CUTESY NAME TBD (BUMM- BASED UPON MELODY MATCHER): A VISUAL ANALYSIS OF THE INTELLIGIBILITY OF SONG LYRICS USING A MUSIC-LINGUISTIC APPROACH

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jennifer "Jenee" Gayle Hughes

June 2012

COMMITTEE MEMBERSHIP

TITLE:	cutesy name tbd (BUMM- Based Upon Melody Matcher): A Visual Analysis of the Intelligibility of Song Lyrics Using a Music-Linguistic Approach
AUTHOR:	Jennifer ”Jenee” Gayle Hughes
DATE SUBMITTED:	June 2012
COMMITTEE CHAIR:	Zoë Wood, Ph.D.
COMMITTEE MEMBER:	Franz Kurfess, Ph.D.
COMMITTEE MEMBER:	John Clements, Ph.D.

Abstract

cutesy name tbd (BUMM- Based Upon Melody Matcher): A Visual Analysis of the Intelligibility of Song Lyrics Using a Music-Linguistic Approach

Jennifer ”Jenee” Gayle Hughes

In this project, we developed a visualization of the ambiguity of written prose. Given a textual phrase, our program determines all oronyms for that phrase, or possible ways that that phrase is likely to be misheard. It then creates an oronym parse-tree visualization, with each branch fork indicating that a phonetic sequence can be interpreted in more than one way. Each branch segment represents an orthographic word, and the branch radius is scaled to the word's frequency of use in everyday language.

Given any valid English phrase, our system will first generate all possible correct phonetic sequences for a General American accent. Then, it parses through these phonetic transcriptions depth-first, looking for valid orthographic words for each subsequent phonetic subsequence, generating full and partial phrases from these words. While it is doing so, a tree branch is generated on screen for each possible orthographic divergence. In the event that a branchs phonetic tail is not orthographically interpretable, we visually dead-end the branch by drawing a red sphere. In the event that the entire phonetic sequence can be parsed into a valid orthographic phrase, we indicate this successfully-found oronym with a green sphere.

This visual representation allows users to see how many ways a phrase can be interpreted, and most novely, where dead-end interpretations of the phrase's phonetic sequence exist. A particularly strong orthographic partial phrase before a phonetic dead-end can mislead a listener, causing them to lose track of the

words in the rest of the phrase.

Our visual representation does not take into account n-gram word-proximity, which causes the visual representation to incorrectly weight some branch paths. However, overall, it does a fairly good job of weighting the likelihood that a listener will follow an oronym branchs particular interpretation as they listen to a phrase.

In addition to implementing the visualization, we did a multi-phase user study, incorporating over 500 data points from over 157 test subjects. In it, we tested the validity of our oronym generation by having people read a oronym phrase aloud and send us the recording, and by having people transcribe pre-recorded oronyms. In the first phase, we generated oronym strings for the phrase a nice cold hour, and had over 15 people make approximately 65 recordings of the most common oronym phrases. We then compared their pronunciations to the pronunciations we were expecting, and found that in all cases, the recorded phrase's phonemics matched our expectations. In the second phase, we selected fifteen of the phase one recordings, and had over 30 different people transcribe each one. In the aggregated transcriptions, the most commonly transcribed phrases roughly corresponded with our metric for the most likely oronym interpretation of the phrase in the recording. Though we are happy with our findings, we believe that we could create even better likelihood metrics with the integration of n-grams, and would suggest this for future work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Human brains are built to come to single conclusions about things that have more than one interpretation. The way that you come to this end conclusion is dependent upon your experiences, cutural immersion, and language familiarity (I CAN TIE IN INDIA HERE!) [11]. WHen attempting to write English phrases that will be read aloud and heard by people with other linguistic biases than you, it's important to make your prose as deterministically understandable as possible. THe first towards this is understanding and identifying how many ways your stuff can be misheard, and why.

1.1 You and me...and Leslie?

In the song *"Groovin' (on a Sunday Afternoon)"*, by the Young Rascals, there's a part in the bridge that many people hear as *"Life would be ecstasy, you an' me an' Leslie"*. In fact, the line is *"Life would be ecstasy, you and me endlessly"*. The confusion lies with the last three syllables of the phrase. The pronunciation of each version, if spoken normally, is as follows:

Alphabetic:	and Les- lie	end- less- ly
SAMPA:	@nd “lEs li	“End l@s li

In the song, the singer is doing what many singers are taught to do, to make it easier to sustain the singing of words that end with unsingable consonants: the unsingable consonant is displaced onto the front of the next word. In this case, the consonant “d” is not singable, so he displaces it onto the next syllable, when he can: “and ME” becomes “an dME”, and “end LESS” becomes “en dLESS”.

Basically, singers are *born* to ignore syllable boundries. So, our singer can effectively think of the sung phrase as:

YOU an dME en dLESS lee

This doesn’t cause confusion for listeners, because they’re used to hearing it. This does mean, however, that lyric placement does not provide an accurate barometer to a listener of where a word actually ends.

In addition, the singer is singing fudging his vowels, like singers are taught to do, so “and” and “end” sound almost indistinguishable. So, really, what listeners are hearing is this:

YOU en dME en dLESS lee

Now, the listener’s brain has to take this syllabic gobbledy-gook, and parse it into something useful. They’ve currently got this mess to deal with (represented in SAMPA syllables):

ju *En* ***dmi*** *En* ***dl@s*** *li*

They parse the first part just fine, because the emphases match:

you and **me** *En* ***dl@s*** *li*

But no one says endLESSly. People say ENDlessly. So, the listeners don't recognize it. They have to work with what they have. They already turned one "En d" into an "and", so they do it again:

you and **me** and ***l@s*** *li*

Now, they're just left with LESS lee. And that fits Leslie, a proper noun that fits in context and in emphasis placement. So, the final heard lyric is:

you and **me** and **Les-** lie

The misunderstanding can be traced back to improper emphasis placement. The songwriter probably didn't even think of that, and now he's stuck: a one-hit-wonder with a misunderstood song. We bet that in interview after interview, someone asks him who Leslie is. It's probably very frustrating — especially since he could have just moved the word an eight note later, and it would have been understood perfectly.

That's the sort of situation this program is going to help avoid.

