Vani-An Indian Language Text To Speech Synthesizer

Harsh Jain

Varun Kanade

Kartik Desikan

Final Stage Report

Vani Team
Department of Computer Science and Engineering
Indian Institute of Technology Mumbai
India

April 2004

Dedicated to

Our beloved country...

Vani - An Indian Language Text To Speech Synthesizer

Vani Team

Abstract

A Text to Speech(TTS) Synthesizer is a computer application that is capable of reading out typed text. Vani is such a synthesizer primarily developed for Hindi, but with minor modification could directly be modified for any language which is phonetic in nature, i.e. what is written is exactly what is read out. This is not true for languages like English, in which what is written is significantly different from what is read out, in the sense that the same characters will be pronounced differently depending on context.

Vani exploits the phonetic nature of Hindi, which makes possible a TTS that requires very little of language processing. A TTS sytstem consists of mainly two parts, a language processing module and a signal processing module. In languages such as English, language processing is a major part. Hindi is read almost as it is written, so the amount of language processing is very little.

All TTS systems exisiting allow users to specify what is to be spoken, but do not give any control on how it has to spoken. For Vani we introduce a new encoding schema called *vTrans* for giving this control completely to the user. A vTrans file allows a person to encode exactly what he wants to speak, the way he wants to speak.

A signal processing module, then will bring out this speech by making appropriate variations to the sound database. It will then be possible for our program to sing/speak in a fashion that one desires.

Declaration

The work in this thesis is based on research carried out at the New Software Lab, the Department of Computer Science and Engineering, IIT Bombay, India. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all our own work unless referenced to the contrary in the text on joint research.

Acknowledgements

We would like to thank **Prof. Sivakumar** was giving us the idea for this project, and helping us throughout to make this project a reality. We would like to thank **Prof. Ramesh** for allowing us to go ahead with this project and also for his encouragement. We would like to thank **Prof. Aniruddha Sen, TIFR Bombay** for helping us in the beginning of this project, and providing insight into the area. We would like to thank Pranav, Kshitiz for recording voice samples for us, and allowing us to play with their voice :). We would finally like to thank the **Department of Computer Science and Engineering, IIT Bombay** for all the support and resources that were made available to us.

Contents

List of Figures

Chapter 1

Study of a TTS

Introduction

Incorporation of human facilities like speech and vision in to machine is a basic issue of artificial intelligence research. The capabilities of a computer to generate speech output is termed as speech synthesis. [1] It requires an indepth understanding of speech production and perception and has always been a topic of great interest in speech and cognitive sciences.

In its simplest form computer speech can be produced by playing out a series of digitally stored segments of natural speech. The segments are coded for storage efficiency. The technique involved is elementary and much of the work done in this field specially in our country focuses around here.

Types of speech synthesis

Concatenative Synthesis

In concatenation synthesis, limited number of stored segments obtained from ral speech are used. A critical issue involved in design of concatenative TTS synthesizer is selection of unit size to store the segments. The text to speech synthesizer developed at IIIT, Hyderabad[1] used diphonemes as their fundamental unit, where

[1]

the end point of the splices are in the steady region of speech, so that transitions are not missed [2] .

The basic advantage of the concatenative method is its simplicity. As the units are taken from real speech, the cumbersome task of generating them is avoided. Transitions like C to V are directly captured from the speech data, and the rules to concatenate are elementary.

However the use of a database, restricts the type of speech that can be generated. Hence a concatenative speech synthesis is a limited TTS system.

Formant Synthesis

This method curently used at TIFR, Bombay [2] is based on generation of speech from a production model which has formant frequencies, energies, voice control parameters and few other acousticl, phonetic parameters as control variables. It is possible to generate any arbitary speech by applying a set of context sensitive rules on the stored phonemic data for enacting contextual modifications and for generating transition segments. With single phonemes as the basic unit, rules for smooth concatenation can be formulated. With only a few additional rules and little additional data consonant clusters can be handled elegantly. Most importantly the effort neede to switch over to another similar language with a marginally different phoneme set is also minimal. Overall this synthesizer has the potential to surpass a concatenation synthesizer in both quality and versetality.

