An Interactive Articulation-to-Area-Function Phonetics Modelling Tool

MÁTYÁS JANI

KTH Computer Science and Communication

Master of Science Thesis
Stockholm, Sweden 2011
An Interactive Articulation-to-Area-Function Phonetics Modelling Tool

MÁTYÁS JANI

Master’s Thesis in Music Acoustics (30 ECTS credits) at the School of Computer Science and Engineering Royal Institute of Technology year 2011
Supervisor at CSC was Sten Ternström
Examiner was Sten Ternström

TRITA-CSC-E 2011:081
ISRN-KTH/CSC/E--11/081--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc
An interactive articulation-to-area-function phonetics modelling tool

Abstract

For speech synthesis, the concatenative approach is currently the most widely adopted, however, it is recognized that concatenated speech has several inherent disadvantages when compared to what might be achieved using articulatory synthesis. As articulatory speech synthesis receives more attention, many articulatory models are developed. While articulatory models may eventually be used for speech synthesis, they are already valuable as research and pedagogical tools. For example, they can be applied to explore formant – cavity relationships and other articulatory aspects of the human sound production system. The main objectives of this work were to rewrite and modernize APEX, one of several current articulatory models, and at the same time to explore the feasibility of using the SuperCollider development environment as an interactive platform for voice modelling. SuperCollider is a programming environment for composition and sound processing. It follows a client-server model, where the client has an interpreted programming language to control the server, which has natively implemented signal processing functions. Initially, only the client side was used, but later to achieve better performance, the time consuming numeric computations were implemented using native code in the server. The architecture of this second version is described in details in the Implementation section. It was found that real-time simulation in SuperCollider is possible, but only if the code is carefully optimized and structured. Basic speed benchmarks are presented in the results section. The resulting software inherits the portability of SuperCollider, so it should be easy to transfer it to other platforms. The architecture makes it easy to change part of the software, for example to implement a new synthesizer.

There are also recommendations for further work, including suggestions for an improved architecture and a discussion of how the project would benefit from a 3D model.
Ett interaktivt verktyg för fonetisk modellering av areafunktioner

Sammanfattning


# Table of contents

Introduction ..........................................................................................................................................................1
Literature...........................................................................................................................................................1
  Human voice production ..........................................................................................................................1
  Speech synthesis methods .........................................................................................................................2
  Overview of articulatory models ..............................................................................................................4
APEX – background and principles.............................................................................................................5
Statement of the Problem / Objectives .......................................................................................................7
Method ...................................................................................................................................................................8
  Articulation .......................................................................................................................................................8
  Area calculation ..............................................................................................................................................10
Implementation ................................................................................................................................................12
  Introduction to SuperCollider .....................................................................................................................12
  Implementing the software ..........................................................................................................................13
    Development approaches .......................................................................................................................13
    Architecture of the solution ....................................................................................................................13
User interface ..................................................................................................................................................17
  Area mapper ...................................................................................................................................................18
    Constructing the midline ..........................................................................................................................19
    Measuring mid-sagittal distances .............................................................................................................20
    Calculating intersection points ...............................................................................................................20
    Mapping distances to areas .....................................................................................................................22
    Input parameters for the area mapper ......................................................................................................22
Synthesizer ......................................................................................................................................................24
  Formant frequency and bandwidth calculation .....................................................................................24
  Source-filter synthesizer ...........................................................................................................................25
    Input parameters for the synthesizer ........................................................................................................25
Results ................................................................................................................................................................27
  Quality factors ..............................................................................................................................................27
  Achieved goals ............................................................................................................................................29
Discussion .......................................................................................................................................................30
  Future work ..................................................................................................................................................30
Acknowledgements ............................................................................................................................................32
References ..........................................................................................................................................................33
Introduction

Literature

Human voice production

The following introduction is adapted from Stevens (1998) [1]. The human speech production system can be divided into several parts with respect to their different functions. The system below the larynx, called the subglottal system, provides the driving air pressure for the speech. The larynx and the supraglottal structures are responsible for the production of audible sound.

The subglottal system is the part of the airways below the larynx, with the trachea at the top. It branches downwards into two smaller tubes called bronchi. These branch further to smaller and smaller tubes. The leaves of this structure are the alveolar sacs. When breathing in, we activate the diaphragm and external intercostal muscles. During inspiration, these muscles contract so as to expand the volume in the thoracic cavity, so that air is sucked into the lungs. On expiration, the contraction of internal intercostal and abdominal muscles decreases the volume of the lungs, so that air is pressed out. The exchanged air volume during normal breathing is much smaller than the vital capacity, which is the maximum expirable air volume after a deep breath. During speech, the lung volume usually changes more than in normal breathing.
The airway is constricted at the larynx. This is where the vocal folds are located. During voiced sounds in speech (phonation), the vocal folds are set into vibration, with a fundamental frequency determined by their tension and stiffness. For a model, one must use a simplified abstraction of the real physical world. In the case of speech production the vocal folds can be considered as a source, as the events in the supraglottal parts are to a large extent independent of this source.

The pharynx is the portion of the vocal tract immediately above the larynx, in the supraglottal area. An almost perpendicular curve follows this vertical tube in normal head position. Under the soft and hard palate the oral cavity can be found. The shape of the vocal tract can be changed by the movement of the tongue, the opening of the jaw and also by changing the shape of the lips. For the production of nasalised sounds, the speaker also lowers the soft velum, opening a port to the nasal cavity.

The acoustic properties of the supraglottal region are governed by the shape of the vocal tract. Assuming plane wave propagation, this part of the speech production system can be modeled as a series of short interconnected tubes with the same cross-sectional area as the real-life shape (at frequencies < 4 kHz, because at higher frequencies the diameter of the vocal tract is in the same magnitude as the wavelength).