1.2 Why it breaks down

Imagine the case where a jingle-writing company must market a

There are a ton of places where an author must speak to their intended audience through an intermediary source, with whom they may have little to no contact. When this is the case, it is of the utmost importance to ensure that their written text is as deterministic as possible.

There are two points at which the author's intendeded phrasing can be muddled" : First, when the author's orthographic text becomes an orator's spoken (phonemic) interpretation, and second, when the orator's phonemic interpretation is translated by an audience into a perceived orthographic phrase. Both of these conversions/translations/transmutations/ICAN"T REMEMBER THE WORK must be made succesfully in order for the author's intended meaning to be conveyed.

The phrase "iced ink" undisputedly succeeds in the first translation, but fails on the second. Iced ink can only be pronouced one way, but it can be heard multiple ways–the most notable of which is "I stink", not "iced ink".

The phrase "a nice cold hour" can fail on both parts. First, the orator could have accidentally-capitalized the word Nice, and made it sound like Nice, the city in Rome. An audience would likely hear this as "niece", and would be confused, at best. Even if the orator pronouces the phrase as the author intended, they audience could hear multiple orthographic phrases in the same phonemic sequence: "a nice cold hour", "an ice cold hour", or even "a nigh scold our".

A third, more rare and nefarious type of audience misunderstanding can be caused by parse-tree misdirection, where an audience member is absolutely sure they're hearing one phrase, only to get lost halfway through the lyric

1.3 Intended audience

1. Playwrights

2. Speechwriters

3. Jingle writers

4. Lyricists

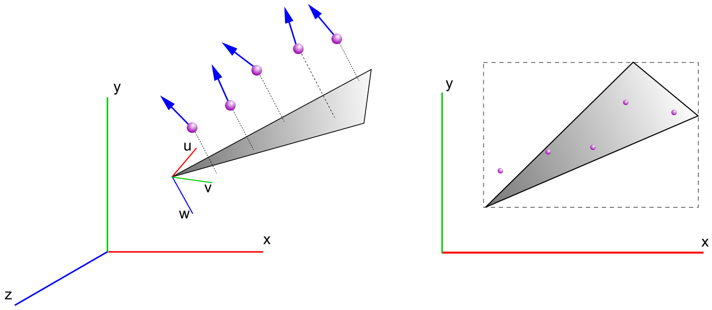

Figure 1.1: This is an example figure. This graphic was drawn in InkScape (http://www.inkscape.org/), a free vector drawing application. Figures can be exported from InkScape as PDF images or as EPS (depending upon if you want LaTeX to generate a PDF or a PostScript document.)

Chapter 2

Background

2.1 Mondegreens

2.2 Oronyms

2.3 Types of Phrase Representations

There are two different phrase representations that we deal with in our project. The first, orthographic, is written word. Consists of letters from the alphabet, and you've been learning it since you started to read. Its letters do not tend to directly corrolate to a single sound, and that is why we have the second phonemic representation of a phrase

A phonemic representation consists of a string of phonemes, which are the smallest possible sound fragments that still carry meaning. Each phoneme correlates directly to a single phonetic sound.

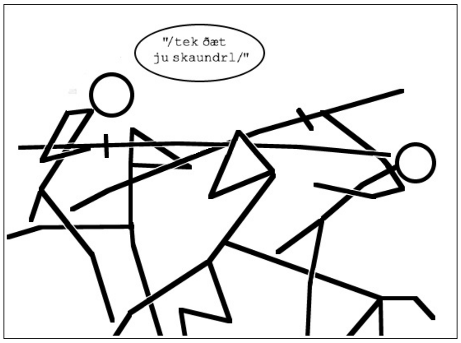

Figure 2.1: Level of sound specificity provided by phonemes[5]

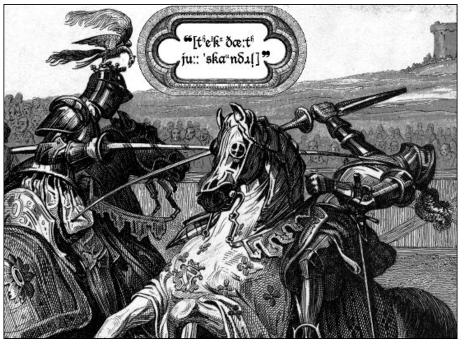

Figure 2.2: Level of sound specificity provided by phones[6]

2.3.1 Phonetic vs Phonemic

Though they are often used interchangably, the words ”phonemic” and ”phonetic” (and their corresponding sound building blocks, ”phone” and ”phoneme”) indicate a different stages of sound parsing. Phonemes are idealized sounds; phones are the actual sounds that come out of a person’s mouth. Figures 2.1 and 2.2 provide a illustrative metaphor of the difference.

Phonology only deals with idealized sound fragments in a vacuum; phonetics deals with the way that those sound fragments are delivered, taking into account the intonation and the separation of words and syllables. We primarily deal with phonemes only in our project, though we take into account some context in the form of word familiarity.

2.4 Phonemes

As we stated in Section 2.3, phonemes are the atomic building blocks of words. Think of phonemes as an alphabet where every sound has its own letter.

For example, the phonemic spelling of the name ”Jenee” is dZe

See 7 for a full table of each phoneme, its description, and its sub-parts.

2.5 Phonetic Alphabet

To discover oronyms for a phrase, we must first to translate our phrase from a series of orthographic words to a sequence of phonemes. These phonemes can be represented by several phonetic alphabets. The most common phonemic alphabet is the IPA (Internation Phonetic Alphabet).

2.6 Phonetic Dictionary

The UNISYN dictionary is used primarily to phonetically translate words into multiple accents. It has its own formated dictionary, with a bunch of wildcards in it. They also provide some semi-functioning perl scripts that allow you to specify a dialect youd like to use (For example, I would say cooking differently than someone from the Deep South, and we both would say it differently than someone from London. However, were all speaking English. The UNISYN dictionary facilitates this translation).

$$\begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix}$$

Figure 2.3: This is a sample matrix equation.

2.7 This is a new section

LaTeX is a document markup language and document preparation system for the TeX typesetting program.

It is widely used by mathematicians, scientists, philosophers, engineers, and scholars in academia and the commercial world, and by others as a primary or intermediate format (e.g. translating DocBook and other XML-based formats to PDF) because of the quality of typesetting achievable by TeX. It offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout and bibliographies.

