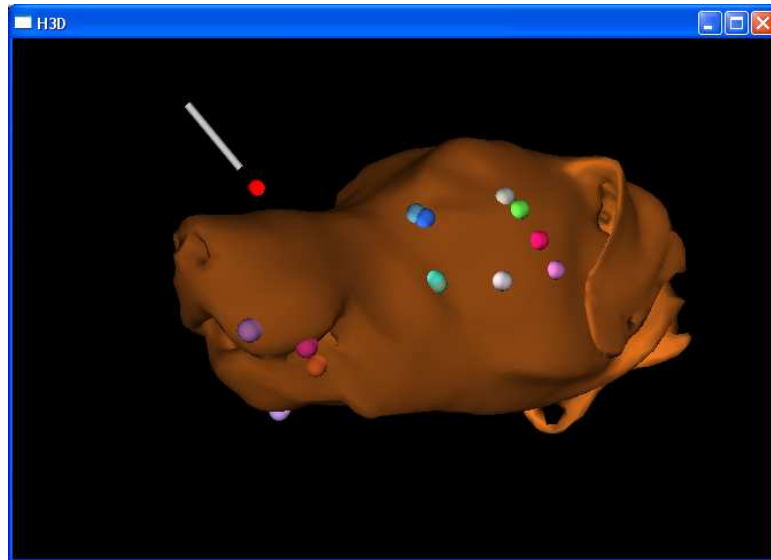


# SimPooch

Canine Medical Acupuncture Training & Simulation System



**Brendan Dahl - Base System and Haptics**

dahlb@engr.colostate.edu

**Sean Thomas – User Interface**

sptbear@gmail.com

**Bryan Landwehr – Dog Model**

blandwehr@msn.com

Advised By:

**Professor Peter Young - ECE**

pmy@engr.colostate.edu

Sponsored By:

**Narda Robinson - CVMBS**

nrobinso@lamar.colostate.edu

**Regina Schoenfeld-Tacher - CVMBS**

reginast@colostate.edu

May 4th, 2008

Colorado State University  
Department of Electrical & Computer Engineering  
College of Veterinary Medicine and Biomedical Sciences

## Abstract

*In the course of creating a simulation to train veterinary students to perform canine acupuncture there are several obstacles that must be overcome. One such obstacle is training a student in such a way that canine acupuncture can be practiced in a real world scenario, on a life-like dog replica, without the ethical issue of inexperienced practice on a live dog. In the SimPooch project we have used the Python programming language to create a computer program that encases a life-like mannequin of a MRI image of a dog with a three dimensional object. In this fashion, the program, through the use of a haptic device, can simulate and detect acupuncture punctures on the mannequin. The final goal of the project is to develop a process to manufacture a canine acupuncture simulator which has the ability to grade an acupuncture student's performance.*

## Table of Contents

Title	1
Abstract	2
Table of Contents	3
List of Figures and Tables	4
I. Summary	5
II. Introduction	6
A. Background of Acupuncture	6
B. Need for an Acupuncture Simulator	6
III. Problem Statement	8
IV. Constraints	9
V. Design Detail	10
A. The Physical System	10
B. Data Conversion	10
C. The Base Unit	12
D. The Virtual System	17
E. Point Data Analysis	23
VI. Conclusions and Future Development	25
Appendix A. Software Samples	27
Appendix B. Budget	32
References	33
Acknowledgements	34

## List of Figures & Tables

Figure 1.	Project Design Layout	10
Figure 2.	Distorted Walker Hound	11
Table 1.	Average Hounsfield Units for Pertinent Materials	11
Figure 3.	Final Triangle Mesh Model	12
Figure 4.	Stationary Head Options	12
Figure 5.	Haptic Device and Mannequin	13
Figure 6.	Initial Stand Design	13
Figure 7.	Assembled Chassis	14
Figure 8.	Coupled Magnetic Encoder	15
Figure 9.	Base Unit Output as a Wave	16
Table 2.	Haptic Device Comparison	18
Figure 10.	Sensible Phantom Omni	18
Figure 11.	Conceptual AMIRA Mesh (*.am) File	19
Figure 12.	Conceptual Wavefront Object (*.obj) File	20
Figure 13.	AMIRA Screenshot	20
Figure 14.	Haptic Programming API Comparison	21
Figure 15.	Current SimPooch Test Program Structure	22
Figure 16.	Voronoi Map with Regions	24
Table 3.	Spring 2008 Budget	32

## I. Summary

The SimPooch senior design team's purpose is to design and fabricate a canine acupuncture simulation system. The system was first conceptualized by Dr. Narda Robinson and Dr. Regina Schoenfeld-Tacher of the CSU College of Veterinary Medicine and Biomedical Sciences. They expressed a need for such a system due to the fact that they had no save, reliable and repeatable way to train students to perform canine acupuncture. Project requirements and goals were determined after meeting with Dr. Robinson and Dr. Schoenfeld-Tacher in the spring of 2006. The team has since pursued development along two separate avenues. The physical canine head modeling process was the primary concentration of the Fall 2006/Spring 2007/Fall 2008 SimPooch Team.

The goal of the Physical development track was to create an anatomically correct canine mannequin for the use of palpation during acupuncture. The team used a CT scan of a Labrador Retriever's head to obtain three-dimensional data for use in both the Physical and Virtual systems. The team then used this data to create a prototyped skull and a canine head. Through the use of a computer program called Amira, the team was able to convert the CT scan data in to a .STL file. This file allowed the team to order two rapid prototypes from Protogenic: a canine head and a canine skull. The team then used the canine head to create a negative mold. They then placed the canine skull inside this negative mold and filled the negative mold with a soft Silicon material. The soft silicon material then solidified in the shape of the canine head, representing the dog's flesh layer. This process resulted in a canine mannequin that would be used for palpation in the simulator. For a more in depth analysis of the Physical System, please read the SimPooch Spring 2007 Engineering report.

The goal of the Virtual development track was to create a computer program that records an acupuncture student's touches on the physical mannequin. The team used same data that was obtained from the CT scan to create a three-dimensional haptics object. This object, when used with the Sensible Phantom Omni Haptics Stylus, would represent the canine mannequin inside the computer program. The program itself should perform two tasks. The first task is to provide means to grade an acupuncture student on how accurately they performed the acupuncture. The second is to provide the student with a correct sense of touch feedback while virtually penetrating the canine

The project's final goal is to integrate these two development tracks into a final system. This integration gave rise to a third area of development; the Base Stand. The goal of the Base Stand Development is to create a mounting device for the canine mannequin from the Physical system and to provide the Virtual system with the rotational position of the mannequin.

The Physical development of the mannequin and its associated development of a process to create such a mannequin were completed during the Spring 2007 semester. During the Spring 2008 semester, the team was able to continue and finish the development of the Base Unit. The Spring 2008 team was also able to create vital, base functionality in the Virtual computer program. The program now renders the onscreen canine object, rotates that object based on the signal from the Base Unit, and logs touches upon that object by the virtual needle. Several goals were set for next semester's team, as this project has the potential to develop into a very accurate, safe training device for acupuncture students.

## II. Introduction

### A. Background of Acupuncture

The science of acupuncture has been practiced for over 4,500 years, and it is still used as a clinical remedy for many illnesses and conditions. It has typically been understood to relieve the “qi” in a body through energy points of mystical power. This misconception has led many to accept acupuncture as a holistic practice rather than a scientific process.

Acupuncture, more specifically, is the ability to precisely place fine tipped needles in locations that stimulate a nervous or muscular reaction. The proper placement and combination of pins can ease a wide range of conditions, including decreasing nausea and soreness, improving recovery treatment and reducing stress.

“Animal acupuncture points have been derived from Chinese point drawings as well as from transposing one or more systems of human acupuncture points onto animal anatomy.” This caused an inconsistency in point locations. Thus, two points with the same name have markedly different locations between humans and non-humans and, as a result, will have different physiologic effects and applications.

Canines undergo acupuncture to strengthen the animal’s immune system, relieve pain, and improve the function of organ systems. Acupuncture can help such fundamental problems as paralysis, arthritis, gastrointestinal problems, certain reproductive problems, and pain. The treatments stimulate nerves, increase blood circulation, relieve muscle spasms, and cause the release of such hormones as endorphins and cortisol.

### B. Need for an Acupuncture Simulator

An acupuncturist requires substantial knowledge and experience for optimal medical benefit. The acupuncturist must understand human/animal anatomy and physiology in order to identify the location of a nerve and its effect on the body when stimulated by a needle. In addition, she must also recognize the body’s response to the needle at each stimulation point. Due to anatomical differences within one species, the location of each point may vary from patient to patient. “The variability in veterinary acupuncture point locations has drawn considerable criticism, not only from skeptics but also from students who earnestly desire to master the art, yet become frustrated when they learn that many instructors and practitioners place the points in different locations,” comments Dr. Narda Robinson. Thus, it is critical to study the body’s response to determine if the acupuncturist inserted the needle correctly. For example, each tissue of the body (skin, muscle, bone, etc.) will provide a different resistance to the needle allowing the acupuncturist to determine needle location by the “feel” of the insertion. Also, when correct insertion is achieved, nerve stimulation by small movements of the needle will result in a small muscle contraction at the acupuncture point. For the patient, this may result in a “tingling” or “cold” sensation at the insertion point. Human patients are able to verbally communicate with the acupuncturist about the proper placement. The ability to recognize each response requires practice that, at this point in time, cannot be simulated on anything other than a human. This proves to be the difficulty of canine acupuncture; no response can be given by the animal. Slight movement may occur if the animal experiences discomfort, but reasons cannot be identified.