**Speech synthesis methods**

There exist several approaches to synthesize human speech. The input for these methods is some kind of textual data (a list of phonemes), often supplemented with some notation of extralinguistic features, for example tone, stress and intonation. The output is a voice signal. The implementation of these approaches can be different in the involved knowledge, in the algorithmic complexity and in the required storing capacity (memory). The involved knowledge can be the functioning of the human voice production system, for a full-scale simulation; or only its output, for what is known as a terminal analogue model. The former usually has more flexibility and sometimes requires less memory, but on the other side the latter does not involve too complex computations. [3]

**Formant synthesis**

In phonetics, a formant is generally understood as one of the resonances in the vocal tract. The resonance frequencies are quite independent of the fundamental frequency and its harmonics. (In general acoustics, a ‘formant’ is any particularly reinforced part of a spectrum; not necessarily caused by a single resonance [4].)
Formant synthesis uses resonant low-pass filters to make the spectrum of the signal similar to what was observed for different vowels and consonants. The base signal is some kind of pulse signal at the fundamental frequency which is passed through filters matching the formant frequency and bandwidth parameters observed in real speech. An advantage of this method is that it is quite simple to implement (although more complex and better sounding solutions exist too) and also does not involve too much computation. On the other hand, the simple versions usually sound less than natural; and there is no direct relationship between the synthesis domain and the physics of the real vocal tract, which makes it harder to alter the characteristics of the result voice without further data gathering.

Concatenative synthesis
Perhaps the most straightforward way to produce speech is to record and play it back. For this method a large collection of recorded speech components is needed. To synthesize a sentence, the required prerecorded components are concatenated together. The selection of units (phones, diphones, triphones) is a matter of choice, with pros and contras for either option. Usually longer units sound better but more samples are required, so with shorter units less storage space is needed. For some usage scenarios, a few predefined sentence structures are sufficient – for example, in a train station speaker – in these cases the concatenative synthesis is usually the best choice. Nowadays, it is the dominant method, but it offers little flexibility and the recording phase is very time consuming. To have a new speaker voice, all components must be recorded again. Another drawback is the disfluency that is easily caused by discontinuous concatenation points.

Articulatory synthesis
The aim of articulatory synthesis is to model the human vocal tract as closely as it is necessary for a required quality of speech. It is clear that a very detailed model based on physical principles will produce high quality results, however it is hard to implement even a modest model, because the computations rapidly become very complex and extensive. The rapid development in computer hardware can be expected to lead to more widespread usage of this method.

Implementations can model the vocal articulators with a set of area functions with rules for parameters like lip and tongue positions, shapes, and so on. Data for these parameters and shapes can in principle be derived from X-ray or MR images, as has been done in the APEX project (more information below).
A major advantage of this method is that it connects the produced sound to a real vocal tract model. For example, in this way, the transitions between different observed positions can be more similar to the real life equivalent. Also it has the advantage that it can be modified using a small set of topological parameters, to sound like a non-existing extraordinary speaker or someone with disordered speech. [5]

**Overview of articulatory models**

There are a lot of recent and on-going research projects about articulatory models, because these may be the future of speech synthesis [5]. This subchapter provides an overview of three projects.

**Vocaltract Lab**

A three-dimensional vocal tract articulatory and control model have been introduced by Kröger and Birkholz [6]. The shapes of the fixed parts have been extracted from volumetric Magnetic Resonance Imaging (MRI) data. The articulation is defined by the position and rotation of the rigid parts and the shapes of the deformable structures. An articulation is described by 24 vocal tract parameters. The method also calculates the midline and measures the cross-sectional areas on planes perpendicular to the midline. The derived area function is mapped to tube sections to build a tube model of the vocal tract. For sound generation this tube model is converted into an electrical transmission line network.

For speech movements, a control gesture based concept has been used. Learning algorithms have been used to extract control rules for achieving more natural sounding results.

**Olov Engwall’s model**

This model has been developed under the KTH 3D Vocal Tract project by Olov Engwall [7]. It is also a 3D model, working very similarly to the Vocaltract Lab. The data has been extracted from MR images. The articulatory parameters are the height of the larynx, the jaw opening, lip protrusion and rounding, velum and five more parameters to define the tongue shape [8]. These parameters affect different vertices of the mesh, with weighted coefficients. The cross-sectional areas are derived from the 3D configuration, using planes perpendicular to the calculated midline. The audible feedback has been generated using an external toolkit called Snack. The quality of the resulting sounds has been measured by listening tests. An interesting result is also that using a 2D model with power equation (see below) for calculating cross-sectional areas from mid-sagittal distances resulted in higher errors in the first two formants, but with smaller errors in the third and fourth formants, than this 3D approach.
APEX

The goal of the original APEX project was to implement an articulation-to-formants tool with voice feedback of the configuration [9]. In the program a virtual 2-dimensional vocal tract is modeled after a real-life one. The geometry of the vocal tract is derived from a large number of X-ray images from an individual subject. APEX makes several steps to generate the sound from articulatory positions. From the positions and state of lips, tongue tip, tongue body, jaw opening and larynx height, APEX constructs an articulatory profile, with an artificial midline which is placed equidistantly between the front and back side of the vocal tract. Then through an applied coordinate system to this profile the mid-sagittal distances are measured at a number of positions along the midline. These distances are then converted into cross-sectional areas using special rules. These cross sectional areas form an area function along the vocal tract which is used to calculate formant frequencies. For sound generation APEX uses speech synthesizer called SENSYN.

In this work the APEX model is implemented so it is described in more detail in the next section.

APEX – background and principles

Data extraction

Articulatory data have been derived from several different sources for the APEX model. For the contours of the vocal tract shape X-ray images have been used [10]. The main problem with the use of X-ray images is the radiation hazard that affects the recorded speaker. Current technologies are better in this respect, but there are stringent safety directives concerning the length and the absorbed dose of an exposure. The accuracy of these traced contours is about 0.5 – 1 mm.

The coefficients for the area calculation have been determined from MR cross-sectional images at several places of the vocal tract [11]. The speech material for the recordings covered a great part of the vowel space of Swedish language. During the MRI acquisition video and audio recordings were also collected.

Exponent mapping of distance to area

The 2-dimensional methods can only use directly mid-sagittal distances because the cross-sectional shapes are unknown for the model. As APEX uses a 2-dimensional vocal tract model it must use some equation to calculate the areas from the observed distances.

