Audio Speech Research Note

Ryan J. Cassidy (ryanc@ieee.org)
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University*
Stanford, CA 94305

Abstract

This document discusses some topics important to the understanding of the production, analysis, and modeling of human speech. Topics presented include the anatomy of the vocal tract, methods of describing speech sounds, the analysis and synthesis of vowel sounds, and the synthesis of vowel sounds using the CCRMA Common Lisp Music (CLM) package and the CCRMA Synthesis Tool Kit (STK).

Contents

1 Introduction

This document discusses some topics important to the understanding of the production, analysis, synthesis, and modeling of human speech. After a presentation of background topics (vocal tract anatomy, introductory phonology, etc.), three main vowel-sound-synthesis schemes are examined: formant-filter-based vowel synthesis, cascaded-tube-section vowel synthesis, and vowel synthesis via frequency modulation. Examples and demonstration software are featured wherever possible to reinforce the results presented.

2 Anatomy of the Human Vocal Tract

Fig. 1 shows a diagram of the human vocal tract, with some acoustically-significant features labeled. Acoustically-significant features not shown on the diagram include the lungs, the trachea (or windpipe—this is the tube which connects the lungs to the larynx), the glottis (or opening between the vocal chords or glottal folds), and the uvula (or fleshy prolongation that hangs from the soft palate). A more authoritative source for information about vocal tract anatomy may be found in [Netter 2003].

*http://www.stanford.edu/

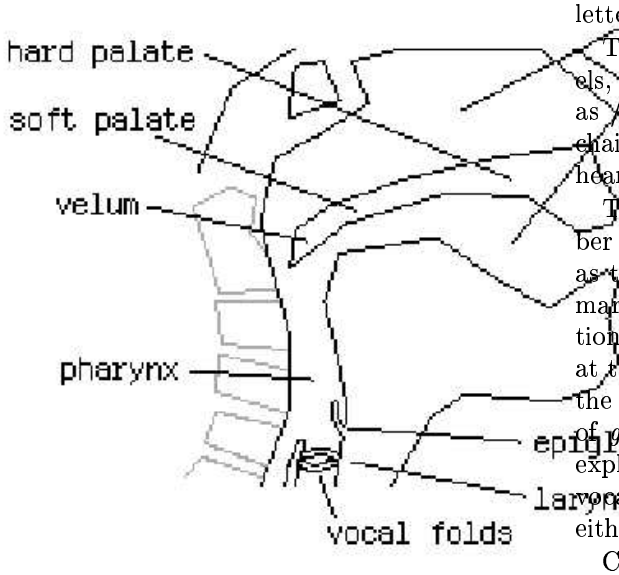

Figure 1: Diagram of the human vocal tract, showing some acoustically-significant features (adapted from [Cook 1990]).

3 Describing Speech Sounds

Much of the summary that proceeds has been inspired by [Fletcher 1953].

The basic linguistic unit is called a phoneme, denoted by a character enclosed with forward slashes or square braces (e.g. the symbol /iː/ represents the vowel sound heard in the word *team*).[1] Phonemes may be classified and described according to a number of criteria. They may be divided, for example, into vowels and consonants. While vowels are produced primarily by the vibration of the vocal folds, consonants are the speech sounds produced by the turbulent or explosive flow of air through constricted or obstructed parts of the vocal tract, known as articulators. Note that consonants may also involve vibration of the vocal folds, as in the phoneme /v/, heard in the word *voice*. Such consonants are called voiced consonants, while consonants such as the /f/ of the word *fish* are referred to as unvoiced, since the vocal folds are simply held open during the production of these sounds. A tentative rule of thumb is that vowels consist of the sounds represented by the letters a, e, i, o, u, and sometimes y.

The vowels may be further divided into pure vowels, each consisting of a single voiced sound, such as /ʊ/ of the word *took*, and dipthongs, created by chaining two pure vowels together, such as the /aɪ/ heard in the word *time*.

The consonants may also be divided along a number of lines. Fricative consonants, or spirants, such as the aforementioned /f/ of *fish* and /s/ of *sit*, are marked by a steady, turbulent flow of air at a constriction created somewhere in the vocal tract other than at the vocal chords. Stop consonants, or plosives, on the other hand, such as the /p/ of *push* or the /g/ of *goat*, are produced by the build-up and sudden, explosive release of air pressure at some point in the vocal tract. Fricatives and stop consonants may be either voiced or unvoiced.

Certain terms used in the description of speech sounds may be applied to both vowels and consonants. Nasal sounds are those in which the nasal cavity plays a role in the transmission and broadcast of the vocal sound, whereas non-nasal sounds occur when the nasal cavity is cut off from the vocal tract by the velum during sound production. Continuants are those speech sounds that involve the continuous, steady flow of air from lungs to the environment, while stops involve the complete closure or obstruction of the vocal cavities at some point in the production of the sound.

Finally, there are some classes of phonemes that do not fit neatly into the vowel-consonant classification scheme described above. The sounds /l/, /r/, /m/, /n/, and /ng/, for example, though often thought of as consonants, are referred to as liquids or semi-vowels. The sounds /w/, /y/, and /h/ are referred to as transitionals in [Fletcher 1953]. Speech sounds referred to as affricates consist of a plosive or stop consonant immediately followed by a fricative or spirant, such as the German /pf/.

4 Analysis and Synthesis of Pure Vowels

Though the discussion that proceeds treats pure vowels (defined in §3), almost identical principles may be applied to the liquids or semi-vowels, and diphthongs (previously explained) may be thought of as two pure vowels chained together.

During the phonation of pure vowels, a roughly periodic acoustic pressure wave is produced at the vocal chords, and subsequently transmitted through

[1] Unfortunately, different characters are used for certain phonemes, depending on which alphabet the author chooses. For example, Webster's dictionary uses a different set of symbols than the International Phonetic Association. A table of symbols used by Webster's Dictionary, Bell Labs, and the International Phonetic Association may be found in [Fletcher 1953]. This document uses the IPA symbols.

the vocal tract. It is then broadcast from the oral and/or nasal cavities to the environment. The vocal folds may thus be thought of as a sound source, and the vocal tract (including the oral and/or nasal cavities) functions as a filter (hence the phrase "source-filter description of speech sounds" used by Fant in [Fant 1960]). The waveform produced by the vocal chords looks roughly like a lowpass-filtered band-limited impulse train, with fundamental frequency f_0 that gives rise to the apparent pitch of the voiced sound (not necessarily constant). More information may be found in [Fant 1960].