[2]

Chapter 2

Motivation

The existing TTS systems available today all make use of concatenation synthesis. Concatenation sythesis though easy to implement, does not give control over what is to be spoken. What can be spoken depends on the available data base of sound sample. This is always finite, and hence it is a limited synthesis.

Why Vani?

The aim of Vani, was to allow complete specification of speech. To the best of our knowledge, there is no system existing today that allows this control. What this means is that one can anyway get the software to speak exactly what they want to. This means typically that the software can also sing if so desired. Also emotions as can be expressed. In a simple concatenative synthesizer, there is no way you can express one word in two different ways, unless you have them in your database.

Chapter 3

Language Processing

Vani was primarily intended for Hindi. Our current implementation of Vani, supports language specific features only for Hindi. Hindi is a phonetically almost sound. Hence very little language processing is required as opposed to languages like English. However some language processing is required even in Hindi, to make it sound natural as we see below.

3.1 Schwa Deletion

Sanskrit which is probably from where Hindi originated, is phonetically perfect. So for a TTS in Sanskrit, no language processing is required. Hindi on the other hand, does require some language processing. The vowel "a" which inherently occurs in all consonants is called as *schwa*. There are some cases in Hindi when schwa after certain characters are not pronounced. As an example consider *ka-ma-la* it would be pronounced as *ka-ma-l*. The "a" following the "l" is deleted.

There has been some work on developing Schwa deletion algorithms for Hindi. Two most important works are those by Narsimhan B., Sproat R. and Kiraz G. [3] and that by Anupam Basu and Monojit Choudhary [4]. The first one combines morphological analysis with finite state transducers and cost models. The accuracy is about 89%. The second one is a rule-based algorithm. If used without a morphological analyser it gives about 96.12% accuracy and with one it gives about 99.89% accuracy. We will be using the second one in Vani, though without a morphological

analyser.

3.2 The Algorithm

This is the rule based algorithm developed by Monojit Choudhary and Anupam Basu [4].

Input: String of graphemes (word without schwa deleted eg. *aa-ma-ntra-Na*)
Output: The word with schwas deleted appropriately *aa-ma-ntra-N*

1. **Mark all the full vowels, viz. vowels not associated with consonants as full and also all consonants followed by vowels other than the inherent schwa and all "h"s as full, unless explicitly marked half by use of halant.**

   This is because of the empirical observation that the schwa following h is always retained. Mark all consonants foolowed by consonants or halants as half. Mark all remaining as undetermined.

2. **If in the word, y is marked as undetermined and is preceded by i,I,ri,u or U, mark it as full.**

   The consonant "y" is a glide from a high vowel to a medium vowel. Therfore if schwa is deleted in the context when "y" is preceded by a high vowel, this glide will be lost, and y will not be appropriately pronounced. Eg. in "tRîtIya" the "a" following "y" needs to be retained. But it may be deleted from "hoya".

3. **If "y","r","l" or "v" are marked as undetermined and preceded by consonants marked half, then mark them as full.**

   This is because of phonotactic constraint. Eg. "kAvya", "samprati", "ashva".

4. **If consonant marked as undetermined is followed by a full vowel, then mark it as full.** This is to maintain lexical distinctions. Eg. if from "baDhaI", the schwa after Dh is deleted, then "badhaI" will be indistinguishable from "baDhi".

5. **While traversing from left to right, if a consonant marked undetermined is encountered before any consonant or vowel marked full, then mark it as full.**

   The schwa following the first syallable is never deleted. This may result in illegal consonant clusters and may change the identity of the word. Eg. "kalama", if the schwa following "k" is not retained it will be "klama". This changes the identity of the word.

6. **If the last consonant is marked undetermined, mark it half.**

   This is again empirically observed. Eg. "kalama" may be "kalam", "banda" should be "band" and so on.