Although educational acupuncture on animals and cadavers allows a student to practice on real bodies with true anatomy, there is no way to determine when correct location of the needle is achieved.

An acupuncture simulator would be an ideal tool in accelerating the learning process of students that attend acupuncture training sessions across the country. A simulator can be designed to generate feedback when determining the proper placement of a needle. A certification mode will be required to test the student’s ability to correctly identify and penetrate several acupuncture points given no feedback other

than the sense of touch from the needle. Tracking the student's performance will be essential upon certification. The testing method also removes examiner bias, often found in testing situations. The simulator will allow the students repetition while training, refining the students motor skills. This is a critical aspect to successfully mastering the advance techniques in acupuncture. The simulator also provides a safer alternative than in vivo (alive) instruction.

It is the goal of this project to design a prototype of an acupuncture simulator that can be used to certify students in basic acupuncture techniques, more specifically proper point location. Anticipated success of the simulator will be determined by its ability to provide an accurate sense of touch feedback using tissue resistance and visual or audio notification when to the acupuncturist inserts the needle properly.

### III. Problem Statement

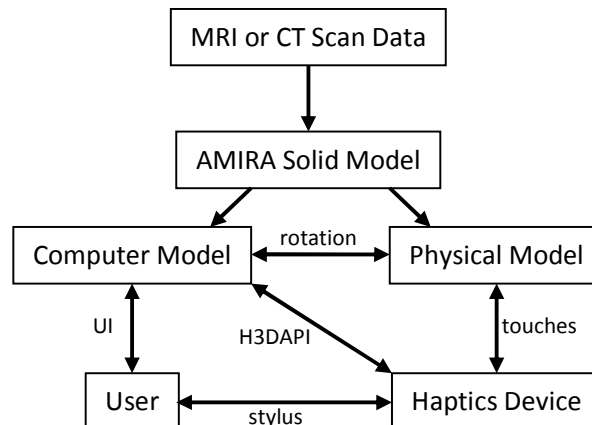
The goal is to create a medical acupuncture simulator comprised of a life-size, anatomically correct, physical model with a computer-based haptics interface providing sense of touch force feedback for students to learn and practice methods of traditional Chinese acupuncture. Competency testing and certification examinations will compare the student's responses with the correct point location and deliver feedback to the user (student or instructor) as appropriate.

#### IV. Constraints

1. Proper location of 83 cranial acupuncture points must be identified and implemented in the model so that it may serve as an accurate tool for education.
2. Each point must respond to 800 cycles in order to maintain effectiveness as a training aid. The material needs to be healable, pliable, but resistant to deformation from creep.
3. The surface of each point must not show wear to the visible eye at 0.5 meters.
4. For ease of transport, the simulator must weigh less than 20 lbs.
5. Cost of the simulator must not exceed \$25,000.
6. The simulator must accurately represent the contours of the canine.
7. Each point must be properly located within the anatomical layers of the simulated canine skull.
8. Each anatomical layer must simulate the tactile response of an applied load.
9. The skull must be self supporting
10. Each point must be properly located within a 5mm sphere of the anatomical layers of the simulated canine skull.

## V. Design Detail

Below is a flow diagram of the preliminary design of the project. To create anatomical likeness, it was determined that a CT or MRI scan be used for the data. With this data the team can use AMIRA, an advanced 3D visualization and modeling system, to create a .STL file. Ultimately, the mannequin and the materials can be mapped to vertices to create the computer model.



*Figure 1. Project Design Layout*

The user will use the mannequin to palpate and find the appropriate acupuncture point. Then, the haptics device will be used to perform the acupuncture procedure on that point. The haptics device will be mapped to the computer model, so the points can be traced, and the appropriate sense of touch feedback will be given to the student. Based on the above procedure, the student's abilities can be graded.

### A. The Physical System

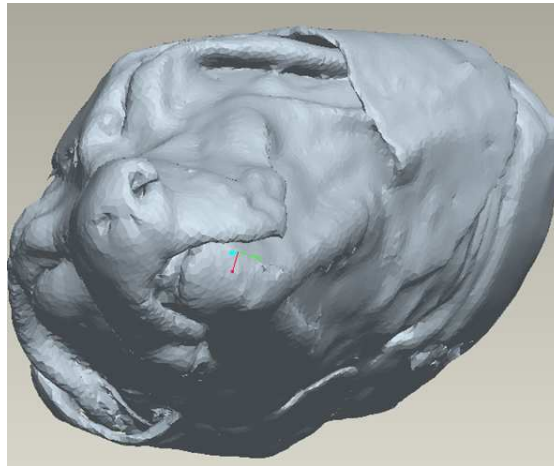
The physical system is comprised of a stand that holds the mannequin. The physical model allows the user to perform palpation to identify the correct acupuncture location. With anatomically correct features, the mannequin will provide the user with an accurate palpation experience. The mannequin was developed from a CT scan, which was converted in Amira. Details of the design, analysis and procedure of the physical system can be found in the SimPooch Spring 2007 Engineering Report.

### B. Data Conversion

To create the anatomically correct mannequin, it was chosen to use a Computed Tomography (CT) of a canine. This will allow for exact data directly from the canine. A Magnetic Resonance Image (MRI) was also an option. A research study was taking place at the CSU Veterinary Hospital using Walker Hounds, so MRI scans were available. Dr. Sue Kraft and Mrs. Betsy Sestina offered their expertise in reading MRI scans and identifying pertinent features of the canines head. After the MRI was taken, Wes Womack, a graduate student at Colorado State University provided his time and effort while using Amira. Amira, as mentioned above is an advanced 3D visualization and modeling system that is very useful for displaying and converting 3D image data. The program was used to convert the MRI & CT scans to .OBJ (object) and .STL files.

The first MRI that was taken was hard to open, as there were several series of scans in all X, Y, and Z directions mixed together. If an MRI is used, the user needs to recognize the series that corresponds

to the data going front to back, or top to front should be used--there are several free programs that can be used to determine this (ImageJ, Idioimaging,etc). Once the data was finally recognized and opened, the head was squished and hard to decipher different materials. Apparently, the deceased canine was wrapped in a plastic wrap that deformed the flesh. The hydrogen content of the wrapped material was very close to that of flesh, so trying to identify the materials proved to be nearly impossible. After hours of identifying the differences, the following image was obtained:



**Figure 2. Distorted Walker Hound**

If the data of this Walker Hound was used to create the mannequin, the student would not experience an accurate palpation experience. Anatomical likeness would not be achieved if this data were used.

We happened to have a CT of a Labrador that one of his peers performed a spine analysis. The one problem with the CT was the top of the canines head was not in the initial frame of the CT, so the image wasn't complete. Fortunately, by going through every slide, and identifying the top of the head on a previous slice, one can interpolate and create a very accurate representation of the missing aspect. Also, Amira allows the user to identify materials based on gray value (also known as Hounsfield Units) or selection techniques. The gray scale method was used first because it narrowed down a lot of small areas. ImageJ was used to obtain values for the pertinent materials, so easy identification would occur while examining the scans. An example of the values obtained, see the table below.

<b>Material</b>	<b>Min Value</b>	<b>Max Value</b>
Air	-1150	-300
Fat	-299	20
Muscle	21	99
Bone	100	2350

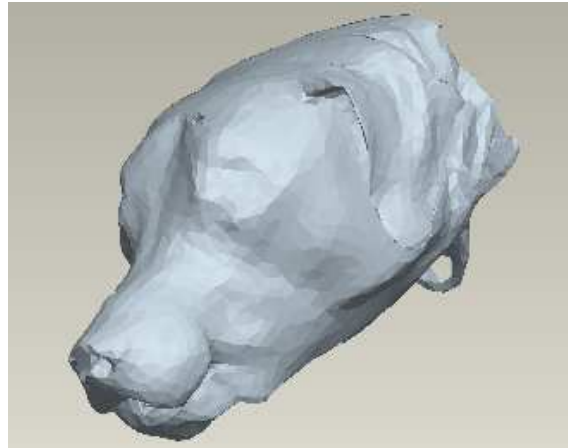
**Table 1. Average Hounsfield Units for Pertinent Materials**

Amira converts the gray values to positive values, as usually seen on MRI scans. First, the bone layer was identified because it has the gray scale value, and it will be simple to remove and select all of the other materials. Only bone and flesh were identified because palpation only concentrates on the underlying anatomical features of the skull. Therefore, muscle and fat are not necessary for palpation purposes.

The development of the program will eventually require identification of separate materials including muscle and fat but may be in semesters to come and the MRI will be needed for an accurate

identification. Once the bone was identified, the material was completely removed and everything else was identified as the “fleshy layer.”