LaTeX is intended to provide a high-level language to access the power of TeX. LaTeX essentially comprises a collection of TeX macros, and a program to process LaTeX documents. Since TeX's formatting commands are very low-level, it is usually much simpler for end-users to use LaTeX.

LaTeX was originally written in 1984 by Leslie Lamport at SRI International and has become the dominant method for using TeXfew people write in plain TeX anymore.

LaTeX is based on the idea that authors should be able to focus on the meaning of what they are writing, without being distracted by the visual presentation of the information. In preparing a LaTeX document, the author specifies the logical structure using familiar concepts such as chapter, section, table, figure, etc., and lets the LaTeX system worry about the presentation of these structures. It therefore encourages the separation of layout from content, while still allowing manual typesetting adjustments where needed. This is similar to the mechanism by which many word processors allow styles to be defined globally for an entire document, or the CSS mechanism used by HTML.

1. Here is a list item.

    (a) Here is a sub list item.

        i. Here is a sub sub list item.

Chapter 3

Implementation

This program has three major functional parts: a custom phonetic dictionary, a command-line oronym generator, and a OpenGL oronym-parse-tree visualization generator.

3.1 Customized Phonetic Dictionary

In order to discover oronyms for each phrase, we first needed to determine how each phrase is pronounced. Pronunciation can vary depending on the speakers accent, so it was important for us to (1) chose an accent that we could easily replicate and (2) find a dictionary that supported that accent.

We decided to utilize a General American accent, due to its ubiquity in media and news sources. The General American accent, also known as the "Standard American English" dialect, is not spoken by the majority of people in America, but is used as a sort of "average accent". It most closely resembles the Midwestern accent using in the area in Figure **??** and more commonly recognized as "the

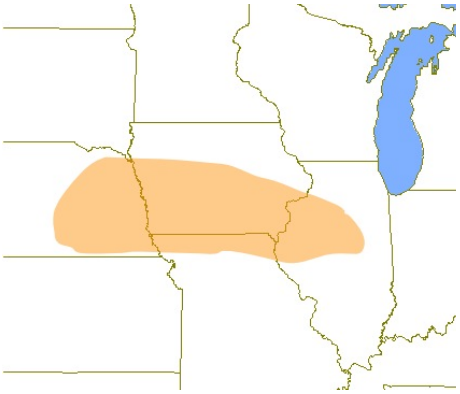

Figure 3.1: This is the geographic area whose accent most closely resembles the General American Accent [4]

newscaster accent". Newscasters learn this accent for national TV, because it is the "least-accented" of the American accents.

3.1.1 Dictionary Options

We considered using three different phonetic dictionaries: the CMU dictionary, LC-STAR dictionary and UNISYN dictionary[7] [1] [10] [8]. We started out by looking at the LC-STAR dictionary, but quickly decided that it wasnt going to be as useful to us, because the LC-Star project is relatively focused on Speech-to-Speech or Text-to-Speech tech. Also, the website had not been maintained since 2006. We then tried the CMU dictionary, which, for a while, seemed like it was going to work. It had a very simple way of encoding words: first the word, then the identifier number in parens (if needed), then a space, then a one-to-two char code for each sound in the word, with the numbers 0, 1, 2 appended to indicate emphasis (if needed), separated by spaces. An example of a CMU dictionary entry can be seen in Figure 3.2.

```
ABBREVIATE  AH0 B R IY1 V IY0 EY2 T
```

Figure 3.2: Here is the CMU dictionary entry for the word ”abbreviate”

The problem that arose with this format, was that there was no explicit definition of where to hyphenate the word when splitting it up. This causes problems if I want to use the word in lyrics, where each note has is own syllable underneath it, and each syllable might have many different sounds. The benefits of the CMU dictionary over some other dictionaries were that (1) it was actively maintained, (2) it included proper nouns, which are often found in lyrics, but not in dictionaries, and (3) it was ridiculously easy to read.

The downsides were that (1) it included no part of speech data or hyphenation data, and (2) it used non-standard symbols for its phonetic alphabet. With the downsides and benefits in mind, the CMU dictionary could not be used in isolation, especially if I someday want to attempt generation of original lyrics (which part of speech data would be vital for).

The UNISYN dictionary is used primarily to phonetically translate words into multiple accents. It has its own formated dictionary, with a bunch of wildcards in it. They also provide some semi-functioning perl scripts that allow you to specify a dialect youd like to use (For example, I would say cooking differently than someone from the Deep South, and we both would say it differently than someone from London. However, were all speaking English. The UNISYN dictionary facilitates this translation).

It had all the information I needed, and then some. However, it was case-insensitve, meaning that it didn’t make it easy to differenciate pronunciations for some words. For example, the word ”nice” is pronounced differently from the city "Nice", but they were both stored as "nice" the orthography of UNISYN. The CMU dictionary did. The obvious conclusion, then, was to combine them. I decided that anything that was in the CMU dictionary, but not in the UNISYN dictionary, would be counted as a proper noun. And with that, I started to massage both data sets into mergable forms.

However, I ran into a setback, mentioned in the very first article I found references to both dictionaries in: the dictionaries were inconsistent. They didnt always put stresses in the same place, nor did they always have the same pronunciation. Because of this, it was difficult to match wordsespecially words that were homographic heteronyms (same writing, different sounds, like "Do you know what a buck *does* to *does*?"). Because of this, we decided to use the UNISYN dictionary exclusively.

3.1.2 Formatting the dictionary

We used the UNISYN dictionary, using General American SAMPA output, as the base for our phonetic dictionary.

First, we downloaded the dictionary and related files from the UNISYN website.

Once we'd extracted them, we attempted to run the provided perl scripts to turn the UNISYN meta-phonetic symbols into SAMPA phonetic symbols, using the procedure specified in Section 6.1 of the UNISYN Documentation, and the towncode gam (which indicated that we want pronunciations using the General AMerican accent)[9].

There were a few errors in translation we fixed manually:

- The translation script had erroneously substituted the character *d* instead of the digit *4* , which primarily affected the unique identifiers to disambiguate words and the word-frequency counts. We did a simple search-and-replace to fix this, excluding the orthographic, extended orthographic, and SAMPA spellings.