The cross-sectional areas can be approximated in several ways, using the cross-dimensions on
the mid-sagittal section with some coefficients [12]. Probably the most widely used one is the so-called power equation by Heinz and Stevens (1964, 1965):

\[ A = K \cdot d^\alpha \]  

(1)

where \( A \) is the cross-sectional area, \( d \) is the cross-dimension perpendicular to the airway, \( K \) and \( \alpha \) are coefficients that depend on the speaker and on the position along the midline in the pharynx. This equation can also be used in the oral cavity but with different \( \alpha \) and \( K \) coefficients.

**Principal component analysis of tongue shape**

The parameters for the tongue body articulation were determined previously by principal component analysis. About 400 tongue contours were obtained from X-ray images and then each was sampled at 25 roughly equidistant points [13]. The principal component analysis results in a linear, weighted combination of a small number of base functions.

The equation obtained by principal component analysis is the following:

\[ V(x) = N(x) + c_1(v) \cdot PC_1(x) + c_2(v) \cdot PC_2(x) + ... \]  

(2)

where \( x \) is the index of the point on the sampled tongue contour, \( V(x) \) is the calculated tongue shape, \( N(x) \) is a “neutral” tongue shape (average of the observed shapes), and \( PC_i(x) \) is the \( i \)-th base function. The coefficient denoted by \( c_i \) is the weight for the \( i \)-th base function. \( c_i \) is a 2-dimensional vector value which depends on the vowel, and it is used as a parameter for calculating the tongue shape.

Accuracy: using a single PC base function is found to account for 85.7% of the variance, while two principal components achieved 96.3%. [13]
Statement of the Problem / Objectives

Why a 2-dimensional approach?

Recent projects on articulatory synthesis have usually taken a 3-dimensional approach, which is very useful for acquiring cross-areas directly. While this is a very important advantage, the old APEX project also had some advantages which made it unique. The original goal of the APEX project was not to make a synthesizer, but a research and pedagogical tool for researching the connection between articulation and formants. The synthesizer part was only for audible feedback. The model tries to address the formant-cavity relationship with control parameters that are physiologically true. It works with realistic deformation of the tongue and realistic movements of the jaw and the lips rather than just geometrical transformations. The formant-articulator relationship provides some natural constraints on the class of 'humanly possible' area functions. The APEX project also has access to a rich source of speaker specific data (MRI, X-ray, other measurements).

The main goal with this project was to try to implement APEX model in a modern environment, SuperCollider, and to explore whether or not SuperCollider is a suitable environment for this kind of problem. The 3D model would have been unnecessary for these goals and it would have been much more complex to implement than the 2-dimensional model.

Problems with the legacy implementation of APEX:

1. it has been abandoned, and there have been updates on the model and on the background data
2. the source code has become obsolete (Win16/MFC) and difficult to maintain
3. it was platform dependent

Objectives for this new implementation of APEX:

1. incorporate new available measurements and models for calculation of area functions
2. make it platform independent
3. separate the synthesizer part into a different module for the possibility of using alternative synthesizers (source-filter, waveguide)
4. documentation for further development
5. to explore if SuperCollider is a suitable environment for this work
Method

Articulation

The voice production apparatus consists of several parts, some of which can be considered fixed: the posterior pharyngeal wall (“back wall”), the hard palate and upper jaw; and others which are movable: the larynx with the vocal folds, the pharyngeal side walls, the epiglottis, the tongue with body and tip, the soft palate or velum, the mandible or lower jaw, and the lips.

By “articulation” the phonetician means the positioning of the movable parts, which results in a characteristic geometry of the vocal tract for each different phoneme spoken. There are many components and hence many degrees of freedom. We seek only the phonetically relevant positions, which are a subset of all possibilities, so it is important to find a small set of parameters which can describe this geometry efficiently.

The parameters and the methods for the articulation are inherited mainly from the old APEX. Typically, there is a neutral shape for the organs which is extracted from X-ray or MRI data, and then with some algorithm these shapes can be modified to the required articulation.

In the present model, the articulation is 2-dimensional, and it consists of several parts: the back wall shape, the larynx, the tongue body, the tongue blade, the tongue tip, and the movement of the jaw.

Back wall

This is the dorsal side of the 2-dimensional vocal tract from the larynx to the palate. It is considered as a fixed shape and its coordinates have been extracted from observed data. These data are from X-rays of one individual subject.

Larynx

The shape for the larynx has been extracted from observed data. This shape, too, is fixed, but the vertical position is adjustable. The height (vertical position) of the larynx varies a little during speech, but most importantly it varies between different speakers. Females typically have a higher larynx position (and children even higher) and a shorter vocal tract which causes the formant frequencies to be scaled up relative to the male voice.
**Tongue body**

The shape of the tongue body is the most deformable part of the vocal tract. The control parameters have been determined by principal component analysis as was described in the Introduction. Three base functions have been used in the final model.

**Tongue tip, tongue blade**

The point where the tongue blade under side is connected to the mouth floor was chosen as a fixed point in the lower jaw coordinate system (point $C$ in Figure 2). The tongue body ends at point $A$, and the tongue tip is at point $B$ in the figure. For the upper contour of the tongue blade Hermite’s interpolation was used between point $A$ and point $B$. To make the connection smooth, the first derivative of the two connected curves are set to be equal at point $A$. For the lower contour, observed data was transformed to match the ending points, $B$ and $C$.

![Figure 2 Tongue blade configuration. The thin continuous curve is the lower jaw, the thin dashed curve is the tongue body, the dashed curve between point $A$ and $B$ is the upper side of the tongue blade, and the fine dashed line between $B$ and $C$ is the under side of the tongue blade.](image)
Jaw movement

The jaw movement is the translation and then the rotation of the lower jaw coordinate system. It moves some part of the ventral (frontal) vocal tract including the tongue body and blade as well as the mouth floor and teeth. The angle for the rotation is determined by the following equation:

\[ \alpha_{\text{deg}} = \frac{j}{2} + 7 \]

where \( \alpha_{\text{deg}} \) is the angle in degrees, and \( j \) is the jaw opening, i.e., the distance between the upper and lower teeth, in mm. The effect is shown in Figure 3. The blue curve (the outer curve) is the back part of the vocal tract (fixed); the point \( U \) is the origin of the upper-jaw coordinate system. If the jaw opening is set to \( j \), then the distance between \( U \) and the lower-jaw origin \( L \) equals to \( j \). All the denoted angles in the figure equals to \( \alpha \). The red dashed curve (inner curve) is the original shape of the tongue body, translated by \( j \) to the correct position, and the solid red curve (between the dashed and the outer curve) is the rotated tongue body.

