocess_formula(string raw_formula);
IsoOperation *parse_formula(const IsoConfig &cfg);

/* Variable Operation
* This is a trivial operation, which produces a result equal to the
variable it represents.
* The result is a vector. It takes no inputs. */
class IOVariable : public IsoOperation
{
public:
IOVariable(string varname_in);

protected:
IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
bool validatePriv();
};

/* Literal Operation
* This is a trivial operation, which allows constant scalars to appear
in a formula. It produces a result

```

```

    * that is equal to the number it represents in the formula. Its value
    can be defined as either a symbolic
    * constant (from the command line or input file), or as a literal
    number in the formula itself.
    * The result is a scalar. It takes no inputs. */
class IOLiteral : public IsoOperation
{
private:
    double my_literal;
    bool is_symbol;
    string symbol_name;

public:
    IOLiteral(double literal_in);
    IOLiteral(string symbol_name_in);

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Positive Operation
* If the input is positive, the result is equal to the input.
Otherwise, the result is 0. It can be used to
* operate on only those values that are positive.
* The result is a scalar. It takes a single scalar input. */
class IOPos : public IsoOperation
{
public:
    IOPos(istream &instream) : IsoOperation(instream, "pos", false)
    {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Negative Operation
* If the input is negative, the result is equal to the input.
Otherwise, the result is 0. It can be used to
* operate on only those values that are negative.
* The result is a scalar. It takes a single scalar input. */
class IONeg : public IsoOperation
{
public:
    IONeg(istream &instream) : IsoOperation(instream, "neg", false)
    {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Absolute value operation
* If the input is positive, the result is equal to the input.
Otherwise, the result is the input multiplied by -1.
* The result is always greater than or equal to 0.

```

```

    * The result is a scalar. It takes a single scalar input. */
class IOAbs : public IsoOperation
{
public:
    IOAbs(istream &instream) : IsoOperation(instream, "abs", false)
    {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Vector Normalization Operation
 * This operation produces a result equal to the normalized input
vector. It produces a unitless vector with magnitude 1.0, with
 * the same direction as the input.
 * The result is a vector. It takes a single vector input. */
class IONormalize : public IsoOperation
{
public:
    IONormalize(istream &instream) : IsoOperation(instream, "norm",
true) {}

protected:
    IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Vector Magnitude Operation
 * This operation's result is the magnitude of the input vector.
 * The result is a scalar. It takes a single vector input. */
class IOMagnitude : public IsoOperation
{
public:
    IOMagnitude(istream &instream) : IsoOperation(instream, "mag",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Operation to Convert Cartesian Components into a Vector
 * The three scalar inputs are used to create a vector. The first
value is the X component, followed by Y, then Z.
 * The result is a vector. It takes exactly three scalar inputs. */
class IOVectorize : public IsoOperation
{
public:
    IOVectorize(istream &instream) : IsoOperation(instream, "vect",
true) {}

protected:
    IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

```

```

/* Component Retrieval Operation
 * A single component of the input vector is selected by the input
integer, and produced
 * as the result.
 * The result is a scalar. It takes one vector input, followed by one
integer input. */
class IOComponent : public IsoOperation
{
public:
    IOComponent(istream &instream) : IsoOperation(instream, "comp",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Addition Operation
 * All of the input scalars are added and produced as the result.
 * The result is a scalar. It takes at least two scalar inputs. */
class IOAddScalar : public IsoOperation
{
public:
    IOAddScalar(istream &instream) : IsoOperation(instream, "+",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Vector Addition Operation
 * All of the input vectors are added and produced as the result.
 * The result is a vector. It takes at least two vector inputs. */
class IOAddVector : public IsoOperation
{
public:
    IOAddVector(istream &instream) : IsoOperation(instream, "+v",
true) {}

protected:
    IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Subtraction Operation
 * The second input scalar (and all other inputs following it) are
subtracted from the first input scalar.
 * The result is a scalar. It takes at least two scalar inputs. */
class IOSubtractScalar : public IsoOperation
{
public:
    IOSubtractScalar(istream &instream) : IsoOperation(instream, "-",
false) {}

protected:

```

```

    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Subtraction Operation
 * The second input vector (and all other inputs following it) are
subtracted from the first input vector.
 * The result is a vector. It takes at least two vector inputs. */
class IOsubtractVector : public IsoOperation
{
public:
    IOsubtractVector(istream &instream) : IsoOperation(instream, "-
v", true) {}

protected:
    IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Multiplication Operation
 * All of the input scalars are multiplied together to produce the
result.
 * The result is a scalar. It takes at least two scalar inputs. */
class IOMultiplyScalar : public IsoOperation
{
public:
    IOMultiplyScalar(istream &instream) : IsoOperation(instream, "*",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
};

/* Vector Scaling Operation
 * The input vectors are multiplied together (if applicable) and used
to scale the input vector to produce the result.
 * The result is a vector. It takes one or more scalar inputs and one
vector input, in any order. */
class IOMultiplyVector : public IsoOperation
{
public:
    IOMultiplyVector(istream &instream) : IsoOperation(instream,
"*v", true) {}

protected:
    IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Division Operation
 * The first input scalar is divided by the second input scalars (and
all other inputs following it) to produce the result.
 * The result is a scalar. It takes at least two scalar inputs. */
class IOdivideScalar : public IsoOperation
{

```

```

public:
    IODivideScalar(istream &instream) : IsoOperation(instream, "/",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Vector Division Operation
* The scalar inputs are multiplied together, and the input vector is
scaled by the multiplicative inverse of that.
* The magnitude is effectively divided by all of the scalars.
* The result is a vector. It takes one vector input, and one or more
scalar inputs. */
class IODivideVector : public IsoOperation
{
public:
    IODivideVector(istream &instream) : IsoOperation(instream, "/v",
true) {}

protected:
    IsoVector evaluateVector(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Dot Product Operation
* The result is the dot product of the input vectors.
* The result is a scalar. It takes exactly two vector inputs. */
class IODotProduct : public IsoOperation
{
public:
    IODotProduct(istream &instream) : IsoOperation(instream, "dot",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Average Operation
* The result is the average of the input scalars.
* The result is a scalar. It takes one loop variable and exactly one
input operation, which evaluates to a scalar. */
class IOAverageScalar : public IsoOperation
{
public:
    IOAverageScalar(istream &instream) : IsoOperation(instream,
"avg", false) {}

protected:
    double evaluateLoop(const vector<double> &operands);
    bool validatePriv();
};

/* Vector Average Operation

```

```

    * The result is the average of the input vector. The formula is
    similar to a scalar average. The input
    * vectors are added together. Then, that sum is divided by the number
    of input vectors.
    * The result is a vector. It takes one loop variable and exactly one
    input operation, which evaluates to a vector. */
class IOAverageVector : public IsoOperation
{
public:
    IOAverageVector(istream &instream) : IsoOperation(instream,
"avgv", true) {}

protected:
    IsoVector evaluateLoop(const vector<IsoVector> &operands);
    bool validatePriv();
};

/* Iterating Scalar Sum
 * This operation computes the input operation once per each canal, and
    sums the result of each iteration
    * to produce the result.
    * The result is a scalar. It takes one loop variable and exactly one
    input operation, which evaluates to a scalar. */
class IOSumScalar : public IsoOperation
{
public:
    IOSumScalar(istream &instream) : IsoOperation(instream, "sum",
false) {}

protected:
    double evaluateLoop(const vector<double> &operands);
    bool validatePriv();
};

/* Iterating Vector Sum
 * This operation computes the input operation once per each canal, and
    sums the result of each iteration
    * to produce the result.
    * The result is a vector. It takes one loop variable and exactly one
    input operation, which evaluates to a vector. */
class IOSumVector : public IsoOperation
{
public:
    IOSumVector(istream &instream) : IsoOperation(instream, "sumv",
true) {}

protected:
    IsoVector evaluateLoop(const vector<IsoVector> &operands);
    bool validatePriv();
};

/* Iterating Scalar Product
 * This operation computes the input operation once per each canal, and
    multiplies the result of each iteration
    * to produce the result.
    * The result is a scalar. It takes one loop variable and exactly one
    input operation, which evaluates to a scalar. */
class IOProduct : public IsoOperation
{
public:

```

```

        IOProduct(istream &istream) : IsoOperation(istream, "prod",
false) {}

protected:
    double evaluateLoop(const vector<double> &operands);
    bool validatePriv();
};

/* Scalar Maximum Operation
 * This operation finds the maximum value within the input values.
 * The result is a scalar. It takes one or more scalar inputs. */
class IOMaximum : public IsoOperation
{
public:
    IOMaximum(istream &istream) : IsoOperation(istream, "max",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Minimum Operation
 * This operation finds the minimum value within the input values.
 * The result is a scalar. It takes one or more scalar inputs. */
class IOMinimum : public IsoOperation
{
public:
    IOMinimum(istream &istream) : IsoOperation(istream, "min",
false) {}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

/* Scalar Exponentiation Operation
 * This operation raises the first input to the power of the second
input.
 * The result is a scalar. It takes exactly two scalar inputs. */
class IOPow : public IsoOperation
{
public:
    IOPow(istream &istream) : IsoOperation(istream, "pow", false)
{}

protected:
    double evaluateScalar(const map<string,IsoVector> &bound_vars,
const vector<IsoVector> &canal_arr, map<string,double> &bound_scalars);
    bool validatePriv();
};

#endif /* ISOMAP_PROG_H */

```

## ▣ ppm2bmp.cpp

```

#include <iostream>

using namespace std;

/*****
 * NAME: printrec
 *
 * DESCRIPTION: This function prints the commands
 *             for converting a single image file.
 *
 * RETURNS: N/A
 *****/
void printrec(
    const char *filename_base, /* The base name of the output
file, excluding the parent directory, output suffix, and file
extension. */
    const char *variant ) /* The output suffix used in isomap-
gen to generate the output. */
{
    /* Open the next image, without requiring any user action. */
    cout << "(let* ((image (car (gimp-file-load RUN-NONINTERACTIVE
\"output/" << filename_base << "." << variant << ".ppm\" \"\")))\n";
    /* Get a handle to the image data itself. */
    cout << "      (drawable (car (gimp-image-active-drawable
image))))\n";
    /* Save the entire file as-is, but in the BMP format. */
    cout << "      (gimp-file-save  RUN-NONINTERACTIVE image
drawable \"output/" << filename_base << "." << variant << ".bmp\"
\"\")\n";
    /* Crop the image data to remove the left and right-hand regions
of the image, which duplicate the grayscale and have now contour lines.
*/
    cout << "      (gimp-image-crop image 1024 512 512 0)\n";
    /* Save the cropped image in the BMP format. */
    cout << "      (gimp-file-save  RUN-NONINTERACTIVE image
drawable \"output/" << filename_base << "-cropped." << variant <<
".bmp\" \"\")\n";
    /* Unload the image data. */
    cout << "      (gimp-image-delete image)\n";
    cout << ")\n";
}

/*****
 * NAME: main
 *
 * DESCRIPTION: This is the main routine of the
 *             PPM-to-BMP helper program. This program is
 *             only a build tool. The first command-line
 *             argument is the output suffix. Every
 *             subsequent command-line argument is the base
 *             name of a dataset.
 *
 *             This program outputs commands for GIMP's
 *             (GNU Image Project) LISP script interpreter to
 *             stdout. The stdout of this program can be
 *             piped directly to GIMP.
 *
 * RETURNS: -1 if there were too few arguments.  0
 *****/

```

```

* otherwise.
*****/
int main( int argc, char **argv )
{
    if (argc < 3)
    {
        cerr << "Too few arguments.  USAGE:\n";
        cerr << argv[0] << " suffix dataset1 [dataset2...]\n";
        return -1;
    }

    for (int i = 2; i < argc; ++i)
    {
        printrec(argv[i], argv[1]);
    }

    cout << "(gimp-quit 0)\n";

    return 0;
}

```

## vector.cpp

```

#include "vector.h"
#include <math.h>

#ifdef WIN32
#include <float.h>
#else /* WIN32 */
using namespace std;
#endif /* WIN32 */

/*****
* NAME: IsoVector (default)
*
* DESCRIPTION: This is the default constructor for
* the IsoVector class. The X, Y, and Z
* components are all initialized to 0.
*
* RETURNS: N/A
*****/
IsoVector::IsoVector()
{
    x = 0.0;
    y = 0.0;
    z = 0.0;
}

/*****
* NAME: IsoVector
*
* DESCRIPTION: This constructor initializes the
* vector from the three Cartesian components, X,
* Y, and Z.
*
* RETURNS: N/A
*****/

```

```

*****/
IsoVector::IsoVector(
    double _x, /* The X component of the new vector. */
    double _y, /* The Y component of the new vector. */
    double _z) /* The Z component of the new vector. */
{
    x = _x;
    y = _y;
    z = _z;
}

/*****
* NAME: IsoVector
*
* DESCRIPTION: This constructor creates a unit-
* length vector from the angular spherical
* components. It is especially useful for
* creating the rotation axis at each sample.
*
* RETURNS: N/A
*****/
IsoVector::IsoVector(
    double bearing, /* The bearing component, in degrees. */
    double pitch) /* The pitch component, in degrees. */
{
    /* Convert from degrees to radians, so they can be
    * used in the upcoming trigonometric functions. */
    bearing = bearing * RADIANS_PER_DEGREE;
    pitch = pitch * RADIANS_PER_DEGREE;

    /* This formula actually does two transformations at once,
    beginning with the [1,0,0] vector:
    * 1. The cos(pitch) and sin(pitch) terms rotate the vector up,
    around the Y axis.
    * 2. The cos(bearing) and sin(bearing) terms rotate that around
    the Z axis. */
    x = cos(bearing) * cos(pitch);
    y = sin(bearing) * cos(pitch);
    z = sin(pitch);
}

/*****
* NAME: IsoVector
*
* DESCRIPTION: This constructor creates a unit-
* length vector from an X-Y pixel location in the
* output image space. It works by converting the
* pixel X-Y into spherical angular coordinates,
* and then to Cartesian coordinates.
*
* RETURNS: N/A
*****/
IsoVector::IsoVector(
    int pixel_x, /* X coordinate of the pixel, in image space.
*/
    int pixel_y, /* Y coordinate of the pixel, in image space.
*/
    int horiz_resolution) /* Horizontal resolution of the
image, excluding the duplicate regions on the left and right.

```

```

                                * The vertical resolution is half
this value. */
{
    /* Convert the pixel values into bearing and pitch. This mapping
is easy, since
    * the image was projected such that X represents bearing and Y
represents pitch. */

    /* The entire range of X represents a span of 360 degrees.
    * Subtract 180 degrees to give it a range of -180 to 180
degrees.
    * pixel_x / resolution produces a range of 0.0 to 1.0, which is
then scaled
    * up to 360. */
    double bearing = -180.0 + 360.0 * pixel_x / horiz_resolution;

    /* The entire range of Y represents a span of 180 degrees.
    * Y=0 is the top of the image, which represents pitch=90, and the
pitch decreases as Y increases..
    *
    * The vertical resolution is half as big as big as the
horizontal resolution, so
    * pixel_y / (resolution/2) represents a range of 0.0 to 1.0, and
then scaled up to
    * 180. That is equivalent to pixel_y * 2 / resolution. */
    double pitch = 90.0 - 180.0 * 2.0 * pixel_y / horiz_resolution;

    /* The spherical coordinates are now ready, so just use the same
algorithm as the constructor from
    * spherical coordinates above. */

    bearing = bearing * RADIANS_PER_DEGREE;
    pitch = pitch * RADIANS_PER_DEGREE;

    x = cos(bearing) * cos(pitch);
    y = sin(bearing) * cos(pitch);
    z = sin(pitch);
}

/*****
* NAME: dotProduct
*
* DESCRIPTION: Compute the dot product between
* this vector and rhs.
*
* NOTES: The dot product is a well-studied
* operation. One of its most useful properties
* is that the result of dot_product(A,B) =
* magnitude(A)*magnitude(B)*cos("the angle between A and B")
*
* By operating on normalized vectors, or dividing
* out the magnitudes of A and B, you can isolate
* the cosine of the angle between them.
*
* RETURNS: The dot product result.
*****/
double IsoVector::dotProduct(
    const IsoVector &rhs) /* The right-hand-side operand of the
dot product. */
    const

```

```

{
    return x*rhs.x + y*rhs.y + z*rhs.z;
}

/*****
* NAME: crossProduct
*
* DESCRIPTION: Compute the cross product between
*   this vector and rhs.
*
* NOTES: The cross product is a well-studied,
*   standard operation. The result of
*   cross_product(A,B) has these properties:
*   1. The result's direction is perpendicular to
*   both A and B.
*   2. Since there are two such solutions to item 1,
*   this version of the cross product
*   disambiguates by using the right-hand rule.
*   3. The result's magnitude is:
*   magnitude(A)*magnitude(B)*sin("the angle between A and B")
*
* RETURNS: The cross product result.
*****/
IsoVector IsoVector::crossProduct(
    const IsoVector &rhs) /* The right-hand-side operand of the
cross product. */
const
{
    return IsoVector(y*rhs.z - z*rhs.y,
                    z*rhs.x - x*rhs.z,
                    x*rhs.y - y*rhs.x);
}

/*****
* NAME: scale
*
* DESCRIPTION: Return a copy of this vector that is
*   scaled by scalar value s. The result has the
*   same direction (or opposite if s is negative),
*   and the magnitude is multiplied by s.
*
* RETURNS: This vector scaled by s.
*****/
IsoVector IsoVector::scale(
    double s) /* The amount to scale this vector. */
const
{
    return IsoVector(x*s, y*s, z*s);
}

/*****
* NAME: add
*
* DESCRIPTION: Compute the vector sum of this
*   vector and rhs.
*
* RETURNS: The sum of this vector and rhs.
*****/
IsoVector IsoVector::add(

```

```

        const IsoVector &rhs) /* The right-hand-side operand of the
addition. */
    const
{
    return IsoVector(x+rhs.x, y+rhs.y, z+rhs.z);
}

/*****
* NAME: subtract
*
* DESCRIPTION: Compute the vector difference of
*   this vector and rhs.
*
* RETURNS: This vector, with rhs subtracted from
*   it.
*****/
IsoVector IsoVector::subtract(
    const IsoVector &rhs) /* The right-hand-side operand of the
subtraction. */
    const
{
    return IsoVector(x-rhs.x, y-rhs.y, z-rhs.z);
}

/*****
* NAME: normalize
*
* DESCRIPTION: Compute a unit-length vector that
*   has the same direction as this vector.
*
* NOTES: Normalizing a vector always makes it
*   unitless, representing a direction and nothing
*   else.
*
* RETURNS: This vector normalized.
*****/
IsoVector IsoVector::normalize() const
{
    /* Scale the magnitude such that the magnitude cancels out. */
    return scale(1.0 / magnitude());
}

/*****
* NAME: magnitude
*
* DESCRIPTION: Compute the magnitude of this vector.
*
* RETURNS: The magnitude of this vector.
*****/
double IsoVector::magnitude() const
{
    /* This formula is just the Pythagorean theorem, extended to 3D.
*/
    return sqrt(x*x + y*y + z*z);
}

/*****
* NAME: getBering
*
* DESCRIPTION: Compute the bering (in the X-Y

```

```

*   plane) of this vector.
*
* RETURNS: The bearing of this vector.
*****/
double IsoVector::getBearing() const
{
    /* Plain atan(y/x) can only resolve a range of -90 to 90 degrees,
but
    * atan2(y,x) performs an additional quadrant check to completely
define
    * the angle from -180 to 180 degrees.
    *
    * atan2() returns radians, so convert to degrees before
returning. */
    return atan2(y, x) * DEGREES_PER_RADIAN;
}

/*****
* NAME: getPitch
*
* DESCRIPTION: Compute the pitch (elevation above
the X-Y plane) of this vector.
*
* RETURNS: The pitch of this vector.
*****/
double IsoVector::getPitch() const
{
    /* Use the inverse sine to extract the pitch, then convert it
    * from radians to degrees. */
    double result = asin(z / magnitude()) * DEGREES_PER_RADIAN;

    /* If asin()'s input was not inside the range of -1.0 to 1.0,
which may arise from
    * rounding errors, the result is mathematically undefined. No
value c will ever make
    * sin(c) be outside that range.
    *
    * Inverse sine reacts by returning a special number, NaN (Not a
Number). Detect this
    * scenario, using the function appropriate for the computer's
libraries. */
#ifdef WIN32
    if (_isnan(result))
#else /* WIN32 */
    if (isnan(result))
#endif /* WIN32 */
    {
        /* The value was undefined. Clamp it to either +90 or -90,
depending
        * on which side of the range it is on. */
        if (z > 0.0)
        {
            /* Z was positive, so the input to asin() must've
been greater than 1.0.
            * A true 1.0 would result in 90, so clamp it to
that. */
            result = 90.0;
        }
        else
        {

```

```

        /* Z was negative, so the input to asin() must have
been less than -1.0.
        * Clamp it to -1.0, which would result in -90. */
        result = -90.0;
    }
}

return result;
}

/*****
* NAME: getPixelX
*
* DESCRIPTION: Compute the pixel X coordinate of
* this vector, when it is projected onto the
* image.
*
* RETURNS: The pixel X coordinate.
*****/
int IsoVector::getPixelX(
    int horiz_resolution) /* The horizontal resolution of the
image, excluding the duplicate regions on the left and right. */
const
{
    /* Convert the -180 to 180 range of the bearing to the 0 to
horiz_resolution range
    * of the pixel X coordinate. This is a simple offset and scale.
*/
    return (int)((this->getBearing() + 180.0) / 360.0 *
horiz_resolution);
}

/*****
* NAME: getPixelY
*
* DESCRIPTION: Compute the pixel Y coordinate of
* this vector, when it is projected onto the
* image.
*
* RETURNS: The pixel Y coordinate.
*****/
int IsoVector::getPixelY(
    int horiz_resolution) /* The horizontal resolution of the
image, excluding the duplicate regions on the left and right. */
const
{
    /* The vertical resolution is half as big as the horizontal
resolution. */
    int vert_resolution = horiz_resolution / 2;
    /* The pitch decreases as the pixel Y increases. Remove the +90
offset,
    * invert the pitch's direction, scale to cancel out the 180
degree range,
    * and scale back up to produce the 0 to vert_resolution range.
*/
    return (int)((90.0 - this->getPitch()) / 180.0 *
vert_resolution);
}

/*****

```

```

* NAME: correctAxes
*
* DESCRIPTION: Permute and transform the familiar
*   skull-centric coordinates into coordinates
*   suitable for OpenGL. OpenGL defines its
*   coordinate system differently, but the mapping
*   is easy.
*
* RETURNS: This vector, represented in OpenGL's
*   coordinate system.
*****/
IsoVector IsoVector::correctAxes() const
{
    return IsoVector(x, z, -y);
}

```

## vector.h

```

#ifndef ISO_VECTOR_H
#define ISO_VECTOR_H

/* Ratio of radians per degree. Multiply an angle by this value to
convert from degrees to radians. */
#define RADIANS_PER_DEGREE 3.14159265 / 180.0
/* Ratio of degrees per radian. Multiply an angle by this value to
convert from radians to degrees. */
#define DEGREES_PER_RADIAN 180.0 / 3.14159265

/* Isomap Vector
* This class represents 3D vectors, and provides utilities and
operations
* to manipulate them. Internally, the vectors are stored in Cartesian
* components, but it provides functions to convert to/from spherical
coordinates.
* It also implements the standard 3D vector operations. */
class IsoVector {
public:
    double x;
    double y;
    double z;

    IsoVector();
    IsoVector(double _x, double _y, double _z);
    IsoVector(double bearing, double pitch);
    IsoVector(int pixel_x, int pixel_y, int horiz_resolution);

    double dotProduct(const IsoVector &rhs) const;
    IsoVector crossProduct(const IsoVector &rhs) const;
    IsoVector scale(double s) const;
    IsoVector add(const IsoVector &rhs) const;
    IsoVector subtract(const IsoVector &rhs) const;
    IsoVector normalize() const;
    double magnitude() const;
    double getBearing() const;
    double getPitch() const;
    int getPixelX(int horiz_resolution) const;
    int getPixelY(int horiz_resolution) const;
    IsoVector correctAxes() const;

```