The resonances (or modes) of the vocal tract give rise to peaks in the spectrum of the vowel sound, or "formants." It is thought that the formants of a vowel sound play a primary role in distinguishing it from other vowels, as any two vowels may well have the same fundamental frequency. The term "formant" has evolved over the past two centuries[Dunn 1950], as phoneticians/phonologists first used the term loosely to describe the spectral properties of vowel sounds, and acousticians, physicists, and electrical engineers subsequently used it to denote spectral peaks created by vocal tract resonances. The earliest formal use of the term, according to the Oxford English Dictionary, was in 1901, and the first technical use was in 1952[Simpson and Weiner 1989]. Fant gives a precise definition in [Fant 1960]: "The spectral peaks of the sound spectrum $|P(f)|$ are called *formants*," where $P(f) = S(f)T(f)$, $S(f)$ is the spectrum of the glottal waveform, and $T(f)$ is the transfer function of the vocal tract. Note that although, strictly speaking, the term formant refers to the peaks of $|P(f)|$, it is often used simply to refer to the peaks of the vocal tract transfer function, $|T(f)|$[Fant 1960].

The aforementioned source-filter model of speech production lends itself nicely to implementation on a digital computer. Such an implementation is described in [Klatt 1980]. A band-limited impulse train with fundamental frequency of approx. 100–200 Hz may be created and subsequently lowpass-filtered. This signal may then be applied to a network of second-order all-pole filters. Finally, the network output may be filtered to simulate radiation of the sound from the nose and mouth.

5 Formant-Filter Based Vowel Synthesis Examples

The vowel synthesis technique described in [Klatt 1980] uses biquadratic filters (arranged either in parallel or cascade) to simulate the formants of the human vocal tract. This technique will hereafter be described as *Formant-Filter Based Vowel Synthesis*. The following sections give examples of such a technique in action.

5.1 CLM Formant-Filter Based Vowel Synthesis

Due to recent developments on a project at CCRMA related to sonification of complex data[Ben Tal *et al.* 2001], it is now possible to synthesize simple vowel sounds using techniques described in §4 with simple scripts written in Common Lisp[2]. To create and run such scripts, it is first necessary to obtain and install the Common Lisp Music (CLM) package[3], and then obtain the `vowel` function, written by Michelle Daniels[Daniels 2001].

Here is a short Lisp script that generates a short vowel sound in RIFF (WAV) file format:

```
(compile-file "~/audio/speech/vowel/clm_vowel/danielsm/vow
(load "~/audio/speech/vowel/clm_vowel/danielsm/vowel")

(with-sound (:header-type mus-riff
     :output "~/audio/speech/vowel/clm_vowel/rjc_clm_vowel
  (vowel 0 10 10 :pulse-freq 100))
```

The first two lines compile and load the file `vowel.ins`, which contains the definition of Daniels' vowel instrument (this file may be obtained from Michelle Daniels' CLM vowel site[4]), which, in this example, resides in the author's `~/audio/speech/vowel/clm_vowel/danielsm/` directory. The function `with-sound` is a CLM routine that generates sound output. The optional parameter `header-type` has been set to `mus-riff` to indicate WAV output is desired, and the optional parameter `output` has been set to the desired file name. Finally, the `vowel` call within the `with-sound` function creates a vowel sound starting at the beginning of the file (the value 0 has been provided as the first argument to indicate this), 10 seconds long (as specified by the second argument), and of amplitude equal to 10. The optional parameter `pulse-freq` has been set to 100 Hz (the default is 200 Hz).

[2] For an introduction to Common Lisp, see [Touretzky 1990]. More detailed information may be found in [Steele 1990] or [Secretariat 1994]. For information on installing Common Lisp on a variety of computer platforms, see http://www.franz.com/downloads/, http://www.cons.org/cmucl/, or http://clisp.cons.org/.

[3] http://www-ccrma.stanford.edu/software/clm/

[4] http://www-ccrma.stanford.edu/~danielsm/soni/clmvowel.html

In order to run the script, it is first necessary to set up the Common Lisp environment to load the appropriate CLM libraries. Under XEmacs, this may be accomplished by prepending the bundled `.emacs`[5] file into one's `.emacs` file, starting XEmacs, and pressing `C-x C-l`, where `C` denotes the `Ctrl` key on the keyboard.[6]

The WAV file produced by the script may be heard at
http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/clm_vowel.wav

5.2 STK Formant-Filter Based Vowel Synthesis

The Synthesis Tool Kit (STK)[7], a CCRMA-created collection of C++ classes for the synthesis and processing of musical instrument sounds, contains a C++ class `VoicForm` for the synthesis of vowel sounds based on formant filtering of a band-limited impulse train.

The following short C++ program generates a short clip of the vowel /iː/:

```
#include "VoicForm.h"
#include "WvOut.h"

#define OUTPUT_LENGTH_SEC       5
#define NUM_OUTPUT              (Stk::sampleRa

#define VOWEL_FREQ              75.0
#define VOWEL_AMP               1.0

int main(void)
{
  WvOut wvo("test_vowel.wav");
  VoicForm vf;

  vf.noteOn(VOWEL_FREQ, VOWEL_AMP);
  vf.speak();

  for (int i=0; i<NUM_OUTPUT; i++)
  {
    wvo.tick(vf.tick());
  }

  wvo.closeFile();

  return 0;
}
```

[5] http://ccrma-www.stanford.edu/software/clm/clm/DotEmacs

[6] Note that in order for the resultant `.emacs` file to work properly, it is necessary to install the CLM package on one's system. The CLM package may be obtained from http://www-ccrma.stanford.edu/planetccrma/software/.

[7] http://ccrma-www.stanford.edu/software/stk/

The `NoteOn()` member of `VoicForm` sets the frequency (in Hz) (default is 75 Hz) and amplitude (a non-dimensional, linear value) of the vowel. Next `VoicForm::speak()` activates vowel sound output with subsequent calls to `VoicForm::tick()` (`tick()` is the name of the method given to most STK classes that accept or produce audio samples). Finally, the `for` loop produces the vowel output samples and stores them in an STK WAV file object, `wvo`.

The WAV file produced by this program may be heard at
http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/stk_vowel/

5.3 GUI Application Formant-Filter Based Vowel Synthesis

A GUI-driven application has been written to allow users to experiment with formant bandwidths, center frequencies, and amplitudes. The GUI has been implemented in Tcl/Tk, with vowel synthesis handled by the `VoicForm` class described in §5.2. A screen-shot of this application is shown in Figure 2.