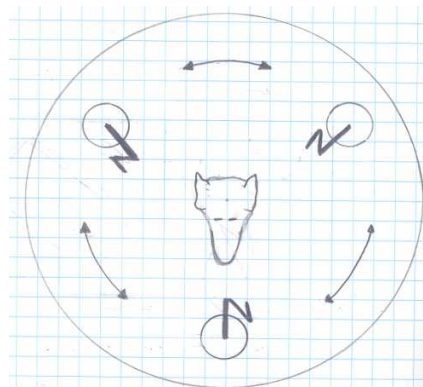
From this data and the identified materials, the skull and the fleshy later with the skull were made into .STL files for the creation of a rapid prototype. The final data used can be seen in the figure below.



**Figure 3. Final Triangle Mesh Model**

### C. The Base Unit

Since the haptics device has a limited workspace, two options were available. One was to create a rotating mannequin, with a stationary machine. This would basically cause the head to rotate around the haptics device. A second option was to create a program through the haptics device with several zeros, and have the haptics device travel around the mannequin.

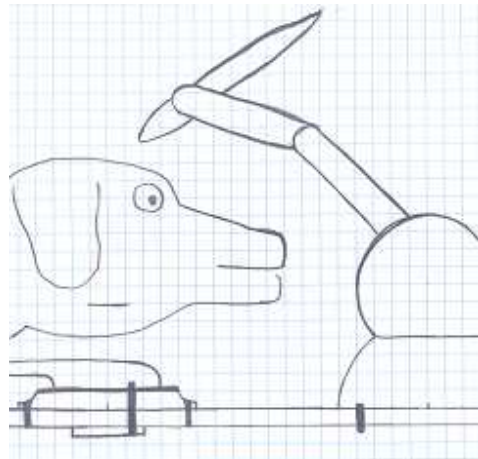


**Figure 4. Stationary Head Option**

Since little information was known about the programming, it was thought that for simplicity, the head rotate with a stationary haptics device. This would require the physical model to rotate simultaneously with the computer model. While, ultimately, it integrates the models, it's shown on the flow chart as just a part of the physical system.

When designing the stand, the team consider several constraints. The stand needed to be a self-contained workstation for acupuncture students while housing several components. The constraints and

criteria of the stand design included considerations for many pieces of the project. Constraints and criteria were determined through the mannequin model, haptics device, drive train, computer and electrical interfaces, controls, cooling of the circuitry and input from Dr. Narda Robinson. Essentially, the purpose of the stand was to provide a rigid link between the haptics device and the canine mannequin. The rigid link is essential for the project because the virtual model of the canine shares the same space as the physical mannequin. The two models need to be geometrically coincident with respect to the haptics device to provide an accurate interface between the students determined location of the acupuncture point and the geometrical coordinate recorded by the haptics device.

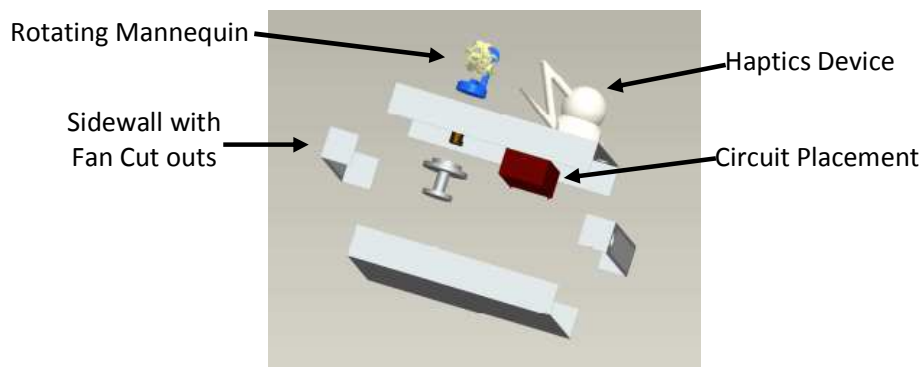


**Figure 5. Haptic Device and Mannequin**

A major concern of Dr. Narda Robinson was that the stand and components would be durable enough to withstand several years of use by her veterinary students. Eight hundred cycles was the estimated as the life requirement of the system. Also, the stand assembly needed to be easily transportable.

**Initial Design**

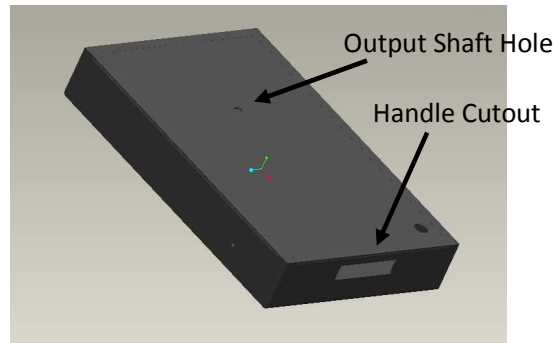
Below was the initial stand design in an exploded view with the general components labeled. Functions of these components can be found below.



**Figure 6. Initial Stand Design**

**Stand Features**

The figure below shows the assembled chassis design. The key features are the handle cut outs on the end and the hole on the top surface where the output shaft comes through to connect to the mannequin-mounting bracket.



*Figure 7. Assembled Chassis*

### **Stand Design Iterations**

The rotating mannequin requirement added a few more constraints to the design. The stand needed to be large enough to accommodate the space needed to allow the mannequin to rotate about a center axis. The mannequin also needed to be suspended above the stand to allow room for palpation under the mannequin's jaw and head. Also, the piece that connected the mannequin to the driven shaft could not impair the student's ability to palpate as well as remaining clear of the haptics device. Accurate recording of the mannequin's angular orientation was an important constraint to consider so the virtual and physical models continue to remain aligned as the mannequin changes orientation. The tolerance of the acupuncture points greatly influenced the tolerance of the angular positioning as well as the deflection in the X, Y, and Z planes. An estimated angular positioning tolerance needed to be within 3 degrees, while the deflection in the X, Y, and Z planes needed to be less than 5 mm.

The computer and electrical interface needed to neatly and easily connect to the stand. The locations of the connections needed to be near the haptics device to limit the bulk produced by the power and data chords. Additionally, the power and data chords needed to be easily unplugged from the stand to make transport of the stand easier. The power supply as well the other internal components of the circuit need to free from obstructing moving components of the stand while not overheating.

Planning to accommodate all the physical requirements of the stand proved to be a valuable step. The stand was needed to house many components such as the DC stepper motor, pinion and driven gears, circuit boards, power supply, and motor control. Other features that are attached to the external casing includes: two cooling fans, USB connector, and a power entry module. Handles on the sides of the stand were incorporated to make it easy to lift. All electrical connections are made on the panel with plugs, thus eliminating loose wires hanging from the stand during transportation.

### **Chassis Design**

The design process for the chassis began by examining existing chassis designs. The final design subsequently used features found on chassis at T.D.P. Pro Engineer's sheet metal feature was used to create the chassis design. Several iterations were made before reaching our final chassis design. The final stand design consists of four pieces. The largest piece forms three sides – the bottom and two sides. The front, back, and top of the stand are made separately and bolted together.

The design of the front panel called for an assembly using standoffs to provide the air-intake for cooling and provide a location for potentially adding a glowing effect at the junction using LED's. Dual fans pull air through the gap at the faceplate to create convection current that cools circuit components. Slots for handles, USB, power entry module and a rocker switch were strategically placed on the design to allow for a clean finish and for great ease of use.

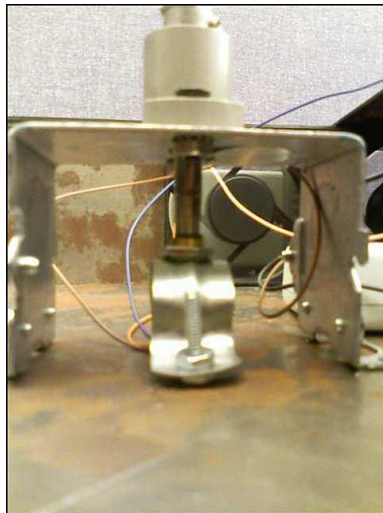
The construction process of the stand began by having the individual pieces cut from 14 gauge steel sheet metal. Metal Tech Ind., a local vendor cut our pieces with a CNC laser cutting machine and placed the notches and bends with a Bend Press. A vendor was used because the stand had complicated geometrical cuts on large pieces of sheet metal that would be difficult to cut and hold securely in the mill. Once the individual pieces were prepared, the stand was assembled with #6-32 screws and press-on nuts. The press-on nuts are advantageous because they make assembly and disassembly very easy. The final surface finish on the stand was applied with black paint.

The haptics device does not have a large enough reach to cover the entire mannequin surface so the mannequin needed to be able to rotate. The mannequin spins on a eight inch radius so the placement of the haptics device was designed to sit on the corner of the user side as close to the mannequin as possible without interfering with its path. Haptics device mounting holes have not been placed on the top of the stand to allow for better control of the placement of the haptics device in the future.

The locations of internal circuitry were determined after the drive train was installed to insure that the two do not interfere. Once the locations were determined, transfer punches were used to place a mark on the drill spot. After the holes were drilled and countersunk the circuitry was installed using standoffs of allow for more airflow beneath the circuit components. Making the electrical connections was the last step in the assembly process.