![Figure 3 Effect of the jaw opening.](image)

Area calculation

The area function is an abstraction of the vocal tract, which represents the cross-sectional areas as a function of distance from the larynx. It is a very useful representation of the vocal tract in that; although some information is lost (the exact curve), it preserves the acoustic properties quite well, thus it can be used for voice signal generation. A limitation of using a single area function is that it
can not handle side branches, such as the nasal cavities, so for that some extension would be needed.

When the articulation is defined and a vocal tract configuration is set, it is possible to calculate the area function from the geometry. It is not obvious how to go along the tract to retrieve the cross-sectional areas. The approach used in this work is to create first a curve called middle line or simply midline, which should be halfway between the anterior/front (larynx, tongue body, tongue blade, lower jaw) and the posterior/back side of the vocal tract, and then draw perpendicular lines to this midline at equidistant positions. The cross-sectional distances are the lengths of the sections on these lines from the intersection points with the front and the back shapes. To calculate the areas the power rule was used (see Introduction).
Implementation

Introduction to SuperCollider

SuperCollider is a platform independent programming language and server-client environment for real time audio processing and synthesis. It is an open source project licensed under the General Public License (GPL) and it is freely available [14]. It is widely used by scientists, artists and musicians because it is good for algorithmic composition as well as for signal processing. The project has online documentation and tutorials, and very recently an “official” handbook was published, The SuperCollider Book [15].

SuperCollider has a server-client architecture, which makes it possible to separate the controlling client(s) from the sound generating and processing server(s). It has some benefits: multiple clients, remote control over network/internet; and also some drawbacks: latency, asynchronous execution. The communication protocol between the client and the server is a modified version of the Open Sound Control (OSC) protocol, a protocol for communicating between synthesizers and multimedia devices [16]. Thus it is possible to create a customized client (or server) for special purposes and use it with the original server (or client) using this protocol for communicating.

The client provided by the SuperCollider community contains an interpreter for the programming language called SCLang. The language itself is an object-oriented Smalltalk-like language with special extensions for composition and audio synthesis. The language has some interesting control structures for these cases. For example, there are special functions which save the exit point on return and on the next call the execution continues from there. Also there are a lot of built-in classes for data storage in memory (collections), for creating a graphical user interface (gui) and for generating and modifying sound signals. The latter has a large collection of sources, filters, delays, buffers, etc. These are called unit generators (UGens), and the real implementations of the UGens are compiled into the server, so in the client there are only interface classes for them.

It is also possible to extend the class library by writing new classes, however before using them the whole class library has to be recompiled, which is not possible at run-time. Other things, like writing new or modifying existing functions can be done at run-time, for instance, at live performances. The interpreter also has a garbage collection feature.
The server is written in C++ and can be extended, for example by writing new UGens. The UGens are compiled as libraries which can be loaded dynamically at startup. This raises issues of portability, as a compiled library may not work under different operating systems and on different hardware; it must be recompiled for each target platform.

Concerning the graphical user interface, the client has some built-in classes to create windows, buttons and other widgets in a platform-independent way. The implementing GUI kit may depend on the platform where the program is executed, and the GUI classes are redirected to the platform specific implementations. At the time of this project, there were two GUI kits available – Cocoa for Mac and Swing for other platforms – but in coming development versions there is a third – Qt – which can be used on all three platforms.

Implementing the software

Development approaches

Software development can basically be approached in two ways. One way is to define the exact requirements and then try to divide the task into smaller sub-tasks. This approach is called Top-Down (TD) software development. The other way is to start from basic blocks and try to reach the proposed object by combining the building blocks. This is called the Bottom-Up (BU) approach.

Software developed by TD approach can usually meet more of the starting criteria. On the other hand the resulting software is hard to maintain, and needs a lot of effort to meet new or changed requirements.

When using BU approach, it is harder to reach the exact desired goals, and it is possible that the resulting software will not meet the starting criteria. However, in the long-run it is much easier to change the software and maintain it, and probably the user can be tempted to accept the software as it is. [17]

In the case of this project I tried to do the work in a BU way. Due to the limited time and unsure success some TD approach was brought in. For the basic components – like vector and matrix operations, paths of curves and geometry computation – it was easy to follow the BU way. For more complex parts it was perhaps less successful.

Architecture of the solution

In software technology, modularity has a great impact factor on various quality criteria [18]. It usually increases maintainability, re-usability and many other factors as well. The software system is modular
when the cohesions in the components are high and the coupling between them is low. In some cases it can affect performance in a negative way and also the design time can be longer for such systems, because the interface between the modules must be designed carefully. In the case of this project the available time was limited, and also the environment had limitations. Still, several attempts were made to make it modular.

I tried to make a distinction between the computation, control and user interface part. This is called the Model – View – Control architecture: the Model is the part where the background computation is done, the View is responsible for the user interface and graphical output, and the Control puts it all together and controls the processes.

For the construction of the modular architecture, the concept was to define a workflow to find the basic components and the dependencies between them. In Figure 4 the first box is the GUI input, where the user sets the parameters needed to calculate the shapes of the vocal organs and for the synthesis. The second box calculates the area function from the geometry of the shapes. This module contains several subtasks, which will be discussed later. After the area function is available, there are multiple possibilities for voice synthesis. Two examples are a digital waveguide mesh and source-filter synthesis. The former is rather a simulation method and it was a research topic at KTH; there are two recent reports about this topic [2][19]. The source-filter method was also a research topic for a long time; and it was used by the old APEX for sound generation. This method was implemented in the program. The synthesizer part has subtasks, such as calculating formant frequencies and bandwidths from the area function, and the synthesis itself.
In practice the architecture is somewhat more complex. There are input parameters to the synthesizer from the GUI, for example fundamental frequency or the propagation speed of the sound wave. For the visualisation – which is very important since one of the goals is for APEX to be a pedagogical tool – the states of the shapes and the formant frequencies have to be loaded back to the GUI.

