

The Festival Speech Synthesis System

Seminar One:

An introduction



CSTR

What is this course about?

The Festival speech synthesis system

- What Festival is, how it does things
- General principles of speech synthesis
- Building *Multisyn* voices
- What resources are available to you

You will not learn everything. That takes about 5 years!

CSTR

Who are we? What do we do?

Rob Clark

- Festival developer and coordinator at CSTR
- Interests include: intonation for speech synthesis

Korin Richmond

- Festival developer and researcher at CSTR
- Interests include: unit selection, using information from the articulatory domain

Other current Festival developers:

- Alan Black (CMU): Original project developer
- Volker Strom (CSTR): Prosody and unit selection

CSTR

Course Outline

1. Introduction to Festival and building multisyn voices
2. More unit selection and voice building
3. Using and evaluating voices

CSTR

1. Introduction to Festival and voice building

We will look at:

- What Festival is and what it does
- The basics of using Festival
 - Introduction to scheme
- Data needed to build voices.

You get to:

- Start using Festival
- Start building a voice.

CSTR

In the beginning...

Festival started life about 8 years ago:

- Developed by Alan Black & Paul Taylor at CSTR, University of Edinburgh
- A Diphone based Synthesiser for English

Since then:

- Other synthesis methods & languages
- Various derived commercial products

CSTR

What is Festival? (Festival 2.0)

- A development environment for speech synthesis research
- A medium scale reasonably robust TTS system.

What is festival not?

- Officially supported under windows
- A robust large scale TTS server environment

CSTR

Festival/Festvox

Festival

- The synthesiser
- From: CSTR, University of Edinburgh

Festvox

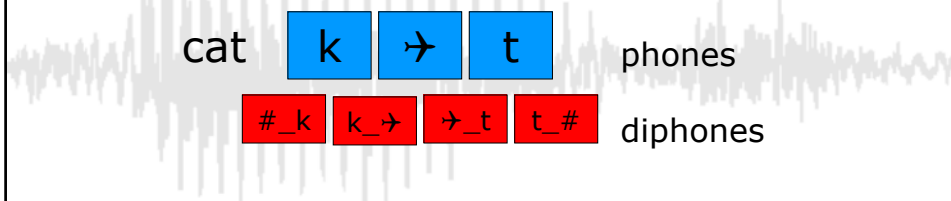
- Voice development tools and documentation
- From: CMU speech group

CSTR

Concatonative Speech Synthesis

Diphones are generally the type of units used

- A diphone is a chunk of speech which starts at the centre of one phone and ends at the centre of the next



How many diphones does a language have?

- No. of phones squared?

CSTR

Types of synthesis Festival provides

Diphone

- Record 1 instance of each diphone

Cluster units (clunits)

- Multiple instances of each diphone

General unit selection (multisyn) [New in Festival 1.95]

- Multiple instances of each diphone

HMM Based (HTS) [Heiga Zen]

- Trained HMMs generate the speech

CSTR

Running Festival

Various modes

- Interactive *scheme* shell
- Batch processing
- Server/Client
- C/C++ API

We concentrate on the interactive
scheme mode

CSTR

An introduction to scheme

What is scheme?

- An interpreted shell language
- A variant of LISP

Why scheme?

- Allows maximum flexibility
 - Rapid prototyping
 - Flexible configuration
- Is completely embedded in Festival, no package dependencies.

CSTR

The good the bad and the ugly

The downside is that scheme and LISP are a bit obscure, and not the easiest language to work with.

- The brackets will drive you mad!!!

... So we provide a gentle introduction here. This is intended to be enough information (often simplified) to enable the use and understanding of scheme in Festival rather than anything else.

CSTR

Scheme for beginners

There is only one type of statement in scheme, it looks like this:

```
(function_name arg1 arg2 ...)
```

And the only thing scheme does it to take expressions like the one above and **evaluate** them, replacing the statement with the result of the evaluation

CSTR

Scheme for beginners

Some examples:

```
> (+ 1 2)
```


```
3
```

```
> (print "hello")
```

```
"hello"
```

```
nil
```

```
> (SayText "Hello world")
```

```
#<Utterance 0xa184560> 
```

```
> (+ 1 (+ 1 2))
```

```
4
```

CSTR

Data structures in scheme

Simple *atomic* data types:

- t, nil

- 1,2,3,...

Creating new atomic data items

```
> (quote hi)
```

```
hi
```

The quote function is the only function to not evaluate its argument!

Short hand: `hi, `a, `b, `1, `2, `label, `strawberry

The point of atoms is that they evaluate to themselves.

CSTR

Data structures in scheme

Strings:

"hi", "label", "strawberry"

Strings are not atoms!

"strawberry" ≠ `strawberry

Although many festival functions can take either as their arguments

CSTR

Variables and function names

Names that are *unquoted* are assumed to be variables or function names.

Variables can be set with the command **set!**

```
> (set! v1 "hello")
```

```
"hello"
```

```
> (set v1 (+ 1 2))
```

```
3
```

```
> v1
```

```
"hello"
```

```
> v2
```

```
3
```

Functions can be defined with the command **define** which we will hear more about later.

CSTR

Complex data types

Lists are the main data type in scheme.