- The translation script had erroneously substituted the digit *5* instead of the character *l* (lower case L). This interfered with the phonetic/sampa spellings of the words. Fortunately, 5 is not a valid SAMPA phoneme or orthographic letter value, so we removed all instances of it from the ortho, SAMPAspelling, and extendedOrtho columns

- The translation script had erroneously substituted the character sequence *‘r*\ instead of *‘r* . We were able to fix this by simply removing all \characters from our output.

Here is the format for the raw fields, after we were done with fixing the UNISYN output:

<ortho> : **<uniqueID>** : **<partOfSpeech>** : **<SAMPAspelling>** : **<SAMPAnoEmph>** : **<extendedOrtho>** : **<freq>**

Example: transfer:2:VB/VBP: trns”f3‘r :trans==fer:7184

<ortho> is the regular spelling of the word

<uniqueID> is a number (and optional string) used to differentiate homographs.

<partOfSpeech> is used to identify the specific part of speech

<SAMPAspelling> is the breakdown of the word, phonetically. It uses the

SAMPA alphabet, and separators to show where breaks in the word are, and how theyre emphasized. If a separator is ' $ ', the following phones (until the next separator) are not emphasized. If it's ' % ', then they are the secondary emphasis. If it's ' " ', then they are the primary emphasis.

<SAMPAnoEmph> is the same as *¡SAMPASpelling¿*, but with all emphasis characters stripped out. We chose to add this field so that we could more-easily look up phonetic sequence matches.

<extendedOrtho> has no particular use to me, at this time. It indicates the morphological breakdown of the word, allowing for stemming analysis of words. Since we arent convering that, we never use this field.

<freq> is the frequency at which the word occurs in language, according to UNISYN. The frequency count is "taken from a composite of a number of on-line sources of word-frequency. It includes frequencies from the British National Corpus and Maptask, and frequencies derived from Time articles and on-line texts such as Gutenberg. They were weighted to give more importance to sources of spoken speech, and also to increase the numeric frequency of smaller corpuses"[9].

3.2 Oronym Generation

3.2.1 Step 1: Get all phonetic sequences for a given orthographic phrase

The first step to discovering the oronyms for the phrase is to determine all the possible pronunciations. The pseudocode to do this is in 3.2.2.

```
vector< vector < phone > >findAllPhoneSeqsForOrthoPhrase( string orthoPhrase) {
  vector< vector < phone > > allFullPhrasePhoneSeqs
  vector<string> orthoWords = split(orthoPhrase);

  int origNumFullPhrases = 0;
  for (  int i = 0 to orthoWords.size() ) {
    string orthoWord = orthoWords[i] ;
    Vector<vector <phone > > nextWordSampaPhoneSeqs (orthoWord)

    if ( this is the first orthoWord weve looked up this loop ) {
      for( vector<phone> phoneSubSeq: nextWordSampaPhoneSeqs ) {
        allFullPhrasePhoneSeqs[i].append(phoneSubSeq);
      }
    } else {
      origNumFullPhrases = allFullPhrasePhoneSeqs.size();
      if theres more than one vector <phone> in nextWordSampaPhoneSeqs
        then we need to create duplicates of all existing allFullPhrasePhoneSeqs
    }
    for( int m = 0 to allPhrasePhoneSeqs.size() ) {
      int phraseToAppendIndex = m / origNumFullPhrases;
      vector<phone> phoneSeqToAppend = nextWordSampaPhoneSeqs[phraseToAppendIndex]
      append phoneSeqToAppend to the end of allFullPhrasePhoneSeqs[m].
    }
  }

  return allFullPhrasePhoneSeqs;
}
```

Figure 3.3: Algorithm to get all phonetic sequences for an orthographic phrase.

3.2.2 Text-based approach

3.3 Visual Representation

```
vector<string> discoverOronymsForPhrase( string origOrthoPhrase, bool includeDeade
   vector<string> orthoMisheardAsPhrases;
   vector<vector<phone> > allPhoneSeqsOfOrigPhrase = findAllPhoneSeqsForOrthoPhras

   int numUniquePhoneticInterpretations = allPhoneSeqsOfOrigPhrase.size();
   for(int i = 0; i < numUniquePhoneticInterpretations; i++) {
      vector<phone> curPhoneSeqWithEmph( allPhoneSeqsOfOrigPhrase.at(i) );
      string strOfCurPhoneSeq = phoneVectToString( curPhoneSeqWithEmph );

      //remove emphasis marking for easier lookups
      vector<phone> curPhoneSeq = getNoEmphsPhoneVect( curPhoneSeqWithEmph );

      vector<string> altOrthoPhrases = findOrthoStrsForPhoneSeq( curPhoneSeq );

      for( int j = 0; j < altOrthoPhrases.size(); j++) {
         string altOrthoPhrase = altOrthoPhrases.at(j);

         //ensure it contains valid ortho text in all cases, and if includeDeadend
         if ( ( includeDeadends == true && altOrthoPhrase != deadEndDelim1
                 && altOrthoPhrase != deadEndDelim2 )
            || ( altOrthoPhrase.find( deadEndDelim1 ) == string::npos
                 && altOrthoPhrase.find( deadEndDelim2 ) == string::npos ) ) {

            orthoMisheardAsPhrases.push_back( altOrthoPhrase );
         }
      }
   }
   //deduplicate orthoMisheardAsPhrases by putting in a set and back again
   set<string> tempSetForDeduplication( orthoMisheardAsPhrases.begin(), orthoMishe
   orthoMisheardAsPhrases.assign( tempSetForDeduplication.begin(), tempSetForDedup
   return orthoMisheardAsPhrases;
}
```

Figure 3.4: Algorithm to get all oronyms for an orthographic phrase.

```
void buildAndDrawFullTree( string orthoPhrase ) {
   vector< string > fullPhrases = discoverOronymsForPhrase( orthoPhrase , true );
   getMaxAndMinFreqForAllOrthoPhrases( fullPhrases, &maxWordFreq, &minWordFreq);


   //draw the tree's seed

   glPushMatrix();
   {
      glTranslated(0.0, -1.0* DEFAULT_BRANCH_LEN , 0.0);
      materials(GreenShiny);
      drawSphere(DEFAULT_RADIUS);
      materials(allMaterials.at( mat % allMaterials.size () ) );

      drawBranchesAtFork ( fullPhrases, DEFAULT_RADIUS );

   }
   glPopMatrix();
}
```