```
};  
#endif
```

## ▣ **venice-main.cpp**

```
#include "view.h"  
#include "geom-gen.h"  
#include <stdlib.h>  
#include <string.h>  
#include <iostream>  
  
using namespace std;  
  
/*****  
 * NAME: main  
 *  
 * DESCRIPTION: This is the main routine for Venice.  
 * It initializes GLUT, generates the geometry  
 * from the input file, prints the keybindings,  
 * and finally passes control to GLUT. From then  
 * on, GLUT executes its own main loop, and  
 * notifies Venice whenever events occur.  
 *  
 * RETURNS: 0 if the program exited normally. A  
 * non-zero exit value indicates an input error or  
 * abnormal termination.  
 *****/  
int main( int argc, char **argv )  
{  
    /* Without an input file, there is nothing to do. */  
    if (argc < 2)  
    {  
        cout << "ERROR: Expected input filename.\n";  
        exit(-1);  
    }  
  
    /* Install the destructor to execute if whenever the program  
exits.  
 *  
 * This is critical, since the OS does not release OpenGL  
resources upon exit  
 * (like it does for heap space and file handles). Failing to  
release OpenGL  
 * resources will leak graphics memory, and can only be fixed by  
rebooting the  
 * system. */  
    atexit(destroy_isomap);  
  
    /* Let GLUT initialize itself and OpenGL. */  
    glutInit( &argc, argv );  
  
    /* Initialize Venice's OpenGL resources. */  
    init_display();  
  
    /* Initialize the display matrices. */  
    recalc_matrices();
```

```

/* Load the image and create the geometry from it. */
load_isomap(string(argv[1]));

/* Print out the keybindings for the user's reference. */
cout << "\n\n";
cout << "*****\n";
cout << "      Controls      \n";
cout << "*****\n";
cout << "Escape      : Quit program\n";
cout << "Arrow keys  : Move camera\n";
cout << "a          : Show/hide axis arrows\n";
cout << "s          : Use sphere/lobed model\n";
cout << "f          : Move camera forward\n";
cout << "b          : Move camera back\n";
cout << "i          : Look in -X direction\n";
cout << "I          : Look in +X direction\n";
cout << "j          : Look in -Y direction\n";
cout << "J          : Look in +Y direction\n";
cout << "k          : Look in -Z direction\n";
cout << "K          : Look in +Z direction\n";
cout << "*****\n";
cout << "Green arrow is +X\n";
cout << "Red arrow is +Y\n";
cout << "Blue arrow is +Z\n";

/* Fake a keypress to force the camera to a reasonable spot. */
kb_handler( 'O', 50, 50 );

/* Defer execution to GLUT's main routine. */
glutMainLoop();

/* Should be unreachable. */
return 1;
}

```

## view.cpp

```

#include "view.h"
#include "geom-gen.h"
#include "vector.h"
#include <math.h>
#include <iostream>

using namespace std;

int window_handle;
GLfloat window_width;
GLfloat window_height;
GLfloat ambient_color[4] = { 1.0, 1.0, 1.0, 1.0 };

/* The user's position in spherical coordinates.
 * All angles are degrees. */
double cam_pos_bering = 0.0;
double cam_pos_pitch = 0.0;
double cam_pos_dist = 15.0;

bool enable_lobed_sphere = false;

```

```

bool enable_axis_arrows = true;

/*****
 * NAME: display_frame
 *
 * DESCRIPTION: This is the handler to display the
 * geometry in the window. It is installed as a
 * callback into GLUT, and will be automatically
 * called whenever the screen must be updated or
 * refreshed.
 *
 * RETURNS: N/A
 *****/
void display_frame( void )
{
    /* Clear the screen to start with a clean slate. */
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* Display the geometry of the new frame. */
    display_isomap();

    /* Reset the material. */
    GLfloat mat_zero_emission[4] = { 0.0, 0.0, 0.0, 0.0 };
    glMaterialfv( GL_FRONT, GL_EMISSION, mat_zero_emission );
    /* Flush the rendered frame to the monitor. */
    glFlush ();
}

/*****
 * NAME: reshape
 *
 * DESCRIPTION: This function is installed as a
 * callback into GLUT, to respond to the user
 * resizing the window. If the window is resized,
 * the aspect ratio must be updated, which
 * requires constructing new projection matrices.
 *
 * RETURNS: N/A
 *****/
void reshape (int w, int h)
{
    /* The user resized the window, so pass the command to
     * oGL. Whatever size the user chose is probably fine. */
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    recalc_matrices();
}

/*****
 * NAME: recalc_matrices
 *
 * DESCRIPTION: Use the user's position to construct
 * all of the transformation matrices again. Every
 * matrix is recomputed, so this function is
 * when the program starts up, whenever the user
 * moves, or resizes the window.
 *
 * RETURNS: N/A
 *****/
void recalc_matrices()

```

```

{
    /* Convert the user's position from spherical coordinates to
    Cartesian coordinates. */
    IsoVector camera_isov(cam_pos_bering, cam_pos_pitch);
    camera_isov = camera_isov.scale(cam_pos_dist).correctAxes();

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();

    gluPerspective( 40.0, (GLfloat)( window_width / window_height ),
    0.5, 200.0 );

    //get ready to make the new camera matrix
    IsoVector lookdest = IsoVector(0.0, 0.0, 0.0);

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt( camera_isov.x, camera_isov.y, camera_isov.z, //camera
location
    lookdest.x, lookdest.y, lookdest.z, //place the
camera is looking at
    0.0, 1.0, 0.0 ); // "up"
vector, for tilt; don't modify!
}

/*****
* NAME: init_display
*
* DESCRIPTION: Initialize the window, and provide
the display and window configuration. It also
* installs the OGL callbacks.
*
* RETURNS: N/A
*****/
void init_display()
{
    window_width = 512;
    window_height = 512;

    /* spawn and initialize the window itself. */
    glutInitWindowPosition(0, 0);
    glutInitDisplayMode( GLUT_RGB | GLUT_DEPTH | GLUT_SINGLE );
    glutInitWindowSize( (int>window_width, (int>window_height );
    window_handle = glutCreateWindow( "Venice" );

    /* Set the hooks to respond to events. */
    glutKeyboardFunc( kb_handler );
    glutSpecialFunc( special_kb_handler );
    glutDisplayFunc( display_frame );
    glutReshapeFunc(reshape);

    glEnable( GL_TEXTURE_2D );

    /* Set the "blank" color to white. */
    glClearColor(1.0, 1.0, 1.0, 0.0);

    /* When triangles are viewed from their front, their vertices
wind in
    * the counter-clockwise direction. */
    glFrontFace( GL_CCW );

```

```

    /* Always make the closer polygons cover the farther ones. */
    glDepthFunc( GL_LESS );
    glEnable( GL_DEPTH_TEST );

    /* Use the Gouraud model for simplicity.
    * The only lights used are ambient, so this lighting model is
    * more than sufficient. */
    glShadeModel( GL_SMOOTH );

    /* Configure the lighting, which consists of only one ambient
light.
* Other lighting models will interfere with the meaning of dark
and light
* pixels, which are supposed to represent sensitivity values. */
    glLightfv( GL_LIGHT0, GL_AMBIENT, ambient_color );
    glEnable( GL_LIGHT0 );
    glEnable( GL_LIGHTING );
    glEnable( GL_COLOR_MATERIAL );
}

/*****
* NAME: kb_handler
*
* DESCRIPTION: This function is a callback for GLUT
* to report the user's keypresses. This one
* handles the non-special keys. The arrow keys
* are handled by oGLSpecialKeyboardHandler.
*
* RETURNS: N/A
*****/
void kb_handler(
    unsigned char key, /* The character for the key pressed by
the user. */
    int x, /* The mouse X coordinate when the key was pressed.
Ignored. */
    int y) /* The mouse Y coordinate when the key was pressed.
Ignored. */
{
    /* Update the global variables based on the key that was pressed.
    * If the key is not mapped to any action, silently ignore it. */
    switch( key )
    {
        case IsoKeyBindings::ToOrigin:
            cam_pos_bering = 0.0;
            cam_pos_pitch = 0.0;
            cam_pos_dist = 20.0;
            break;

        case IsoKeyBindings::ExitProgram:
            exit(0);
            break;

        case IsoKeyBindings::GoForward:
            cam_pos_dist -= 0.1;
            break;

        case IsoKeyBindings::GoBackward:
            cam_pos_dist += 0.1;
            break;
    }
}

```

```

        case IsoKeyBindings::LookPosX:
            cam_pos_pitch = 0.0;
            cam_pos_bering = 180.0;
            break;

        case IsoKeyBindings::LookNegX:
            cam_pos_pitch = 0.0;
            cam_pos_bering = 0.0;
            break;

        case IsoKeyBindings::LookPosY:
            cam_pos_pitch = 0.0;
            cam_pos_bering = -90.0;
            break;

        case IsoKeyBindings::LookNegY:
            cam_pos_pitch = 0.0;
            cam_pos_bering = 90.0;
            break;

        case IsoKeyBindings::LookPosZ:
            cam_pos_pitch = -90.0;
            cam_pos_bering = 0.0;
            break;

        case IsoKeyBindings::LookNegZ:
            cam_pos_pitch = 90.0;
            cam_pos_bering = 0.0;
            break;

        case IsoKeyBindings::ToggleLobedSphere:
            enable_lobed_sphere = !enable_lobed_sphere;
            break;

        case IsoKeyBindings::ToggleAxisArrows:
            enable_axis_arrows = !enable_axis_arrows;
            break;
    };

    /* Refresh the display, with the parameters updated by the user.
*/
    redisplay();
}

/*****
* NAME: special_kb_handler
*
* DESCRIPTION: Handle key presses of "special"
* keys that are not mapped to ASCII characters.
* This function is installed as a callback into
* GLUT as the special key handler. The only
* special keys to handle are the arrow keys.
*
* RETURNS: N/A
*****/
void special_kb_handler(
    int key, /* An enumerated value representing which key was
pressed. */
    int x, /* The mouse X coordinate when the key was pressed.
Ignored. */

```

```

    int y) /* The mouse Y coordinate when the key was pressed.
Ignored. */
{
    /* Choose which direction to move the user, if at all,
    * based on which key was pressed.
    *
    * Moving the user, while keeping the lookat point
    * fixed at the origin, has the effect of rotating
    * the model.
    *
    * When no other landmarks exist, each are equally valid.
    * Moving the camera is more natural for the user, so
    * assign the arrow directions that way. */
    switch( key )
    {
    case IsoKeyBindings::MoveUp:
        cam_pos_pitch += 1.0;
        break;

    case IsoKeyBindings::MoveDown:
        cam_pos_pitch -= 1.0;
        break;

    case IsoKeyBindings::MoveLeft:
        cam_pos_bering -= 1.0;
        break;

    case IsoKeyBindings::MoveRight:
        cam_pos_bering += 1.0;
        break;
    };

    /* Refresh the display, with the parameters updated by the user.
*/
    redisplay();
}

/*****
* NAME: redisplay
*
* DESCRIPTION: Error-check the viewpoint and
* request for GLUT to refresh the display. GLUT
* will call the display function when it's ready.
*
* RETURNS: N/A
*****/
void redisplay()
{
    /* Never allow the user to look straight up or down.
    * The cross products required to generate the
    * lookat matrix will explode, and the system
    * will render garbage. */
    if (cam_pos_pitch > 89.99)
    {
        cam_pos_pitch = 89.98;
    }
    else if (cam_pos_pitch < -89.99)
    {
        cam_pos_pitch = -89.98;
    }
}

```

```

    /* Keep the bearing within the range of 0-360 degrees.
    * There is no hazard like the pitch has, but it keeps the
    * bearing from getting too large and losing precision. */
    if (cam_pos_bearing > 360.0)
    {
        cam_pos_bearing -= 360.0;
    }
    else if(cam_pos_bearing < -0.001)
    {
        cam_pos_bearing += 360.0;
    }

    /* Never let the user get so close to the origin that the user
    travels
    * inside the model. The user will get lost because OpenGL will
display
    * nothing. */
    if (cam_pos_dist < 8.5f)
    {
        cam_pos_dist = 8.5f;
    }

    /* Ask GLUT to re-display the geometry. GLUT will do so as soon
    as possible,
    * but when it's convenient. */
    glutPostRedisplay();
}

```

## view.h

```

#ifndef __ICG_VIEW_H__
#define __ICG_VIEW_H__

#include <stdlib.h>

#ifdef linux
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#endif

#ifdef darwin
#include <GLUT/glut.h>
#endif

/* Key Bindings Assignments
* These constants allow the key bindings to be used symbolically.
* Only the keys handled by Venice are listed.
*/
namespace IsoKeyBindings
{
    /* Resets the view in case the user gets lost. */
    const char ToOrigin = 'O';
    /* Exit the program (escape key). */
    const char ExitProgram = 0x1b;
}

```

```

/* Go forward, closer to the sphere. */
const char GoForward = 'f';
/* Go backward, away from the sphere. */
const char GoBackward = 'b';

/* Look direction presets.
 * The view is constrained to always look at the origin (where the
 * model is).
 * These 6 options move the user to the position such that looking at
 * the origin
 * means the user is looking in the specified direction. */
const char LookPosX = 'I';
const char LookNegX = 'i';
const char LookPosY = 'J';
const char LookNegY = 'j';
const char LookPosZ = 'K';
const char LookNegZ = 'k';
/* Toggle between the pure sphere and lobed models. */
const char ToggleLobedSphere = 's';
/* Toggle between enabling and disabling the axis arrows being
 * displayed. */
const char ToggleAxisArrows = 'a';

/* Special keys.
 * These are the arrow keys. They move the user around the model in
 * the
 * direction specified. As usual, the user's look direction stays
 * fixed
 * at the origin. */
const int MoveUp = GLUT_KEY_UP;
const int MoveDown = GLUT_KEY_DOWN;
const int MoveLeft = GLUT_KEY_LEFT;
const int MoveRight = GLUT_KEY_RIGHT;
};

extern bool enable_lobed_sphere;
extern bool enable_axis_arrows;

void display_frame( void );
void recalc_matrices();
void kb_handler(unsigned char key, int x, int y);
void special_kb_handler( int key, int x, int y );
void init_display();
void reshape (int w, int h);
void redisplay();

#endif

```

## 5. ■ Makefile

The Makefile file tells the computer how to turn all that source code into a program you can run.

```

DEBUG_FLAG = -g
CPPFLAGS = -Wall

```

```

INCLUDEPATH = -I.
GEN_OBJS = obj/isomap-gen.o obj/vector.o obj/isomap-prog.o obj/isomap-
config.o
VENICE_OBJS = obj/geom-gen.o obj/venice-main.o obj/vector.o obj/view.o
VENICE_LIBFLAGS = -L/usr/X11/lib -lX11 -lXi -lXmu -lglut -lGL -lGLU -lm
all : isomap-gen ppm2bmp

isomap-gen : ${GEN_OBJS}
    g++ ${DEBUG_FLAG} -o $@ ${GEN_OBJS}

venice : ${VENICE_OBJS}
    g++ ${DEBUG_FLAG} ${VENICE_LIBFLAGS} -o $@ ${VENICE_OBJS}

obj/%.o : src/%.cpp
    g++ -c ${DEBUG_FLAG} -o $@ ${CPPFLAGS} ${INCLUDEPATH}
    ${DEBUG_FLAG} $<

ppm2bmp : src/ppm2bmp.cpp
    g++ ${DEBUG_FLAG} -o $@ $<

clean :
    rm -f ${GEN_OBJS} ${VENICE_OBJS} isomap-gen ppm2bmp output/*.ppm
    output/*.bmp output/*-hist.*.txt output/prime-dir-summary.csv venice

render : isomap-gen ppm2bmp render.sh input-list.dat
    ./render.sh

output/%.f.bmp : output/%.f.ppm ppm2bmp
    ./ppm2bmp f $* | gimp -i -d -b -

output/%.fpd.bmp : output/%.fpd.ppm ppm2bmp
    ./ppm2bmp fpd $* | gimp -i -d -b -

output/%.f.ppm : input/%.txt isomap-gen
    ./isomap-gen --dataset $* --preset FINAL --out-suffix f --auto-
color

output/%.fpd.ppm : input/%.txt isomap-gen
    ./isomap-gen --dataset $* --preset FINAL --out-suffix fpd --auto-
color --prime-dir

% : output/%.f.bmp output/%.fpd.bmp
    echo making $@

```

## 6. render.sh

```

#!/bin/sh
IFS=$': \n\t'
TARGETS=`cat input-list.dat`

SUMMARY_FILE="summary"
GAIN_VARS="--default-arg gain_slope 0.23 --default-arg gain_ofs 0.09"

```

```

echo "Specimen,Output Suffix,Canal,Radius,Regular X,Regular Y,Regular
Z,Prime X,Prime Y,Prime Z,Angle" > output/$SUMMARY_FILE.csv

for fileroot in $TARGETS; do
    ./isomap-gen --dataset $fileroot --preset RODGERS_ABS_GAIN --out-
suffix gabs2 --auto-color --summary-file $SUMMARY_FILE $GAIN_VARS
    if [[ "$?" != "0" ]]; then
        exit 1;
    fi
done

cat input-list.dat | xargs ./ppm2bmp gabs | gimp -i -d -b -

rm -f isomap-output.zip isomap-output-ppm.zip
cd output
zip -9 isomap-output *.bmp *-hist.*.txt output/$SUMMARY_FILE.csv
zip -9 isomap-output-ppm *.ppm *-hist.*.txt output/$SUMMARY_FILE.csv
mv isomap-output.zip isomap-output-ppm.zip ..
cd ..