7. **If any consonant is marked undetermined and immediately followed by a consonant marked half, then mark it as full.**

   This is mainly because of phonotactic constraints. Eg. "sAphalya", the schwa after "y" needs to be produced, to make the word sound. This is because pronouncing "phly"is not possible.

8. **While traversing from left to right, for every consonant marked undetermined, mark it half if it is preceded by full, and followed by undetermined or full, otherwise mark it as half**

9. **For all consonants marked half, if it is followed by a schwa in the original word, then delete the schwa from the word. The resulting word is the required ouput.**

Chapter 4

vTrans

As we have seen earlier the motivation behind Vani, was to give complete freedom of expression to the user. For this we need to develop some encoding scheme. This is *vTrans*. For representing text in Indian languages a standard encoding scheme called *iTrans* has been developed which is standardly used. However for speech, we need to represent more than just characters, in order to give complete freedom to the user.

4.1 What vTrans contains

vTrans document will contain a head and a body. In the head there will be parameters and styles defined. Parameters used will be those which need to be changed for changing speech. Parameters that we are using in Vani to control speech are pitch, volume and duration. However, in principle vTrans allows you to use any number of parameters to vary speech. Also for parameters one should specify whether or not that parameter should be allowed to vary with other parameters. This may be done using the attribute *allowstyle*. Also along with parameters, there are styles which specify, how a parameter should vary. In the definition style is simply defined as a step function. Since step-size is arbitary you may define anyfunction you want as accurately as you want. The function should be defined from unit interval `[0,1]` to R. This will later be scaled as required.

Apart from this the body section contatins various tags. They may be nested but text may be put only in the innermost of the tags. The text in the tags should be iTrans encoded. The tags may be any of the parameters defind in the head section. The attributes assigned to the tag maybe, "style" which determines what style to use, if the parameter definition so allows. Also the attribute args allows the user to scale and translate the function used. The latest values of all parameters will be used while producing the output sound.

4.2 The formal definition

The vTrans document must contain and element document which contains the head and the body tag.

head

Head element may or may not be present. If *head* is not present then the contents of the body is pure iTrans and no tags are used. The head contains elements *parameters* and *styles*.

parameters

Parameters may or may not be present. Again if there are no parameters defined then no tags should be present in body section. It will be purely iTrans text.
Parameters will contain several elements *parameter*. A parameter will have three attributes *name*, *default* and *allowstyle* which determine the name of the parameter, the default value to be used for the parameter if no value is specified by user, and whether the parameter is allowed to vary with other parameters or not. The value of default should be a double, and allowstyle should be "yes" or "no".

styles

Styles will contain several elements *style*. Each style will contain an attribute *name* which determines the name of the style. In addition it will contain several elements *point* which has attributes x and y which determine the (x,y) of the function. The x

s should be in increasing order and all between 0 and 1. The y s should be positive real numbers. This defines a step function which is y1 for 0 to x_1, y_2 for x_1 to x_2 .. and so on, and y_n for x_{n-1} to x_n which should be 1.

body

Whatever text is present in the *body* will be read out. The tags contained in the *body* should only be those for parameters defined the *parameters* section. No text should be present excpet in the innermost of the nested tags. The parameter tags could have attribute of *val* if the *allowstyle* was defined to be "no". In this case the value of the *val* attribute will be used as the value, if absent default value will be used. If *allowstyle* was defined to be "yes", then it may contain the attributes *style* which should be `name:param` where name is the name of the style to be used, and param is the parameter with which it should vary. If no style is present, then default funtion which is contant in `[0,1]` and value 1 is used. Also the atribute *args* defined as `a:b` is used to scale the funtion to f*a + b. If no args is present then default value for a, and 0 for be will be used.

Chapter 5

Generation of Unlimited Speech

The next step to the completion is the generatio of the speech from a given vTrans file. This chapter and our work does not deal with the generation of vTrans representation, but with the conversion of vTrans to unlimited speech. We first begin with our assumptions based on observations and experiments and then describe How we are able to generate any speech sample from any given frequency and volume distribution functions.