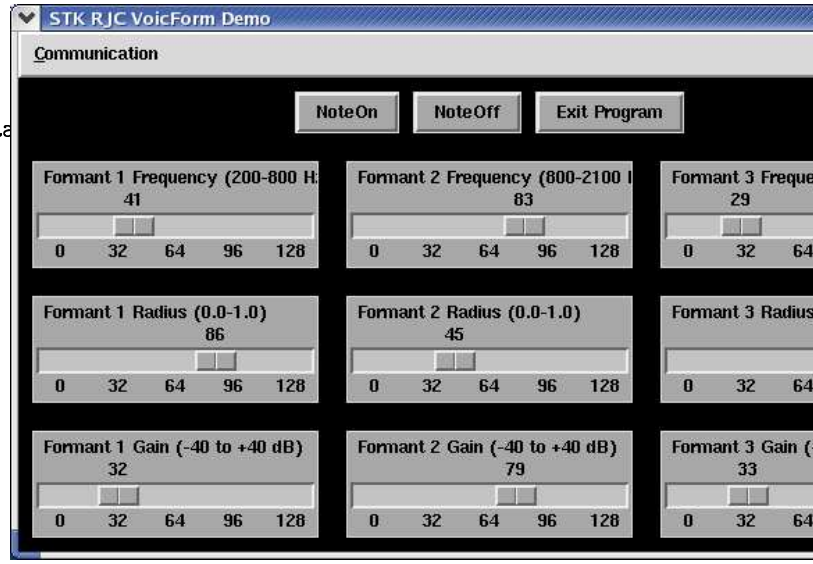

Figure 2: Screen-shot of the GUI application demonstrating the STK `VoicForm` class.

This GUI may be downloaded at
http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/VoicFormI

6 Digital Waveguide Modeling of the Vocal Tract

In his thesis [Cook 1990], Perry Cook describes a method of vocal tract modeling superior to the pre-

viously described formant-filter based approach. The method involves approximating the vocal tract by a series of acoustic tube sections, each with a radius that varies from one vowel sound to the next. As shown in [Cook 1990], the radii of adjacent tube sections govern the transmission and reflection of acoustic energy at the junction between such sections. For each tube section, discrete-time delay elements are used to model the forward- and reverse-traveling wave components of the digital waveguide simulation[8]. Between the delay elements, a scattering junction is used to handle the change in radius from one tube section to the next. A block diagram is shown in Figure 3.

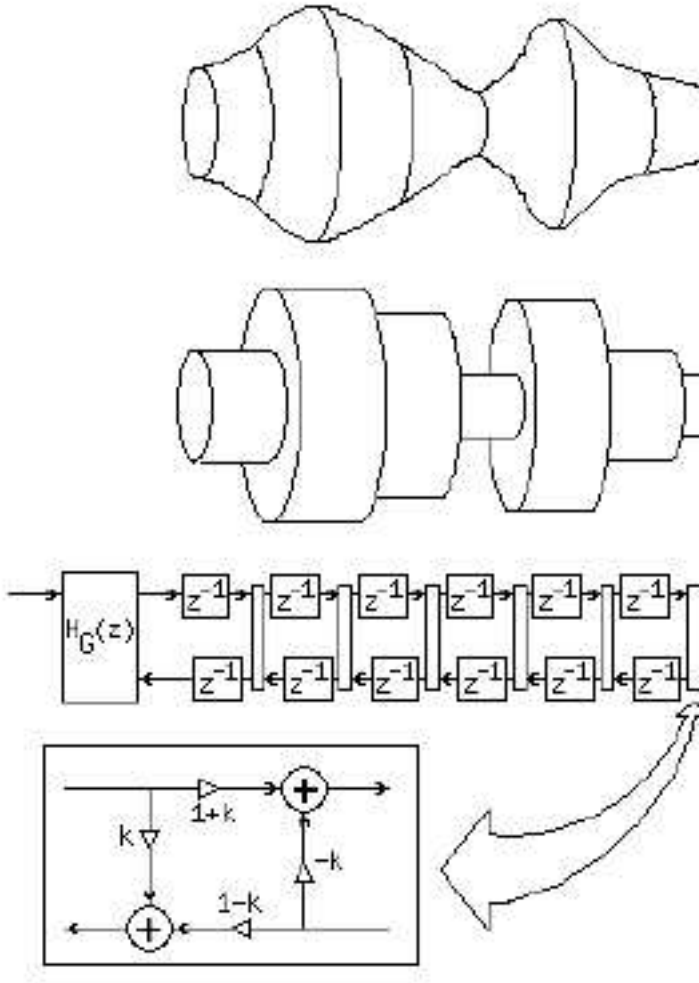

Figure 3: Digital waveguide model of the human vocal tract, with scattering junctions between adjacent tube sections to account for changing radii (adapted from [Cook 1990]).

6.1 Derivation of Acoustic Wave Propagation in a Tube and the Cascaded-Tube Section Scattering Relations

An excellent derivation of the acoustic wave equation in a tube may be found in [Cook 1990]. The main points are summarized here:

We require the formulae for point-wise conservation of momentum and mass within the tube:

$$a(x)\frac{\partial p(x,t)}{\partial x} = -\rho\frac{\partial u(x,t)}{\partial t}, \tag{1}$$

and

$$\frac{\partial u(x,t)}{\partial x} = -\frac{a(x)}{\rho c^2}\frac{\partial p(x,t)}{\partial t}, \tag{2}$$

where $a(x)$ is the cross-sectional area of the tube at point x (in square meters), $p(x,t)$ is the longitudinal pressure (in Newtons per square meter), ρ is the density of the fluid in the tube (in kilograms per cubic meter), $u(x,t)$ is the volume velocity of the fluid in the tube (in cubic meters per second), and c is the speed of sound in the fluid.