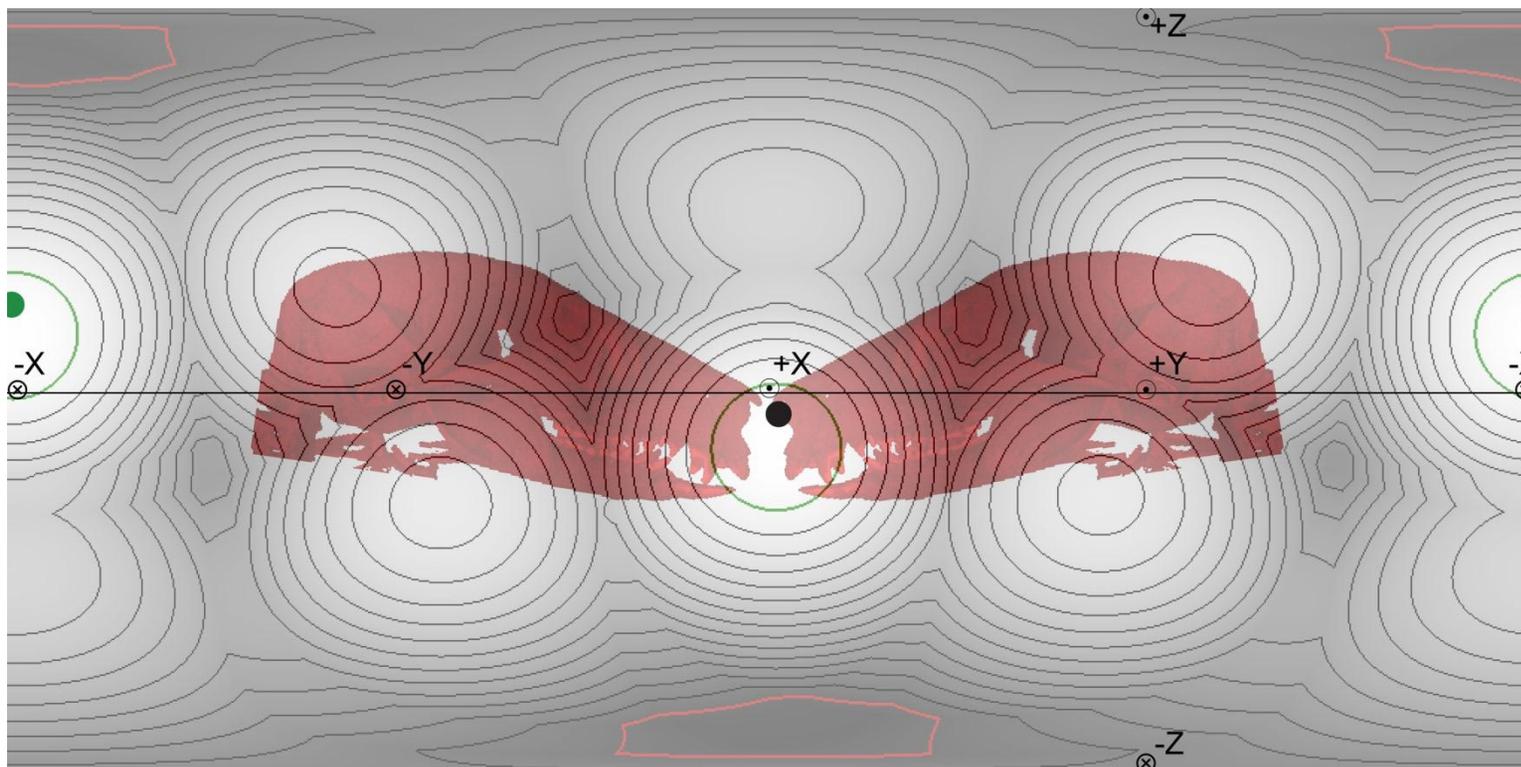
```

**APPENDIX 7. SENSITIVITY RESULTS FOR ALL TAXA. HEAD-CENTERED SENSITIVITY RESULTS GIVEN IN (SPIKES • SEC<sup>-1</sup>)/(DEGREES • SEC<sup>-1</sup>), WITH THE RATIO OF THE MAXIMUM/MINIMUM VALUES. RESULTS ARE SHOWN GRAPHICALLY IN APPENDIX 8.**

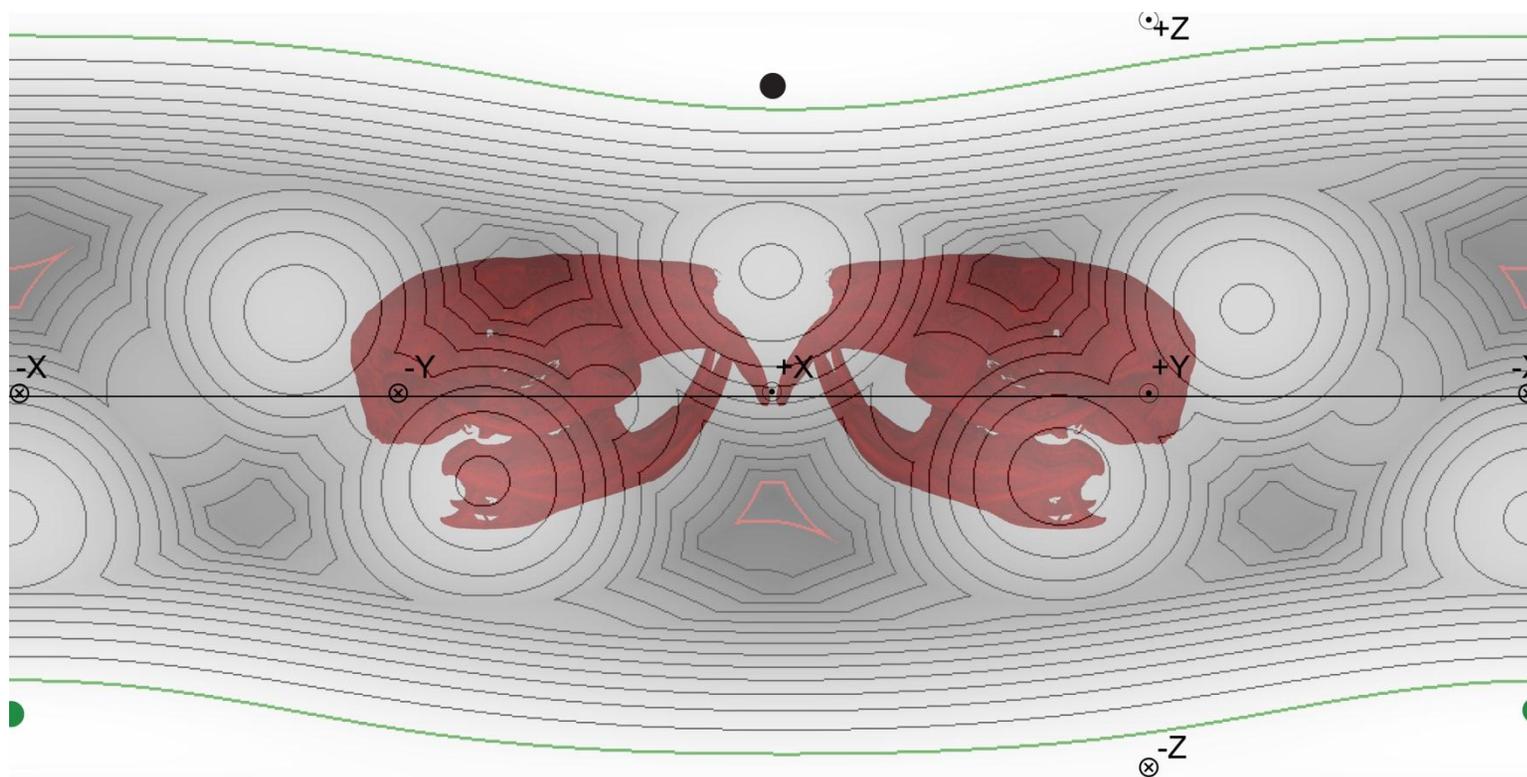
Name	Maximum	Minimum	Maximum/ Minimum	Contour Interval	Maximum Contour	Minimum Contour
<i>Acrobates</i>	0.42	0.21	2.05	0.01	0.40	0.22
<i>Allactaga</i>	1.35	0.72	1.88	0.04	1.27	0.76
<i>Anomalurus</i>	1.23	0.63	1.93	0.04	1.15	0.67
<i>Caluromys</i>	0.68	0.39	1.75	0.02	0.64	0.41
<i>Cercartetus</i>	0.40	0.19	2.15	0.01	0.38	0.20
<i>Chironectes</i>	0.71	0.35	2.01	0.02	0.67	0.38
<i>Chrysochloris</i>	0.38	0.14	2.83	0.02	0.35	0.15
<i>Crocuta</i>	2.17	1.08	2.00	0.07	2.04	1.15
<i>Dactylopsila</i>	0.87	0.40	2.20	0.03	0.81	0.43
<i>Dromiciops</i>	0.38	0.20	1.88	0.01	0.36	0.21
<i>Enhydra</i>	1.62	0.86	1.87	0.05	1.52	0.91
<i>Glaucomys</i>	0.88	0.53	1.66	0.02	0.84	0.55
<i>Hemibelideus</i>	1.52	0.70	2.17	0.05	1.42	0.75
<i>Heterocephalus</i>	0.41	0.21	1.91	0.01	0.38	0.23
<i>Lepus</i>	1.35	0.69	1.95	0.04	1.27	0.73
<i>Monodelphis</i>	0.46	0.23	1.98	0.01	0.43	0.25
<i>Notoryctes</i>	0.38	0.16	2.36	0.01	0.35	0.17
<i>Pedetes</i>	1.58	0.86	1.82	0.04	1.49	0.91
<i>Petauroides</i>	1.11	0.56	1.98	0.03	1.04	0.59
<i>Petaurus</i>	0.73	0.33	2.22	0.02	0.68	0.35
<i>Petroseudes</i>	1.16	0.63	1.84	0.03	1.09	0.66
<i>Potorous</i>	1.20	0.56	2.16	0.04	1.12	0.60
<i>Pseudocheirus</i>	1.16	0.53	2.19	0.04	1.08	0.57
<i>Pseudochirops</i>	1.11	0.51	2.19	0.04	1.04	0.55
<i>Pseudochirulus</i>	1.00	0.45	2.24	0.03	0.93	0.48
<i>Sciurus</i>	1.65	0.88	1.89	0.05	1.56	0.92
<i>Talpa</i>	0.59	0.30	1.96	0.02	0.56	0.32
<i>Tarsipes</i>	0.26	0.11	2.47	0.01	0.24	0.12
<i>Thylacinus</i>	1.40	0.60	2.32	0.05	1.30	0.65
<i>Vulpes</i>	1.62	0.87	1.87	0.05	1.52	0.91
<i>Wallabia</i>	1.81	1.04	1.74	0.05	1.71	1.09

**APPENDIX 8. ISOMAPS FOR ALL TAXA IN THIS STUDY. BLACK LINE, XY REFERENCE PLANE; GREEN CONTOUR AND DOT,  $S_{MAX}$ ; RED CONTOUR, ZONE OF  $S_{MIN}$  SENSITIVITY; BLACK DOT,  $180^\circ$  POLE TO  $S_{MAX}$  AXIS; MAXIMA AND MINIMA GIVEN IN APPENDIX 7. RED SKULLS ARE GIVEN FOR ORIENTATION PURPOSES ONLY, AND DO NOT REPRESENT SCALED FIGURES, GRAY SCALES INDICATE INCREASING AND DECREASING SENSITIVITY WITH HIGHEST SENSITIVITY IN LIGHTEST TONES.**

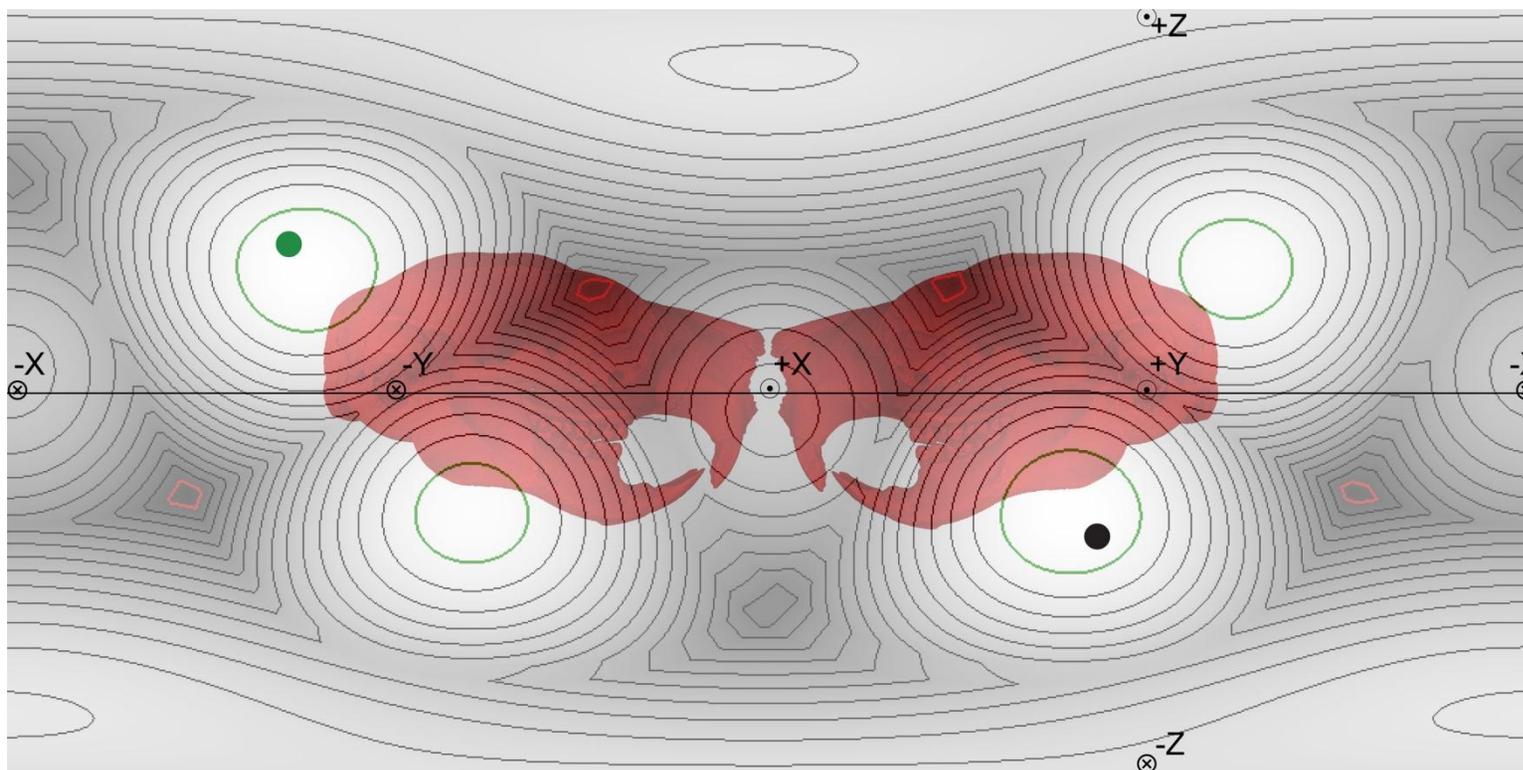
*Acrobates*



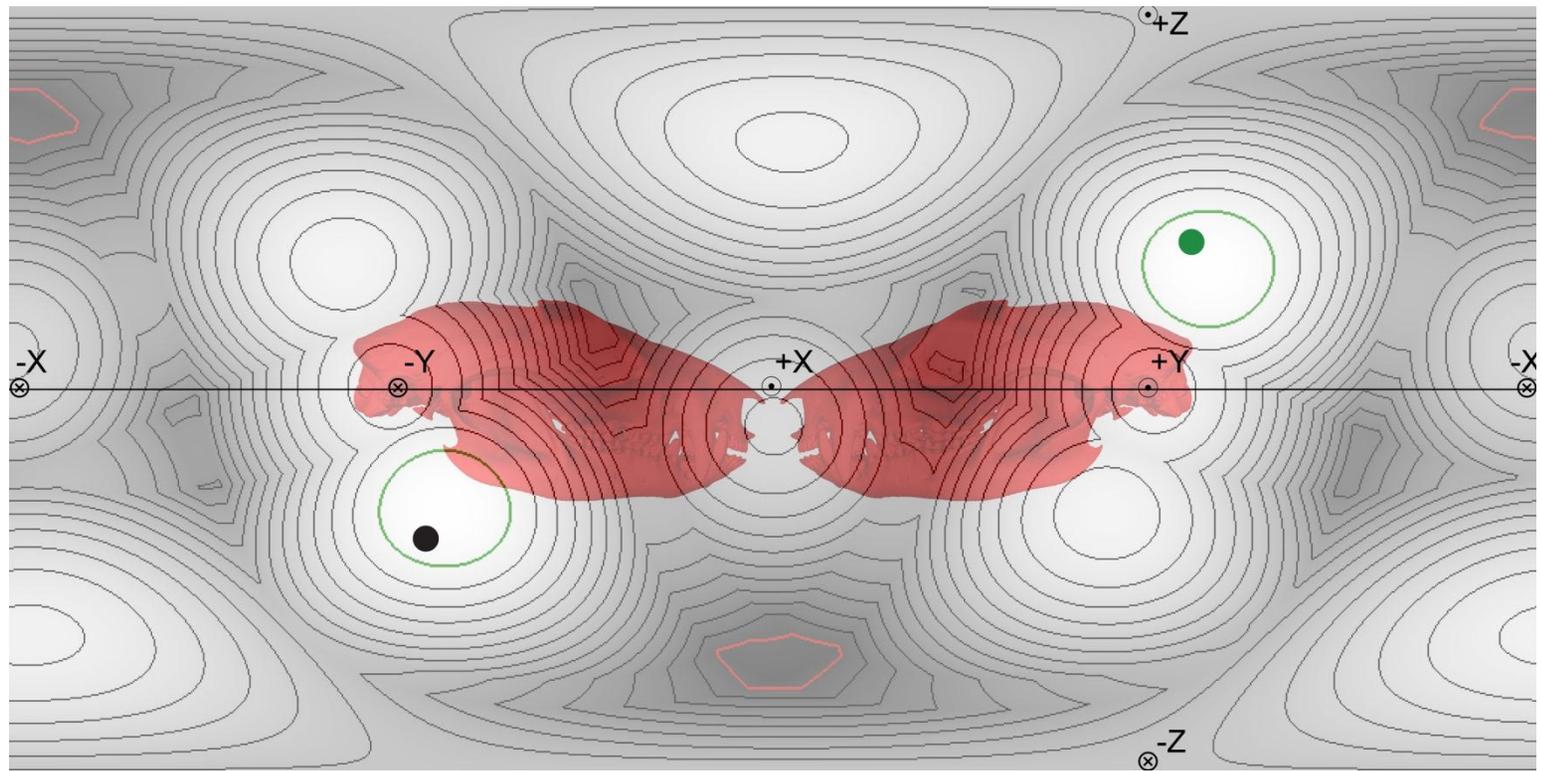
*Allactaga*



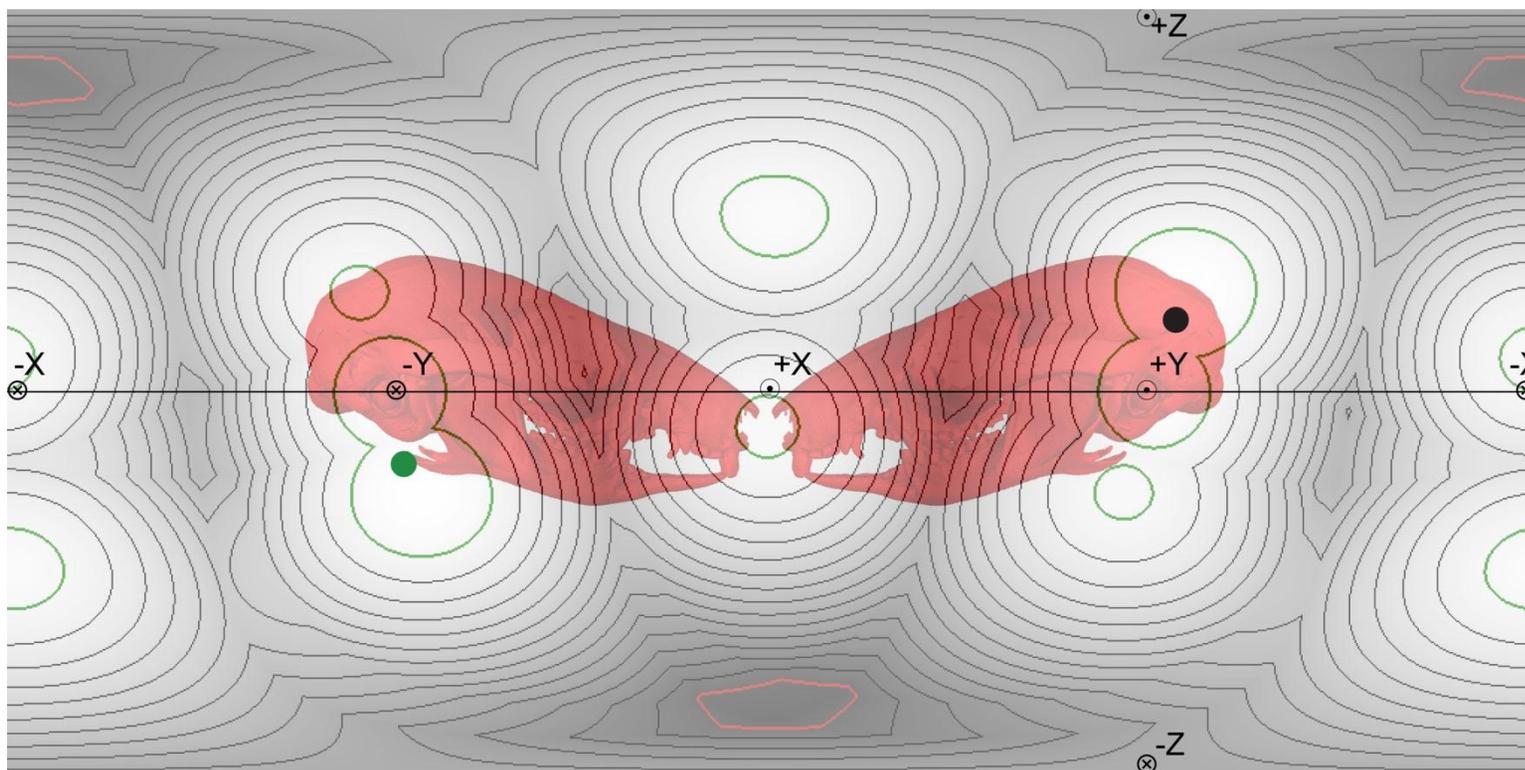
*Anomalurus*



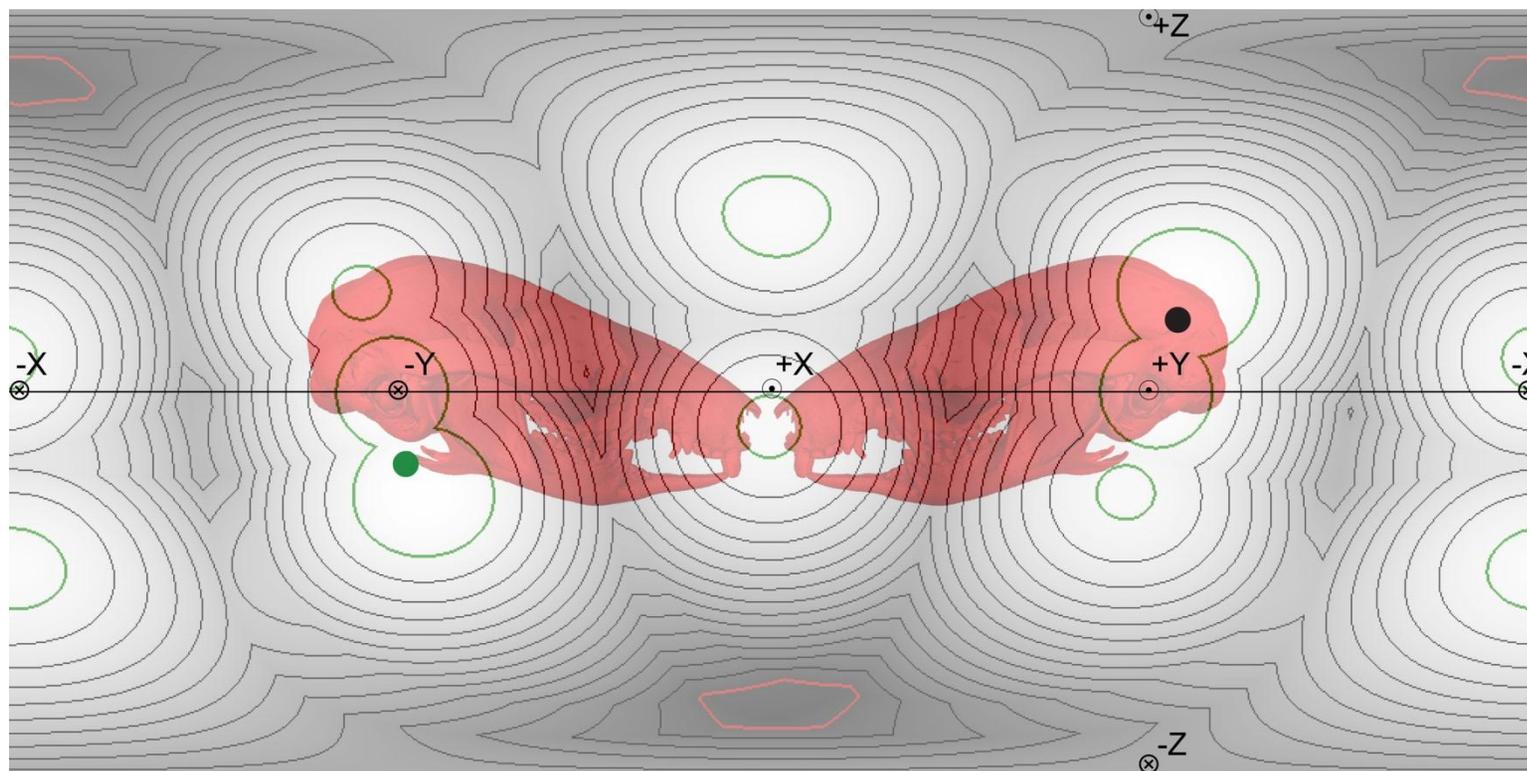
*Caluromys*



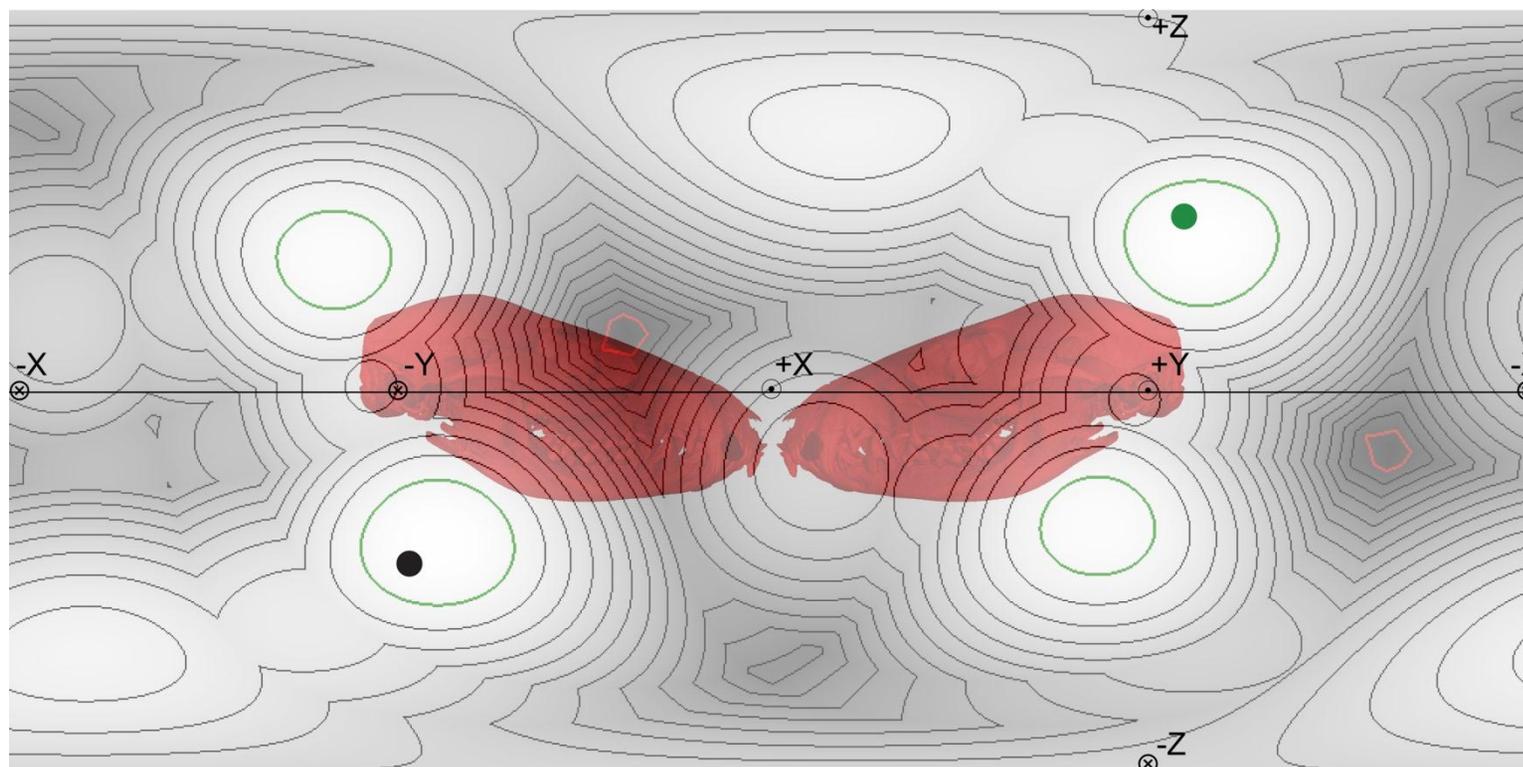
*Cercartetus*



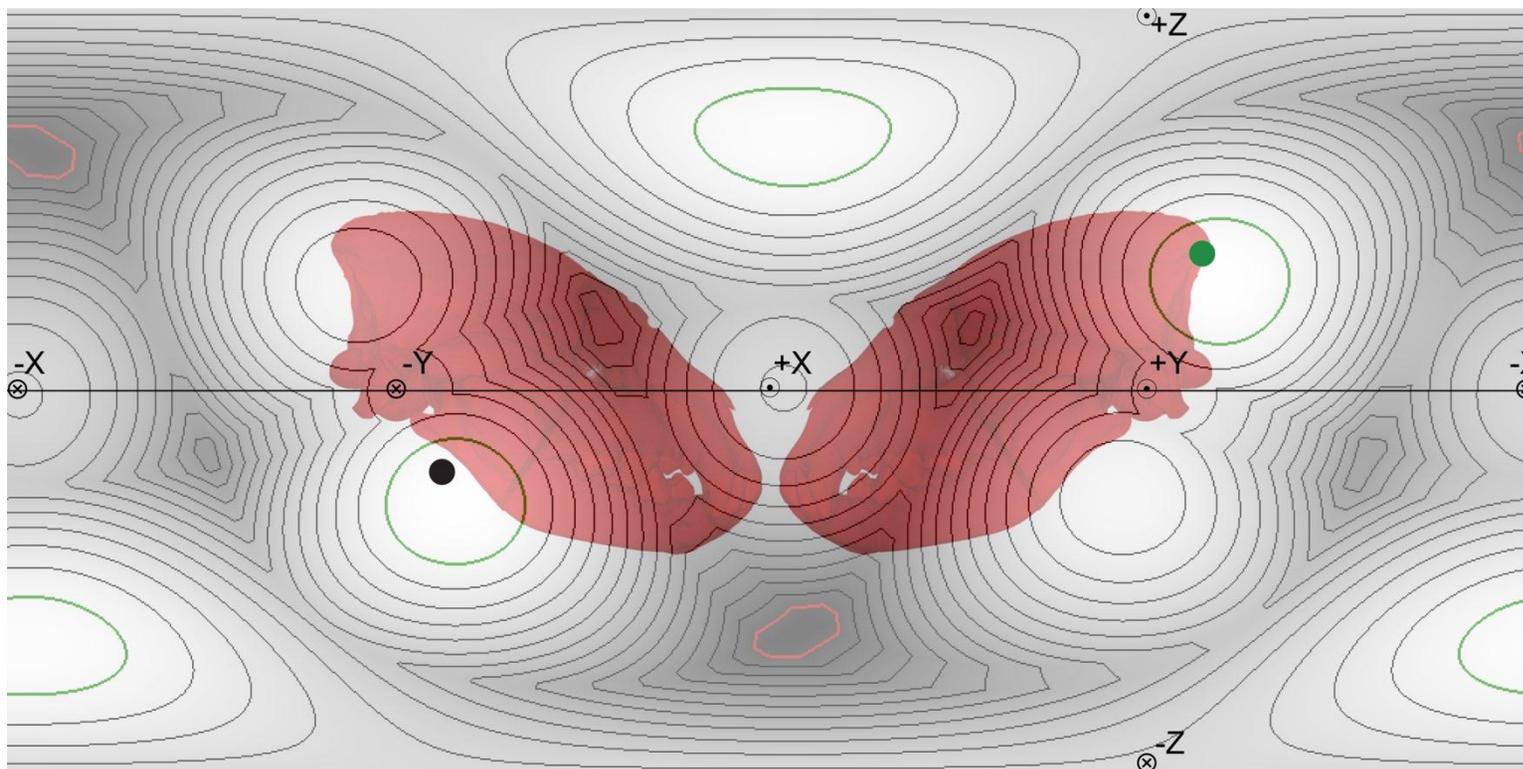
*Chironectes*



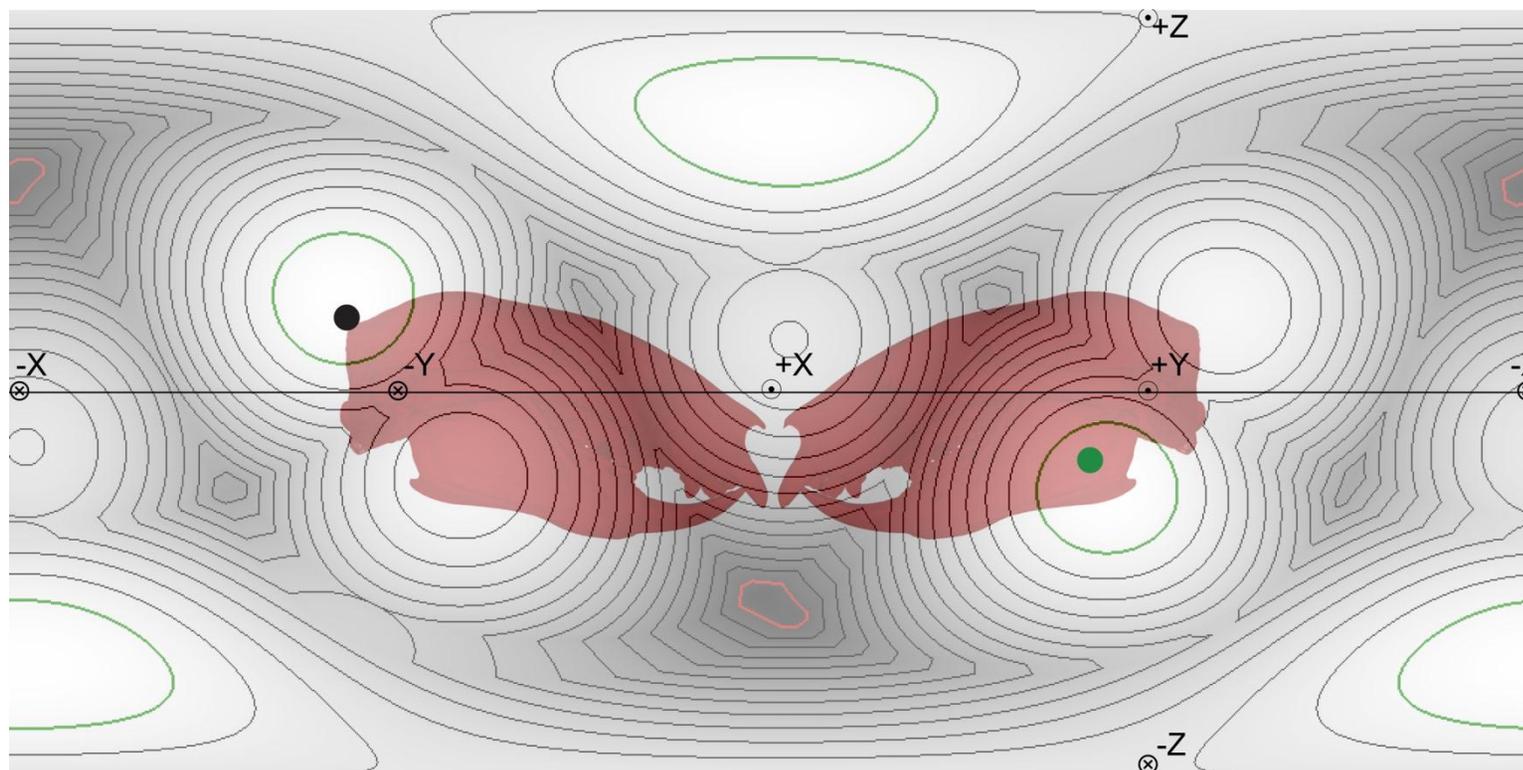
*Chrysochloris*



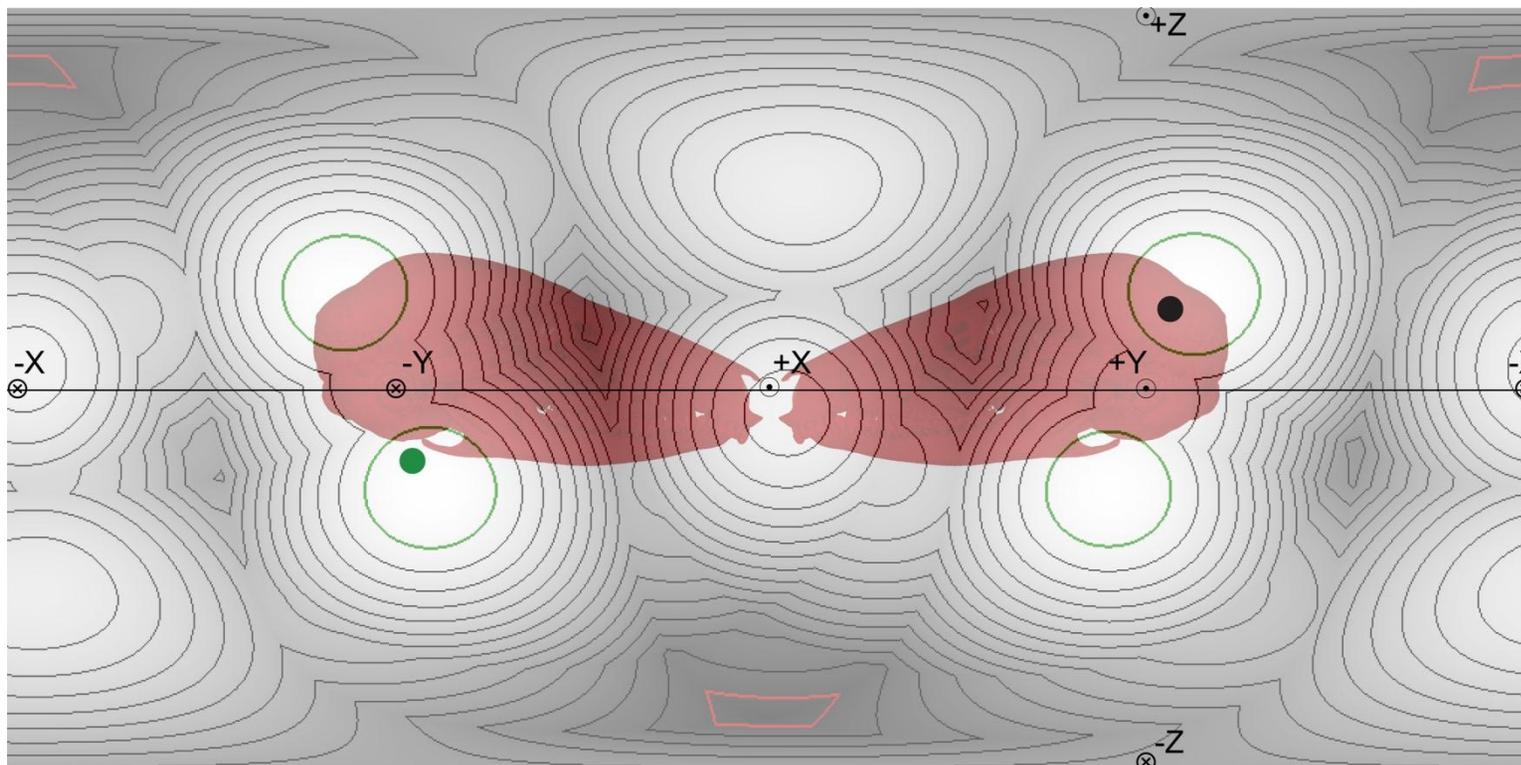
*Crocuta*



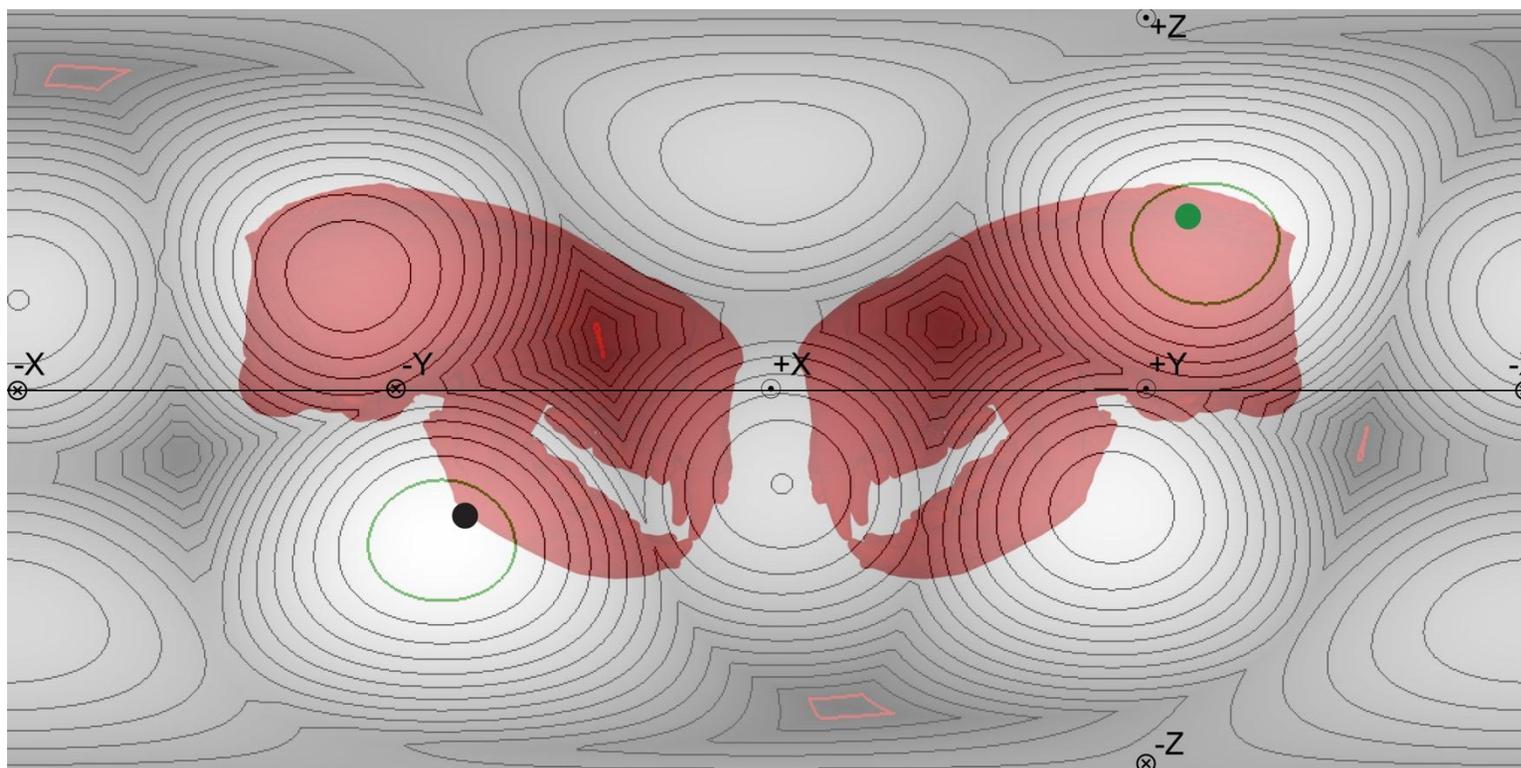
*Dactylopsila*



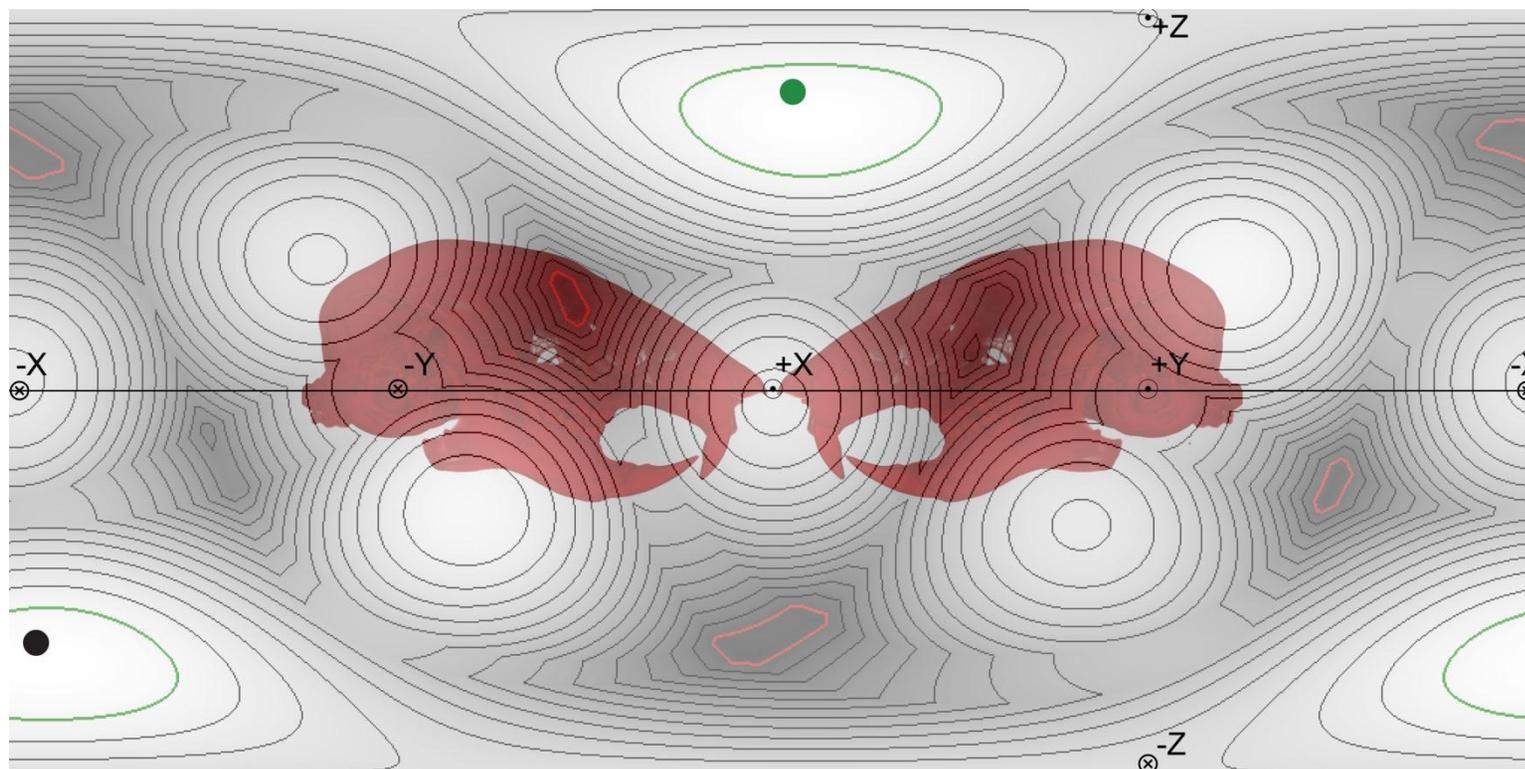
*Dromiciops*



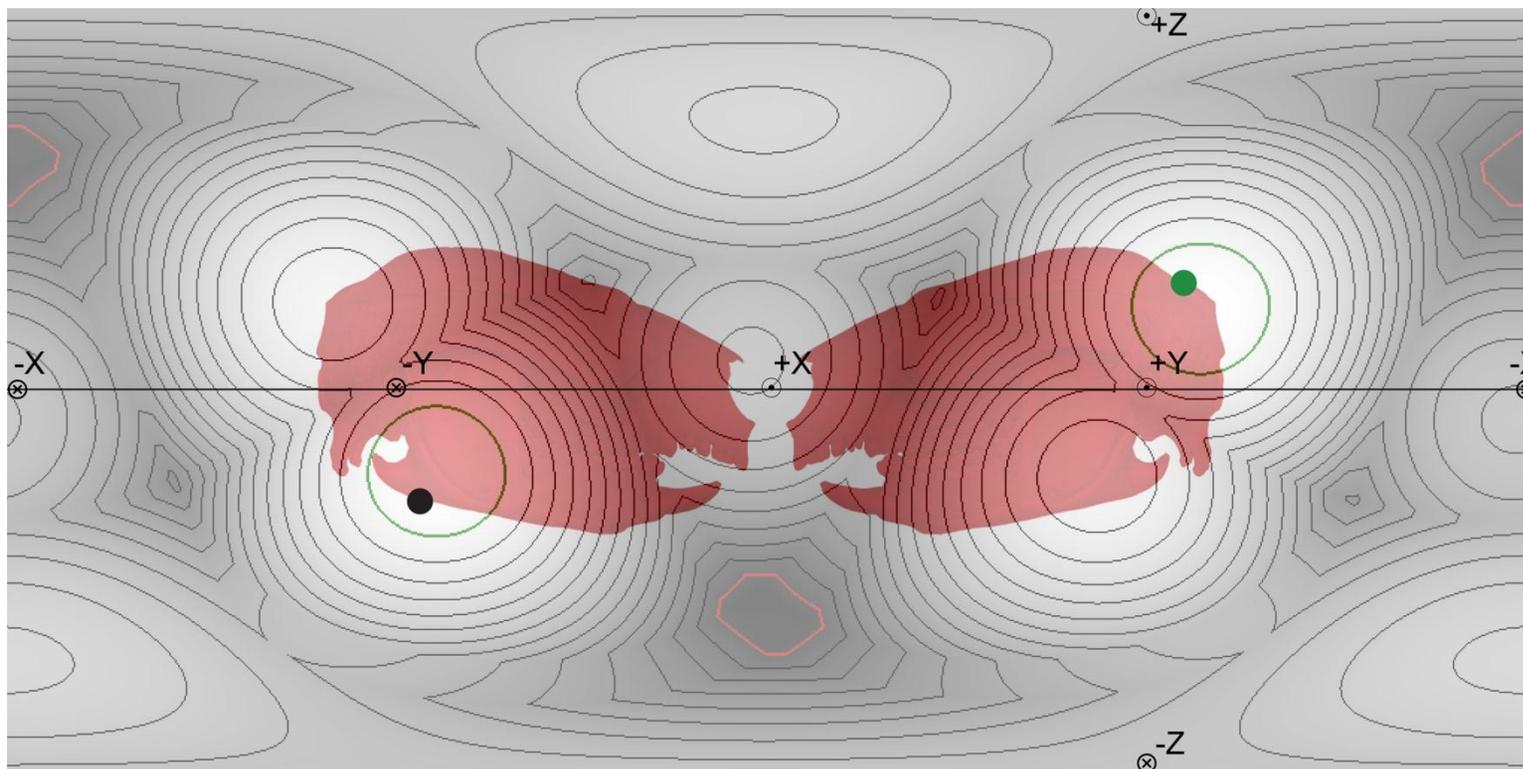
*Enhydra*



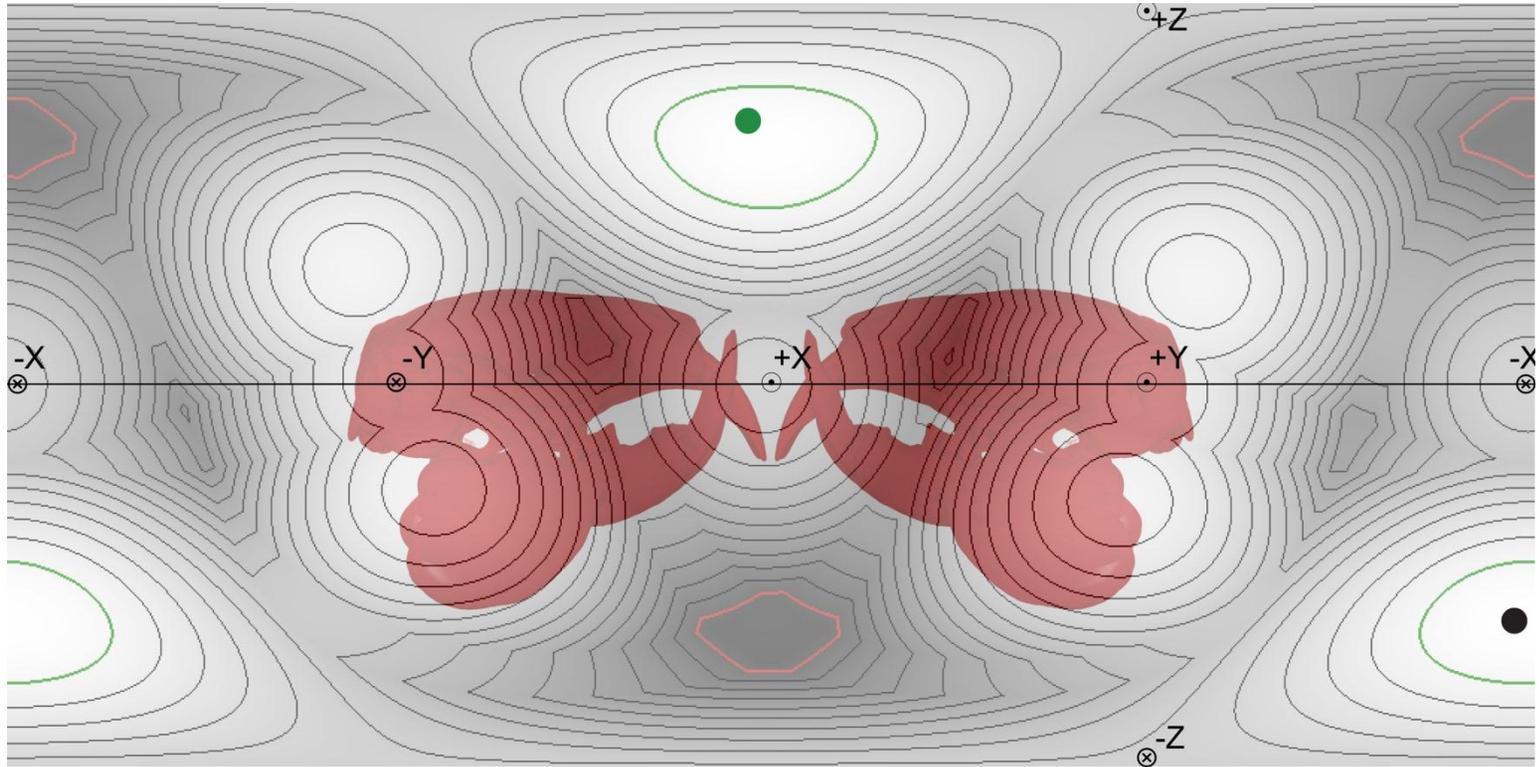
*Glaucomys*



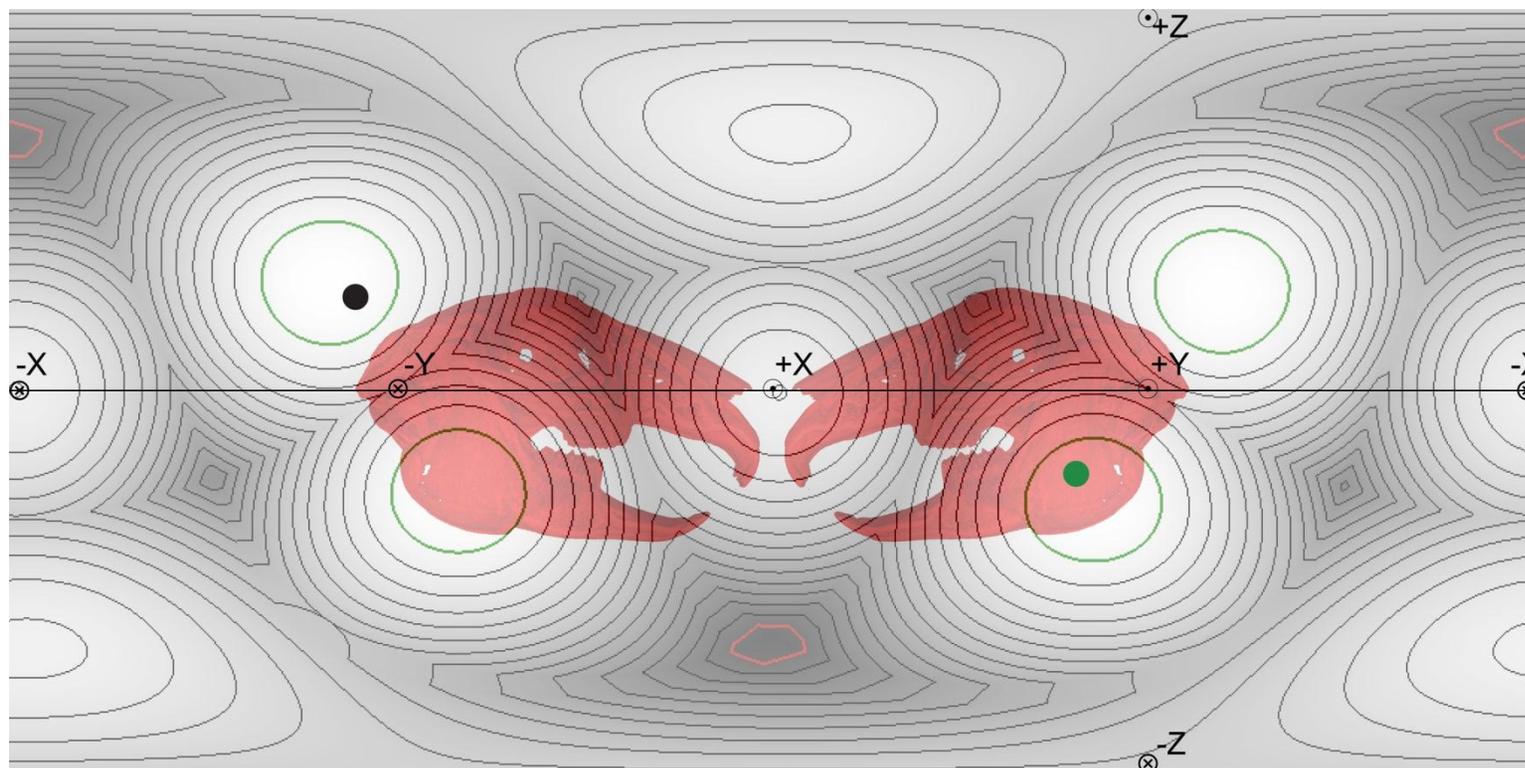
*Hemibelideus*



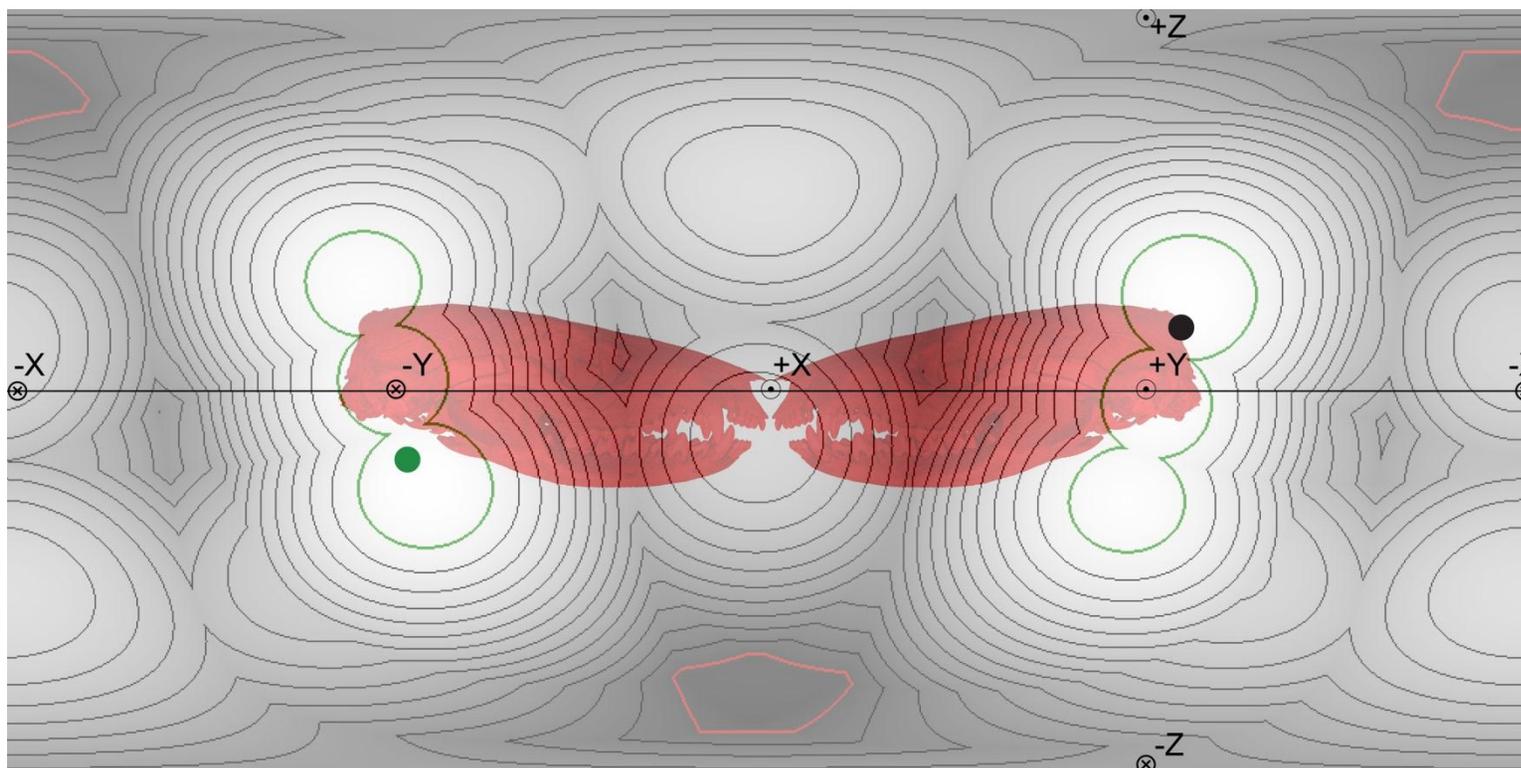
*Heterocephalus*



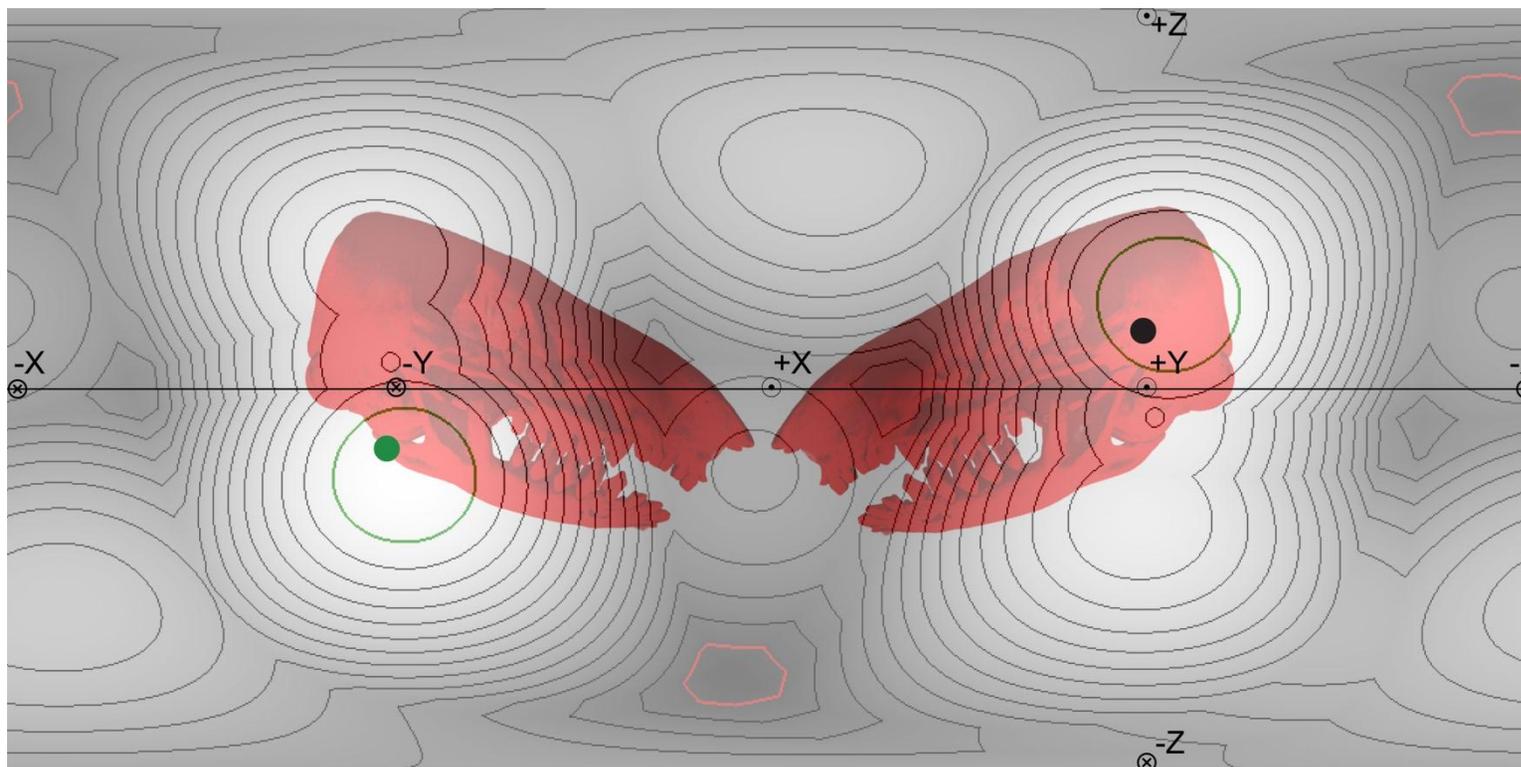
*Lepus*



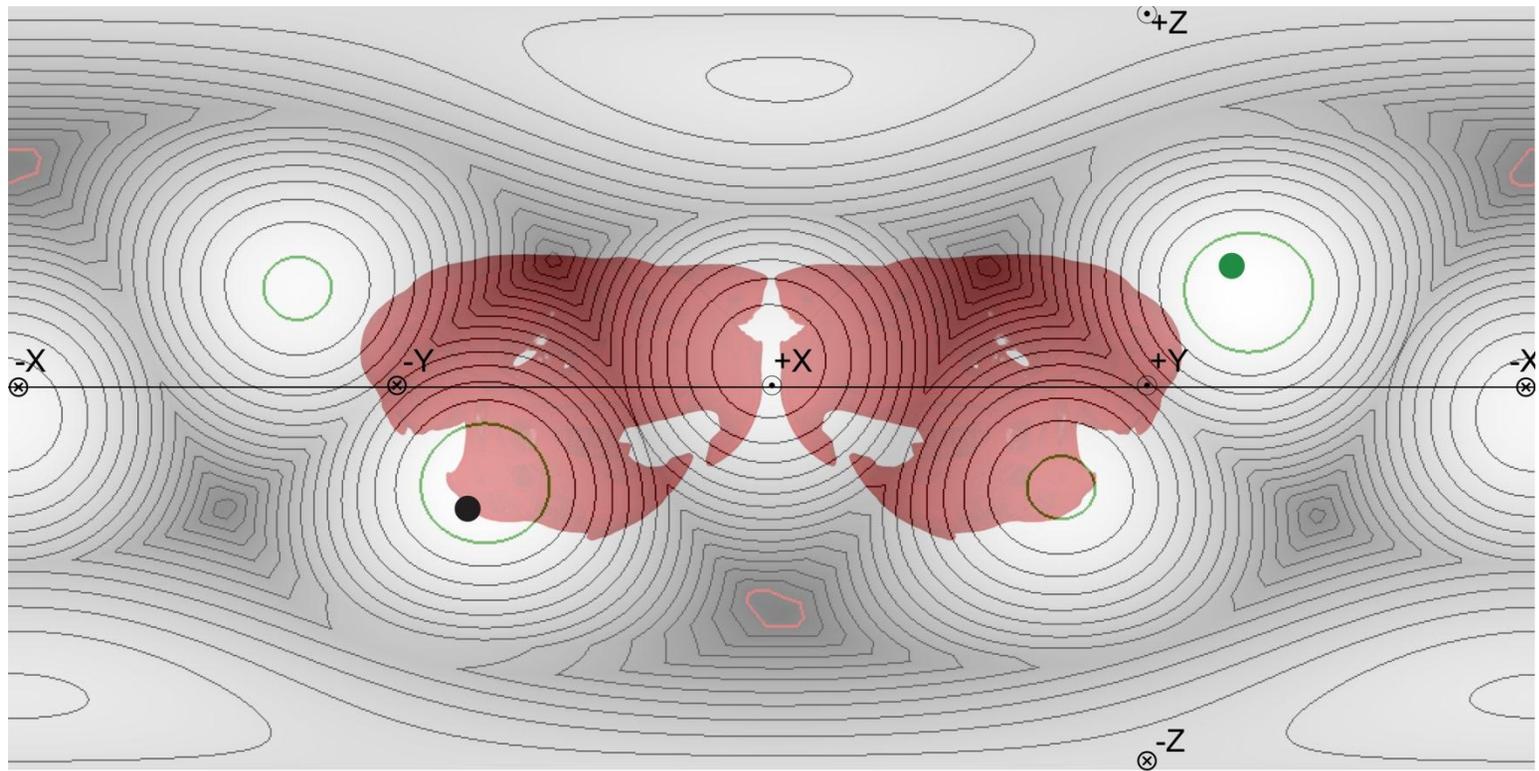
*Monodelphis*



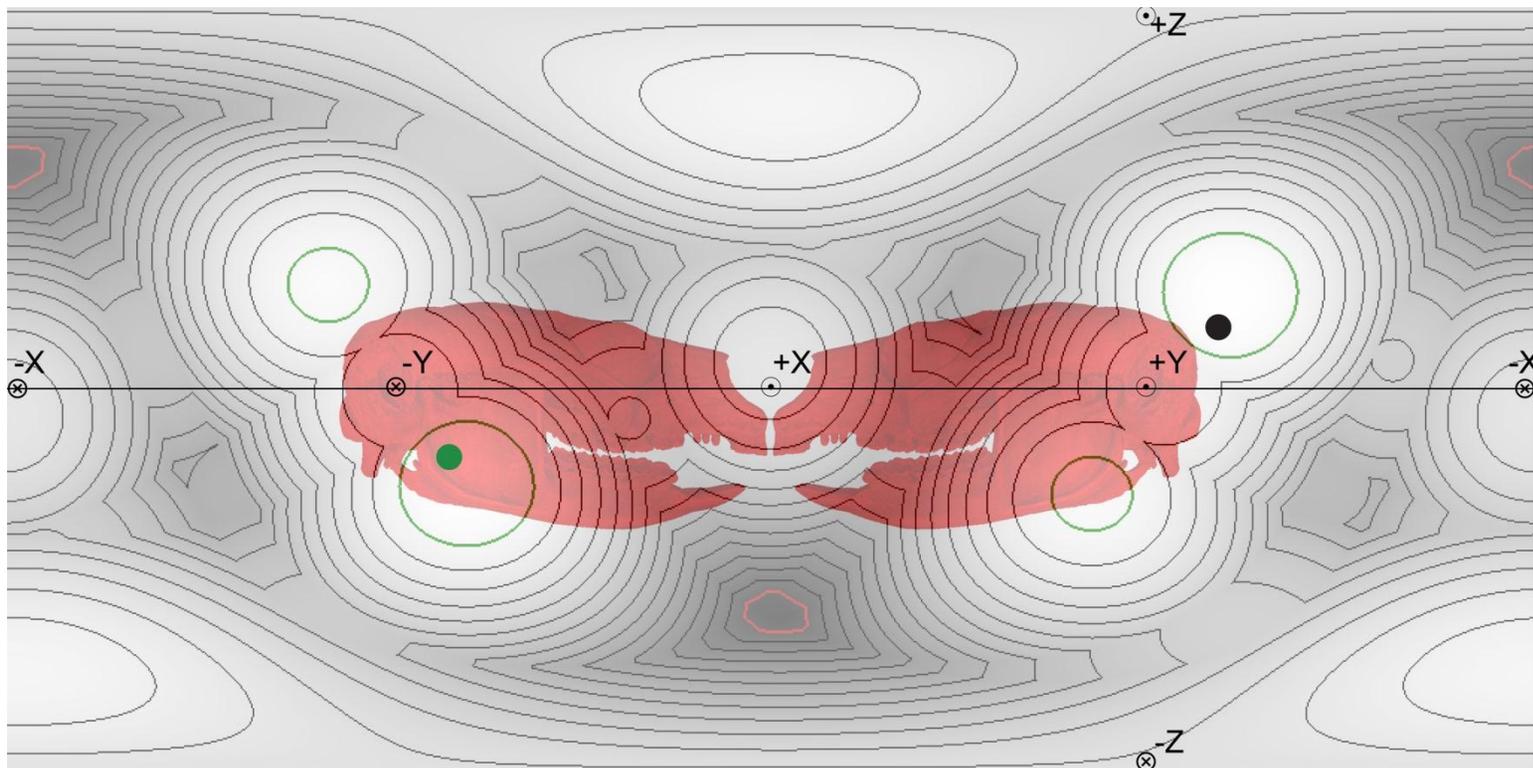
*Notoryctes*



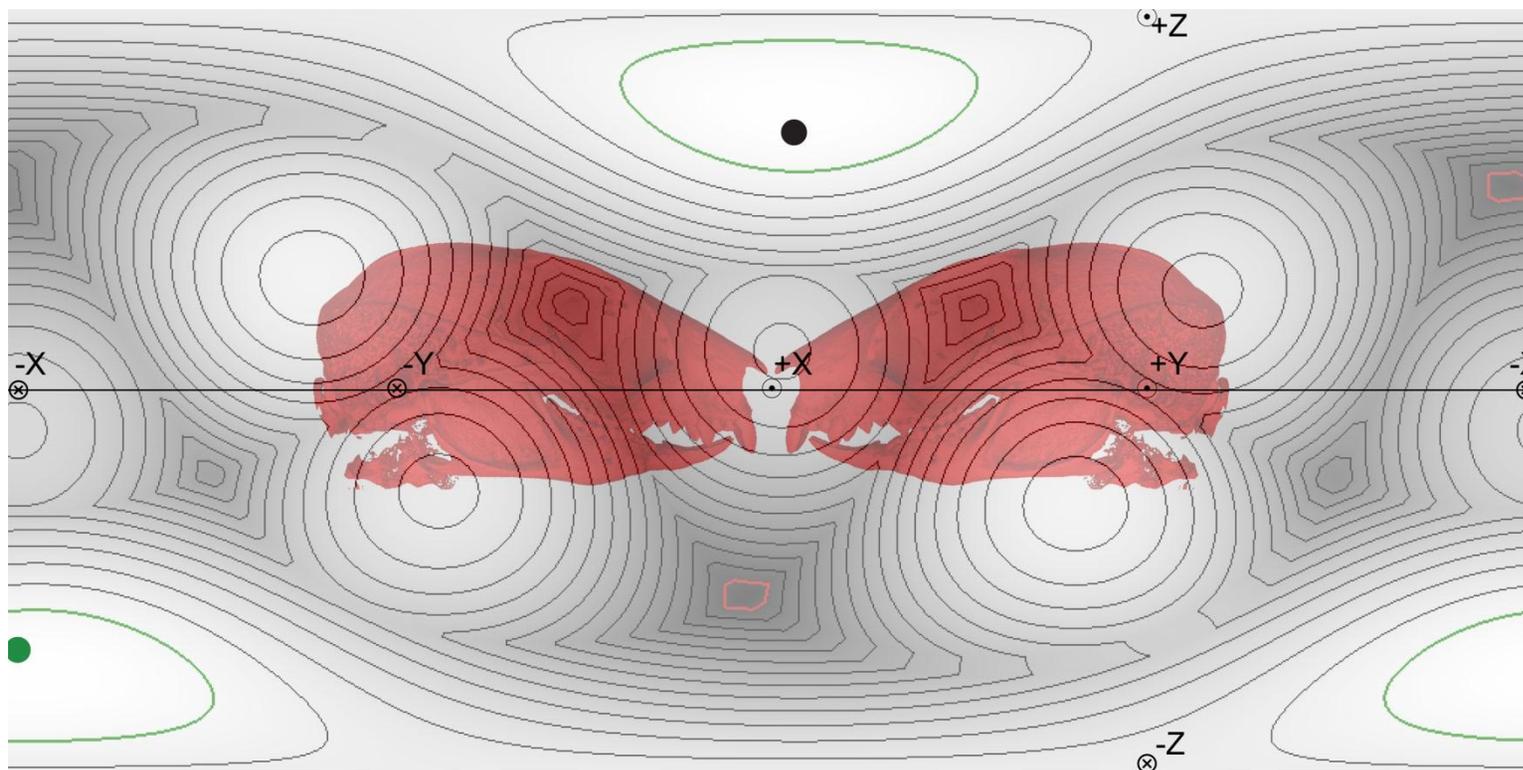
*Pedetes*



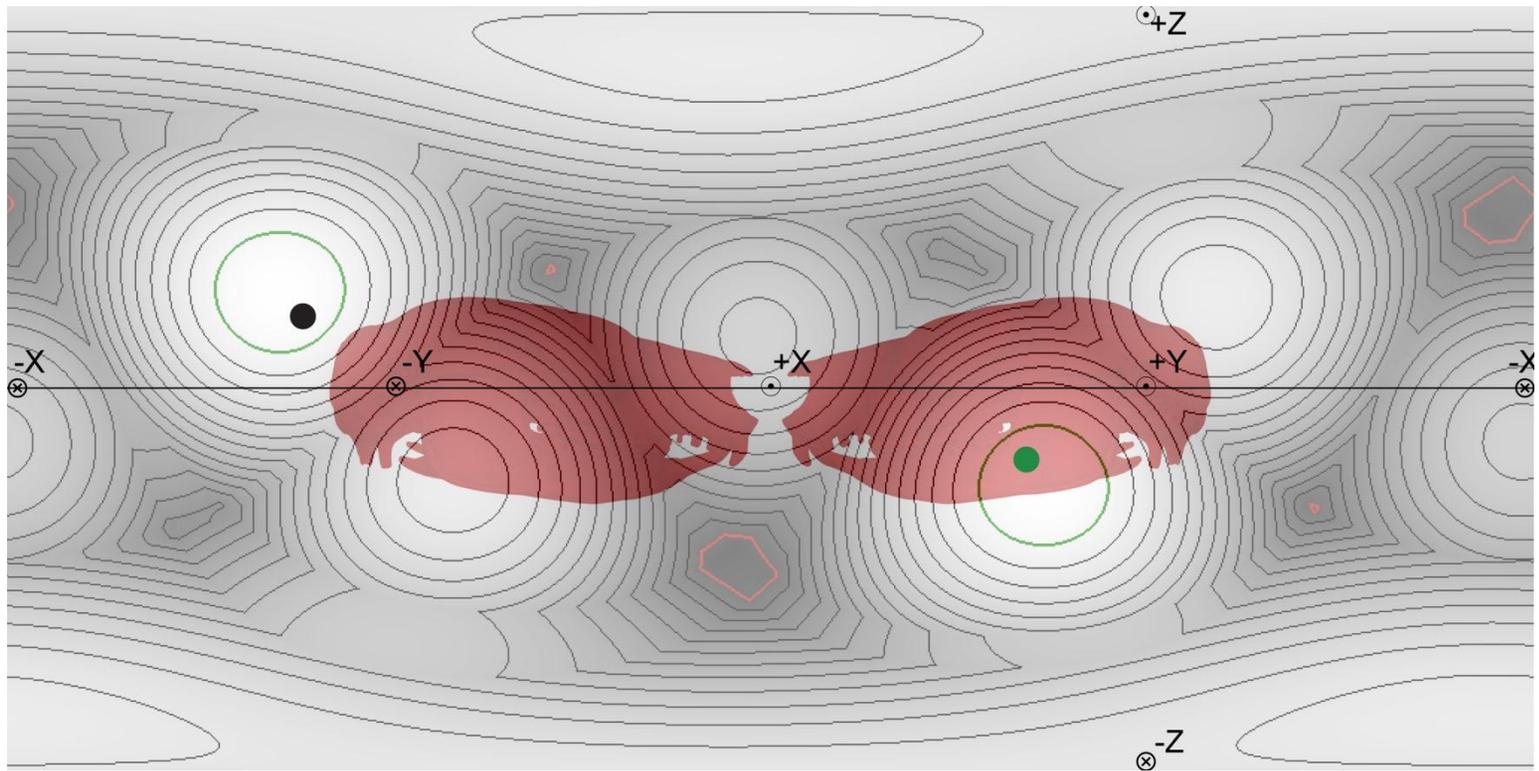
*Petauroides*



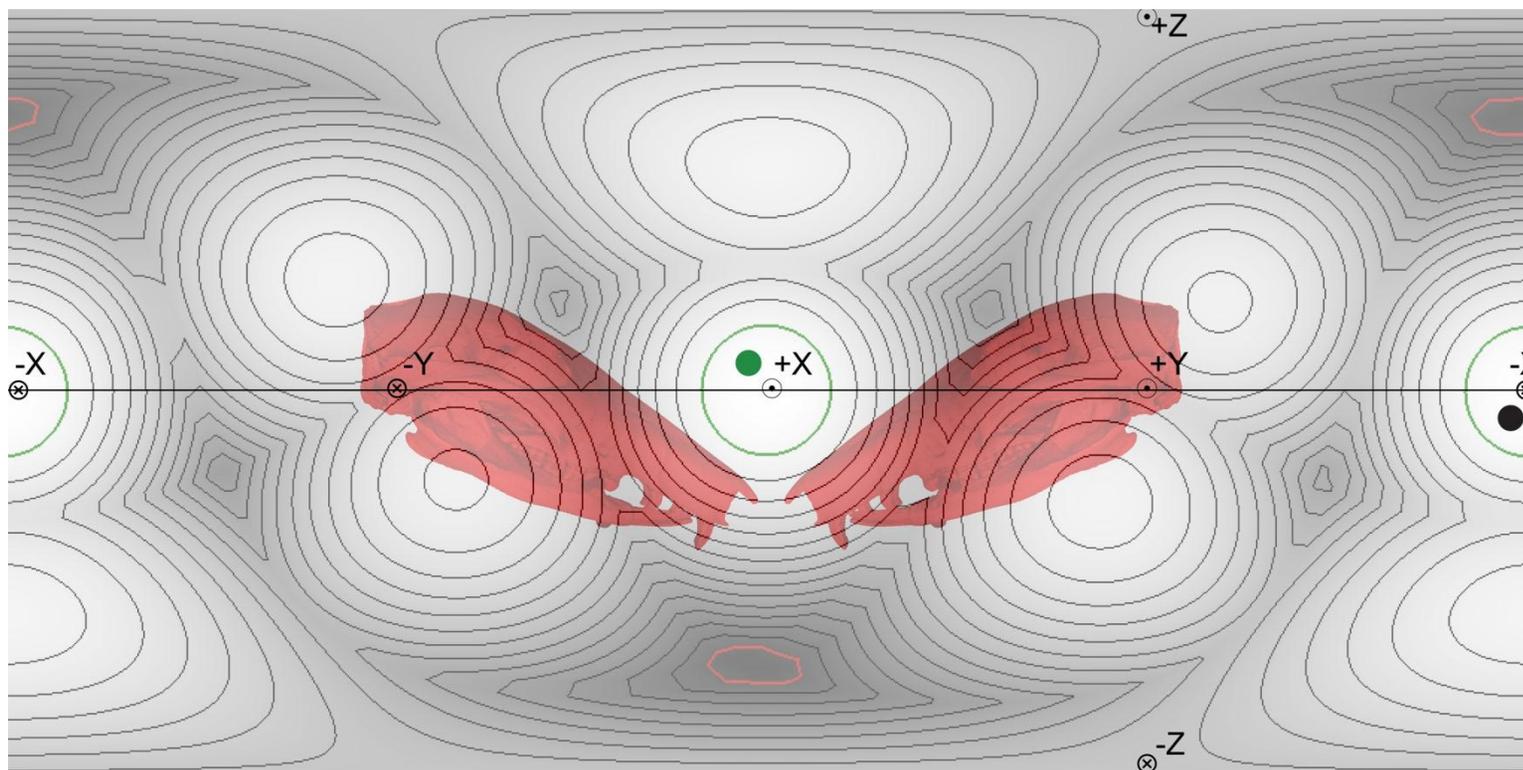
*Petaurus*



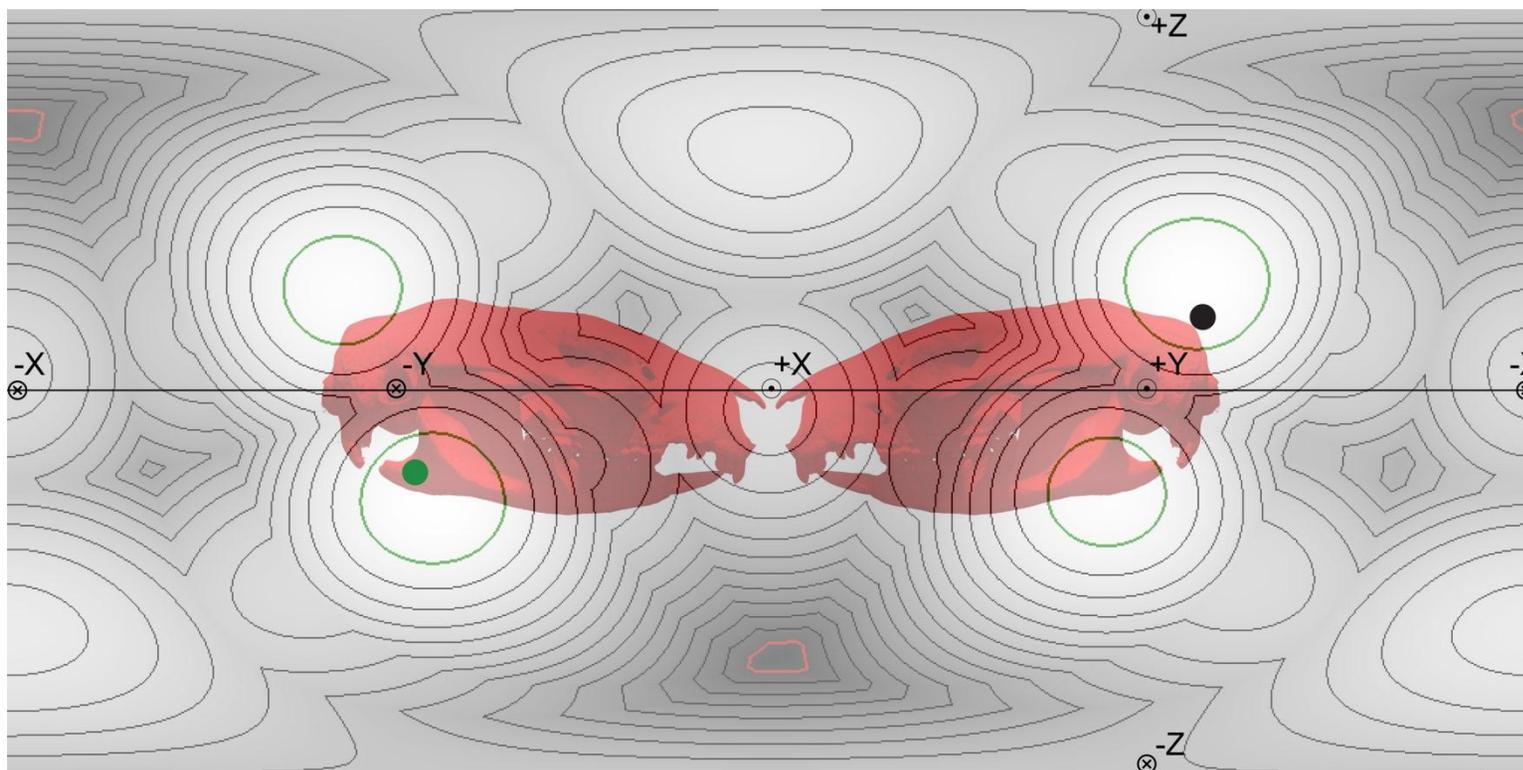
*Petropseudes*



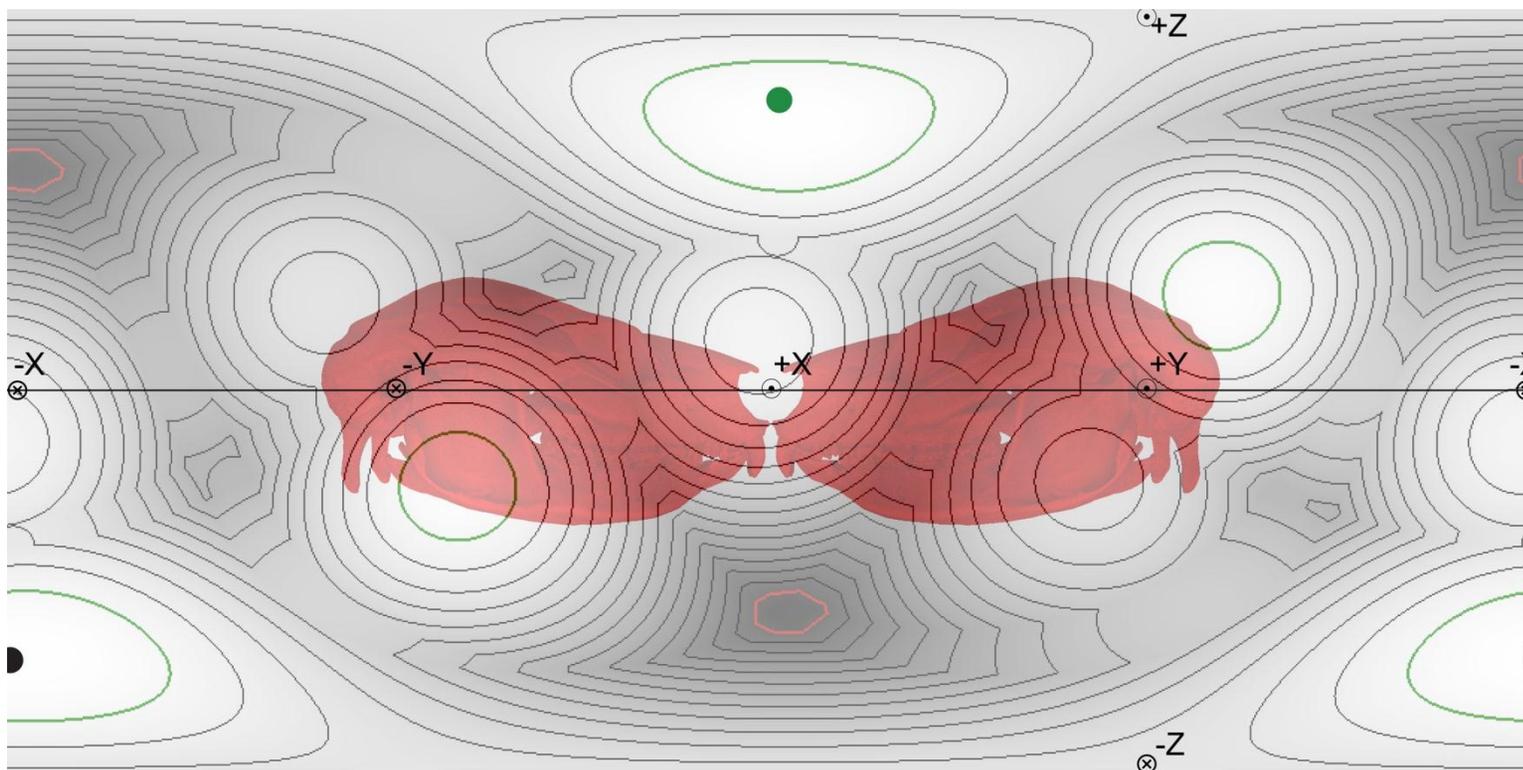
*Potorous*



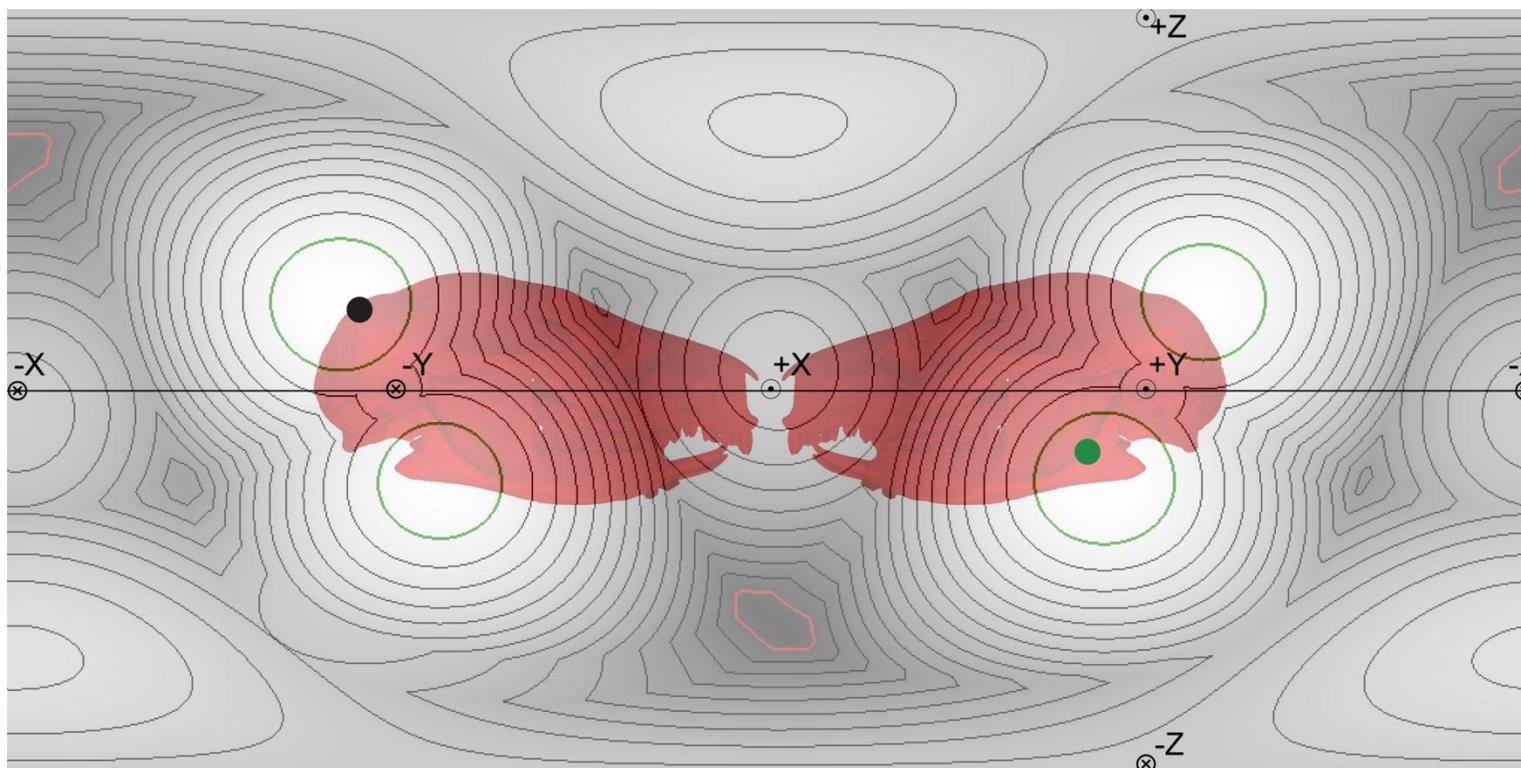
*Pseudocheirus*



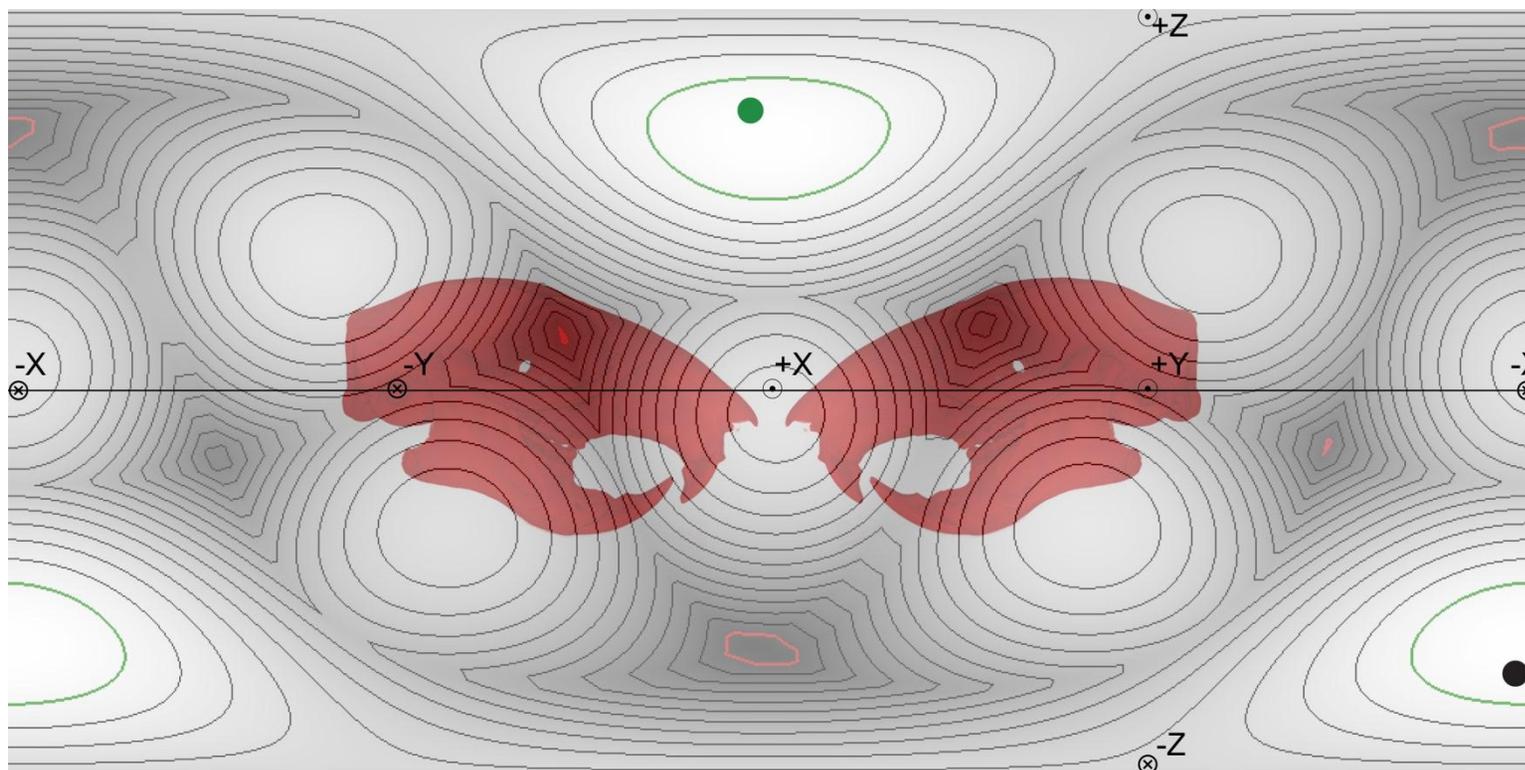
*Pseudochirops*



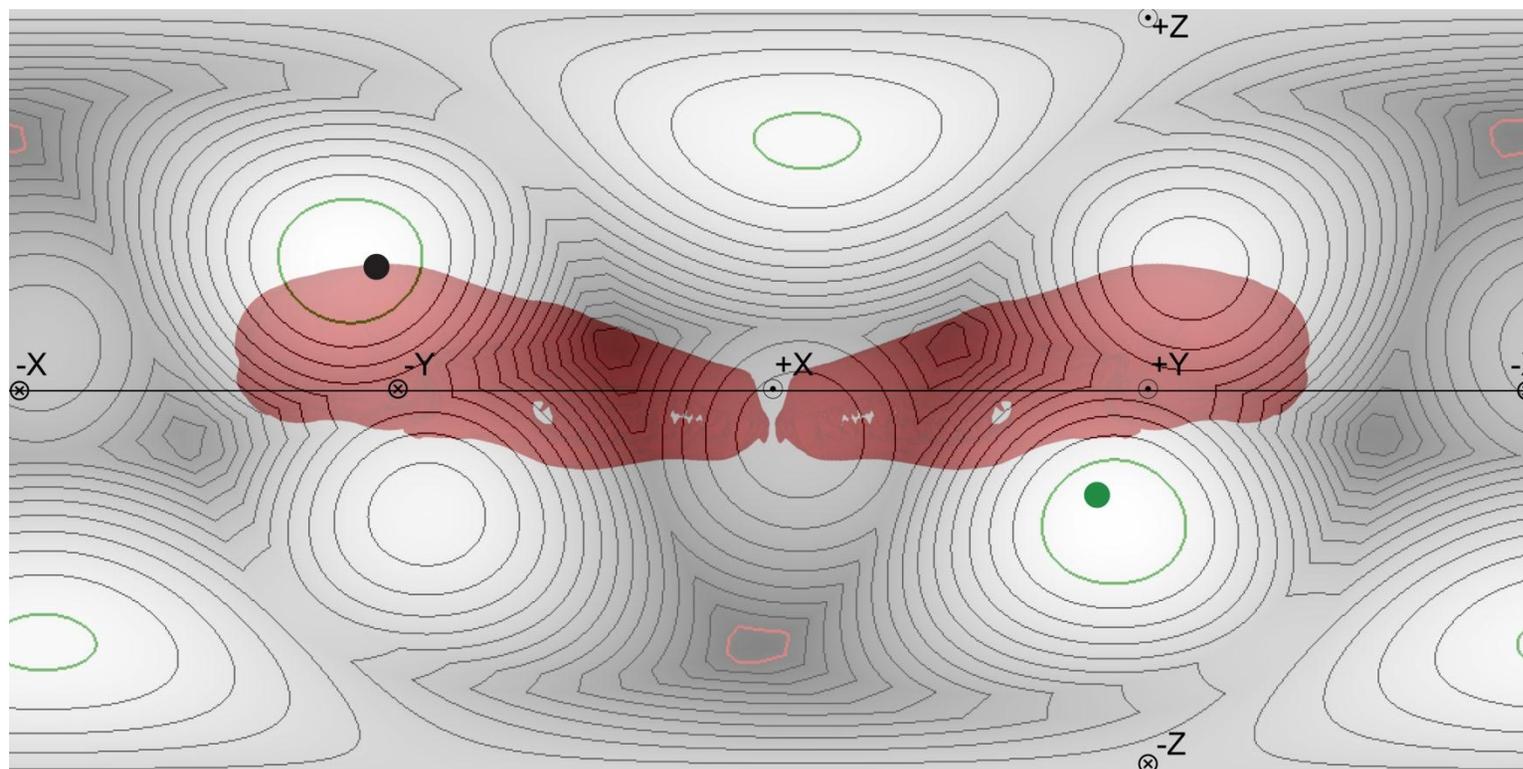
*Pseudochirulus*



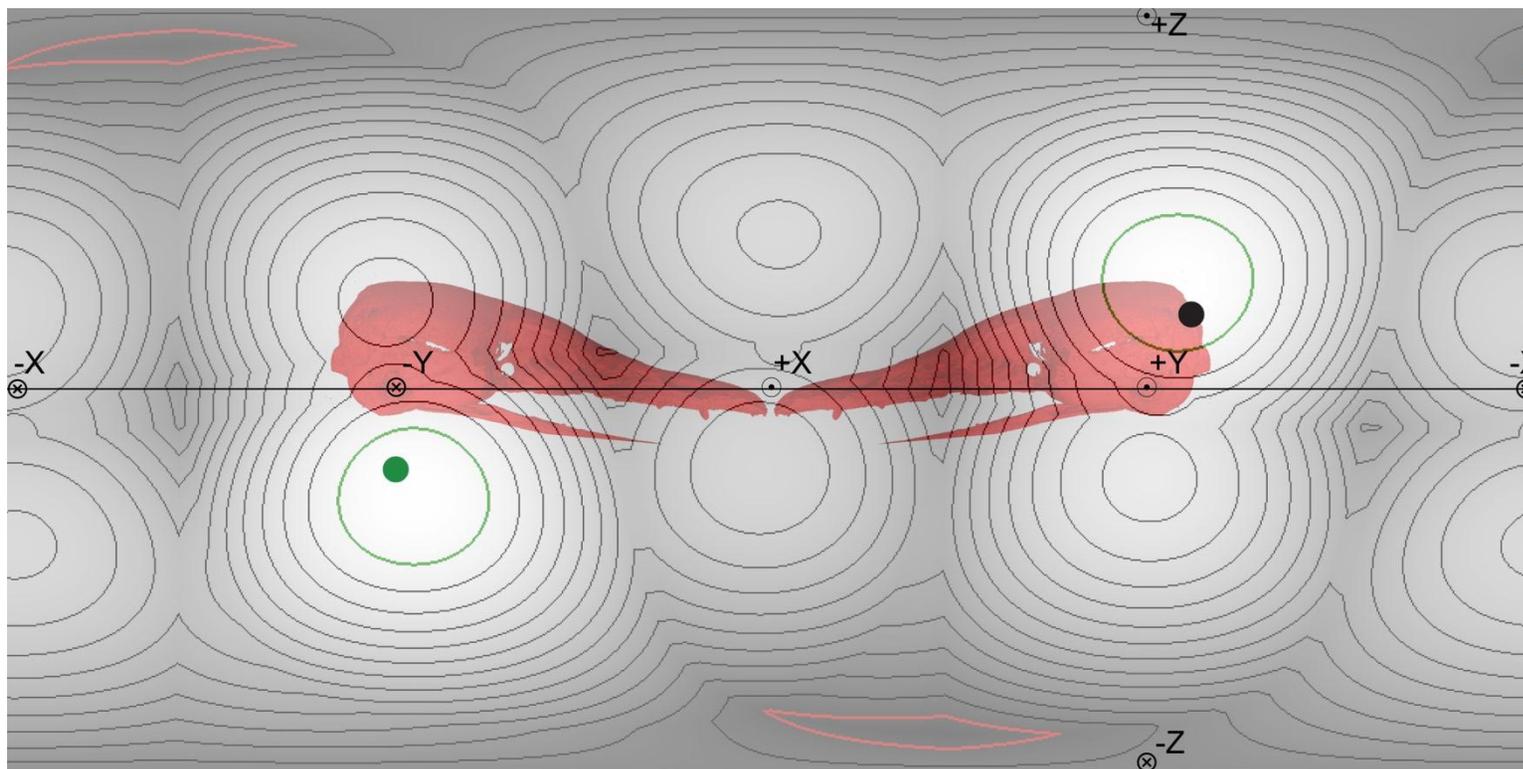
*Sciurus*



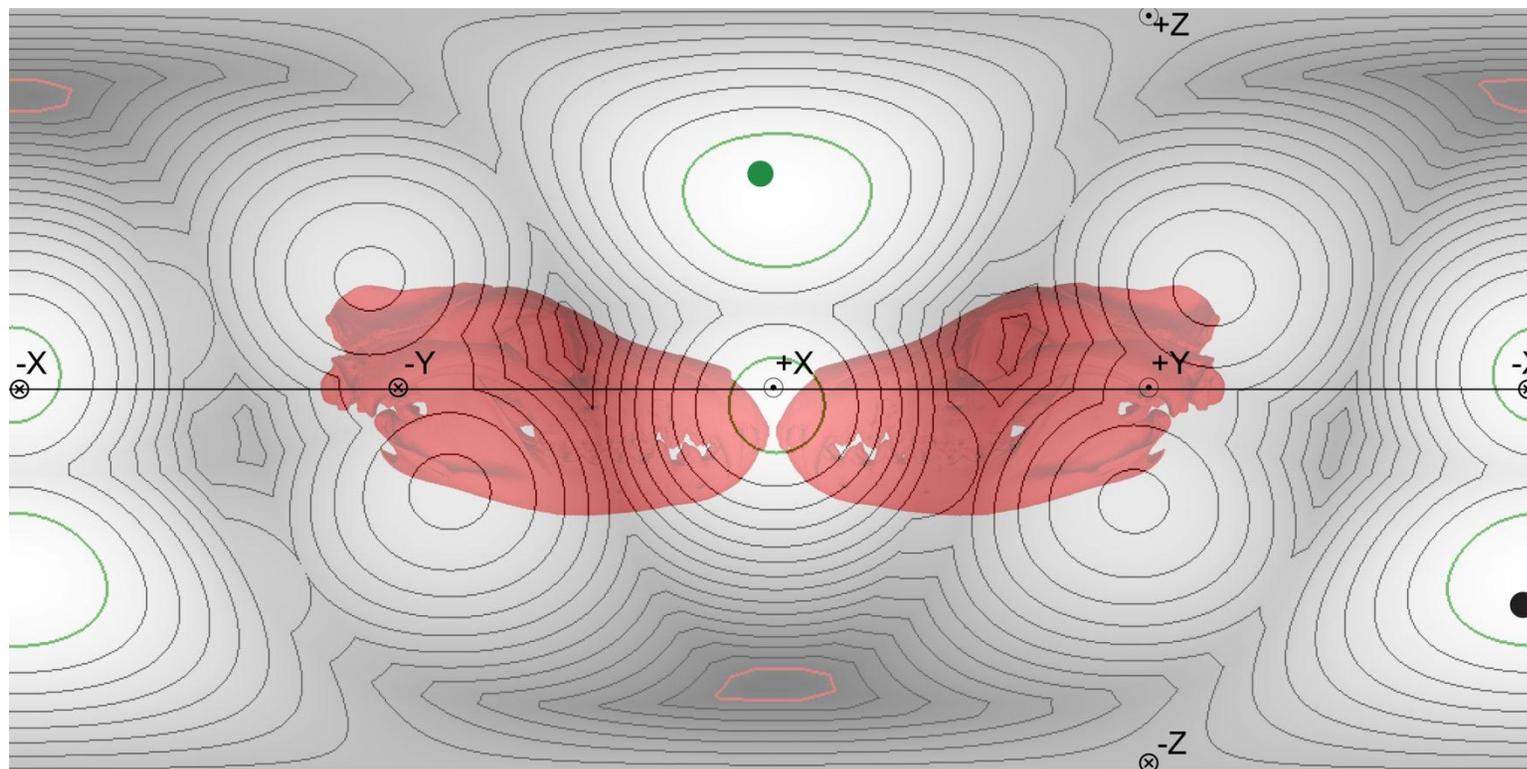
*Talpa*



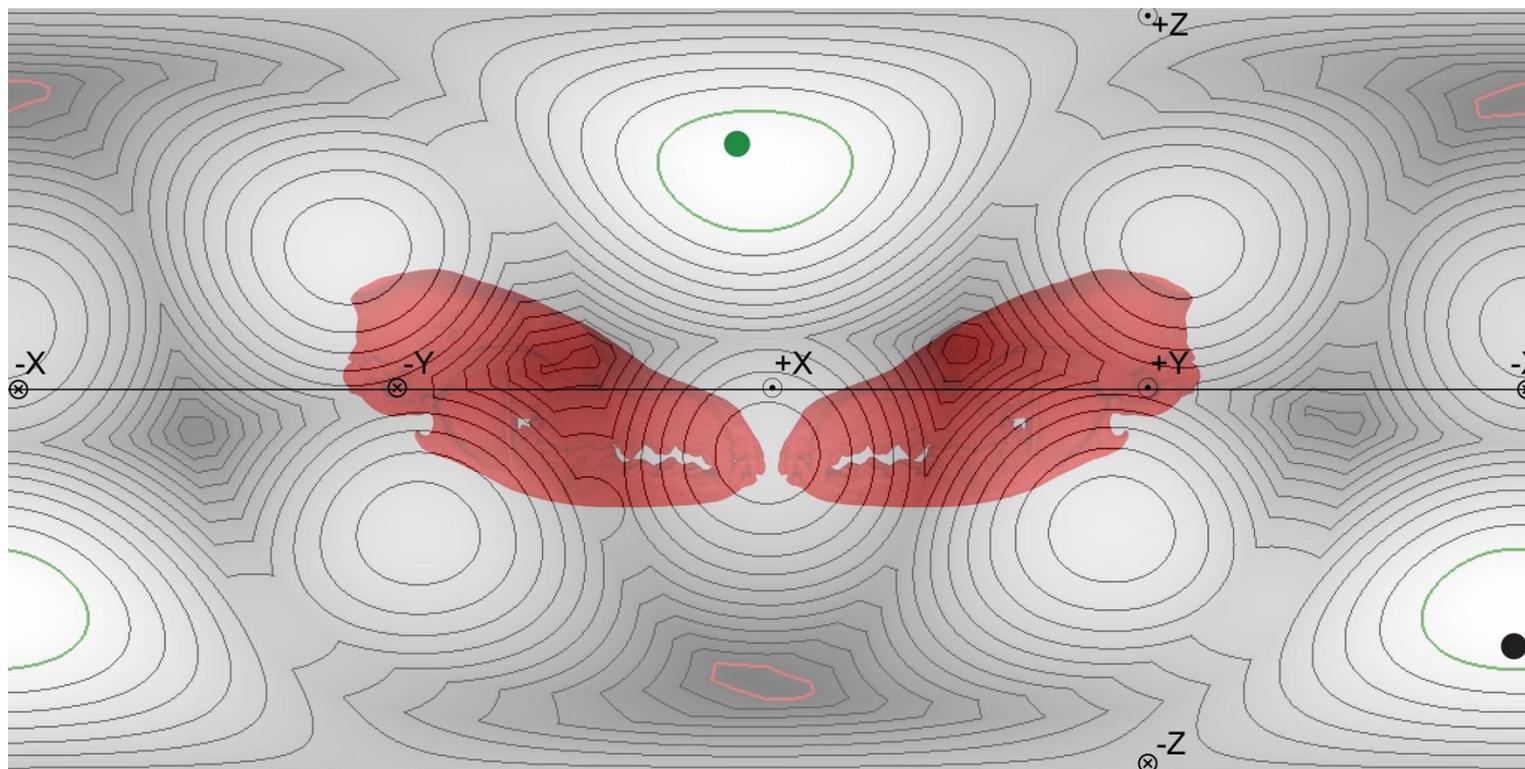
*Tarsipes*



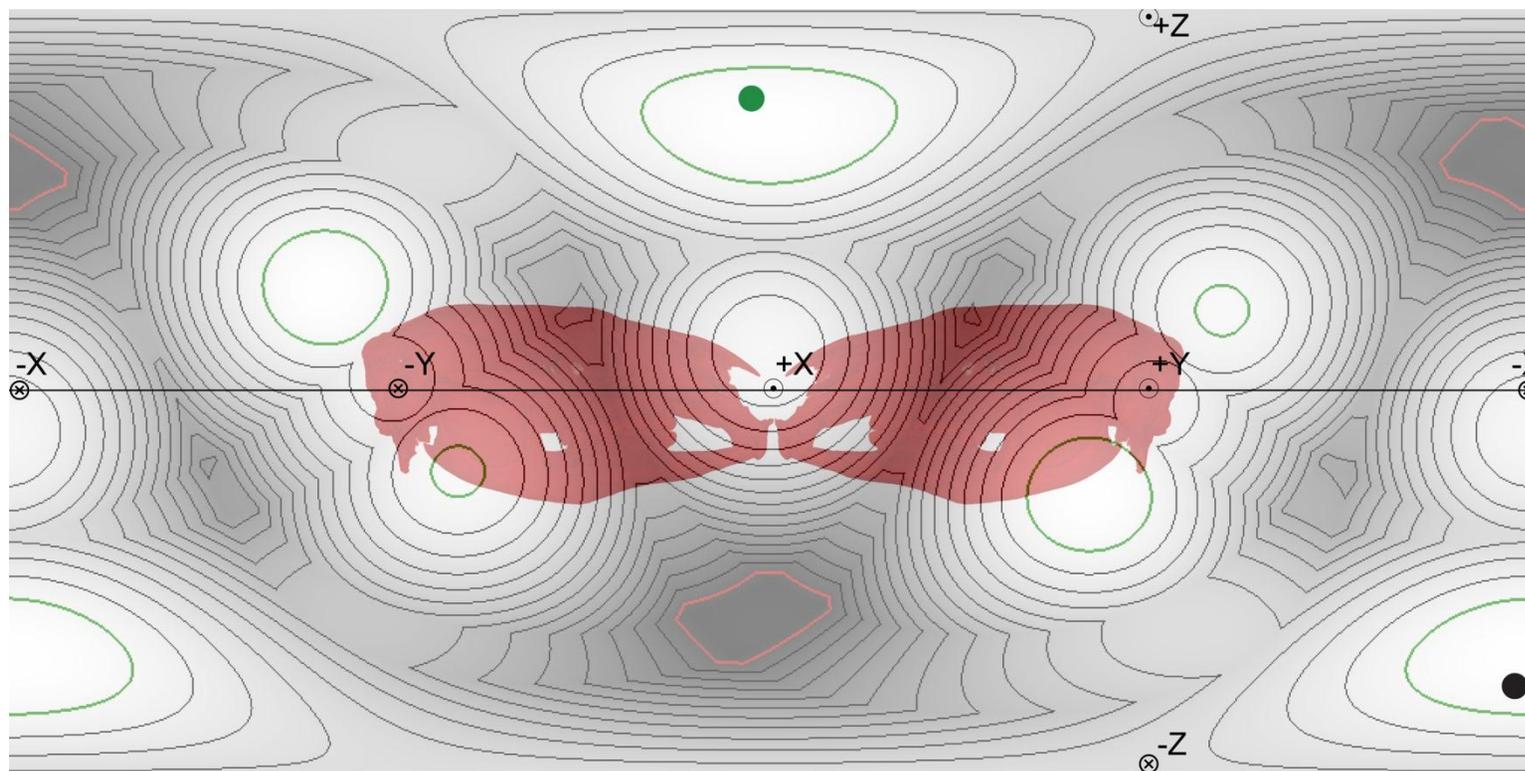
*Thylacinus*



*Vulpes*



*Wallabia*



## References

- Angelaki, D. E., and K. E. Cullen. 2008. Vestibular system: the many facets of a multimodal sense. *Annual Review of Neuroscience* 31:125-150.
- Anson, B. J., and J. A. Donaldson. 1981. *Surgical Anatomy of the Temporal Bone*, Third Edition. WB Saunders, Philadelphia, Pennsylvania. 734 pp.
- Ashwell, K. W. S. 2010. Cerebellum, vestibular and precerebellar nuclei; pp. 82-94 in K. W. S. Ashwell (ed.), *The Neurobiology of Australian Marsupials*. Cambridge University Press, Cambridge, UK.
- Bast, T. H., and B. J. Anson. 1949. *The Temporal Bone and the Ear*. Charles C. Thomas, Springfield, Illinois. 478 pp.
- Beraneck, M., and F. M. Lambert. 2009. Impaired perception of gravity leads to altered head direction signals: what can we learn from vestibular-deficient mice? *Journal of Neurophysiology* 102:12-14.
- Blanks, R. H. I., and Y. Torigoe. 1989. Orientation of the semicircular canals in the rat. *Brain Research* 487:278-287.
- Blanks, R. H. I., I. S. Curthoys, and C. H. Markham. 1972. Planar relationships of semicircular canals in the cat. *American Journal of Physiology* 223:55-62.