Figure 3.5: Given an orthographic phrase, this function prepares to draw the tree

```
void drawBranchesAtFork( vector< string > fullPhrases, double lastRadius) {

   if( fullPhrases.size() == 0 ) {
      return;
   }

   //use a set to ensure no duplicates
   set< string > firstWords;

   //put the first word of each phrase into the set
   for(int i = 0; i < fullPhrases.size(); i++){
      if( fullPhrases.at(i).size() > 0 ) {
         string firstWord = FirstWord( fullPhrases.at(i) );
         firstWords.insert( firstWord );
      }
   }

   //calculate positioning variables for the spread of branches for firstWord

   set<string>::iterator curFirstWordIter;
   int i = 0;

   //for each firstWord in the set
   for ( curFirstWordIter = firstWords.begin(); curFirstWordIter != firstWords.end
      string curFirstWord = *curFirstWordIter;

      //calculate the branch radius using the scaled frequency of curFirstWord
      int firstWordFreq = queryDBwithOrthoForFreq ( curFirstWord );
      double firstWordRadius = scaleFreqToRadius( firstWordFreq );

      double newAdditiveRadius = firstWordRadius;

      glPushMatrix();
      {

         //translate and rotate into place


         //if firstWord indicates a dead end ( xxx or fff, defined in wordBreakdow
         if(curFirstWord == deadEndDelim1  || curFirstWord == deadEndDelim2 ) {
            //draw a red sphere at the end of the last branch
         } else if (curFirstWord == successDelim ) {
            //draw a green sphere at the end of the last branch
         } else {

            //draw a branch
            drawBranch( radiansToDegrees( tiltAngle ), curXOffset, curYOffset, new
```

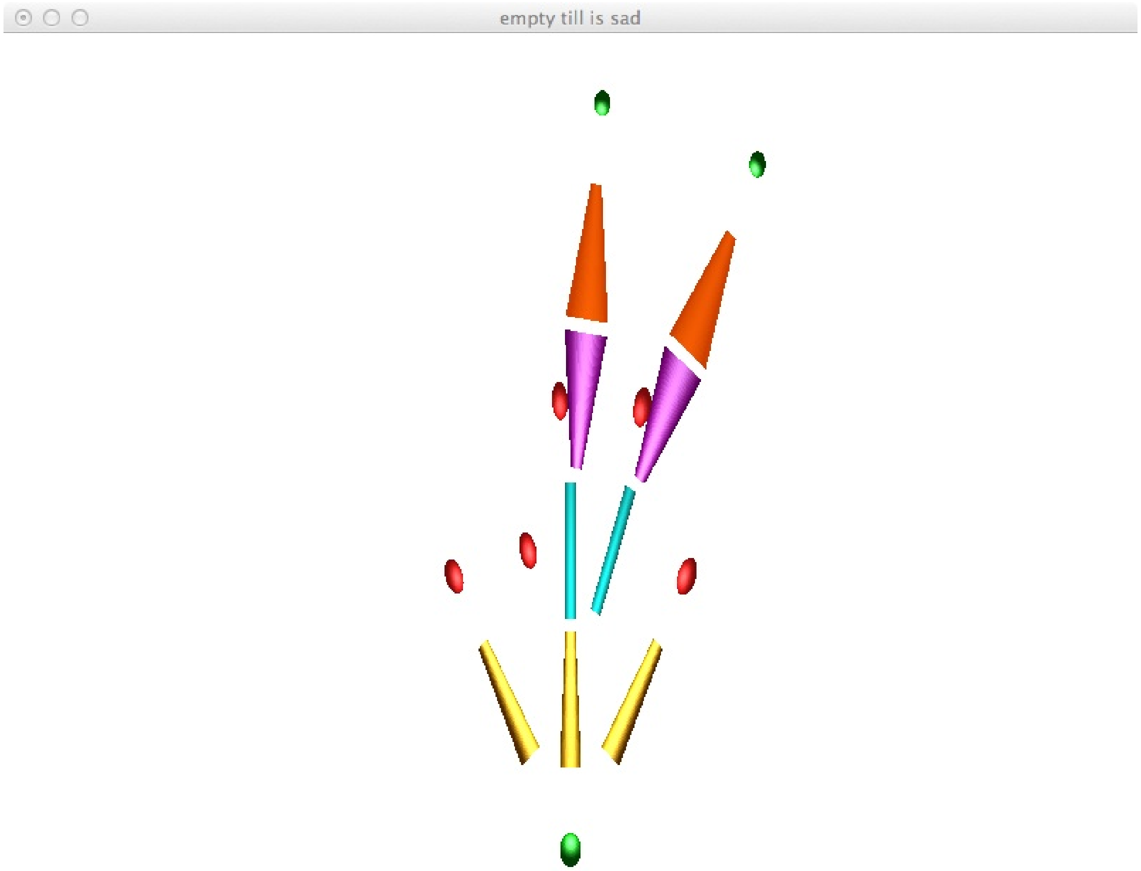

Figure 3.7: This is the parse tree for the phrase ”empty till is sad”

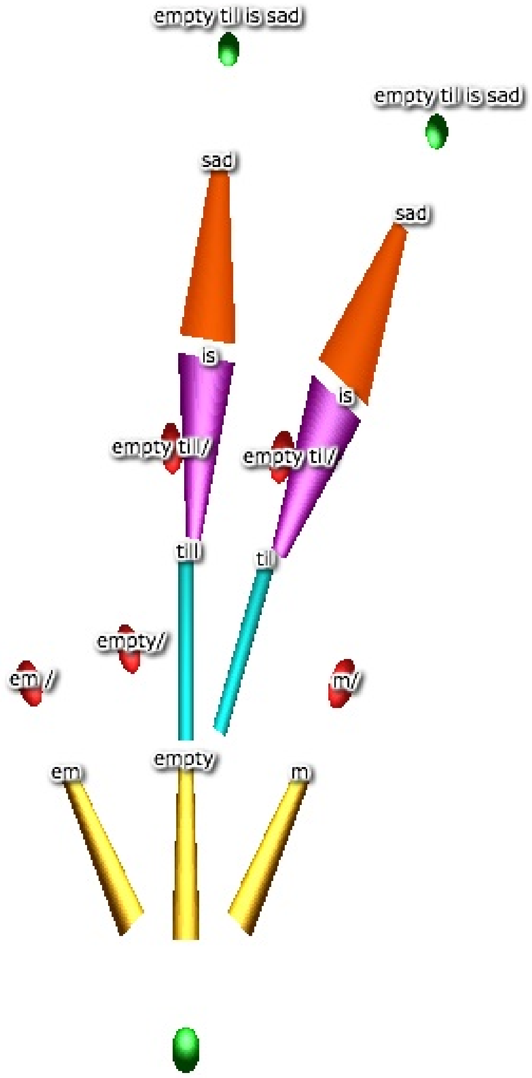

Figure 3.8: This is the annotated parse tree for the phrase ”empty till is sad”