### **Connected motion**

The mounting bracket connects the canine mannequin to the output shaft. It is made from  $\frac{3}{4}$  inch square aluminum tubing connected to two collar clamps. The aluminum tubing was cut into 4 and 8-inch sections with a  $45^\circ$  angle cut at one end of each. The other end was cut with an end mill to the outer dimensions of the collars. The smaller clamp attached to the  $\frac{1}{4}$  inch output shaft and was welded to the 8-inch section. The 4-inch section was welded to the larger clamp that attaches to the  $\frac{3}{8}$ -inch rod imbedded in the mannequin's skull. The figure below shows the clamp that was used in the design. The two aluminum sections were welded together at the  $45^\circ$  angles to create a  $90^\circ$  junction. After the aluminum cooled, the welds were cleaned up using a belt sander then the part was primed and painted black.



*Figure 8. Coupled Magnetic Encoder*

The mounting bracket remained relatively close to the initial design but it was the last part to be designed and built. The absence of the mannequin made it difficult to determine the required dimensions. The mounting bracket needed to attach the mannequin to the output shaft without interfering with palpation. The design to attach the bracket at the back of the head made the most sense because it

remains out of the way and provides access to underneath the head. The design was created using Pro Engineer but we were not able to get the Pro Mechanical verification to work correctly. In order to keep the design process moving manual calculation were performed to estimate the size of the aluminum tubing. The mounting surface between the bracket and the mannequin saw a last minute design change. The original design is shown in below. The original design called out a bolted connection between the bracket and the mannequin. The design was changed to use a larger size of the same collar used to connect the mannequin to the output shaft. This design change allowed for the collar to be welded directly to the bracket as well as creating a more streamline appearance.

### Rotational Measurement

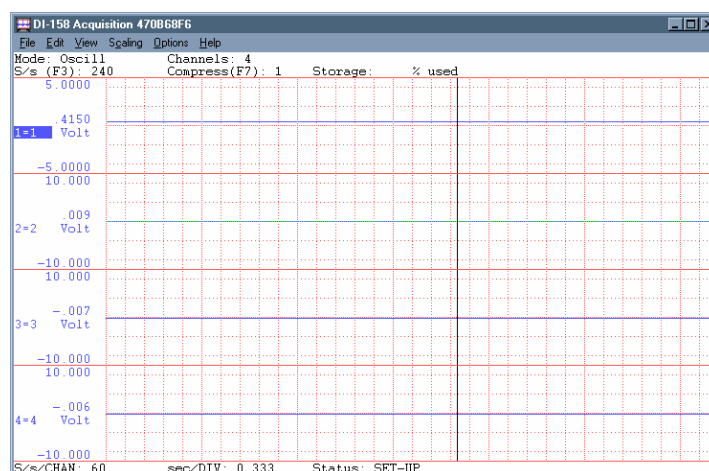
The initial idea for the angular sensor was to install PIC controller to the Base Unit's gear system that would be able to detect how much the mannequin rotated after a given amount of drive time. The PIC controller would then send a reading through an RS232 Signal encapsulated in a USB Device. After a great deal of testing, the team decided upon a much more efficient and cost effective way to accomplish the task of rotational measurement.

### Magnetic vs. Optical Encoder:

The original idea was to use a magnetic or optical encoder which would send back a voltage through an analog to digital converter. Because the team decided to dedicate time to re-design the internal circuitry, the new goal was to come up with a product that could reliably give us a reading within a 1/2 degree of accuracy instead of the proscribed 3 degrees. First research was done on a rotary encoder. It turned out that this solution would not end up having a high enough resolution, as it would not even be accurate to one degree. Looking further into a magnetic encoder it was found that the accuracy was much higher. A miniature absolute magnetic shaft encoder (MA3) manufactured by US Digital, was found to be ideal for our project. This magnetic encoder will have an output transition noise of +/- .06 degrees.

### Analog to Digital converter:

After doing research on the A-D converter, an initial model (DU-148U made by DataQ instruments), costing 50 dollars plus shipping, was selected that allows for +/- 10v analog signal conversion. The resolution of this converter was 19.7mV. Accounting for an input voltage range of 0-5 volts, this method would only allow for us to have approximately 250 different readings, which would not be suitable for the project. Instead, the team selected a slightly more expensive model (DU-158U) which would have a resolution of <5mV. This would allow for approximately 1250 readings allowing for measurements with an accuracy of approximately 0.29 degrees.



**Figure 9. Base Unit Output as a Wave**



### **Attachment and Testing:**

After deciding on the internal circuitry, the team had to design how to mount the encoder into the preexisting Base Unit setup, and how to ensure there was a certain amount of tension to prevent the head from freely spinning during simulation. The initial plan was to attach the MA3 with a belt and pulley, while maintaining the motor drive to prevent free-spin. An image of this setup is included below prior to deciding this was not a practical idea. After each rotation the encoder was not consistent in its data output because of the belt's tendency to slip. The second concept tested was to attach the encoder underneath the current shaft using a coupling as depicted in the images below. After mounting the new system, testing illustrated that it was indeed an extremely accurate solution.

### **D. The Virtual System**

The virtual system is comprised of the haptics device, a computer and computer model of the canine. The haptics device is used by the student to perform acupuncture. As mentioned, no material is penetrated, it's all virtual. The haptics device will provide the user with a different force for the different materials contacted with the stylist. The feedback allows the user to know if the target point is contacted with the virtual needle.

### **Computer Hardware Requirements**

While the specifications of the computer were loosely defined by the customer, the acupuncture team arrived at several requirements that would need to be fulfilled in order for the purchase to be a success. The underlying basis for all conclusions is that the computer must handle a large amount of visible data in a "real time" fashion. In other words, no visible or physical lag must be present when processing the program. The following sections highlight a few of the categories that were emphasized in the computer design.

#### **Processor**

The processor that was chosen was the Intel(tm) Core™2 Duo Processor E6700 processor installed by Dell. This provides two cores of processing power at 2.67 GHz, 1066 MHz front side bus and 4mb of L2 cache. Simply, for our given price range, this was the most efficient use of the funds, as there would not be a processor too powerful and our inability to know exact necessary qualifications at the time. The high FSB was critical in that it is often a limiting factor in the effective speed of the package.

#### **Video Card**

The main criterion for the video card would be its ability to handle large amount of visual vertices and at what pace. The video card that was chosen was approved for compatibility with the purchased Dell system. The video card, GeForce 7950GX2, is especially capable of processing large data structures, suggested at by its 1GB memory capacity. The card is rated at being able to process 2 billion vertices per second; a critical characteristic of the card. Furthermore, with a 1.2 GHz effective ram speed, dual video outputs and 512 bit resolution, it would be able to not only render the objects for the test subject, but also on a projector or second monitor for a class or lecture hall to view. By using a video card with such a high effective memory rate, the video card would not be limiting the speed from the other computer components.

#### **Memory**

One distributor of the SensAble product recommended at least 2 GB of memory, an increase of the manufacturer's requirement of 512 MB. We chose to install 2 GB DDR2 SDRAM at 667 MHz. We have the capability to expand the system up to 8 GB of similar memory

### Other Hardware Requirements

- The SensAble system requires a IEEE Firewire port for its communication
- While the models can be as large as 750MB, many can still be stored on a standard hard drive readily available. We chose a 160 GB high speed drive (10,000 RPM) hard drive with a transfer rate of 1.5 GB/s to ensure that the data could be retrieved without overloading the cache and slowing down the system.
- We chose a 19" LCD monitor for both a minimal desktop footprint and large viewing area for test observation.
- The system has a DVD burner to allow transfer of the large files without multiple CDs.

### Haptics Device

The haptics device is the main component of the virtual system. The user moves the stylist and virtually penetrates the canine, and the haptics device gives the users hand a force feedback respective to the material the student has penetrated.

The choice to pursue a haptics system was made because of the high reliability, consistency and reproducibility that would be available with such a system. The ability to use the haptics device to virtually recreate the desired surface was a large factor in the decision. Initially, it was perceived that a haptics device would be chosen so that it would be able to operate in a workspace large enough to encompass all regions of the canine skull. The skull was approximated as a volume 10 in x 8 in x 6 in. The group decided that to allow full accessibility, at least 3 inches of workspace would have to be available in all directions around the skull. This provided us a working space of 13 in x 11 in x 9 in. After an introductory research, it was quickly realized this would not be a realizable criteria. Below is a brief description of the available systems that would fulfill such a requirement.

Model	Workspace	Cost
Sensible Premium 3.0	33" x 23" x 16"	\$60,500
Haption Virtuose	42.5" x 35.5" x 23.6"	\$112,000
Sensible PHANTOM 3.0	33"x 23" x 16"	\$79,000
Force Dimension	14.2" x 14.2" x 12"	\$28,000

*Table 2. Haptic Device Comparison*

Immersion, another independent firm that develops haptic devices, provides a glove that would give a workspace of approximately 18" in each direction, although the glove weighs nearly 20lbs and costs \$54,000. This was also eliminated due to both cost and weight criteria violations.