- Blanks, R. H. I., I. S. Curthoys, and C. H. Markham. 1975. Planar relationships of the semicircular canals in man. *Acta Otolaryngologica* 80:185-196.

- Blanks, R. H. I., I. S. Curthoys, M. L. Bennett, and C. H. Markham. 1985. Planar relationships of the semicircular canals in rhesus and squirrel monkeys. *Brain Research* 340:315-324.
- Breuer, J. 1874. Über die Function der Bogengänge des Ohrlabyrinthes. *Wien Medizinische Jahrbücher* 4:72-124.
- Breuer, J. 1875. Beiträge zur Lehre vom statischen Sinne (Gleichgewichtsorgan, Vestibularapparat des Ohrlabyrinths). Zweite Mitteilung. *Wien Medizinische Jahrbücher* 5:87-156.
- Brodal, A., and B. Høivik. 1964. Site and mode of termination of primary vestibulocerebellar fibers in the cat. An experimental study with silver impregnation methods. *Archives Italiennes de Biologie* 102:1-21.
- Brown, A. C. 1874. On the sense of rotation and the anatomy and physiology of the semicircular canals of the internal ear. *Journal of Anatomy and Physiology* 14:327-331.
- Caix, M., and G. Outrequin. 1979. Variability of the bony semicircular canals. *Anatomia Clinica* 1:259-265.
- Calabrese, D. R., and T. E. Hullar. 2006. Planar relationships of the semicircular canals in two strains of mice. *Journal of the Association for Research in Otolaryngology* 7:151-159.

- Canalis, R. F., E. Mira, L. Bonandrini, and R. Hinojosa. 2001. Antonio Scarpa and the discovery of the membranous inner ear. *Otology & Neurotology* 22:105-112.
- Cartmill, M. 1992. New views on primate origins. *Evolutionary Biology* 1:105-111.
- Clark, C. T., and K. K. Smith. 1993. Cranial osteogenesis in *Monodelphis domestica* (Didelphidae) and *Macropus eugenii* (Macropodidae). *Journal of Morphology* 215:119-149.
- Clark, G. 1939. The use of the Horsley-Clarke instrument on the rat. *Science* 90:92.
- Cox, P. G., and N. Jeffery. 2008. Geometry of the semicircular canals and extraocular muscles in rodents, lagomorphs, felids, and modern humans. *Journal of Anatomy* 213:583-596.
- Cox, P. G., and N. Jeffery. 2010. Semicircular canals and agility: the influence of size and shape measures. *Journal of Anatomy* 216:37-47.
- Curthoys, E. J., R. H. I. Blanks, and C. H. Markham. 1977. Semicircular canal radii of curvature (R) in cat, guinea pig, and man. *Journal of Morphology* 151:1-16.
- Curthoys, I. S., E. J. Curthoys, R. H. I. Blanks, and C. H. Markham. 1975. The orientation of the semicircular canals in the guinea pig. *Acta Otolaryngologica* 80:197-205.
- David, R., J. Droulez, R. Allain, A. Berthoz, P. Janvier, and D. Bennequin. 2010. Motion from the past. A new method to infer vestibular capacities of extinct species. *Comptes Rendus Palevol* 9:397-410.

- Day, B. L., and R. C. Fitzpatrick. 2005. Virtual head rotation reveals a process of route reconstruction from human vestibular signals. *Journal of Physiology* 567:591-597.
- De Miguel, C., and M. Henneberg. 1997. Encephalization of the koala, *Phascolarctos cinereus*. *Australian Mammalogy* 20:315-320.
- Della Santina, C. C. 2010. Regaining balance with bionic ears. *Scientific American* 302:68-71.
- Della Santina, C. C., V. Potyagaylo, A. A. Migliaccio, L. B. Minor, and J. P. Carey. 2005. Orientation of human semicircular canals measured by three-dimensional multiplanar CT reconstruction. *Journal of the Association for Research in Otolaryngology* 6:191-206.
- Ekdale, E. G. 2005. Ontogeny of the inner ear of mammals, implications for the phylogenetic assessment of fossils. *Journal of Vertebrate Paleontology* 25 (Supplement to 3):53A.
- Ekdale, E. G. 2009. Variation within the bony labyrinth of mammals. PhD Dissertation, Jackson School of Geosciences, The University of Texas, Austin. 439 pp.
- Ekdale, E. G. 2010. Ontogenetic variation in the bony labyrinth of *Monodelphis domestica* (Mammalia: Marsupialia) following ossification of the inner ear cavities. *Anatomical Record Part A* 293:1896-1912.

- Ezure, K., and W. Graf. 1984. A quantitative analysis of the spatial organization of the vestibulo-ocular reflexes in lateral- and frontal-eyed animals -- I. Orientation of semicircular canals and extraocular muscles. *Neuroscience* 12:85-93.
- Filan, S. L. 1991. Development of the middle ear region in *Monodelphis domestica* (Marsupialia, Didelphidae): marsupial solutions to an early birth. *Journal of Zoology, London* 225:577-588.
- Fitzpatrick, R. C., J. E. Butler, and B. L. Day. 2006. Resolving head rotation for human bipedalism. *Current Biology* 16:1509-1514.
- Gannon, P. J., A. R. Eden, and J. T. Laitman. 1988. The subarcuate fossa and cerebellum of extant primates: comparative study of a skull-brain interface. *American Journal of Physical Anthropology* 77:143-64.
- Glickstein, M., N. Gerrits, I. Kralj-Hans, B. Mercier, J. Stein, and J. Voogd. 1994. Visual pontocerebellar projections in the macaque. *Journal of Comparative Neurology* 349:51-72.