5.1 Theoritical Aspects

A vTrans has theoritical capability of representation of any voice sequence. The algorithms to convert it to an audible form have to be sound and capable of carrying out any desired transforations.

5.1.1 Observations

The waveforms of a spoken hindi file is not very difficult to analyze. The consonants are arbritary samples of very small durations where vowels are continuous repeatations of a simple waveform with some variances. More than 90 percent of the CV pair is dominated by V alone, which looks like a continuous repeatation of a very small segment, which taken together will be called as fract-phoneme. A CV pair consists of a consonant fract-phoneme and 50-52 repeatations of vowel fract phonemes. The repeatations are not monotonous, and there are variances in pitch and volume but

the fundamental structure remains as same. This implies, our fundamental assumption, that "Any unlimited speech can be generated using fract-phonemes as a basic unit."

5.1.2 Interpretations

Duration of a consonant is very small. The variance in durations and pitch need not consider/change the duration of a consonant. Hence only volume variances are considered. Any desired effect can be achieved by varying pitch, duration and volume of fract-phonemes of respecting vowels and consonants.

To understand this, consider the waveform of vowel 'a' in the figure. The first figure is a recorded waveform of 'a'. The second waveform is fract-phoeneme of 'a' which is extracted somewhere from between. The waveform of a human voice can be divided into some 50 similar fract-phonemes. All these fract-phoenemes share similar structure with fract-phoneme of 'a' with some variances. Now the variance can be either in x-direction or in y-direction[1]. Variance in x-direction is obtained by varying pitch and in y-direction is obtained by varying volume. Hence, if we are able to generate a fract phoneme with any arbitrary distribution of volume and pitch, we will be able to generate Unlimited Speech. What that function is ?, is not a problem we are trying to answer. vTrans allows definition of any arbitrary function and we present a model for applying it to generate the required waveform.

5.2 Methodology of Generating a phoneme from fract-phoneme

Given the volume and frequency curves and the duration a phoneme can be generated using fract-phoneme as per the following procedure.

- Step 1: Calculation of virtual-duration.
- Step 2: Generation of a monotonous-phoneme of virtual-duration.

[1]Any modification in y-scale is constrained from -1.0 to 1.0

- Step 3: Applying the pitch transformation.
- Step 4: Applying the volume transformation.

We shall now look into each steps in detail.

5.2.1 Calculation of virtual-duration

virtual-duration is defined as the required duration to which a phoneme has to be generated so that after applying the pitch transformation we get the phoneme of the required(actual) duration. Its calculated as a solution of the equation

$$\int_0^1 t\frac{1}{f(x)}\,dx = T.$$

where,

f(x) is the frequency function curve defined over 0 to 1

t is the virtual-duration

T is the actual-duration

5.2.2 Generation of a monotonous-phoneme of virtual-duration

Monotonous phoneme is the phoneme of duration actual-duration without any variations of pitch and/or volume. After the calculation of virtual-duration its generation is simply done by concatenating the base fract-phoneme repeatedly until a phoneme of required duration is obtained.

5.2.3 Applying the pitch transformation

This step involves first the conversion of the generated phoneme-wave to time domain. Let us call denote the representation of the monotonous-phoneme, we obtained in step 2 as o(t). The frequency function is f(x):[0,1]-¿R which will be appropriately scaled to be applied to o(t). We need to calculate n(t) the new wave which will be generated after the application of f(x)[2] to o(t).

[2]f(x) is defined over 0 to

Duration of o(t) is t.[3] and

Duration of n(t) is T[4]. The equation which we derived to generate the n(t) is :-

$$n(\int_0^t \frac{1}{f(x)}\,dx) = o(t)$$

Note that this equation has no trivial solution and the computation of n(t) has to be done numerically. Due to computational restriction no algorithm can successfully determine n(t). We devised a heuristic algorithm for its calculation. The new wave generated is of duration T, as required.

5.2.4 Applying the volume transformation

Till this step we have generated a sound wave with applied frequency distribution. Application of volume function is fairly straight forward, since we only need to scale the amplitude according to the function. The equation is simply

$$n(t) = v(t) * o(t)$$

Finally, we have the required phoneme with the desired features that can be used in speech synthesis.