*Figure 10. Sensable Phantom Omni*

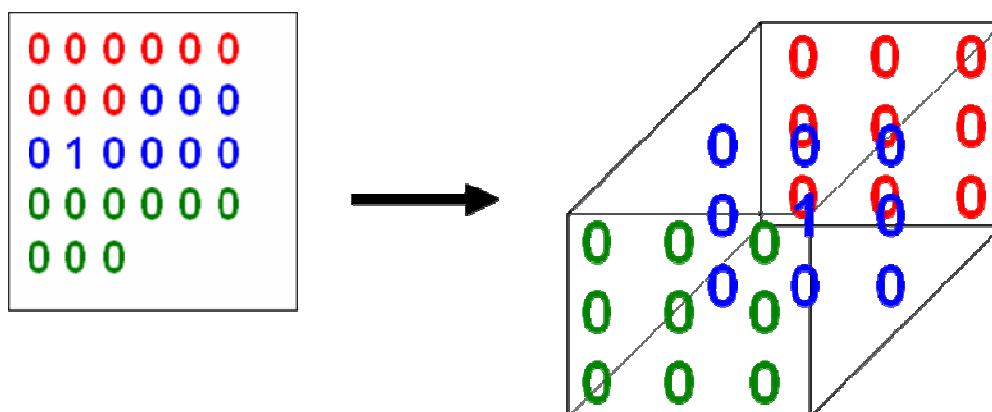
### Conceptual Development Program

To begin the development of the large haptics program, a conceptual development program was designed as a building block for the larger program. At first it was thought that an orange could be used as the development program object. The simple spherical shape and layers would serve as fundamental aspects to focus on for the program. Thus, the team also performed a CT Scan on an orange. This CT model would serve as a simple object for the purpose of rapid, iterative development.

### Test Development Program

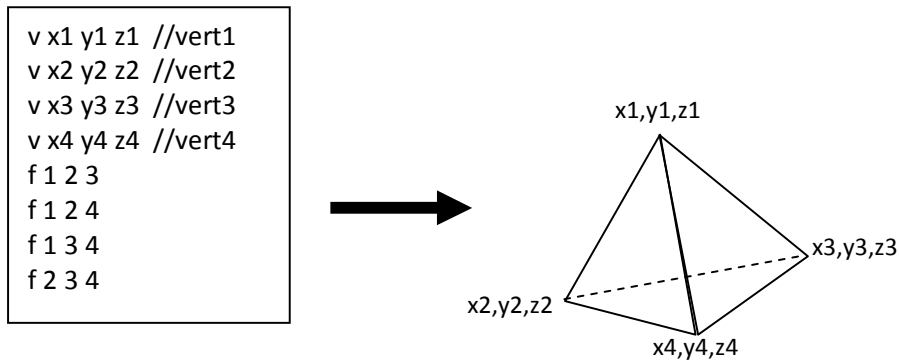
The test program consists of a dog object rendered in a three dimensional haptics environment. This environment has a cursor in the shape of a sphere which represents the current position of the point of the haptics device in relation to the dog. The haptics device is a Sensable PHANTOM Omni. This device is connected to our Computer through a firewire port and acts as a three dimensional cursor. To begin development, only the tools provided by Sensable were available. Along with the drivers needed to operate the Omni, Sensable sold a software package called the OpenHaptics Toolkit. In this toolkit were various examples (both source code and compiled executables) through which the Omni device could become familiarized. These examples consisted of modules such as: CannedForceEffect, Events, and DeformableSurface. The module that showed the most promise was one named HapticViewer. The HapticViewer example creates a 3D Haptics Environment and allows the user to import and interact with a Wavefront \*.obj file.

The CT scan data we received was a file in the \*.am format. This \*.am file is an Amira Mesh file which is essentially a list of integers which describes a three dimensional image. Each different integer in the file describes a different material. In the orange.am file, the different integers from 1 to 3 describing the peel, the inside of the peel, and pins (0 represents empty space). For example, the following file describes a single black point in a 3x3 Amira Mesh of white points (each integer in the file is color coded with that same point in the diagram, these colors do not appear in the \*.am file):



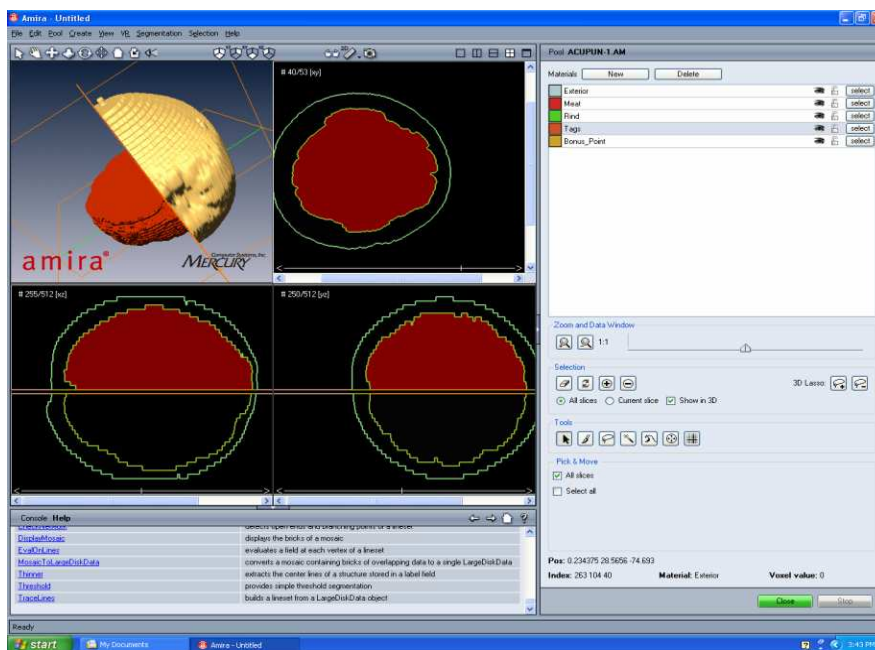
*Figure 11. Conceptual AMIRA Mesh (\*.am) File*

The Wavefront \*.obj file is significantly different. It represents a 3D object through describing surface triangles. These triangles are made up of vertices and faces. In a Wavefront \*.obj file, the vertices are listed first and then faces comprised by the index values of those vertices. The following example describes a 4-faced pyramid as a Wavefront \*.obj file:



**Figure 12. Conceptual Wavefront Object (.obj) File**

To convert our orange.am file into the orange.obj file, we used a software tool developed by Mercury Computer Systems, Inc. called Amira 3D. Amira 3D allows the user to generate a 3D surface out of a 3D mesh and export that surface as an \*.obj file. Now that we had the orange.obj file, we used the HapticViewer example from the OpenHaptics Toolkit to render it in the 3D Haptic Environment. At this point we had a virtual orange, floating in three space, which we could “touch” with the Omni.

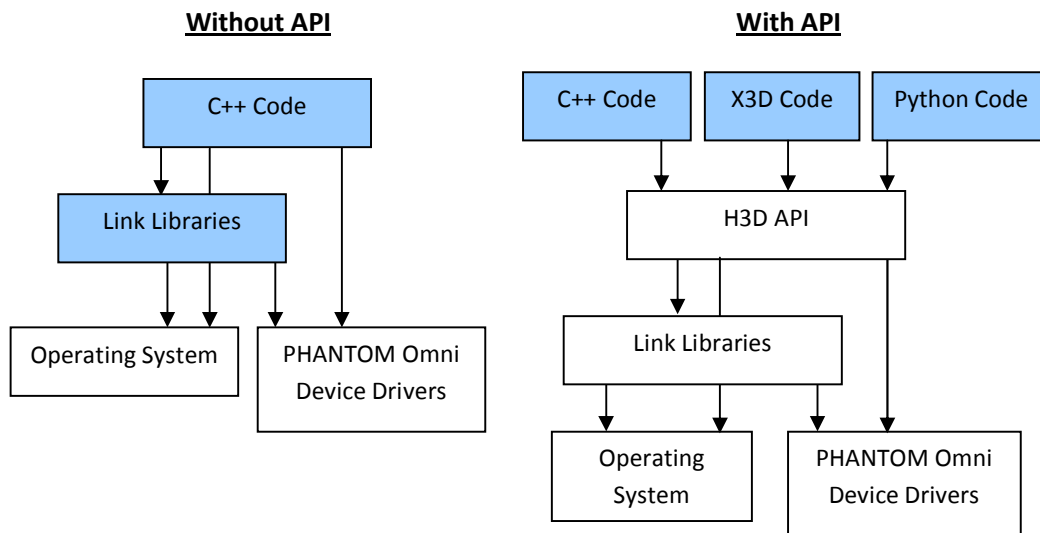


**Figure 13. AMIRA Screenshot**

From there, the next objective was to build HapticViewer from source instead of using the supplied, precompiled executable. This step was required because the HapticViewer could not be changed in any way (like adding forced feedback or event handling) without adjusting its source code. To do this the C++ files were imported into Microsoft Visual Studio 2005. This method of development proved to be tedious. The supplied code would not compile on the system because of various dynamic library linking problems. Visual Studio could compile the supplied C++ code files, but would err when it tried to attach those files to the system files used by the Omni advice.