Chapter 4

User Study

4.1 Structure

We created a multi-wave user study to examine the effectiveness of different parts of our program. I feel like I have multiple personality disorder writing in third person like this.

In the first wave, we had **NUM TO BE COUNTED** people record over 56 different phrases, to see how they pronouced them. This wave served two purposes: one, to gather recordings for the second wave, and two, to see if our phonemic transcriptions were valid.

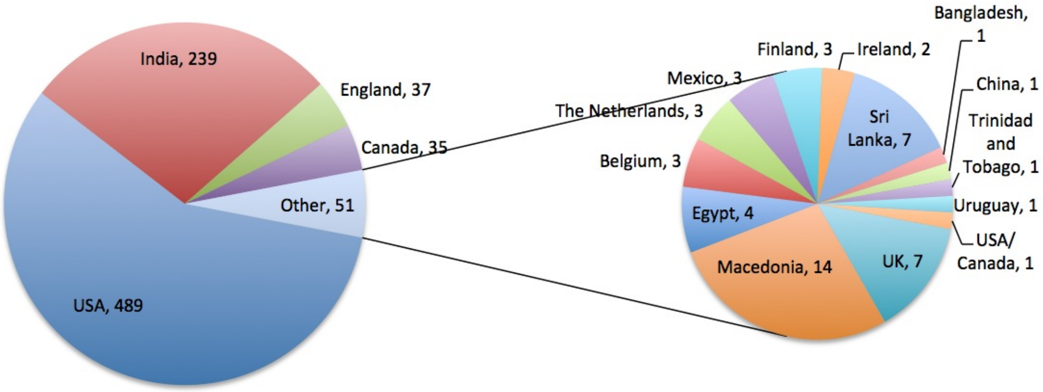

Figure 4.1: Our user study primarily polled people from the United States and India.

4.2 Methodology

4.2.1 First Wave: Recitation

In this wave of the user study, we used a combination of **NUM TO BE COUNTED** Mechanical Turk workers (hired for $ 0.10 per task) and volunteers from Reddit.com [2] [3] to record 58 different phrases.

4.2.2 Second Wave: Transcription

orthoPhrase	numRecordings	phraseID
a nice cold our	3	A.17.51 a nice cold our
an ice cold our	2	A.17.135 an ice cold our
a nye scold our	2	A.17.69 a nye scold our
ah nye scold our	2	A.18.109 ah nye scold our
an eye scold our	2	A.18.125 an eye scold our
on aye scold our	2	A.18.267 on aye scold our
a nigh scold our	2	A.18.65 a nigh scold our
a nye skol dower	2	A.18.71 a nye skol dower
an aye skol dower	2	A.19.119 an aye skol dower
an eye skol dower	2	A.19.127 an eye skol dower
an ice coal dower	2	A.19.133 an ice coal dower
eh nice coal dower	2	A.20.159 eh nice coal dower
ah nice coal dower	2	A.20.89 ah nice coal dower
fourth wry to	2	B.15.19 fourth wry to
fourth wry too	2	B.16.20 fourth wry too
forth right ooh	2	B.17.1 forth right ooh
fourth rite ooh	2	B.17.13 fourth rite ooh
forth wright ooh	2	B.18.6 forth wright ooh
on i scold our	1	A.16.279 on i scold our
an i scold hour	1	A.17.128 an i scold hour
an i skol dower	1	A.17.131 an i skol dower
an ice-cold our	1	A.17.141 an ice-cold our
on i scold hour	1	A.17.278 on i scold hour
on i skol dower	1	A.17.281 on i skol dower
an aye scold our	1	A.18.117 an aye scold our
an ice cold hour	1	A.18.134 an ice cold hour
an ice-cold hour	1	A.18.140 an ice-cold hour
eh nye scold our	1	A.18.179 eh nye scold our

Response By Country	Num Responses
USA	290
India	124
Britain	32
Canada	28
UK	6
Macedonia	5
Egypt	4
Belgium	3
The Netherlands	3
Mexico	3
Finland	3
Ireland	2
Sri Lanka	2
Bangladesh	1
China	1
Trinidad and Tobago	1
Uruguay	1
USA/Canada	1