- Goldberg, J. M., and C. Fernández. 1971. Physiology of peripheral neurons innervating semicircular canals of the squirrel monkey. I. Resting discharge and response to constant angular accelerations. *Journal of Neurophysiology* 38:635-660.
- Goldberg, J. M., C. E. Smith, and C. Fernández. 1984. Relation between discharge regularity and responses to externally applied galvanic currents in vestibular nerve afferents of the squirrel monkey. *Journal of Neurophysiology* 51:1236-1256.

- Gray, A. A. 1907. *The Labyrinth of Animals*. J & A Churchill, London, UK. 198 pp.
- Gray, A. A. 1908. *The Labyrinth of Animals*. J & A Churchill, London, UK. 252 pp.
- Gray, H. 1858. *Anatomy Descriptive and Surgical*. John W. Parker and Son, London, UK. 750 pp.
- Haight, J. R., and J. E. Nelson. 1987. A brain that doesn't fit its skull: a comparative study of the brain and endocranium of the koala, *Phascolarctos cinereus* (Marsupialia: Phascolarctidae); pp. 331-352 in M. Archer (ed.), *Possums and Opossums: Studies in Evolution*. Surrey Beatty & Sons and the Royal Zoological Society of New South Wales, Sydney, Australia.
- Haque, A., D. E. Angelaki, and J. D. Dickman. 2004. Spatial tuning and dynamics of vestibular semicircular canal afferents in rhesus monkeys. *Experimental Brain Research* 155:81-90.
- Hashimoto, S., H. Naganuma, K. Tokumasu, A. Itoh, and M. Okamoto. 2005. Three-dimensional reconstruction of the human semicircular canals and measurement of each membranous canal plane defined by Reid's stereotactic coordinates. *Annals of Otology, Rhinology & Laryngology* 114:934-938.
- Hawkins, J. E., and J. Schacht. 2005. The emergence of vestibular science. *Audiology & Neurotology* 10:185-190.

- Henkel, C. K., and G. F. Martin. 1977. The vestibular complex of the American opossum, *Didelphis virginiana* I. Conformation, cytoarchitecture and primary vestibular input. *Journal of Comparative Neurology* 172:299-320.
- Highstein, S. M. 2004. Anatomy and physiology of the central and peripheral vestibular system: overview; pp. 1-10 in S. M. Highstein, R. R. Fay, and A. N. Popper (eds.), *The Vestibular System*. Springer, New York, New York.
- Hiramatsu, T., M. Ohki, H. Kitazawa, G. Xiong, T. Kitamura, J. Yamada, and S. Nagao. 2008. Role of primate cerebellar lobulus petrosus of paraflocculus in smooth pursuit eye movement control revealed by chemical lesion. *Neuroscience Research* 60:250-258.
- Horsley, V., and R. H. Clarke. 1908. The structure and functions of the cerebellum examined by a new method. *Brain* 31:45-124.
- Hoyte, D. A. N. 1961. The postnatal growth of the ear capsule in the rabbit. *American Journal of Anatomy* 108:1-16.
- Hullar, T. E. 2006. Semicircular canal geometry, afferent sensitivity, and animal behavior. *Anatomical Record Part A* 288A:466-472.
- Hullar, T. E., and C. D. Williams. 2006. Geometry of the semicircular canals of the chinchilla (*Chinchilla laniger*). *Hearing Research* 213:17-24.
- Hullar, T. E., and N. C. Page. 2010. Vestibular physiology and disorders of the labyrinth, sixth edition; pp. 113-133 in A. J. Gulya, L. B. Minor, and D. S. Poe (eds.),

- Glasscock-Shambaugh Surgery of the Ear. People's Medical Publishing House-USA, Shelton, Connecticut.
- Ifediba, M. A., S. M. Rajguru, T. E. Hullar, and R. D. Rabbitt. 2007. The role of 3-canal biomechanics in angular motion transduction by the human vestibular labyrinth. *Annals of Biomedical Engineering* 35:1247-1263.
- Jeffery, N., and F. Spoor. 2006. The primate subarcuate fossa and its relationship to the semicircular canals part I: prenatal growth. *Journal of Human Evolution* 51:531-549.
- Jeffery, N., and P. G. Cox. 2010. Do agility and skull architecture influence the geometry of the mammalian vestibulo-ocular reflex? *Journal of Anatomy* 216:496-509.
- Jeffery, N., T. Ryan, and F. Spoor. 2008. The primate subarcuate fossa and its relationship to the semicircular canals part II: adult interspecific variation. *Journal of Human Evolution* 55:326-339.
- Jerison, H. J. 1973. *Evolution of the Brain and Intelligence*. Academic Press, New York, New York. 482 pp.
- Jones, G. M., and K. E. Spells. 1963. A theoretical and comparative study of the functional dependence of the semicircular canal upon its physical dimensions. *Proceedings of the Royal Society* 157:403-419.

- Kandel, B. M., and T. E. Hullar. 2010. The relationship of head movements to semicircular canal size in cetaceans. *Journal of Experimental Biology* 213:1175-1181.
- Kemp, T. S. 2009. The endocranial cavity of a nonmammalian eucynodont, *Chiniquodon theotenicus*, and its implications for the origin of the mammalian brain. *Journal of Vertebrate Paleontology* 29:1188-1198.
- Kralj-Hans, I., J. S. Baizer, C. Swales, and M. Glickstein. 2007. Independent roles for the dorsal paraflocculus and vermal lobule VII of the cerebellum in visuomotor coordination. *Experimental Brain Research* 177:209-222.
- Larsell, O., and J. Jansen. 1970. *The Comparative Anatomy and Histology of the Cerebellum from Monotremes through Apes*. University of Minnesota Press, Minneapolis, Minnesota. 269 pp.