After a lot of troubleshooting, it was determined that an abstraction layer between the developed code and the Omni System was needed. An abstraction layer, also known as an API (Application

Programming Interface), allows a programmer to speed up his development. The API deals with all the system level problems so that the programmer doesn't have to. It creates an environment that the programmer can program on top of instead of interfacing directly with the operating system and device drivers. The best API we could find was available from H3D.org and was called H3D API. The following diagram illustrates the differences between the two programming methods (the layers shaded in blue are the ones that the development programmer has to write):



**Figure 14. Haptic Programming API Comparison**

The API provides many benefits, the main being the development speed. With the API, the programmer only needs to program for a single interface, instead of having to deal with the link libraries, operating system and Omni device separately. This also makes the end program more portable (ability to move between different operating environments; such as Windows, Mac OS, UNIX, etc) because the program runs on the API, not on the OS directly. In this particular case, the H3D API gives us the added benefit of allowing us to code in three different programming languages. Python code benefits the programmer because it is known for its quick development cycles of small code segments that are easy to read and very powerful. C++ is the most popular programming language and is known for its robustness. C++ allows us to access many code libraries unavailable in other languages. X3D is a XML based language developed by H3D.org and gives the H3D API the ability to utilize fast, object oriented programming.

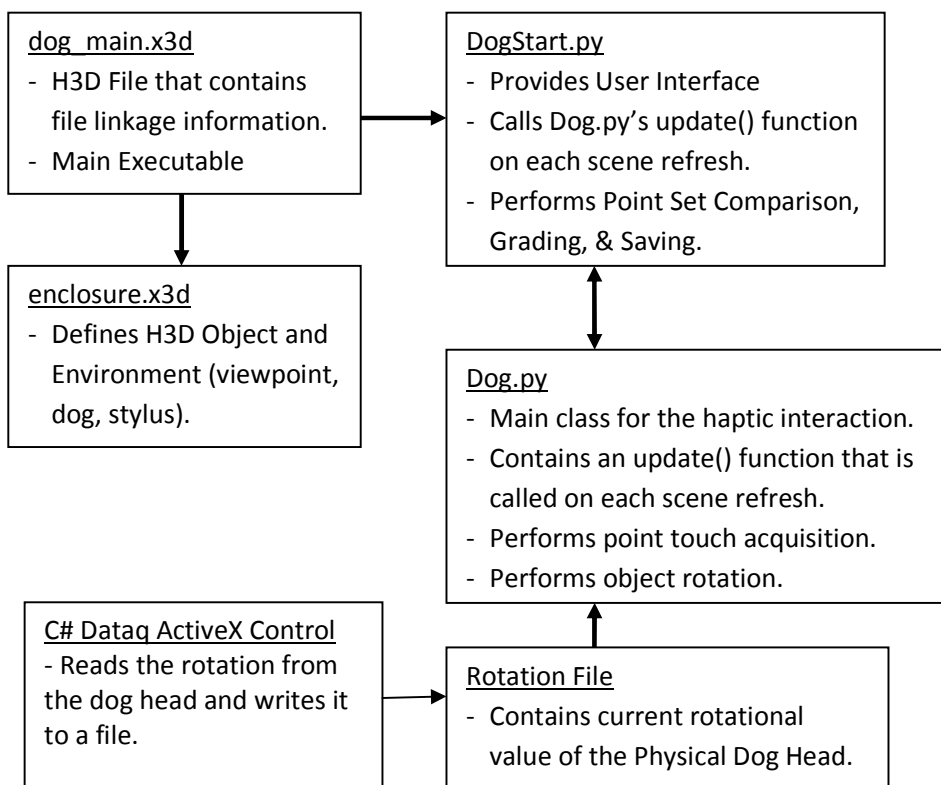
The H3D API functions primarily through x3d files. In an x3d file, objects (known as Nodes) can be created. These Nodes have properties known as Fields which can change dynamically during runtime. The third and final component in an x3d file is a Route. A Route is a way to define dependency between Nodes. For example, a Route from the SurfaceColor Field in Node A to the DetectIfRed Field in Node B can be created. Each time the SurfaceColor Field in Node A changes, Node B will be notified. Node B will then watch as Node A's SurfaceColor changes and indicate somehow (display a message onscreen) when Node A's SurfaceColor becomes red. Nodes can be grouped and abstracted into different files. This allows us to create objects in C++ or python and reference them in the x3d file. X3d files can also reference media files such as sounds or images. It can then use these media files in the program (ex: map an image of fish scales onto a 3D fish object or play a bubble noise when the Omni touches the fish). Thus far, the orange.obj file was converted to an orange.x3d and the dog.obj file converted to a dog.x3d file (a simple copy-paste and formatting change).

An issue arose with the H3D API and these files. The loader will load and render them but, when the user attempts to touch the rendered object with the stylus, the loader begins to throw

HL\_OUT\_OF\_MEMORY exceptions. The reason for this is that the loader only allocates enough memory space for 65,000 vertices when it starts. Since both the orange and dog objects have many more vertices than was allocated, the program attempts to perform an emergency allocation during run time. While, in theory, this is a good solution, the huge number of vertices overwhelms this emergency allocation. In order to continue the test program development, a smaller model was selected. This smaller model was included in the H3D API examples. The fish.x3d file contains approximately three thousand vertices, and therefore, operates correctly. After, consulting with the support services for the H3D API, the team decided to try a different set of drivers. This new driver set was an upgrade provided by Sensible. After changing to these new drivers, the HL\_OUT\_OF\_MEMORY error was eliminated and development could continue.

After addressing the memory allocation issues, the progress has continued on many other segments of the test program. These segments include: detecting Omni collision with the rendered object, evaluating/saving this collision data and rotating the dog object onscreen based on the incoming signal from the Base Unit. A python script was used to detect these collisions. In order for the collision detection to work, a route was setup between the dog object and the python script. Upon a touch of the stylus upon the dog, the dog object sends the point to the python script. The python script then parses this point and outputs the point to the command line (or to a file). This process occurs each time the dog is touched.

Ideally, the test program should be as modular and object oriented as possible. It should have the ability to run the acupuncture simulation regardless of the model that it is fed. As such, the program structure was redesigned based upon one of the example programs provided by the H3D API. The new program structure is as follows:



**Figure 15. Current SimPooch Test Program Structure**

The dog\_main.x3d file is the file that a student executes to start the test program. It contains linking information that initializes and connects the rest of the files to each other. dog\_main.x3d calls a python program named DogStart.py. This program instantiates a Dog.py object and performs all the user interface functionality. For instance, this is the program that prompts the user for their user type (student or teacher) and then grades their output after they have finished touching the dog. Dog.py is a python class that contains all the functionality of the three dimensional haptics environment. This program handles touch identification & acquisition, button presses, mannequin rotation and object positioning. enclosure.x3d is the file that contains all the object data such as the needles size and shape, the dog's triangle mesh, the viewpoint object (the camera), gravity specifications and a directional light object.

## E. Point Data Analysis

The team explored several ideas in deciding how to grade a student's acupuncture performance. After creating a 3D map of the dog, we needed a correct set of surface acupuncture points. For development purposes we chose to generate these points randomly but in the project's final implementation, we will have the teacher place points on the dog. The program will save these teacher points for use by a student. Finally, we also had to simulate student touches upon the dog. We also generated these student points randomly. The specific task of this area of the Virtual system is to create an algorithm with which we can grade the student's touches against those of the teacher.

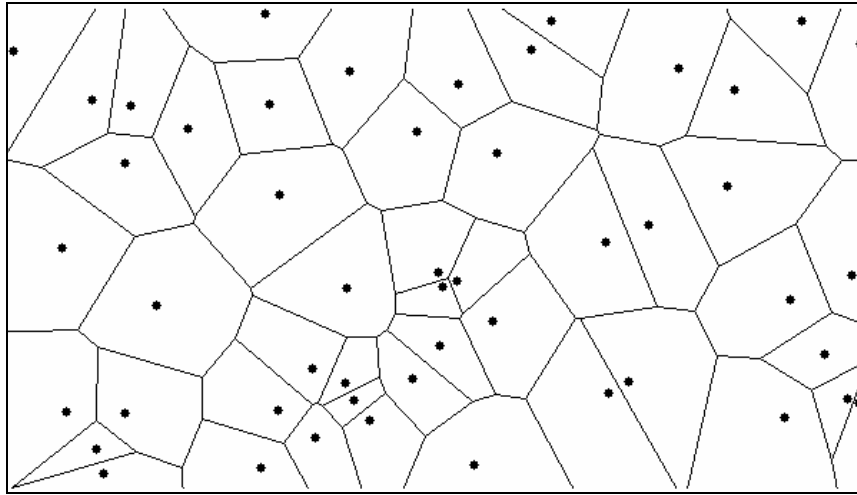
First we explored what is known as the *Closest Point Algorithm*. We quickly found that it was not practical to convert this procedure into what we needed. Next, we researched the *Voronoi Algorithm*. This algorithm creates a regionalized map of the teacher's points so that when a student touches the dog, we can know exactly at which teacher point the student was aiming and how close the student came to that teacher point. Although this works correctly, it became painfully clear that this algorithm was needlessly complicated. It requires a second algorithm called the *Steepest Descent Algorithm* to resolve the issue of associating the student points to the region boundaries. Also, we had trouble expanding this code from two dimensions to three dimensions. Finally, we settled on a brute force approach to this problem because there were relatively few points for a computer to check one by one. This simpler algorithm will work in most situations. There is a particular situation in which this algorithm has certain shortcomings.

Python is a free, open source, object oriented programming language. The team decided to use this language because it is compatible with our haptic device and because Python can easily "talk" to other programming languages.