Table 4.2: Here's a table with the number of responses per country

Chapter 5

Results

PUT NGRAMS DIAGRAM HERE: http://books.google.com/ngrams/graph?content=nice+colo

5.0.3 Individual Recording/Transcription Breakdowns

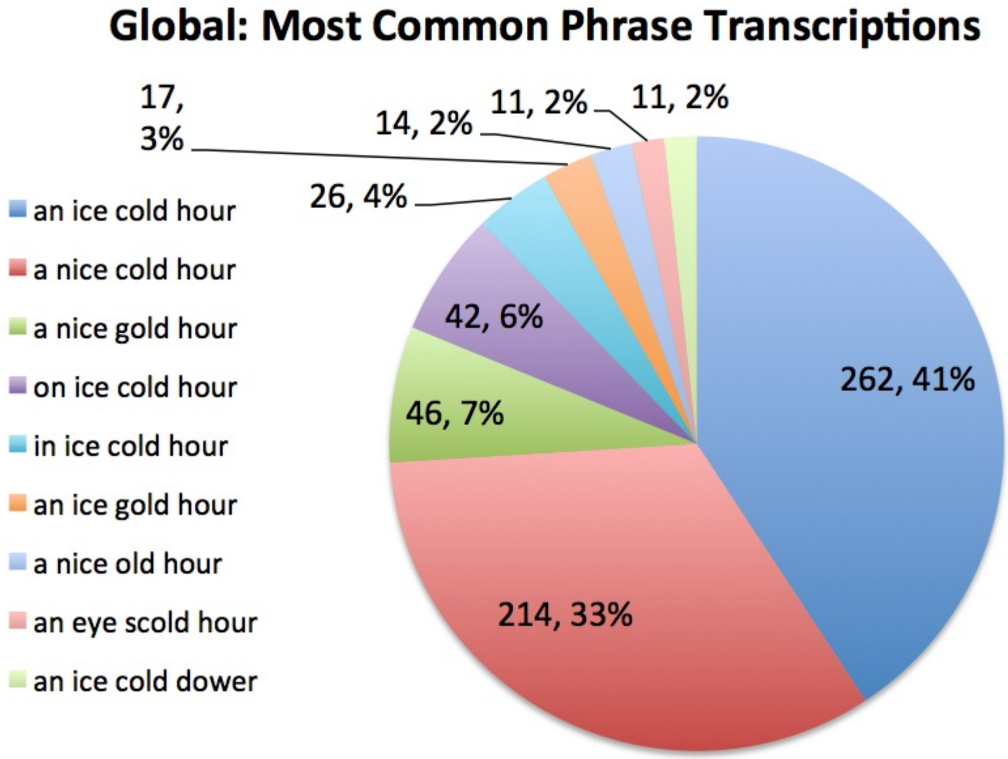

Figure 5.1: Our top two transcriptions were "a nice cold hour" and "an ice cold hour"

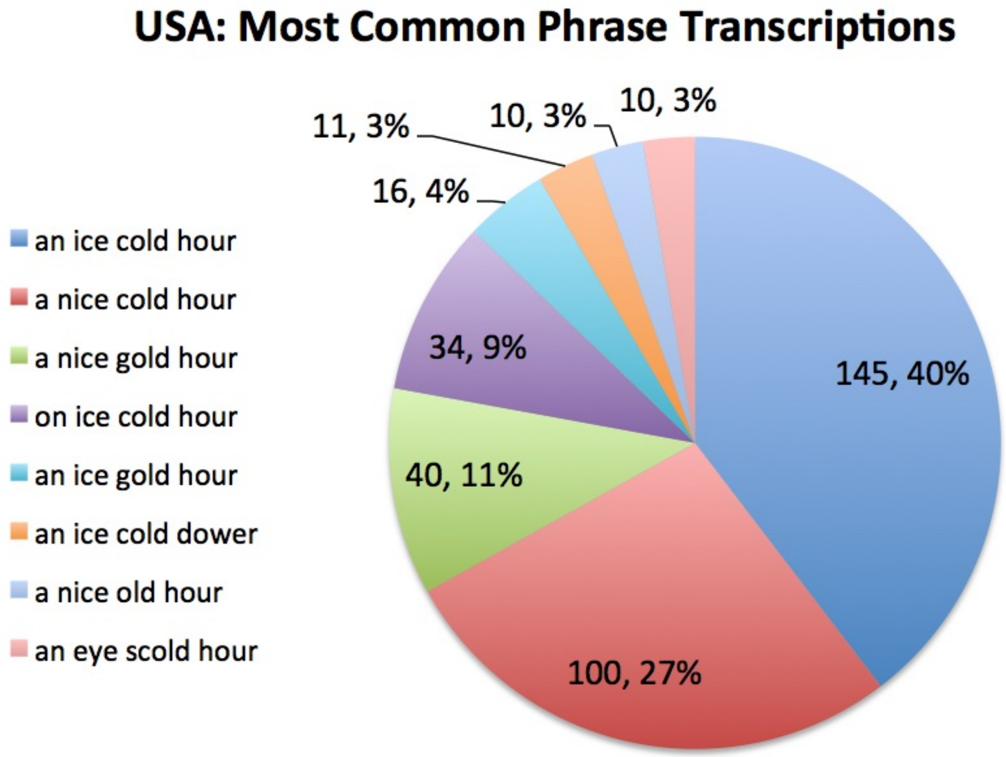

Figure 5.2: Though the breakdown is a bit different than the global transcription breakdown, you can still see the clear trend of "a nice cold hour" and "an ice cold hour" being the most common. There is a slightly larger gap between these two phrase, we hypothesize, because the American transcribers are familiar with what words normally are in proximity to others.

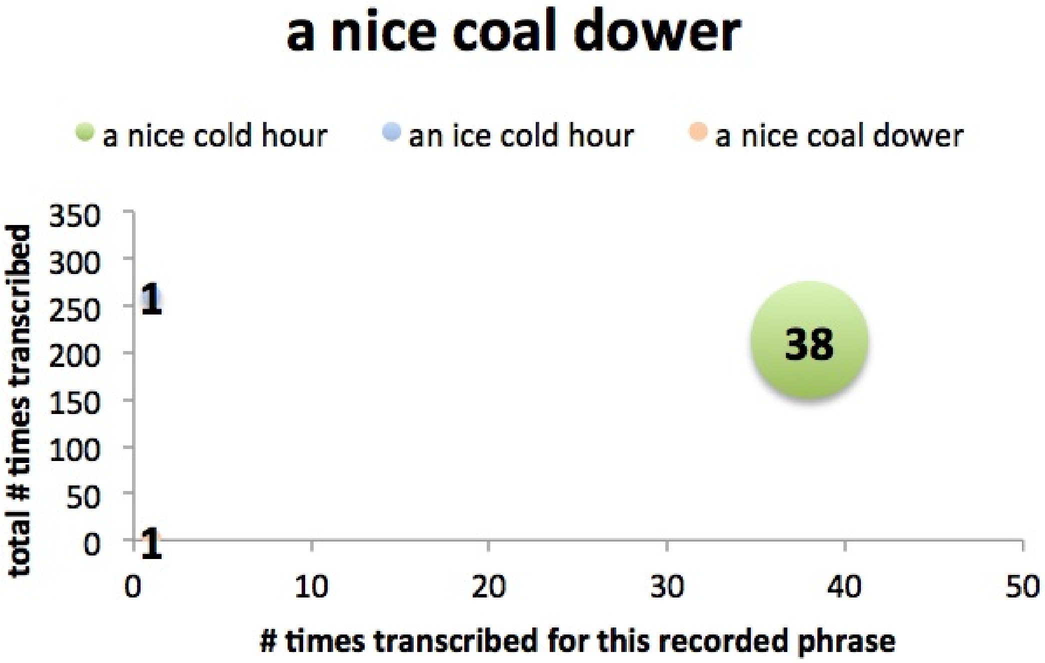

Figure 5.3: This bubble chart represents the most common transcriptions for the recorded phrase ”a nice coal dower”. The size of the bubble and the position along the x axis is indicative of the number of times that phrase was transcribed for this particular recording. The position along the y axis shows how often this phrase was transcribed over all the recordings.