- Lindenlaub, T., H. Burda, and E. Nevo. 1995. Convergent evolution of the vestibular organ in the subterranean mole-rats, *Cryptomys* and *Spalax*, as compared with the aboveground rat, *Rattus*. *Journal of Morphology* 234:303-311.
- Loqman, M. Y., P. G. Bush, C. Farquharson, and A. C. Hall. 2010. A cell shrinkage artefact in growth plate chondrocytes with common fixative solutions: importance of fixative osmolarity for maintaining morphology. *European Cells and Materials* 10:214-227.

- Mach, E. 1873. Physikalische Versuche über den Gleichgewichtssinn des Menschen. Sitzungsberichte der Kaiserlichen Akademie der Wissenschaften. Mathematisch-Naturwissenschaftliche Classe 68:124-140.
- MacIntyre, G. T. 1972. The trisulcate petrosal pattern of mammals. *Evolutionary Biology* 6:275-303.
- Macrini, T. E. 2004. *Monodelphis domestica*. *Mammalian Species* 760:1-8.
- Macrini, T. E., T. Rowe, and J. L. VandeBerg. 2007a. Cranial endocasts from a growth series of *Monodelphis domestica* (Didelphidae, Marsupialia): a study of individual and ontogenetic variation. *Journal of Morphology* 268:844-865.
- Macrini, T. E., G. W. Rougier, and T. Rowe. 2007b. Description of a cranial endocast from the fossil mammal *Vincelestes neuquenianus* (Theriiformes) and its relevance to the evolution of endocranial characters in therians. *Anatomical Record* 290:875-892.
- Malinzak, M., R. F. Kay, and T. E. Hullar. 2011. Semicircular canal orthogonality, not radius, best predicts mean speed of locomotor head rotation: a new hypothesis with implications for reconstructing behaviors in extinct species. *American Journal of Physical Anthropology* 144:204.
- Malinzak, M. D. 2010. Experimental analyses of the relationship between semicircular canal morphology and locomotor head rotations in primates. Dissertation, Department of Biological Anthropology and Anatomy, Duke University, Durham, North Carolina. 236 pp.

- Matano, S., T. Kubo, C. Niemitz, and M. Guenther. 1985. Semicircular canal organ in three primate species and behavioral correlations. *Fortschritte der Zoologie* 30:677-680.
- Mazza, D., and B. J. Winterson. 1984. Semicircular canal orientation in the adult resting rabbit. *Acta Otolaryngologica* 98:472-480.
- McClure, T. D., and G. H. Daron. 1971. The relationship of the developing inner ear, subarcuate fossa and paraflocculus in the rat. *American Journal of Anatomy* 130:235-249.
- McKenzie, D. 1912. The semicircular canals and the sense of position, or orientation. *Proceedings of the Royal Society of Medicine, Otological Section* 5:141-154.
- McVean, A. 1999. Are the semicircular canals of the European mole, *Talpa europaea*, adapted to a subterranean habitat? *Comparative Biochemistry and Physiology Part A* 123:173-178.
- Moore, S. T., E. Hirasaki, T. Raphan, and B. Cohen. 2005. Instantaneous rotation axes during active head movements. *Journal of Vestibular Research* 15:73-80.
- Moore, S. T., H. G. MacDougall, B. T. Peters, J. J. Bloomberg, I. S. Curthoys, and H. S. Cohen. 2006. Modeling locomotor dysfunction following spaceflight with galvanic vestibular stimulation. *Experimental Brain Research* 174:647-659.
- Muir, G. M., J. E. Brown, J. P. Carey, T. P. Hirvonen, C. C. Della Santina, L. B. Minor, and J. S. Taube. 2009. Disruption of the head direction cell signal after occlusion

- of the semicircular canals in the freely moving chinchilla. *Journal of Neuroscience* 29:14521-14533.
- Neu, S. C., D. J. Valentino, and A. W. Toga. 2005. The LONI Debabeler: a mediator for neuroimaging software. *Neuro Image* 25:1170-1179.
- Norris, C. A. 1994. The periotic bones of possums and cuscuses: cuscus polyphyly and the division of the marsupial family Phalangeridae. *Zoological Journal of the Linnean Society* 111:73-98.
- Nowak, R. M. 1999. *Walker's Mammals of the World, Sixth Edition*. Johns Hopkins University Press, Baltimore, Maryland. 1936 pp.
- Olson, E. C. 1944. Origin of mammals based upon cranial morphology of the therapsid suborders. *Geological Society of America Special Papers* 55:136.
- Oman, C. M., and E. N. Marcus. 1980. Influence of semicircular canal ampulla, duct, and utricular shape on endolymph flow dynamics. *Society of Neurosciences Abstracts* 6:558.
- Oman, C. M., E. N. Marcus, and I. S. Curthoys. 1987. The influence of semicircular canal morphology on endolymph flow dynamics. *Acta Otolaryngologica* 103:1-13.
- Rabbitt, R. D. 1999. Directional coding of three-dimensional movements by the vestibular semicircular canals. *Biological Cybernetics* 80:417-431.

- Rabbitt, R. D., E. R. Damiano, and J. W. Grant. 2004. Biomechanics of the vestibular semicircular canals and otolith organs; pp. 153-201 in S. M. Highstein, A. Popper, and R. Fay (eds.), *The Vestibular System*. Springer-Verlag, Berlin, Germany.
- Robinson, E. S., J. L. VandeBerg, and G. B. Hubbard. 1994. Malignant melanoma in ultraviolet irradiated laboratory opossums: initiation in suckling young, metastasis in adults, and xenograft behavior in nude mice. *Cancer Research* 54:5986-5991.