The *Closest Point in a Set Algorithm* works with using a divide and conquer method. It starts with a set  $S$  with  $n$  points in a plane. The algorithm's goal is to find the two closest points,  $p_1 = (x_1, y_1)$  and  $p_2 = (x_2, y_2)$  such that it can deduce at which teacher point the student was most likely aiming. Then, its goal is to find the distance  $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$  such that  $d$  is the minimum among all pairs of points in  $S$ . To accomplish this, we wrote a program that sorted the points along the x-axis and split the points into two equal subsets. This gives a left set and a right set of points. Then the program finds the minimal distances for each side  $d_l$  and  $d_r$ . Then, it computes the minimum distance  $\delta$  between  $S_l$  and  $S_r$ . Thus, the closest points are those whose distance is the minimum between  $\delta$ ,  $d_l$  and  $d_r$ . This was good for finding the two points that were the closest to each other but it was not good for looking at each of the teacher points and each of the student points and comparing them for grading. This led us to the *Voronoi Algorithm*.

The *Voronoi Algorithm* is used to create a Voronoi Diagram. This diagram is a decomposition of space that is determined by the distances to a discrete set of points. The partitioning of a plane with  $n$  points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than to any other generating point. A point location data structure can be built on top of the Voronoi diagram in order to answer nearest neighbor queries. The program looks for the nearest neighbor where a student attempts to hit a teacher point. With this

algorithm, we can not only determine which point the student was aiming for by marking if it's in the region, but also know for sure if a student missed a region altogether.



*Figure 16. Voronoi Map with Regions*

For the aforementioned reasons, the *Voronoi Algorithm* seemed to be the perfect solution to our problem. However, this way of approaching our grading system turned out to overkill the problem and it was very difficult to adapt the algorithm from two to three dimensions. This adaptation would make all the polygons in the two dimensional Voronoi map into polyhedrons.

Since we are building a simulation that can be used and changed by a teacher, we can consider the generating points random. For this reason, we had to find an algorithm that could determine the region the student point was in and calculate the distance to the teacher point. By our programming methods, the generation point was zero distance to itself and the Voronoi region got “steeper and steeper” as distances increased away from the generation point. This would occur until the program finds a boundary and then, it would “descend” back down towards a new generation point. All generation points are created by the teacher. Although this method works, it became too time consuming to manage the code when going from our 2D working model to 3D. To solve this problem we scrapped this method and went for a strait forward way of grading the student that has a small, expectable flaw.

The problem is that, as we go through each of the teacher points one by one with the student, we append the array that shows which point was hit. At the same time, we remove this teacher point from the available teacher point search list. In the end, we will know if there are still teacher points that were not hit, i.e. not removed from the available teacher list. This is a problem because there could arise a situation in which a teacher point was removed on accident. This case would be rare however because the student attempts will be in distances much closer to each teacher point than the teacher points are to each other.

Finally, the code saves the student points as they go through the simulation. Then, the code prints out the coordinates of the teacher point at which the student was aiming, the coordinates of the student attempt at the point, and the distance between the two for comparison. In our case, Dr. Narda Robinson gave us a threshold of 5 mm. An acupuncture puncture within this threshold is defined as acceptable. Overall, we have a successfully working virtual simulation.

## VI. Conclusions and Future Development

SimPooch is currently in its 3<sup>rd</sup> year of development, and much of the work that lies ahead is outlined in this continuation report. There are five areas that need work done: Physical/Virtual Synchronization, Usability, and Physical/Virtual Expansion.

**Physical Synchronization** will be one of the final steps in assimilating all of the pieces together in a coherent prototype. We have to first mount the haptics device to the base unit. This can be accomplished with dowels attached to the bottom of the haptics and holes drilled into the base unit so the haptics device can be pegged in, and would also be removable. The position of the mount would have to be located such that the stylus could reach the ~70 points on the head.

**Virtual Synchronization** is a necessary aspect to join our virtual model with our physical model. We will implement this in the H3D files by modifying the viewpoint and scaling parameters. The head need to be scaled slightly larger so that it contacts simultaneously with the physical model. This brings us to the marriage of these ideas and the final design obstacle: calibration. We will need to make physical and virtual calibration points, fixed points that will align the physical and virtual model. This can be done by constructing a physical stop for the rotation of the dog's head, at the same time programming an initial rotation value. We will need to do the same for the needle position; however the physical calibration point is built into the haptics device.

**Usability** will eventually become important as the prototype is used and tested by users not on the SimPooch team. We would like to start the entire process with the executable H3DLoad.exe and have all of the required operational files be loaded in a batch sequence. A graphical user interface would be programmed to function with all of our modules and in python. Currently pyGTK looks promising as it is compatible with the python language. The GUI would output colored markers to show the points touched while the simulation is running, and when the simulation is completed would show the teacher points and calculate grading based on the voronoi algorithm. See the "UI Considerations Report" for a more in-depth discussion on the graphical user interface, grading, and the voronoi algorithm.

**Physical Expansion** would be accomplished with the assistance of a mechanical engineer. We would like to experiment with a spring loaded needle tip to the haptics stylus. This would more accurately replicate the acupuncture needles used in the field. This spring tip and the physical mounting system and calibration device could more precisely be fabricated in the mechanical engineering workshops.

**Virtual Expansion** includes a number of future modifications that could be completed with the assistance of Dr. Narda and the Vet school. The physical model would be more accurately replicated in the virtual one, with nuances such as the force required to penetrate flesh; the location of bones, tissue, and muscle; and the precise location of acupuncture points in a three-dimensional environment. We would expand the grading process to include application of the proper force, angle of the needle with respect to the skin, depth of the needle, and movement of the needle after it has been inserted. All of these properties will have to be defined by Dr. Narda.

These goals presented offer plenty of opportunity to work in different areas of research, from the physical construction in the workshops, to the python programming, to the mathematical topology of the voronoi

grading system. New team members joining the project will be able to choose what they wish to work on and we will work as a functional team to accomplish these goals outlined above.

## Appendix A. Software Samples

### *dog\_main.x3d:*

```
<Group DEF="G" >1
  <!-- <Viewpoint DEF="VP" position="0 0 1.0" /> -->
  <Viewpoint DEF="VP" position="0 .15 -1" orientation = "0 1 0 3.14" />
  <PythonScript DEF="dog_py" url="dog.py" />
  <PythonScript DEF="dog_start" url="DogStart.py" >
    <Group USE="G" containerField="references" /></PythonScript>

  <KeySensor DEF = "keySensor" />
  <ROUTE fromNode="keySensor" fromField="keyPress" toNode="dog_start" toField="key_manager" />
</Group>
```

### *enclosure.x3d:*

```
<Group>

  <DirectionalLight DEF="LIGHT" intensity="0.2" direction="0 0 -1" />

  <PointLight DEF="POINT_LIGHT" ambientIntensity="0.2" location="0 0.2 -0.3"/>

  <ForceField DEF="GRAVITY"/>
  <ForceField DEF="FORCE"/>

  <!-- Needle -->

  <DynamicTransform DEF="NEEDLE_T">
    <Shape>
      <Appearance>
        <Material transparency="0.0" diffuseColor="1 0 0" />
      </Appearance>
      <Sphere radius = "0.01"/>
      <!--<Cylinder radius = "0.0025"/>-->
    </Shape>
  </DynamicTransform>

  <!-- DOG -->
  <DynamicTransform DEF="TX_DOG" position="0 0 0">
    <Transform DEF="DOG_TRANS" translation="0 0 0" scale="3 3 3">
      <Shape>
        <Appearance>
          <Material DEF="MAT_DOG" diffuseColor="0.54 .27 .07"/>
          <SmoothSurface />
        </Appearance>
        <IndexedTriangleSet DEF="DOG">
          <!--A lot of trinangles -->
          </IndexedTriangleSet>
        </Shape>
      </Transform>
    </DynamicTransform>

  </Group>
```

### *DogStart.py:*

```
from H3DInterface import *
import sys
import Dog
import time
import os
import math

g, = references.getValue()

# globals
positionX = -.09
positionY = .22
positionZ = .05
scale = 2.13
rotate = -1.9
print "\n\nWelcome to the SimPooch Acupuncture Simulator\n\n"

print "Select User Type\n 1. Student\n 2. Teacher\n\n"
#userType = raw_input("selection: ")
userType = '1'

if userType not in ['1','2']:
    print "Invalid User Type Selection, Exiting..."
    sys.exit(0)

# create a new Dog Game
dog = Dog.DogGame( g )

class KeyManager( AutoUpdate( SFString ) ):
    def __init__( self ):
        AutoUpdate( SFString ).__init__( self )
    def update( self, event):
        global positionX, positionY, positionZ, scale, rotate
        key = event.getValue()
        print "Key Pushed: " + key
        if key == "f":
            print "Front Point Defined"
            #global hdev
            #self.frontPoint = hdev.trackerPosition.getValue();
        elif key == "b":
            print "Back Point Defined"
            #self.backPoint = hdev.trackerPosition.getValue();
            #length = self.frontPoint.x - self.backPoint.x
            print "Length: "
            print length
        elif key == "a": #left
            print "Moving Left"
            positionX = positionX - .01
            print positionX
        elif key == "d": #right
            print "Moving Right"
            positionX = positionX + .01
            print positionX
        elif key == "w": #up
            print "Moving Up"
            positionY = positionY + .01
            print positionY
        elif key == "s": #down
            print "Moving Down"
            positionY = positionY - .01
            print positionY
        elif key == "t": #in
            print "Moving Int"
            positionZ = positionZ + .01
            print positionZ
        elif key == "g": #out
            print "Moving Out"
```