If $a(x)$ is assumed to be a constant, A, then combining the two equations yields

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 u(x,t)}{\partial t^2}, \tag{3}$$

or, for pressure,

$$\frac{\partial^2 p(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 p(x,t)}{\partial t^2}. \tag{4}$$

These may each be recognized as dual forms of the homogeneous wave equation, the solution of which allows for arbitrary left- and right-going traveling wave

[8]The musical and acoustic applications of digital waveguide modeling are well-described in [Smith III 2002]

components (these will be hereafter referred to as u^+ and u^- for volume velocity, and p^+ and p^- for pressure). The general solutions to Eq. (3) and Eq. (4) are given by:

$$u(x,t) = u^+(t - \frac{x}{c}) + u^-(t + \frac{x}{c}), \quad (5)$$

and

$$p(x,t) = p^+(t - \frac{x}{c}) + p^-(t + \frac{x}{c}). \quad (6)$$

Since the forward- and reverse-traveling wave components each satisfy the wave equation separately (this can be shown by direct substitution), Eq. (1) yields

$$\begin{aligned} A\frac{\partial p^+(t-\frac{x}{c})}{\partial x} &= -\rho\frac{\partial u^+(t-\frac{x}{c})}{\partial t} & (7)\\ \Rightarrow \frac{\partial p^+(t-\frac{x}{c})}{\partial x} &= \frac{\rho c}{A}\frac{\partial u^+(t-\frac{x}{c})}{\partial x} & (8)\\ \Rightarrow p^+(t-\frac{x}{c}) &= \frac{\rho c}{A}u^+(t-\frac{x}{c}), & (9)\end{aligned}$$

and similarly, for the left-traveling wave component,

$$\begin{aligned} A\frac{\partial p^-(t+\frac{x}{c})}{\partial x} &= -\rho\frac{\partial u^-(t+\frac{x}{c})}{\partial t} & (10)\\ \Rightarrow \frac{\partial p^-(t+\frac{x}{c})}{\partial x} &= -\frac{\rho c}{A}\frac{\partial u^-(t+\frac{x}{c})}{\partial x} & (11)\\ \Rightarrow p^-(t+\frac{x}{c}) &= -\frac{\rho c}{A}u^-(t+\frac{x}{c}). & (12)\end{aligned}$$

Define the characteristic impedance of the tube as

$$\begin{aligned} R &= \frac{\rho c}{A} & (13)\\ \Rightarrow p^+(t-\frac{x}{c}) &= Ru^+(t-\frac{x}{c}), & (14)\\ p^-(t+\frac{x}{c}) &= -Ru^-(t+\frac{x}{c}). & (15)\end{aligned}$$

Consider a pair of tubes (with characteristic impedances R_1 and R_2) in cascade. The scattering relation for the junction between these tubes may be derived as follows:

$$\begin{aligned} 0 &= u_1 - u_2 & (16)\\ &= u_1^+ + u_1^- - u_2^+ - u_2^- & (17)\\ &= G_1(p_1^+ - p_1^-) + G_2(p_2^- - p_2^+) & (18)\\ &= G_1(2p_1^+ - p_J) + G_2(2p_2^- - p_J) & (19)\\ \Rightarrow p_J &= \frac{2p_1^+G_1 + 2p_2^-G_2}{G_1+G_2} & (20)\\ \Rightarrow p_1^- &= p_J - p_1^+ & (21)\\ &= \frac{G_1-G_2}{G_1+G_2}p_1^+ + \frac{2G_2}{G_1+G_2}p_2^- & (22)\\ &= \frac{R_2-R_1}{R_1+R_2}p_1^+ + \frac{2R_1}{R_1+R_2}p_2^-, & (23)\\ \Rightarrow p_2^+ &= p_J - p_2^- & (24)\\ &= \frac{G_2-G_1}{G_1+G_2}p_2^- + \frac{2G_1}{G_1+G_2}p_1^+ & (25)\\ &= \frac{R_1-R_2}{R_1+R_2}p_2^- + \frac{2R_2}{R_1+R_2}p_1^+, & (26)\end{aligned}$$

where u_1 and u_2 are the volume velocities in the two tubes at the junction, u_1^+ and u_1^- are the forward- and reverse-traveling volume velocity wave components in tube 1 at the junction, u_2^+ and u_2^- are the forward- and reverse-traveling volume velocity wave components in tube 2 at the junction, p_1 and p_2 are the pressures in the two tubes at the junction, p_1^+ and p_1^- are the forward- and reverse-traveling pressure wave components in tube 1 at the junction, p_2^+ and p_2^- are the forward- and reverse-traveling pressure wave components in tube 2 at the junction, p_J is the junction pressure (must be identical to p_1 and p_2 to avoid a pressure gradient across a zero-width (and hence, zero-mass) section), and $G_1 = \frac{1}{R_1}$ and $G_2 = \frac{1}{R_2}$ are the acoustic conductances in the two tube sections. Note Eq. (16) expresses the conservation of volume velocity, and Eq. (19) uses the aforementioned continuity of pressure across the junction.

The following subsections deal with sample implementations of this vocal synthesis technique.

6.2 C/C++ Vocal Tract Class

Unfortunately, there is currently no class in STK implementing the aforementioned synthesis technique. For this reason, a C/C++ class has been written from scratch. The class is called `VoicTract`, and has been designed with an interface similar to other STK instruments. The code for this class is shown in Appendix A. It is hoped that this class may soon be included in subsequent releases of the STK.

The class features the ability to modify tube section radii and length. Regarding the modification of section length, tube sections may be set to a non-unity delay (unlike the example shown in Figure 3), and also to fractional delay values for maximum versatility.

6.3 Command-line Tract Model Based Vowel Synthesis

A command-line based demo application has been written to allow users to experiment with the various parameters of the digital waveguide based tract model. The application is called `tract_vowel`, and may be downloaded at
http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/tract_vowe

The application allows for the setting of tract parameters on the command-line via options preceded with a '-' character, or the command-line options may be stored in a text file and then submitted to the program using the '-f' option. More information may be obtained by typing `'./tract_vowel ---help'` in the directory into which the program has been downloaded.

Clips of two vowel sounds ("eee" and "ahh") generated using this program may be heard at http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/tract_vowel/rjc_stk_tract_vowel_eee_ex.snd, and http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/tract_vowel/rjc_stk_tract_vowel_ahh_ex.snd.

7 Vowel Production via FM Synthesis

As a final method for the synthesis of vocal sounds, John Chowning discovered that his FM synthesis technique [Chowning 1973] (a technique for the production of musical sounds) could be used to generate sounds with a vocal texture [Chowning 1989].

A command-line application has been written to allow users to experiment with various FM synthesis parameters for vowel synthesis. The utility is called `fm_vowel`, and may be downloaded at http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/fm_vowel/fm_vowel.tar.gz.

Similar to the `tract_vowel` utility described in §6.3, command-line options may be provided either on the command-line directly, or in a file for subsequent access using the `'-f'` switch. Also, as with `tract_vowel`, more information may be obtained using `'./fm_vowel ---help'` in the directory into which the program has been downloaded.