5.3 An example

Before ending this chapter let us see an example of application of above algorith to generate a phoneme for 'i'. The required DSP Parameters are

Duration : 1000

Volume Distribution

f(t) = 0.6 t < 0.2

f(t) = 1.1 0.2<= t < 0.8

f(t) = 0.8 0.8<=t<=1.0

[3]Virtual Duration, as calculated above

[4]Actual Duration

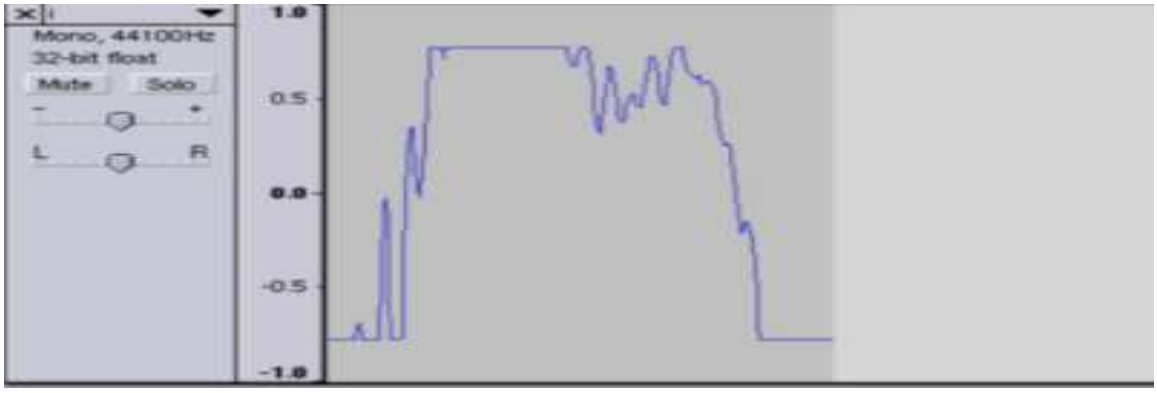

Figure 5.1: fract-phoneme of i

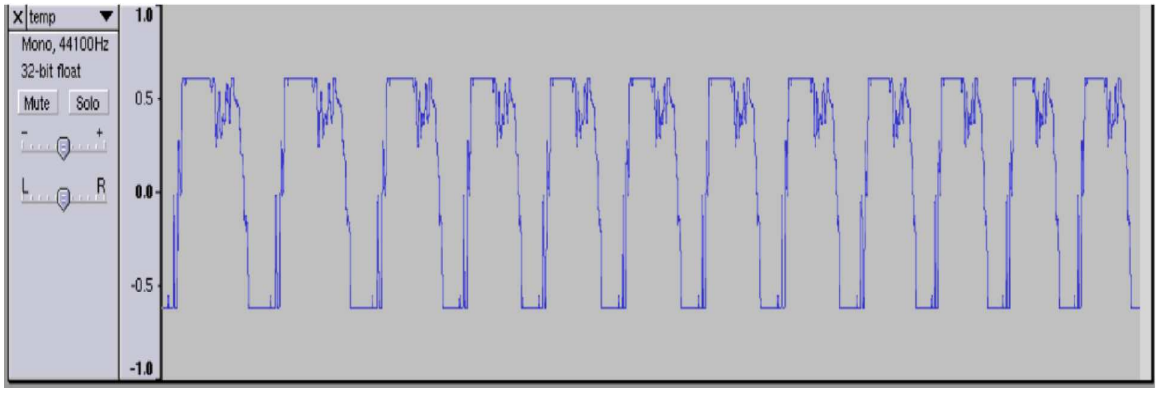

Figure 5.2: after applying pitch transformation

Frequency Distribution

v(t) = 0.6 t<0.2

v(t) = 0.8 0.2<=t<0.7

v(t) = 0.9 0.7<=t<=1.0

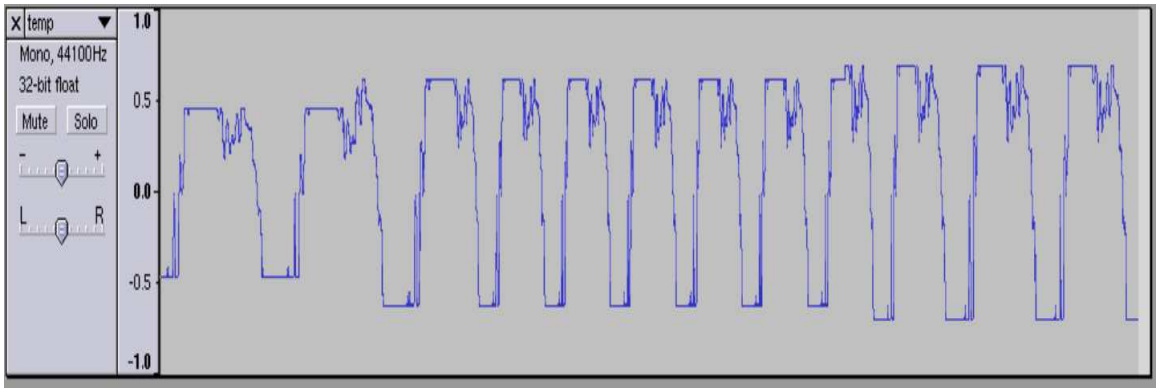

Figure 5.3: finally after applying volume transformation

Chapter 6

Conclusions

In this report we have briefly discussed the development of Vani. The major aspect of this whole project, was the introduction of the vTrans encoding scheme. Currently due to lack of experimentation, and the shortage of a good database of sound, the quality of sound produced by Vani, isn't very good. But we hope to work on this, and soon it will be able to produce good quality speech.

However what we have presented here, is that it is possible, to make arbitary variations in speech, given the appropriate database of phonemes and fract-phonemes, as described in the report. Also we have given an encoding scheme, which enables one to produce such speech. In these aspects, our work differs significantly from other works in this area.

Bibliography

[1] Aniruddha Sen and Xavier A Furtado