Initially, the whole program was implemented in SCLang only. SClang is an interpreted language, and as such it is not very efficient at computations. Indeed, the resulting code turned out to be too slow. I therefore had to re-implement the arithmetically complex parts in the server, as UGens. The problem with this second approach is that the UGens are part of the server and the SCLang is on the client side, and the communication between them is slow.

**Communication between server and client**

The software was separated into two parts. The processing part runs on the server and the GUI and control part on the client. The communication between the server and client is asynchronous, thus commands for example for sending data to the server are not executed immediately when the control flows to the next command in the client.

There is a client command, called `sync` (it is a method of the Server object in the client), for waiting until the server has processed the last sent commands. This can only be used in special control structures, to avoid hanging the main thread. These control structures are the Tasks and

---

*Figure 4 Architecture of the solution.*
Routines. They behave similarly, and because I used the Routine structures, I will describe that here.

Routines are special functions with a special return command called `yield`. Using this for returning from a Routine will store the last position inside the Routine. On the next call it will continue executing the commands in the body of the Routine after the last yield. When starting a Routine with the `run` method, it starts a different thread, which can be paused for a given time interval with the `yield` command. To synchronize the server and the client with the `sync` method, it must be called inside a Routine; the thread of that Routine will be paused until the synchronization is done. It is important to know that the execution will immediately continue after the Routine in the main thread, so commands after the Routine may be executed before any synchronization is done inside the Routine. It is also very important to know that calling other functions or methods inside a Routine will run in the Routine’s thread, so it is possible to use `yield` or `sync` inside those functions or methods too (but of course then those functions or methods can not be called from the main thread, because those commands can not be used in the main thread).

The easiest way to transmit data between the server and the client is to use buffers. Buffers are float arrays stored on the server. They have a header to store the size and other properties of the buffer, but the data is in a simple float array. The buffers have unique identifiers in the server (which are 32-bit unsigned integers). In the client it is possible to access the buffers stored in the server using the Buffer class. Using `sync` after a buffer operation ensures that the state of the buffer in the server is up-to-date, but using it after every buffer operation may slow down the program, so it is important to use it only when it can not be avoided.

The first implementation using only SCLang had the advantage, that the communication between the server and client was minimal (see Figure 5). In the current version all the parameters and the calculated shapes must be transmitted using OSC messages between the server and the client part (see Figure 6).

![Figure 5 Communication between the server and the client in the first version.](image)
There are two different threads handling the communication. The first loads the shapes from the server to visualize the vocal tract, the second sends all the parameters and loads back the formants. There is a third thread to update the formant frequencies displayed, but it does not involve communication with the server. The refresh rates are stored in configuration files, and are held back if the system is slow.

User interface

The Graphical User Interface (GUI) was written in SCLang. A vector input widget was implemented for input parameters, and a display widget for showing different shapes. The present GUI is rudimentary, and intended only for testing purposes.
Figure 7 The graphical user interface.

Area mapper

The task of the area mapper is to calculate an area function for a given articulation. The neutral shape and state of the voice organs are loaded from files and are given to the area mapper as construction time parameters. Other input parameters which are needed at every calculation step are the descriptors of the current positions of the articulators. These parameters can be found in the next section.

When the articulation geometry is calculated, a midline is constructed between the front and the back part of the vocal tract (red, middle line in Figure 8).
Constructing the midline

First we need to define an ‘artificial midline’ curve (this is not the real, resulting midline shown in Figure 8). It is inside of the curve of the back wall (outer black curve in Figure 8) and roughly parallel to it (in reality it is vertical in the lower parts, then it is a quarter circle and then it is horizontal in the oral cavity). When the UGen is created this ‘artificial midline’ curve is passed to the constructor. The constructor of the UGen puts perpendicular planes to this curve at equidistant positions (green on the picture). The number of these planes can be changed by input parameters (see next section, “Planes num”). These planes are treated as segments in reality, and their lengths can be changed by input parameter as well (“Planes width”).

First, the algorithm finds the intersection points of these planes (segments) and the back wall, and puts those points into an array (array $A$). The back wall is considered as a fix shape, so these points are constant, and array $A$ can be calculated at construction time. At runtime, the intersection points between the planes and the front shape (blue) are calculated too, for the given articulation, and stored in array $B$. $A[plane_i]$ is the intersection point of the $i$-th plane and the back wall, $B[plane_i]$ is the intersection point of the $i$-th plane and the front shape. Then the point halfway between $A[plane_i]$ and $B[plane_i]$ will be the $i$-th point of the real midline.
**Measuring mid-sagittal distances**

As mentioned earlier, the distances are measured at equidistant positions on the midline. The distance between two measurement positions can be changed by changing the “Resolution” input parameter (so it is a spatial distance parameter, smaller means more points, higher resolution). It will of course affect the number of measurement positions.

The algorithm is quite similar to the midline constructing algorithm. At the specified positions put a perpendicular line to the midline, and then find the intersection point pairs between this line and the front shape then between this line and the back shape. These point pairs define segments (yellow short segments in Figure 8), and the length of these segments are the mid-sagittal distances at the specified positions.

In the final version these segments are not perpendicular to the midline, but rather perpendicular to a segment between the current and next measurement point on the midline. This makes a difference only when the two measurement points are on different segments of the midline. Using this modification, it is less likely for two yellow segments to intersect each other (but it occurs sometimes).

**Calculating intersection points**

The previous algorithms use a lot of intersection calculation. The intersection point between two lines can be calculated by making the formula of the first line equal to the formula of the second line and then by solving this equation. More steps are required to find the intersection point between a line and a segment, because the intersection point can be on the line of the segment but not on the segment.