A clip of a vowel generated using this utility may be heard at http://www-ccrma.stanford.edu/~rjc/audio/speech/vowel/fm_vowel/fm_vowel_250_Hz.wav,

8 Conclusions

Possibilities for future investigation/development include the refinement and expansion of the Tcl/Tk GUI allowing users to experiment with formant-filter-based vowel sounds, the use of higher-order filters to simulate the radiation of the lips and nose in Cook's vocal tract model, and the further investigation of noise sources in Cook's vocal tract model. Each of the vowel synthesis schemes examined in this paper could probably benefit from the incorporation of slowly-varying random-walk-like processes to give a more "life-like" quality to the sounds.

A C/C++ Code for `VoicTract` Class

A.1 `VoicTract.h`

```
/***************************************************
 * Institution: Stanford University
 * Project: Sonification
 * Author: Ryan Cassidy (05157787)
 * Date: Summer 2003
 ***************************************************/

/*! \class VoicTract
 *  \brief STK class to implement modified Cook tract mode
 *
 * VERSION CONTROL INFORMATION:
 *
 * $RCSfile: VoicTract.h,v $
 *
 * $Author: rjc $
 *
 * $Date: 2003/10/05 05:35:45 $
 *
 * $Locker: rjc $
 *
 * $Log: VoicTract.h,v $
 * Revision 1.2  2003/10/05 05:35:45  rjc
 * Added functions to get number of tract sections.
 * Const correctness stuff.
 * Added ability to return tube section scattering 'k' val
 *
 * Revision 1.1  2003/09/29 21:49:15  rjc
 * Initial revision
 *
 *
 ***************************************************/

#ifndef __VOICTRACT_H
#define __VOICTRACT_H

#include "Instrmnt.h"
#include "VocTract.h"
#include "SingWave.h"
#include "Envelope.h"
#include "Stk.h"
#include "stypes.h"

#include <string>

using namespace std;

class VoicTract : public Instrmnt
{
 public:
```

```
  //! Default constructor.
  VoicTract(int num_tract_sections = DEFAULT_NUM_SECTIONS);

  //! Default destructor.
  virtual ~VoicTract();

/*************************************************************************
 * Functions to set vibrato parameters of voiced/glottal waveform.
 *************************************************************************/
  //! Set amount of vibrato in the input glottal waveform.
  void setVibratoAmt(MY_FLOAT vibratoAmt);

  //! Set the amount of random-ness in the voiced waveform.
  void setRndVibAmt(MY_FLOAT vibratoAmt);

/*************************************************************************
 * Functions to set voiced/unvoiced ratio of glottal waveform.
 *************************************************************************/
  //! Set the gain of the voiced portion of the sound.
  void setVoiced(MY_FLOAT vGain);

  //! Set the gain of the unvoiced portion of the sound.
  void setUnVoiced(MY_FLOAT nGain);

  //! Set both voiced and unvoiced gains.
  void setVoicedUnVoiced(MY_FLOAT vGain, MY_FLOAT nGain);
/*************************************************************************/

  //! Set the rate at which pitches sweep.
  void setPitchSweepRate(MY_FLOAT rate);

  //! Start the singer.
  void speak();

  //! Stop the singer.
  void quiet();

  //! Set the frequency of the underlying glottal waveform.
  void setFreq(MY_FLOAT frequency);

  //! Begin note with specified gain and frequency.
  void noteOn(MY_FLOAT freq, MY_FLOAT amp);

  //! End note currently playing.
  void noteOff(MY_FLOAT amp);

  //! Clock out a sample.
  MY_FLOAT tick();

  //! Control change.
  void controlChange(int number, MY_FLOAT value);

/*************************************************************************
 * Functions to get and set radii of vocal tract sections.
 *************************************************************************/
  //! Set vocal tract radii.
  void setTractRadii(const MY_FLOAT *radii, int num_sectio

  //! Get tract radii.
  const MY_FLOAT *getTractRadii() const;

  //! Set individual tract section radius.
  void setTractSectionRadius(int sec_index, MY_FLOAT radiu

/*************************************************************************
 * Functions to get and set lengths of vocal tract sectior
 *************************************************************************/
  //! Set tube section lengths.
  void setTractLengths(const MY_FLOAT *lengths, int num_se

  //! Return tract section lengths.
  const MY_FLOAT *getTractLengths() const;

  //! Set individual tract section length. \e sec_index is
  void setTractSectionLength(int sec_index, MY_FLOAT lengt
/*************************************************************************/

  //! Get number of sections.
  int numTractSections() const;

  //! Set phoneme.
  int setPhoneme(const char *name, ShapeDataSource sds = S
    {return _tract.setShape(name, sds);}

  const MY_FLOAT *getk() const {return _tract.getk();}

  void setk(const MY_FLOAT *k, int num_k_vals)
    {
      _tract.setk(k, num_k_vals);
    }
protected:
  VocTract _tract;
  string _resource_path;
  SingWave _voiced;
  Envelope _noise_env;
  OnePole _vib_filt;
  MY_FLOAT _perf_vib_amt, _freq, _last_output;
};

#endif
```