- Romer, A. S. 1962. *The Vertebrate Body*. W. B. Saunders Company, Philadelphia, Pennsylvania. 627 pp.
- Ross, M. H., and W. Pawlina. 2011. *Histology: A Text and Atlas with Correlated Cell and Molecular Biology*. Lippincott Williams & Wilkins, Philadelphia, Pennsylvania. 974 pp.
- Rowe, T. 1995. Brain heterochrony and origin of the mammalian middle ear: new data from high resolution X-ray CT. *Journal of Vertebrate Paleontology Supplement to* 15:50A.
- Rowe, T. 1996a. Coevolution of the mammalian middle ear and neocortex. *Science* 273:651-654.
- Rowe, T. 1996b. Brain heterochrony and evolution of the mammalian middle ear; pp. 71-95 in M. Ghiselin, and G. Pinna (eds.), *New Perspectives on the History of Life*. California Academy of Sciences, Memoir 20.

- Rowe, T. 2004. Chordate phylogeny and evolution; pp. 384-409 in J. Cracraft, and M. J. Donoghue (eds.), *Assembling the Tree of Life*. Oxford University Press, Oxford, UK.
- Rowe, T., J. Kappelman, W. D. Carlson, R. A. Ketcham, and C. Denison. 1997. High-Resolution Computed Tomography: a breakthrough technology for Earth scientists. *Geotimes* 42:23-27.
- Rowe, T., T. H. Rich, P. Vickers-Rich, M. Springer, and M. O. Woodburne. 2008. The oldest platypus and its bearing on divergence timing of the platypus and echidna clades. *Proceedings of the National Academy of Science of the United States of America* 105:1238-1242.
- Rowe, T. B., T. P. Eiting, T. E. Macrini, and R. A. Ketcham. 2005. Organization of the olfactory and respiratory skeleton in the nose of the gray short-tailed opossum *Monodelphis domestica*. *Journal of Mammalian Evolution* 12:303-336.
- Sánchez-Villagra, M. R. 2001. Ontogenetic and phylogenetic transformations of the vomeronasal complex and nasal floor elements in marsupial mammals. *Zoological Journal of the Linnean Society* 131:459-479.
- Sánchez-Villagra, M. R. 2002. The cerebellar paraflocculus and the subarcuate fossa in *Monodelphis domestica* and other marsupial mammals -- ontogeny and phylogeny of a brain-skull interaction. *Acta Theriologica* 47:1-14.

- Sánchez-Villagra, M. R., and F. Sultan. 2002. The cerebellum at birth in therian mammals, with special reference to rodents. *Brain, Behavior and Evolution* 59:101-113.
- Sánchez-Villagra, M. R., and J. R. Wible. 2002. Patterns of evolutionary transformation in the petrosal bone and some basicranial features in marsupial mammals with special reference to didelphids. *Journal of Zoological Systematics and Evolutionary Research* 40:26-45.
- Selva, P., C. M. Oman, and H. A. Stone. 2009. Mechanical properties and motion of the cupula of the human semicircular canal. *Journal of Vestibular Research* 19:95-110.
- Shaikh, A. G., F. F. Ghasia, J. D. Dickman, and D. E. Angelaki. 2005. Properties of cerebellar fastigial neurons during translation, rotation, and eye movements. *The Journal of Neurophysiology* 93:853-863.
- Silcox, M. T., J. I. Bloch, D. M. Boyer, M. Godinot, T. M. Ryan, F. Spoor, and A. Walker. 2009. Semicircular canal system in early primates. *Journal of Human Evolution* 56:315-327.
- Silva, M., and J. A. Downing. 1995. *CRC Handbook of Mammalian Body Masses*. CRC Press, New York, New York. 359 pp.
- Sipla, J. S., and F. Spoor. 2008. The physics and physiology of balance; pp. 227-232 in J. G. M. Thewissen, and S. Nummela (eds.), *Sensory Evolution on the Threshold:*

- Adaptations in Secondarily Aquatic Vertebrates. University of California Press, Berkeley, California.
- Son, G. M. L., J.-S. Blouin, and J. T. Inglis. 2008. Short-duration galvanic vestibular stimulation evokes prolonged balance responses. *Journal of Applied Physiology* 105:1210-1217.
- Sorrentino, M., L. Manni, N. J. Lane, and P. Burighel. 2000. Evolution of cerebral vesicles and their sensory organs in an ascidian larva. *Acta Zoologica* 81:243-258.
- Spoor, F. 2003. The semicircular canal system and locomotor behaviour, with special reference to hominin evolution. *Courier Forschungsinstitut Senckenberg* 243:93-104.
- Spoor, F., and F. Zonneveld. 1995. Morphometry of the primate bony labyrinth: a new method based on high-resolution computed tomography. *Journal of Anatomy* 186:271-286.
- Spoor, F., and M. Leakey. 1996. Absence of the subarcuate fossa in cercopithecids. *Journal of Human Evolution* 31:569-575.
- Spoor, F., and F. Zonneveld. 1998. Comparative review of the human bony labyrinth. *Yearbook of Physical Anthropology* 41:211-251.
- Spoor, F., B. A. Wood, and F. Zonneveld. 1994. Implications of early hominid labyrinthine morphology for evolution of human bipedal locomotion. *Nature* 369:645-648.

- Spoor, F., B. A. Wood, and F. Zonneveld. 1996. Evidence for a link between human semicircular canal size and bipedal behaviour. *Journal of Human Evolution* 30:183-187.
- Spoor, F., T. Garland, G. Krovitz, T. M. Ryan, M. T. Silcox, and A. Walker. 2007. The primate semicircular canal system and locomotion. *Proceedings of the National Academy of Science of the United States of America* 104:10808-10812.
- Steinhausen, W. 1933. Über die Beobachtung der Cupula in den Bogengangsampullen des Labyrinths des lebenden Hechts. *Pflügers Archiv* 232:500-512.