Checking whether point $Q$ of a line is between two points $(A$ and $B)$ of the line can be done by checking the sign of a dot product $\langle q-a; q-b \rangle$ (here $q$ is the vector pointing to $Q$, and so on). If it is negative, they point in the opposite directions and $Q$ is between $A$ and $B$ (see part 1 of Figure 9). If the sign is positive, it means that $q-a$ and $q-b$ point in the same direction and the point is outside of the segment $\overline{AB}$ (see part 2 of Figure 9).
There is a faster way to check whether a given line intersects a given segment. First check whether the two points of the segment are on the opposite side of the line. If they are not, then there is no need to calculate the exact intersection point. We need a perpendicular vector \( \mathbf{n} \) to the line, then calculate the vectors \( \mathbf{a}' \) and \( \mathbf{b}' \) between a point of the line \( \mathbf{P} \) and the two points \( \mathbf{A} \) and \( \mathbf{B} \) of the segment \( a' = a - p \); \( b' = b - p \). If the signs of the dot products \( \mathbf{n} \cdot \mathbf{a}' \) and \( \mathbf{n} \cdot \mathbf{b}' \) are not equal, then the points are on the opposite side of the line, so there is an intersection between them (see Figure 10). This is because if the angle between two vectors is smaller than \( 90^\circ \) then the dot product is positive, otherwise it is negative. If it turns out that the segment intersects the line, then the intersection point calculation is done.

To find the intersection points between an array of lines and a curve, the previous algorithm should run on all segments of the curve with all lines. To speed up the process I tried to order the lines and go along the curve only once, but unfortunately it was not successful. It happened
sometimes that a mid-sagittal line which was farther away along the midline from the larynx intersected the front or back path before a closer line.

It happened also that it was not enough to find one intersection point in some cases. For example in the case of the mid-sagittal distances it is the best to find the closest intersection point to the measurement point on the midline, hence more than one intersection point may occur (see Figure 11). So the current version checks all segments of the front and back curve with all lines.

**Mapping distances to areas**

It was described in earlier sections that APEX uses the so-called power rule (here I use $\alpha$ and $\beta$ instead of $K$ and $\alpha$):

$$A = \alpha \cdot d^\beta$$

(4)

The coefficients have been measured using 11 cross-sectional MR images (see introduction chapter) so there are 11 different $\alpha$ and $\beta$ values for 11 different positions. To find a coefficient at any given position between two of the 11 exact positions, a linear interpolation is done between them. It is quite easy to modify the source code to specify different distance-to-area rules for different positions.

**Input parameters for the area mapper**

The input parameters have been determined by previous works and the old APEX project. Some parameters affect the articulation, like the position of the tongue tip, three parameters for the tongue body shape (determined by principle component analysis, see in introduction chapter), one parameter for the larynx height and one for the jaw opening. These can be changed at run-time and will affect the area function. The parameters are summarized in Table 1.
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Value limits</th>
<th>Def. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tongue tip</td>
<td>2D Vector</td>
<td>(mm; mm)</td>
<td>Must be inside mouth</td>
<td>(15; 5)</td>
</tr>
<tr>
<td>Tongue c1</td>
<td>2D Vector</td>
<td>none</td>
<td>Varies on other parameters</td>
<td>(-33.1; 32.1)</td>
</tr>
<tr>
<td>Tongue c2</td>
<td>2D Vector</td>
<td>none</td>
<td>Varies on other parameters</td>
<td>(7.5; -3.3)</td>
</tr>
<tr>
<td>Tongue c3</td>
<td>2D Vector</td>
<td>none</td>
<td>Varies on other parameters</td>
<td>(1; 1)</td>
</tr>
<tr>
<td>Larynx height</td>
<td>Real</td>
<td>mm</td>
<td>-10 … 15 mm</td>
<td>7.6 mm</td>
</tr>
<tr>
<td>Jaw opening</td>
<td>Real</td>
<td>mm</td>
<td>0 … 25 mm</td>
<td>6.8 mm</td>
</tr>
</tbody>
</table>

*Table 1 Input parameters for the articulation. The default values are for the vowel /i/.*

To eliminate memory allocation in the UGen (only allocate memory at construction and free them only at destruction) I had to use arrays with fixed size. This implies that parameters which affect array sizes may be changed only before construction. The “Planes width” and “Skip” parameters do not affect array sizes, but after finding good values for them there is no need to change.

In SuperCollider the processing functions of the UGens are called at fixed time intervals (in the case of control rate UGens, like the area mapper and formant calculator, the default time interval between calls is 1 ms). The area function calculator UGen will skip the number of calls provided by the parameter “Skip” between two calculations. Changing it to a higher number, the refresh rate of the output area function will be rarefied. Changing it to a lower number may hang the application, because it may require more time to calculate the output than the time interval between two calls. The minimum number for this parameter depends on the speed of the computer.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Value limits</th>
<th>Def. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planes num</td>
<td>Integer</td>
<td>None</td>
<td>10 … 35</td>
<td>20</td>
</tr>
<tr>
<td>Planes width</td>
<td>Real</td>
<td>mm</td>
<td>60 … 110</td>
<td>80</td>
</tr>
<tr>
<td>Resolution</td>
<td>Real</td>
<td>mm</td>
<td>1 … 13.5</td>
<td>3</td>
</tr>
<tr>
<td>Skip</td>
<td>Integer</td>
<td>None</td>
<td>1 … 37</td>
<td>20</td>
</tr>
</tbody>
</table>

*Table 2 Construction time input parameters (They are called setup parameters in the program).*
There are other parameters, which cannot be changed for the GUI yet, see Table 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Value limits</th>
<th>Def. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tongue_over</td>
<td>Integer</td>
<td>None</td>
<td>&gt;3</td>
<td>8</td>
</tr>
<tr>
<td>Fps</td>
<td>Real</td>
<td>1/sec</td>
<td>≥0</td>
<td>10</td>
</tr>
<tr>
<td>Fname</td>
<td>Array of Strings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>larynx, tongue_n, …</td>
<td>Integer</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 Other parameters for the area mapper.