A.2 VoicTract.cpp

```
/*******************************************************************************
 * Institution: Stanford University
 * Project: Sonification
 * Author: Ryan Cassidy (05157787)
 * Date: Summer 2003
 *******************************************************************************/
/*! \class VoicTract
 *  \brief STK class to implement modified Cook tract model
 *
 * VERSION CONTROL INFORMATION:
 *
 * $RCSfile: VoicTract.cpp,v $
 *
 * $Author: rjc $
 *
 * $Date: 2003/10/05 05:33:49 $
 *
 * $Locker: rjc $
 *
 * $Log: VoicTract.cpp,v $
 * Revision 1.2  2003/10/05 05:33:49  rjc
 * Fxns to set radii and length now take pointers to const things.
 * Other const-correctness stuff.
 *
 * Revision 1.1  2003/09/29 21:48:09  rjc
 * Initial revision
 *
 *
 *******************************************************************************/

#include "VoicTract.h"

#include "Instrmnt.h"
#include "Stk.h"
#include "SKINI.msg"

#include <string>

using namespace std;

VoicTract::VoicTract(int num_tract_sections /* = DEFAULT_NUM_SECTIONS */)
  :
  Instrmnt(),
  _tract(num_tract_sections),
  _resource_path(""),
  _voiced((_resource_path + "glots/default.raw").c_str(), true),
  _last_output(0.0)
{
//  this->setPhoneme("eee");

  _voiced.normalize();
  _voiced.setGainRate((MY_FLOAT) 0.0005);
  _voiced.setGainTarget((MY_FLOAT) 0.0);
  _voiced.setSweepRate(1.0);
  this->setFreq(250);
  _voiced.tick();
  _voiced.setSweepRate(0.001);

  _noise_env.setRate((MY_FLOAT) 0.001);
  _noise_env.setTarget((MY_FLOAT) 0.0);

  _vib_filt.setPole(0.9999);
  _perf_vib_amt = 0.03;

  this->noteOn(400.0, 0.0);
}

VoicTract::~VoicTract()
{
}

void VoicTract::setVibratoAmt(MY_FLOAT vibratoAmt)
{
  _perf_vib_amt = vibratoAmt;
}

void VoicTract::setRndVibAmt(MY_FLOAT vibratoAmt)
{
  _voiced.setRandomGain(vibratoAmt);
}

void VoicTract::setVoiced(MY_FLOAT vGain)
{
  _voiced.setGainTarget(vGain);
}

void VoicTract::setUnVoiced(MY_FLOAT nGain)
{
  _noise_env.setTarget(nGain);
}

void VoicTract::setVoicedUnVoiced(MY_FLOAT vGain, MY_FLOAT
{
  this->setVoiced(vGain);
  this->setUnVoiced(nGain);
}

void VoicTract::setPitchSweepRate(MY_FLOAT rate)
{
  _voiced.setSweepRate(rate);
}

void VoicTract::speak()
{
  _voiced.noteOn();
```

```
}

void VoicTract::quiet()
{
  _voiced.noteOff();
  _noise_env.setTarget((MY_FLOAT) 0.0);
}

void VoicTract::setFreq(MY_FLOAT frequency)
{
  _voiced.setFrequency(frequency);
  _freq = frequency;
}

void VoicTract::noteOn(MY_FLOAT freq, MY_FLOAT amp)
{
  _voiced.setGainTarget(amp);
  this->setFreq(freq);
}

void VoicTract::noteOff(MY_FLOAT amp)
{
  _voiced.noteOff();
  _noise_env.setTarget(0.0);
}

MY_FLOAT VoicTract::tick()
{
  _voiced.setVibratoGain(_vib_filt.tick(_perf_vib_amt));

//  _tract.setNoiseGain(_noise_env.tick());
  MY_FLOAT vc = _voiced.tick();

  _last_output = _tract.tick(vc) * 0.1;

  return _last_output;
}

#define NORM_7 ((MY_FLOAT) 0.0078125)
void VoicTract::controlChange(int number, MY_FLOAT value)
{
  MY_FLOAT temp;
  int tempi;

#if defined(_STK_DEBUG_)
  printf("VoicTract : ControlChange: Number=%i Value=%f\n",number,value);
#endif
  if (number == __SK_Breath_)
  {
    this->setVoiced((MY_FLOAT) 1.0 - (value * (MY_FLOAT) NORM_7));
    this->setUnVoiced((MY_FLOAT) 0.01 * value * (MY_FLOAT) NORM_7);
  }
  else if (number == __SK_FootControl_)
  {
  }
  else if (number == __SK_ModFrequency_)
    _voiced.setVibratoRate(value * (MY_FLOAT) NORM_7 * (MY
  else if (number == __SK_ModWheel_)
    _perf_vib_amt = value * (MY_FLOAT) NORM_7 * (MY_FLOAT)
  else if (number == __SK_AfterTouch_Cont_)   {
    this->setVoiced(value*NORM_7);
  }
  else
  {
    printf("VoicTract : Undefined Control Number!!\n");
  }
}
/*******************************************************
 * Functions to get and set radii of vocal tract sections.
 *******************************************************/
void VoicTract::setTractRadii(const MY_FLOAT *radii, int n
{
  _tract.setRadii(radii, num_sections);
}

const MY_FLOAT *VoicTract::getTractRadii() const
{
  return _tract.getRadii();
}

void VoicTract::setTractSectionRadius(int sec_index, MY_FL
{
  _tract.setSectionRadius(sec_index, radius);
}
/*******************************************************
 * Functions to get and set lengths of vocal tract sectio
 *******************************************************/
void VoicTract::setTractLengths(const MY_FLOAT *lengths, i
  _tract.setLengths(lengths, num_sections);
}

const MY_FLOAT *VoicTract::getTractLengths() const
{
  return _tract.getLengths();
}

void VoicTract::setTractSectionLength(int sec_index, MY_FL
{
  _tract.setSectionLength(sec_index, length);
}
```

```
int VoicTract::numTractSections() const
{
  return _tract.getNumSections();
}
/*****************************************************************/
```

A.3 VocTract.h

```
/*****************************************************************
 * Institution: Stanford University
 * Project: Sonification
 * Author: Ryan Cassidy (05157787)
 * Date: Summer 2003
 *****************************************************************/
/*! \class VocTract
 *  \brief STK class to implement modified Cook tract model.  All references to
 *  PRC's thesis below denote the thesis by former CCRMAlite Perry R. Cook.
 *
 * VERSION CONTROL INFORMATION:
 *
 * $RCSfile: VocTract.h,v $
 *
 * $Author: rjc $
 *
 * $Date: 2003/10/05 05:31:54 $
 *
 * $Locker: rjc $
 *
 * $Log: VocTract.h,v $
 * Revision 1.2  2003/10/05 05:31:54  rjc
 * Added ability to read shapes from file.
 * VocTract::getk() returns current scattering values.
 *
 * Revision 1.1  2003/09/29 21:49:48  rjc
 * Initial revision
 *
 *
 *****************************************************************/

#ifndef __VOCTRACT_H
#define __VOCTRACT_H

#include "Filter.h"
#include "Stk.h"
#include "DelayA.h"
#include "stypes.h"

#include <vector>
#include <cassert>

using namespace std;

#define DEFAULT_NUM_SECTIONS 8

struct ShapeRadii
{
  const char *name;
  const MY_FLOAT radii[DEFAULT_NUM_SECTIONS];
};

class VocTract
{
 public:
  //! Default constructor initializes vocal tract model to
  VocTract(int num_sections = DEFAULT_NUM_SECTIONS);

  //! Class destructor.
  ~VocTract();

  //! Take sample from glottal waveform, and return sample
  MY_FLOAT tick(MY_FLOAT sample);

  //! Return the number of tract sections in the model (de
  //constructor.)
  int getNumSections() const {return _num_sections;}

  //! Set radius of a tract section specified by index (ze
  void setSectionRadius(int index, MY_FLOAT radius);

  //! Set radii of all tract sections.
  void setRadii(const MY_FLOAT *radii, int num_sections);

  //! Get radii of all tract sections.
  const MY_FLOAT *getRadii() const;

  //! Set length of a tract section specified by index (ze
  //length should be specified in samples of delay.
  void setSectionLength(int index, MY_FLOAT length);

  //! Set lengths of all sections.
  void setLengths(const MY_FLOAT *lengths, int num_sectio

  //! Get lengths of all sections.
  const MY_FLOAT *getLengths() const;

  //! Set phoneme.
  int setShape(const char *name, ShapeDataSource sds = SHP

  const MY_FLOAT *getk() const {return _k;}

  void setk(const MY_FLOAT *k, int num_k_vals)
    {
      assert(num_k_vals == this->getNumSections()-1);
      for (int i=0; i<this->getNumSections()-1; i++)
      {
```