freq	phrase transcribed	total answers
931028	an ice cold hour	262
7851662	a nice cold hour	214
7820004	a nice gold hour	46
2911102	on ice cold hour	42
5503158	in ice cold hour	26
899370	an ice gold hour	17
8013781	a nice old hour	14
892949	an ice cold dower	11
859307	an eye scold hour	11

Table 5.1: You can see, our data isn't too horrible

Chapter 6

Future Work

Chapter 7

Conclusion

SAMPA	Example	Manner of Articulation/ Type	Voiced/ Voiceless	Starts as	Ends as	General type	Weight
p	pen, spin, tip	plosive	voiceless	block	block	Consonant	8
b	but, web	plosive	voiced	block	block	Consonant	7
t	two, sting, bet	plosive	voiceless	block	block	Consonant	8
d	do, odd	plosive	voiced	block	block	Consonant	7
tS	chair, nature, teach	affricate	voiceless	block	continuous frication	Consonant	6
dZ	gin, joy, edge	affricate	voiced	block	continuous frication	Consonant	5
k	cat, kill, skin, queen, thick	plosive	voiceless	block	block	Consonant	8
g	go, get, beg	plosive	voiced	block	block	Consonant	7
f	fool, enough, leaf	fricative	voiceless	continuous frication	continuous frication	Consonant	4
v	voice, have, of	fricative	voiced	continuous frication	continuous frication	Consonant	3
T	thing, breath	fricative	voiceless	continuous frication	continuous frication	Consonant	4
D	this, breathe	fricative	voiced	continuous frication	continuous frication	Consonant	3
s	see, city, pass	fricative	voiceless	continuous frication	continuous frication	Consonant	4
z	zoo, rose	fricative	voiced	continuous frication	continuous frication	Consonant	3
S	she, sure, emotion, leash	fricative	voiceless	continuous frication	continuous frication	Consonant	4
Z	pleasure, beige	fricative	voiced	continuous frication	continuous frication	Consonant	3

SAMPA	Example	Manner of Articulation/ Type	Voiced/ Voiceless	Starts as	Ends as	General type	Weight
h	ham	fricative	voiceless	continuous frication	continuous frication	Consonant	4
m	man, ham	nasal	voiced	redirect	redirect	Consonant	1
n	no, tin	nasal	voiced	redirect	redirect	Consonant	1
N	singer, ring	nasal	voiced	redirect	redirect	Consonant	1
l	left, bell	lateral	voiced	continuous	continuous	Consonant	0
r	run, very	approximant	voiced	continuous	continuous	Consonant	0
w	we	semivowel	voiced	continuous	end	Consonant	2
j	yes	semivowel	voiced	continuous	end	Consonant	2
W	what (Scottish)	approximant	voiceless	continuous	end	Consonant	3
x	loch (Scottish)	fricative	voiceless	continuous frication	continuous frication	Consonant	4
A	father, not, law	short				Vowel	0.25
I	city	short				Vowel	0.25
E	bed	short				Vowel	0.25
3‘	bird, winner	short				Vowel	0.25
‘r	bird, winner	short				Vowel	0.25
{	lad, cat, ran	short				Vowel	0.25
u	soon, through	short				Vowel	0.25
@	about	short				Vowel	0.25
jU	use, pupil	diphthong		syllabic consonant semivowel	short	Diphthong	0.5
ju	use, pupil	diphthong		syllabic consonant semivowel	short	Diphthong	0.5
i	see	long				Vowel	0.75

SAMPA	Example	Manner of Articulation/ Type	Voiced/ Voiceless	Starts as	Ends as	General type	Weight
V	run, enough	short				Vowel	0.25
U	put	long				Vowel	0.75
e	day	long				Vowel	0.75
O	or, shore	long				Vowel	0.75
a	DNE in GenAm	long				Vowel	0.75
aI	my, height	diphthong		long	short	Diphthong	1
OI	boy	diphthong		long	short	Diphthong	1
oU	boat	diphthong		short	long	Diphthong	1
ou	boat	diphthong		short	long	Diphthong	1
aU	now	diphthong		long	long	Diphthong	1.5
=	ridden	syllabic consonant semivowel				Vowel	0.25

le 7.1: Here are the metrics that we use to determine the ghts of the phonemes

Bibliography

[1] The CMU pronouncing dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

[2] Dear R/Assistance, i'm about to finish my master's thesis, but i need your help! (tasks are online; i'm in san luis obispo, CA). : Assistance. http://www.reddit.com/r/Assistance/comments/ubty1/dear_rassistance_im_about_to_finish_m

[3] Dear RecordThis: i'm finishing up my masters thesis, and i need your help! : recordthis. http://www.reddit.com/r/recordthis/comments/ubt9f/dear_recordthis_im_finishing_up_my_ma

[4] File:General american.png - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/File:General_American.png.

[5] knights_emic.gif (GIF image, 552 407 pixels) - scaled (0%).

[6] knights_phonetic.jpg (JPEG image, 552 407 pixels) - scaled (0%).

[7] LCStar project web – schedule. http://www.lc-star.com/schedule.htm.

[8] Unisyn lexicon. http://www.cstr.ed.ac.uk/projects/unisyn/.

[9] S. Fitt. Documentation and user guide to UNISYN lexicon and post-lexical

rules. *Center for Speech Technology Research, University of Edinburgh, Tech. Rep*, 2000.

[10] T. Polyakova and A. Bonafonte. Fusion of dictionaries in voice creation and speech synthesis task. In *Proc. of SPECOM*, 2007.

[11] G. P. Smith. Music and mondegreens: extracting meaning from noise. *ELT Journal*, 57(2):113121, 2003.