The “tongue_over” stores the number of points to be generated by Hermite interpolation for the upper part of the tongue blade. The “fps” parameter sets the refresh rate (frame per second) for the vocal tract visualisation. If it is 0, the vocal tract will not be shown. If the value is too high it will not cause troubles; the resulting refresh rate is constrained by the system capabilities. File names are stored in the “fname” parameter; these are the names of the data files containing the shapes, planes, curves and PCA values. The input file contains other numeric parameters, with names like “larynx”, “tongue_n”, these are simply indices for the “fname” array. So for example the name of the data file containing the shape of the larynx is “fname[larynx]”.

**Synthesizer**

For getting audible feedback, a basic source-filter synthesizer was implemented. The source-filter synthesizer utilizes several formant-filters (second order low pass filters) on a source signal to generate the result signal. The resonance frequencies and bandwidths for the filters (which are the formant frequencies and bandwidths) must be determined from the output of the area mapper, the area function.

**Formant frequency and bandwidth calculation**

The algorithm for the formant frequency calculation is based on the volume velocity transfer through the vocal tract; it is called transfer method [20]. It sweeps through a frequency range and searches for peaks of a calculated function.

There are equations to derive the bandwidths from the formant frequencies [21]. Unfortunately in the paper there was a typographical error. The corrected equations should be these:
\[ B_1 = 15 \cdot \left( \frac{500}{F_1} \right)^2 + 20 \cdot \left( \frac{F_1}{500} \right) + 5 \cdot \left( \frac{F_1}{500} \right)^2 \]

\[ B_2 = 22 + 16 \cdot \left( \frac{F_1}{500} \right)^2 + 12000 \cdot (F_3 - F_2) \]

\[ B_3 = 25 \cdot \left( \frac{F_1}{500} \right)^2 + 4 \cdot \left( \frac{F_2}{500} \right)^2 + 10 \cdot \frac{F_3}{(F_{4a} - F_3)} \]

where \( F_1, F_2 \) and \( F_3 \) are the first three formants and \( F_{4a} \) is a chosen value which is 3400 Hz for male speakers and 3700 Hz for female speakers. The first three formants are sufficient for speech synthesis under 5 kHz, but not for singing.

**Source-filter synthesizer**

The current synthesizer was implemented for testing purposes only. It is quite simple and uses only the formant frequencies because there have been some errors in the formulas for the bandwidth calculation.

The source for the synthesizer is a sawtooth signal that is modulated by fixed time envelopes in amplitude and frequency. The peak frequency of the signal is the fundamental frequency (\( f_0 \)), but it is scaled down through time, to make it more like a real speaker with breathing. The amplitude of the signal is also changing by time to improve that effect. The downscaling is done by smooth envelopes. The implementation in SuperCollider uses the “Saw” UGen. The signal is then filtered by resonance low pass filters with the formant frequencies. The filter UGen is called “RLPF” in SuperCollider.

**Input parameters for the synthesizer**

In the current version of the program the only run-time changeable input parameter for the synthesizer is the fundamental frequency (see Table 4).

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Value limits</th>
<th>Def. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>Real</td>
<td>Hz</td>
<td>10 … 110</td>
<td>110</td>
</tr>
</tbody>
</table>

*Table 4 Input parameters for the synthesizer.*
There are construction time parameters for the synthesizer too. The “Skip” parameter is the same as in the area mapper. The parameter called “sex” changes whether the speaker is male or female; it is only used by the bandwidth calculation, so it has no effect at the moment (the bandwidths are not used currently).

The “Speed of sound”, “Internal end correction” and “Open at left” parameters are needed for the formant frequency calculator algorithm. The “Internal end correction” makes tubes longer which were followed by a tube with larger radius in the tube model of the vocal tract. This corrects some aero dynamical effects. The “Open at left” indicates that the first tube is opened (at the larynx side, so on my visualization it is the right side).

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Value limits</th>
<th>Def. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sex</td>
<td>Integer</td>
<td>none</td>
<td>0 (female) / 1 (male)</td>
<td>1</td>
</tr>
<tr>
<td>Speed of sound</td>
<td>Real</td>
<td>cm/sec</td>
<td>30000 ... 50000</td>
<td>34300</td>
</tr>
<tr>
<td>Internal end correction</td>
<td>Integer</td>
<td>none</td>
<td>0 (no) / 1 (yes)</td>
<td>1</td>
</tr>
<tr>
<td>Open at left</td>
<td>Integer</td>
<td>none</td>
<td>0 (no) / 1 (yes)</td>
<td>0</td>
</tr>
<tr>
<td>Skip</td>
<td>Integer</td>
<td>none</td>
<td>1 ... 30</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5 Construction time parameters for the synthesizer.

Other parameter, which cannot be set from the user interface yet can be found in Table 6.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Value limits</th>
<th>Def. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffersize</td>
<td>Integer</td>
<td>none</td>
<td>≥4</td>
<td>20</td>
</tr>
<tr>
<td>fps</td>
<td>Real</td>
<td>1/sec</td>
<td>≥0</td>
<td>10</td>
</tr>
<tr>
<td>amplitude</td>
<td>Real</td>
<td>None</td>
<td>0 ... 1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 6 Other parameters for the synthesizer.

The first, “buffersize” determines the size of the output buffer for the formant calculation, so the number of calculated formant frequency and bandwidth pairs will be smaller than or equal to this. The current synthesizer uses only 4 formants (and only the frequencies, bandwidths are discarded). The “fps” is the refresh rate for the formant graphical output, and the “amplitude” is the amplitude of the source signal before applying the filters.
Results

The model was mainly inherited from the previous APEX. The result of my work is mainly the implementation of the software.

Quality factors

There are several factors of software quality, which can be divided into two groups: external and internal quality factors. The external quality factors are those, which are important for the users, like integrity, reliability, usability, accuracy. The internal quality factors describe the software from the viewpoint of the programmer or maintainer, and affect the user experience only indirectly. External factors are: efficiency, maintainability, testability, flexibility, interface facility, re-usability, and transferability. These factors can affect each other in an inverse way, so improving the software in one factor may hold it back in others [22]. Some of these factors which could be measured are described in this section.

Efficiency

It is “The volume of code or computer resources (eg. time or external storage) needed for a program so that it can fulfil its function.” McCall et al [22].

The memory requirement of the program is quite low compared to the available memory sizes, therefore it is not so important to analyze. The time requirements are however a primary focus for real-time synthesis.