"Audio Speech Research Note," Ryan J. Cassidy, published electronically by author, July 2003.

```
        _k[i] = k[i];
      }
    }

  static const ShapeRadii _shp_radii[2];

 protected:
  //! Clock wave data through the positive and negative gain delay lines shown
  //in Fig. 1.5 of PRC's thesis.
  MY_FLOAT tractTick(MY_FLOAT sample);

  static const MY_FLOAT _default_delay_lengths[];
  static const MY_FLOAT _default_section_radii[];

  int _num_sections;
  DelayA * _pos_delay;
  DelayA * _neg_delay;
  MY_FLOAT _lip_refl, _last_lip_in, _last_out, _lip_refl_gain, _glot_refl_gain, _tract_minus;
  MY_FLOAT *_k, *_radii, *_lengths;
};

#endif
```

A.4 VocTract.cpp

```
/*****************************************************************
 * Institution: Stanford University
 * Project: Sonification
 * Author: Ryan Cassidy (05157787)
 * Date: Summer 2003
 *****************************************************************/
/*! \class VocTract
 *  \brief STK class to implement modified Cook tract model.  All references to
 *  PRC's thesis below denote the thesis by former CCRMAite Perry R. Cook.
 *
 * VERSION CONTROL INFORMATION:
 *
 * $RCSfile: VocTract.cpp,v $
 *
 * $Author: rjc $
 *
 * $Date: 2003/10/05 05:30:23 $
 *
 * $Locker:  $
 *
 * $Log: VocTract.cpp,v $
 * Revision 1.2  2003/10/05 05:30:23  rjc
 * Added ability to read shapes from file.
 *
 * Revision 1.1  2003/09/29 21:50:33  rjc
 * Initial revision
 *
 *
 *****************************************************************/

#include "VocTract.h"

#include "Stk.h"
#include "ShpFile.h"

#include <cassert>
#include <string>

using namespace std;

#define DEFAULT_SECTION_LENGTH  ((MY_FLOAT) 1.0)
#define DEFAULT_SECTION_RADIUS  ((MY_FLOAT) 1.0)
const MY_FLOAT VocTract::_default_section_radii[] = {0.928
                                                      1.375
                                                      0.629
                                                      0.568
const ShapeRadii VocTract::_shp_radii[] =
{
  {"aah", {0.821402, 0.783922, 1.09815 , 0.993759, 0.81775
           1.07057 }},
  {"eee", {0.928177, 1.37569 , 1.37569 , 0.679558, 0.62983
           0.662983}}
};

// Above are the default values for the vowel 'eee' as obt
VocTract::VocTract(int num_sections /* = DEFAULT_NUM_SECTI
  :
  _num_sections(num_sections),
  _pos_delay(new DelayA[_num_sections]),
  _neg_delay(new DelayA[_num_sections]),
  _lip_refl(0.0),
  _last_lip_in(0.0),
  _last_out(0.0),
  _lip_refl_gain(-0.45),
  _glot_refl_gain(0.7),
  _tract_minus(0.0),
  _k(new MY_FLOAT[_num_sections-1]),
  _radii(new MY_FLOAT[_num_sections]),
  _lengths(new MY_FLOAT[_num_sections])
{
  // Set tract section lengths to default value.
  for (int i=0; i<_num_sections; i++)
  {
    this->setSectionLength(i, DEFAULT_SECTION_LENGTH);
  }

  // If the number of tract sections is equal to the value
  // default for, initialize the tract radii accordingly.
  if (_num_sections == sizeof(_default_section_radii)/size
  {
    this->setRadii(_default_section_radii, _num_sections);
```