- Streit, A. 2001. Origin of the vertebrate inner ear: evolution and induction of the otic placode. *Journal of Anatomy* 199:99-103.
- Summers, A. P., R. A. Ketcham, and T. Rowe. 2004. Structure and function of the horn shark (*Heterodontus francisci*) cranium through ontogeny: development of a hard prey specialist. *Journal of Morphology* 260:1-12.
- Taylor, J., F. Rühli, G. Brown, C. De Miguel, and M. Henneberg. 2006. MR imaging of brain morphology, vascularisation and encephalization in the koala. *Australian Mammalogy* 28:243-247.
- ten Kate, J. H., H. H. van Barneveld, and J. W. Kuiper. 1970. The dimensions and sensitivities of semicircular canals. *Journal of Experimental Biology* 53:501-514.
- Ulinsky, P. S. 1971. External morphology of pouch young opossum brains: a profile of opossum neurogenesis. *Journal of Comparative Neurology* 142:33-58.

- VandeBerg, J. L. 1990. The gray short-tailed opossum (*Monodelphis domestica*) as a model didelphid species for genetic research. *Australian Journal of Zoology* 37:235-247.
- VandeBerg, J. L. 1999. The laboratory opossum (*Monodelphis domestica*); pp. 193-209 in T. B. Poole, and P. English (eds.), *UFAW Handbook on the Care and Management of Laboratory Animals*. Blackwell Science, Oxford, UK.
- VandeBerg, J. L., and E. S. Robinson. 1997. The laboratory opossum (*Monodelphis domestica*); pp. 238-253 in N. Saunders, and L. Hinds (eds.), *Marsupial Biology, Recent Research, New Perspectives*. University of New South Wales Press, Sydney, Australia.
- Venes, D. (ed.). 2005. *Taber's Cyclopedic Medical Dictionary, 20th Edition*. F. A. Davis Company, Philadelphia, Pennsylvania. 2788 pp.
- Welker, K. L., J. D. Orkin, and T. M. Ryan. 2009. Analysis of intraindividual and intraspecific variation in semicircular canal dimensions using High-Resolution X-ray Computed Tomography. *Journal of Anatomy* 215:444-451.
- Wever, E. G. 1975. The evolution of vertebrate hearing; pp. 423-454 in H. Autrum, W. D. Keidel, and W. D. Neff (eds.), *Handbook of Sensory Physiology*. Springer, Berlin, Germany.
- Wible, J. R. 1990. Petrosals of Late Cretaceous marsupials from North America, and a cladistic analysis of the petrosal in therian mammals. *Journal of Vertebrate Paleontology* 10:183-205.

- Wible, J. R. 2003. On the cranial osteology of the short-tailed opossum *Monodelphis brevicaudata* (Didelphidae, Marsupialia). *Annals of Carnegie Museum* 72:137-202.
- Wible, J. R., G. W. Rougier, M. J. Novacek, M. C. McKenna, and D. Dashzeveg. 1995. A mammalian petrosal from the Early Cretaceous of Mongolia: Implications for the evolution of the ear region and Mammaliomorph interrelationships. *American Museum Novitates* 3149:1-19.
- Witmer, L. M. 1995. The extant phylogenetic bracket and the importance of reconstructing soft tissues in fossils; pp. 19-33 in J. Thomason (ed.), *Functional Morphology in Vertebrate Paleontology*. Cambridge University Press, Cambridge, UK.
- Xiong, G., and S. Nagao. 2002. The lobulus petrosus of the paraflocculus relays cortical visual inputs to the posterior interposed and lateral cerebellar nuclei: an anterograde and retrograde tracing study in the monkey. *Experimental Brain Research* 147:252-263.
- Xiong, G., S. Nagao, and H. Kitazawa. 2010. Mossy and climbing fiber collateral inputs in monkey cerebellar paraflocculus lobulus petrosus and hemispheric lobule VII and their relevance to oculomotor functions. *Neuroscience Letters* 468:282-286.
- Yakusheva, T. A., A. G. Shaikh, A. M. Green, P. M. Blazquez, J. D. Dickman, and D. E. Angelaki. 2007. Purkinje cells in posterior cerebellar vermis encode motion in an inertial reference frame. *Neuron* 54:973-985.

Yang, A., and T. E. Hullar. 2007. Relationship of semicircular canal size to vestibular-nerve afferent sensitivity in mammals. *Journal of Neurophysiology* 98:3197-3205.

## **Vita**

The author is a native southern Californian, raised primarily in San Diego. A constant habit of looking at the ground and noticing rocks and fossils spurred her to study them as science projects in elementary school. As an undergraduate at the University of Southern California, her interest in geology won out over other interests such as chemistry, physics, computer science, and ancient art. As a result, she completed a BS in geology and a subsequent MS in structural geology at the University of California, Los Angeles.

After graduate school, she worked as an exploration geologist with an ARCO Exploration in Dallas. She left the company in 2005 to raise a family.

When her children were approaching college age, her interest in science again drove Jeri to graduate school. She entered a doctoral program in January, 2000 to study vertebrate paleontology at University of Texas at Austin. Once again she had fallen under the allure of the fossils that had attracted her as a young girl in San Diego.

Permanent address (or email): [JeriCR55@aol.com](mailto:JeriCR55@aol.com)

This dissertation was typed by Jeri C. Rodgers.