Synthesis of unlimited speech in Indian Languages using formant-based rules ,

[2] S.P. Kishore , Alan W Black

Unit Size in Unit Selection Speech Synthesis,

[3] Narsimhan B., Sproat R. and Kiraz G. ,2001

Schwa Deletion in Hindi Text-to-Speech Synthesis,

[4] Anupam Basu and Monojit Choudhary, 2002

A Rule Based Schwa Deletion Algorithm for Hindi,

Appendix A

A sample vTrans file

<?xml version = "1.0" encoding = "UTF − 8"? >

< documenttype = "text/vtrans" >

< head >

< parameters >

< parametername = "vol" default = "0.8" allowstyle = "yes" >

ToControltheVolume.

< /parameter >

< parametername = "pitch" default = "1.0" allowstyle = "yes" >

Tocontrolthepitch.Itsbasicallythescalingfactor

< /parameter >

< parametername = "duration" default = "1200" allowstyle = "no" >

Tocontroltheduration.

< /parameter >

< /parameters >

< styles >

< stylename = "arbit" >

< pointx = "1.0" y = "1.0" >< /point >

< /style >

< stylename = "jarsj" >

< pointx = "0.2" y = "0.6" >< /point >

< pointx = "0.8" y = "1.1" >< /point >

$< pointx = "1.0" y = "0.8" >< /point >$

$< /style >$

$< stylename = "jarsj2" >$

$< pointx = "0.2" y = "0.6" >< /point >$

$< pointx = "0.7" y = "0.8" >< /point >$

$< pointx = "1.0" y = "0.9" >< /point >$

$< /style >$

$< /styles >$

$< /head >$

$< body >$

$< pitchstyle = "jarsj : duration" >$

$< volstyle = "jarsj2 : duration" >$

$< durationval = "200" >$

i

$< /duration >$

$< /vol >$

$< /pitch >$

$< /body >< /document >$