It is hard to compare the earlier version (which was written only in SCLang), and the new version (which uses server sided UGens as well), because of the different architecture. In the old version the calculation of the area function is only done when at least one of the input parameters have been changed. For speed comparison the time required for one “step” (for one area function and one formant array calculation) have been measured.

<table>
<thead>
<tr>
<th>SCLang version</th>
<th>Resolution = 3</th>
<th>Resolution = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required time:</td>
<td>0.084 s</td>
<td>0.208 s</td>
</tr>
</tbody>
</table>

Table 7 Time required for one calculation step in the earlier version.
In the current version the updates are persistent, running in different threads. The refresh rates of the UGens are quite high, the bottleneck is rather the update of the parameters, to send them to the server and then receive the results. So in the case of the current version the time difference between two updates has been measured.

<table>
<thead>
<tr>
<th>UGen version</th>
<th>Resolution = 3</th>
<th>Resolution = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Delta T), with sync</td>
<td>0.096 s</td>
<td>0.119 s</td>
</tr>
<tr>
<td>(\Delta T), without sync</td>
<td>0.049 s</td>
<td>0.055 s</td>
</tr>
</tbody>
</table>

*Table 8 Time difference between two updates in the current version.*

With synchronization the client will wait until the server verifies that it has received the data, but this requires more time.

The values are the average values of several measurements in both tables. The measurements have been made on a 2.6 GHz Pentium 4 computer with 1 GB RAM and under the Ubuntu 10.10 operating system.

**Transferability**

This factor describes “The cost of transferring a product from its hardware or operational environment to another.” McCall et al [22].

With SuperCollider it is quite easy to develop in a cross-platform way. The client part could be run on every platform supported by SC. In the server part no external libraries have been used, so it could be compiled on any platforms supported by SC, but it has not been tested yet.

**Maintainability and re-usability**

Maintainability means how much trouble it is to find a bug and to fix it. It depends on eg. the documentation and on module coupling and cohesion. Also it covers the consistency of variable and function naming convention and other quality measurements of the source code. This latter part can be improved by refactoring. There have been some efforts done to modularize the software. For example the source filter synthesizer part can be used quite easily without using any other parts of the program. The whole synthesis and also the area mapper part can be replaced by another solution, the only criteria is to use the same interfacing buffers. The quality of the source code and the documentation could be improved.
The re-usability is a measure of how easy it is to use parts of the program (e.g. modules) in a different application. Using the same buffer interface, the different parts can be used in other applications, however I think there are not so many use cases e.g. for the area mapper part. Some basic classes, for example the vector algebra and graphics and different shape or path calculation parts could be re-used on their own in other software.

Achieved goals

The result software is now a platform independent solution, using some recent changes to the APEX model and with the new measurements. A modular architecture has been designed; the synthesizer part is separated from the rest of the software, so it is easy to add an alternative synthesizer without changing much in other parts of the source code.

The most important task was to explore whether SuperCollider is a suitable environment for implementing interactive voice simulations. It was demonstrated that although SuperCollider was not ideal for articulatory modelling, it is possible to use with careful program design.

Some APEX features are missing from the program, so it should not be considered as complete implementation. These are the front cavity rules, which are very important for the 3rd formant.
Discussion

The project aim, to re-implement APEX in the SuperCollider environment, was partially met. SuperCollider was found to be sufficient for this kind of work, with the ability to develop GUI in an interpreted, object oriented language, as well as to use C++ for arithmetic intensive code. There are several possibilities to continue or improve this work.

Future work

Implement the missing parts

The oral cavity rules and lip rounding is missing from the current implementation, due to their unexpected complexity and the lack of time. When every part the APEX model is implemented, it would be nice to validate the accuracy, for example by comparing the calculated formants values to formants in real life recordings.

Playing pre-recorded positions

It would be desirable to be able to specify positions in a configuration file with timing, and then the program would be able to play that file with visual and sound output. Fortunately it is not too difficult to extend the current architecture with this feature.

Other architecture

For GUI programming SC is quite comfortable, but for numeric calculation it was found to be slow. By separating the client and server, extra communication is needed to transfer the shape of the articulation from the server to the client. Using a separate (native, non-interpreted) application for articulation geometry, area function and formant calculations, it would be faster. The visualization could be done in that application directly. It would be possible to use SC for the synthesizer only, sending the formant data by OSC messages from the separate application to the SC server. The communication would be much the same as it was with the first implementation (see Figure 12).

However, it would require learning the OSC messaging, and it would be more difficult to write GUI and process input files, especially in a cross-platform way.
3D vocal tract

While other recent projects use 3D vocal tracts, APEX still uses 2D. This causes few problems for the posterior parts of the vocal tract, but the proposed oral cavity area calculation rules would be quite difficult to implement in 2D, just to avoid 3D calculations. APEX has access to a big collection of X-ray and MRI materials and it has other advantages. It would probably be a good idea to consider the translation of the model to 3D.
Acknowledgements

I would like to thank Sten Ternström, my supervisor, who helped me a lot with the report and also with implementing the synthesizer part of the software, and also for making it possible for me to do this project in Stockholm. I would like to thank Björn Lindblom for his patience in explaining the APEX model to me. Thanks to Gerhard Eckel and Ludvig Elblaus for helping me with SuperCollider. Thanks to György Takács for contacting KTH in the first place and helping me with his advice.

Also it was really helpful to work in a friendly environment. The (almost) weekly innebandy games, the lunch and coffee time conversations, music performances made me feel at home. Thanks to the researchers, guest researchers, PhD students, master students, and other people working at the department (Anders, Anton, Björn, David, Gunnar, Gæl, Johan, Kjetil, Laura, Marco, Raveesh, Roberto, Roxana, Sofia ...).

I am also grateful to the friends from Hotel Månen (especially Dániel Pelyhe), to the friends from St:Eugenie Youth Choir and student’s group, to my neighbour, for the free-time activities, singing and sport activities. Special thanks to my renter, Christopher, who provided me placement for a modest price.

I would like to thank my friends and family, for their continuous support and encouragement.
References