*DogStart.py (continued):*

```
positionZ = positionZ - .01
    print positionZ
elif key == "y":
    print "Scale Up"
    scale = scale + .01
    print scale
elif key == "h":
    print "Scale Down"
    scale = scale - .01
    print scale
elif key == "o":
    print "Rotate"
    rotate = rotate - .01
    print rotate
elif key == "p":
    print "Rotate"
    rotate = rotate + .01
    print rotate
dog.setPosition(positionX, positionY, positionZ)
dog.setScale(scale)
dog.setRotation(rotate)
return "a"
key_manager = KeyManager()

# start the game
dog.restartGame()

# update the spacetennis game once per scenegraph loop. traverseSG is called
# once per scenegraph loop when traversing the PythonScript node in the
# scene graph.
def traverseSG():
    done = dog.update()
    if done:
        pts = dog.getPoints()
        response = raw_input("Do You Really want to Quit? (y/n):")
        #print response
        if response == 'Y' or response == 'y':
            finish(userType,pts)
            #save(dog.getPoints())
            #print "EXITING..."
            #sys.exit(0)

def save(points):
    print "SAVING YOUR POINTS: "
    for p in points:
        print p
    timeleft = 5
    while timeleft > 0:
        print timeleft
        time.sleep(.5)
        timeleft -= 1

def dist(A,B):
    return math.sqrt((A[0]-B[0])**2+(A[1]-B[1])**2+(A[2]-B[2])**2)

def finish(user_type,points):

    #print points
    #print user_type

    if user_type == '1':
        #user is Student
```

### Dog.py:

```
from H3DInterface import *
import H3DUtills
import random
import math
import sys
import os.path
import thread, winsound

#bonustime=15

path = ""
# get the first haptics device and put it into the hdev variable
hdev = None
di = getActiveDeviceInfo()
if( di and len( di.device.getValue() ) > 0 ):
    hdev=di.device.getValue()[0]

# TimerCallback used for delayed events.
timer_callback = H3DUtills.TimerCallback()

sphere = createX3DNodeFromString( ""<Sphere radius="0.005" /> """)[0]

rotationOffset = 1.5

class MainButtonPressed( AutoUpdate( SFBool ) ):
    def __init__( self, parent ):
        AutoUpdate( SFBool ).__init__( self )
        self.game = parent

    def update( self, event ):
        b = event.getValue()
        self.game.mainButton = b
        print self.game.sqlOffset
        if b == 1:
            self.game.sqlOffset = self.game.sqlOffset + 0.05

        return b

class SecButtonPressed( AutoUpdate( SFBool ) ):
    def __init__( self, parent ):
        AutoUpdate( SFBool ).__init__( self )
        self.game = parent

    def update( self, event ):
        b = event.getValue()
        self.game.secButton = b
        print self.game.sqlOffset
        if b == 1:
            self.game.sqlOffset = self.game.sqlOffset - 0.05

        return b

# The main class for the game.
class DogGame:
    def __init__( self, parent):

        self.parent = parent

        self.positionX = 0
        self.positionY = 0
        self.positionZ = 0
        self.scale = 0
        self.rotate = 0

        self.mainButton = 0
        self.secButton = 0
        self.sqlOffset = 0 #- .45
```

*Dog.py (continued):*

```
self.main_button_pressed = MainButtonPressed( self )
    hdev.mainButton.route( self.main_button_pressed )

self.second_button_pressed = SecButtonPressed( self )
hdev.secondaryButton.route( self.second_button_pressed )

self.node, self.dn = createX3DNodeFromURL( path + "enclosure.x3d" )

#print type(parent)
#print type(self)
#self.key_manager = KeyManager( self )
#parent.actionKeyPress.route(self.key_manager)

self.dog_running = 0
self.last_touch_time = time.getValue()
self.deg = 0

#self.mysqlConn =
mysqlHelper.MysqlHelper({'user':'doguser','passwd':'doguser','db':'simpooch'})

self.sqlAdjustment = -2*math.pi/5.0

# make the node visible by adding it to the parent
self.parent.children.push_back( self.node )

# parameters for the game play
self.points_touched = []

print hdev.positionCalibration.getValue()
self.dn["DOG_TRANS"].rotation.getValue()
print self.dn["DOG_TRANS"].rotation.setValue(Rotation(1,0,0,-.3))

def updateScore( self, points ):
    #if not points == [10,10,10]:
    self.points_touched.append(points)

def update( self, s=None ):
    global rotationOffset
    if not self.dog_running:
        return

#print "Dev: ",hdev.deviceOrientation.getValue()
#print "Tra: ", hdev.trackerOrientation.getValue()

# ROTATION:

# get the current angle from the file

f = open("c:/out.txt")
self.deg = (float(f.readline()) - 1.03271) * self.sqlAdjustment + self.rotate
#print rotationOffset
f.close()

oldDeg = self.deg

# now rotate the dog:
sVal = self.dn["TX_DOG"].orientation.setValue(Rotation(0,1,0,self.deg))

# NEEDLE POSITIONING:

needlePos = self.dn["NEEDLE_T"].position.getValue()

#the needle follows the stylus
self.dn["NEEDLE_T"].position.setValue( hdev.trackerPosition.getValue() )
```

## Appendix B. Budget

During spring 2008 semester no money since most of the work required now is programming.

During the fall 2007 semester progress was largely in the form of computer programming, we ended up spending relatively little money. Python is an open source language; as is the H3D API. The computer and the haptics device were one time purchases that were incurred by previous projects.

Most of our budget was spent in purchasing items and hardware for the Base Unit. We purchased two different A/D Converters for testing purposes. All but \$47.76 were covered by the \$50 per senior design student allotment.

Initial Allotment	25,000.00
Fall 2006 - Spring 2007	-9,177.74
Fall 2007 - Spring 2008	-47.76
Python Programming Book	-32.00
Magnetic Encoder	-42.03
A/D Converter	-114.73
Base Unit Hardware	-9.00
A/D Converter	-50.00
Senior Design Student Allotment	200.00
<b>Total</b>	<b>15,774.50</b>

*Table 3. Spring 2008 Budget*

## References

- Berg, Mark de., et al. Computational Geometry: Algorithms and Applications. 2nd ed. Berlin; New York: Springer, 2000.
- Bradney, Tim., et al. Canine Acupuncture Simulator & Trainer 2006-2007. Colorado State University: Fort Collins, CO. 4 May 2007.
- DI-158 Series of Starter Kits. DATAQ Instruments. 2005.  
<<http://www.dataq.com/support/documentation/pdf/datasheets/158ds.pdf>>
- Johns, Derek. Sweep Applet. 2003 <[http://www.cs.magill.ca/~djohns26/Comp\\_644/AppletPage.htm](http://www.cs.magill.ca/~djohns26/Comp_644/AppletPage.htm)>
- Markus. "Re: large x3d object crashes H3DLoad." Online Posting. 15 Nov. 2007.  
<[http://www.h3dapi.org/modules/newbb/viewtopic.php?topic\\_id=527&forum=4](http://www.h3dapi.org/modules/newbb/viewtopic.php?topic_id=527&forum=4)>
- Maruch, Aahz, Maruch, Stef. Python for Dummies. New York: John Wiley & Sons Inc. 2006.
- OpenHaptics(TM) Toolkit Programmer's Guide, Version 2.0. Sensable Technologies. 2005.  
<[http://www.sensable.com/documents/documents/OpenHaptics\\_ProgGuide.pdf](http://www.sensable.com/documents/documents/OpenHaptics_ProgGuide.pdf)>
- Subrata Nandi, Advances in Algorithms, "Closest Pair Among a Point Set", 2006.
- Weisstein, Eric W. "Voronoi Diagram." From MathWorld--A Wolfram Web Resource.  
<<http://mathworld.wolfram.com/VoronoiDiagram.html>>
- van Rossum, Guido. Python Library Reference. Ed. Fred L. Drake, Jr. 19 Sept. 2006.  
<<http://docs.python.org/lib/lib.html>>

## Acknowledgements

The Spring 2008 SimPooch team would like to acknowledge the following individuals and organizations:

Dr. Narda Robinson – Project Sponsor  
Dr. Peter Young – Project Advisor  
Dr. Regina Schoenfeld-Tacher – Project Sponsor  
Olivera Notaros – ECE Senior Design Coordinator

The 2006-2007 SimPooch Team:

Matt Cain  
Renata Voorhees  
Brandon Nino  
Kelly Galloway  
Michelle Dummer  
Tim Bradney

The 2006-2007 Management Team:

Dr. Sue James  
Terry Precht

DATAQ Instruments

The Developers at H3D API.org and Markus at the H3D API Forums

Guido van Rossum and the Developers of Python

The Developers of MySQL

Mercury Systems, Inc. (provided AMIRA Trial Program)

Dr. Sue Kraft and Mrs. Betsy Sestina of the CSU Veterinary Hospital

Mr. Wes Womack