"Audio Speech Research Note," Ryan J. Cassidy, published electronically by author, July 2003.

```
  }
  else
  {
    for (int i=0; i<_num_sections; i++)
    {
      this->setSectionRadius(i, DEFAULT_SECTION_RADIUS);
    }
  }
}

VocTract::~VocTract()
{
  delete [] _pos_delay;
  delete [] _neg_delay;
  delete [] _k;
  delete [] _radii;
  delete [] _lengths;
}

/*****************************************************
 * Functions to handle clocking in and out of samples to the vocal tract.
 *****************************************************/
MY_FLOAT VocTract::tick(MY_FLOAT sample)
{
  // The variable _lip_refl will be used by the upcoming call to tractTick().
  _lip_refl = (_last_lip_in + _pos_delay[_num_sections-1].lastOut())
    * _lip_refl_gain;

  MY_FLOAT temp = _last_lip_in;
  _last_out = temp +
    (_last_lip_in=this->tractTick(sample +
                                  _neg_delay[0].nextOut()) * _glot_radii_ratio);

  return _last_out;
}

// In this function, we implement the delay lines in the discrete-time
// tract model shown in Fig. 1.5 of PRC's thesis.
MY_FLOAT VocTract::tractTick(MY_FLOAT sample)
{
  // First handle positive rail in Fig. 1.5 of PRC's thesis.
  MY_FLOAT pos_out = _pos_delay[0].tick(sample);
  for (int i=1; i<_num_sections; i++)
  {
    // Pass through Kelly-Lochbaum junction.
    MY_FLOAT neg_in = _neg_delay[i].nextOut();

    MY_FLOAT pos_in = (-_k[i-1])*neg_in + (1+_k[i-1])*pos_out;

    // Clock through positive delay.
    pos_out = _pos_delay[i].tick(pos_in);
  }

  // Now clock samples through negative rail of Fig. 1.5 o
  MY_FLOAT neg_out = _neg_delay[_num_sections-1].tick(_lip
  for (int i=_num_sections-2; i>=0; i--)
  {
    // Pass through Kelly-Lochbaum junction.
    MY_FLOAT pos_in = _pos_delay[i].lastOut();

    MY_FLOAT neg_in = (_k[i])*pos_in + (1-_k[i])*neg_out;

    // Clock through positive delay.
    neg_out = _neg_delay[i].tick(neg_in);
  }

  _tract_minus = neg_out;

  return pos_out;
}

/*****************************************************
 * Functions to get and set radii of vocal tract sections.
 *****************************************************/
void VocTract::setSectionRadius(int index, MY_FLOAT radius
{
  assert(index >= 0 && index < _num_sections);
  _radii[index] = radius;

  // Update junction coefficient(s) _k since radii have ch
  // First update the junction coefficient in front of the
  if ((index) < (_num_sections-1))
  {
    _k[index] = (_radii[index+1] - _radii[index]) /
      (_radii[index+1] + _radii[index]);
  }

  // Second update the junction coefficient behind the sec
  if (index > 0)
  {
    _k[index-1] = (_radii[index] - _radii[index-1]) /
      (_radii[index] + _radii[index-1]);
  }
}

void VocTract::setRadii(const MY_FLOAT *radii, int num_sec
{
  assert(num_sections == _num_sections);

  for (int i=0; i<num_sections; i++)
  {
    this->setSectionRadius(i, radii[i]);
  }
}
```

```
const MY_FLOAT *VocTract::getRadii() const
{
  return _radii;
}

/*********************************************
 * Functions to get and set lengths of vocal tract sections
 *********************************************/
void VocTract::setSectionLength(int index, MY_FLOAT length)
{
  assert(index >= 0 and (index) < _num_sections);

  _lengths[index] = length;
  _pos_delay[index].setDelay(length);
  _neg_delay[index].setDelay(length);
}

void VocTract::setLengths(const MY_FLOAT *lengths, int num_sections)
{
  assert(num_sections == _num_sections);

  for (int i=0; i<num_sections; i++)
  {
    this->setSectionLength(i, lengths[i]);
  }
}

const MY_FLOAT *VocTract::getLengths() const
{
  return _lengths;
}

int VocTract::setShape(const char *name, ShapeDataSource sds
                       /* = SHP_FILE */)
{
  bool found_match = false;
  MY_FLOAT *temp_radii;
  ShpFile *sf;

  switch (sds)
  {
  case STRUCT:
    for (unsigned int i=0; i < sizeof(VocTract::_shp_radii)/sizeof(ShapeRadii);
         i++)
    {
      if (string(name) == string(VocTract::_shp_radii[i].name))
      {
        found_match = true;
        // Found a match.
        this->setRadii(VocTract::_shp_radii[i].radii, DEFAULT_NUM_SECTIONS);
      }
    }

  if (!found_match)
  {
    return -1;
  }
  else
  {
    return 0;
  }
  case SHP_FILE:
    sf = new ShpFile(("shapes/" + string(name) + ".shp").c

    if (!(*sf))
    {
      return -1;
    }
    if ((*sf).numSections() != this->getNumSections())
    {
      return -1;
    }

    assert((*sf).numSections() == DEFAULT_NUM_SECTIONS);
    temp_radii = new MY_FLOAT [this->getNumSections()];
    if (!((*sf).getRadii())) return -1;
    memcpy(temp_radii, (*sf).getRadii(),
           this->getNumSections()*sizeof(MY_FLOAT));
    this->setRadii(temp_radii, this->getNumSections());
    delete [] temp_radii;
    delete sf;

    break;
  default:
    assert(false);
    break;
  }

  return 0;
}

/*****************************************************
 * Function to add injection of noise into the vo
 *****************************************************
```

References

[Ben-Tal *et al.* 2001] Ben-Tal, O., M. Daniels, and J. Berger. 2001. "De Natura Sonoris: Sonification of Complex Data.". *Page 330 of:* D'Attellis, C., V. Kluev, and N. Mastorakis (eds), *Mathematics and Simulation with Biological, Economical, and*

Musicoacoustical Applications. Cambridge, MA: WSES Press.

[Chowning 1973] Chowning, J. 1973. "The Synthesis of Complex Audio Spectra by means of Frequency Modulation." *Journal of the Acoustical Society of America*, 21(7):526–534.

[Chowning 1989] Chowning, J. M. 1989. "Frequency Modulation Synthesis of the Singing Voice.". *Pages 57–63 of:* Mathews, M. V., and J. R. Pierce (eds), *Current Directions in Computer Music Research.* Cambridge, MA: MIT Press.

[Cook 1990] Cook, P. R. 1990 (Dec.). *Identification of Control Parameters in an Articulatory Vocal Tract Model, with Applications to the Synthesis of Singing.* Ph.D. thesis, Elec. Engineering Dept., Stanford University (CCRMA). (CCRMA thesis).

[Daniels 2001] Daniels, M. L. 2001. "Research in Sonification and a CLM Vowel Instrument." *Published electronically by author.* Available online at http://www-ccrma.stanford.edu/~danielsm/soni/clmvowel.html.

[Dunn 1950] Dunn, H. K. 1950. "The Calculation of Vowel Resonances, and an Electrical Vocal Tract." *Journal of the Acoustical Society of America*, 22(6):740–753.

[Fant 1960] Fant, G. 1960. *Acoustic Theory of Speech Production.* The Hague: Mouton & Co.

[Fletcher 1953] Fletcher, H. 1953. *Speech and Hearing in Communication.* Princeton, NJ: D. Van Nostrand.

[Klatt 1980] Klatt, D. 1980. "Software for a Cascade/Parallel Formant Synthesizer." *Journal of the Acoustical Society of America*, 67:13–33.

[Netter 2003] Netter, F. H. 2003. *Atlas of Human Anatomy.* Teterboro, NJ: Icon Learning Systems.

[Secretariat 1994] Secretariat, I. T. I. C. 1994. *ANSI X3.226.1994; American National Standard for Information Technology: Programming Language Common Lisp.* Washington, DC: American National Standards Institute. A substantial equivalent to this document is available online at http://www.lispworks.com.

[Simpson and Weiner 1989] 1989. "formant, *n.*". *In:* Simpson, J. A., and E. S. C. Weiner (eds), *Oxford English Dictionary.* Oxford: Clarendon Press. Available online at http://dictionary.oed.com/cgi/entry/00088535.

[Smith III 2002] Smith III, J. O. 2002. *Digital Waveguide Modeling of Musical Instruments.* http://www-ccrma.stanford.edu/~jos/waveguide/.

[Steele 1990] Steele, G. L. 1990. *Common Lisp the Language.* Woburn, MA: Digital Press. Available online at http://www-2.cs.cmu.edu/Groups/AI/html/cltl/cltl2.html.

[Touretzky 1990] Touretzky, D. S. 1990. *Common LISP: A Gentle Introduction to Symbolic Computation.* Redwood City, CA: Benjamin/Cummings Publishing Company. Available online at http://www-2.cs.cmu.edu/ dst/LispBook/index.html.