There are 2 ways to generate lists.

```
> (list 1 2 3 'a 'b 'c)
```

```
(1 2 3 a b c)
```

```
> '(1 2 3 a b c)
```

```
(1 2 3 a b c)
```

```
> (list 1 2 3 '(a b c))
```

```
(1 2 3 (a b c))
```

```
> (list 1 2 3 v1 v2)
```

```
(1 2 3 1 3)
```

CSTR

Processing lists

Two list accessing function car and cdr

- car – return the first item in the list

- cdr – return the *tail* of the list

```
> (set! l1 '(1 ("hello" "world") 2))
```

```
> (car l1)
```

```
1
```

```
> (cdr l1)
```

```
((("hello" "world") 2)
```

```
> (car (cdr l1))
```

```
("hello" "world")
```

CSTR

Defining functions

An example of a function definition:

```
(define (foo a1 a2 a3)
  (let ((v1 (+ a1 a2))
        (v2 nil))
    (set! v2 (+ v1 a3))))
```

```
> (foo 1 2 3)
```

```
6
```

```
> (foo 1 2)
```

```
SIOD ERROR: too few arguments : ...
```

CSTR

More on let

```
(let ((v1 val1) (v2 val2) v3 v4)
  BODY)
```

Let defines the scope of local variables

- It takes 2 arguments: a list of variables and a body of code
 - The variables are defined for the duration of the body
 - The variables can either be lists: where a value is specified
 - or just a variable name

CSTR

Testing equality

(eq? a b)	true if the same object
(equal? a b)	true if recursively equal
(string-equal a b)	true if strings match
(string-matches a b)	true if regex matches
(< a b)	true if $a < b$
(> a b)	true if $a > b$

CSTR

Some scheme programming constructs...

(cond (test1 do1) (test2 do2) ...)

```
(cond
  ((not (number ?))
   (print "x is not a number"))
  ((< 3 x)
   (print "x is < 3"))
  (t
   (print " x is >= 3")))
```

CSTR

Some scheme programming constructs...

(mapcar (lambda (X) DO_EACH_X) LIST_OF_Xs)

```
> (mapcar  
  (lambda (x)  
    (* 2 x))  
  '(1 2 3 4 5))
```

(2 4 6 8 10)

CSTR

To keep the procedural programmers happy...

(while CONDITION BODY)

```
(while (> x 0)  
  (print x)  
  (set! x (- x 1)))
```

(if CONDITION TRUE_DO FALSE_DO)

```
(if (eq? x 1) (print "x is 1") (print "x is not 1"))
```

CSTR

Loading files

(load "path/filename.scm")

Loads the scheme file and evaluates its contents

Useful as long expressions and function definitions are difficult to get right on the command line.

CSTR

Scheme in Festival

In the next session we will learn about manipulating **utterances** in scheme

But we have already seen our first example

(SayText "Hello world")

CSTR

Scheme in Festival

Scheme is used by Festival in a number of ways:

- To control the flow of the synthesis process
- To prototype new methods
- To allow easy access to the data structures as synthesis before, during and after synthesis.

CSTR

Non-interactive festival

Festival can also be run in batch mode

- Used during voice building
- Ok for diphone voices
- Bad for multisyn voices

```
$ festival -b '(some_command arg 1 arg2)'
```

CSTR

Getting help

Most scheme functions have documentation built in. Type the name of the function and press <ESC> followed by H

> **(SayText<ESC>H**

(SayText TEXT)

TEXT, a string, is rendered as speech.

Also, pressing TAB halfway through a function name will give you a list of possible completions

Failing that, the festival manual is online at:

<http://www.cstr.ed.ac.uk/projects/festival/manual>

CSTR

Building a new voice multisyn voice

What do we need to build a new voice?

- The language front end working!
 - Helps to determine what to record
 - Needed to build the linguistic structure of the database
- A speech database which gives *coverage* of the language

CSTR

The stages of building a voice

- Language resource preparation
- Text selection
- Recording
- Processing recorded data
- Labelling data
- Building utterance structures
- Voice definition

CSTR

Designing a voice

Important stage

- A badly designed voice will sound bad
- Once you have recorded your voice it may hard to go back...

CSTR

Text selection

The idea is to have a diphone in our database suitable for every occasion.

To do this we define a set of contexts we wish each diphone to be found in.

- Stressed vs. unstressed
- Different syllable positions
- Different word positions
- Different phrase positions

This is complicated by the fact that we are dealing with diphones!

CSTR

So we just record each diphone we need in each context, right?

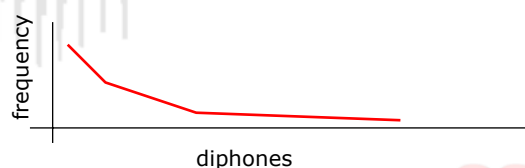
... well we try.

How many diphones are there?

- Word internal diphones
- Inter-word diphones

Distribution of diphones in context

- Zipf like.



CSTR

What sort of data needs to be recorded?

The sort that it is going to be expected to speak!

- Limited domain vs. Completely general

Many sub-databases

- News, flights, meetings, names & addresses, dialogue, lists, email, ...

CSTR

How much data do I need to record?

Voice	phones	words	sentences
nina	175,000	38300	2100
awb	36,000	10000	1134

- Nina probably has too many phones
 - Would be ok with prosody
- Awb has just enough

Both are single domain voices

CSTR

Selecting the text

Ideal situation is to get loads of text and select a subset from it.

- Copyright issues

Select a core set for minimum diphone coverage (no context)

Add diphones in context to this in proportion to their frequency distribution.

(Our tools for this are not yet perfected...)

If data is not available, you may need to design by hand, and test coverage

CSTR

Recording your data

Studio conditions

- Ideal is almost anechoic
 - Most recording studios have a ‘presence’

Nina – recorded in our not so great studio

Awb – laptop, cheap microphone in a quiet room

CSTR

Choosing a speaker

Someone that can read naturally

Someone who's voice quality is consistent

- Not breathy or creaky, constant volume

Someone whose voice you can stand to listen to!

- . Voice talents
- . Acting/drama students

CSTR

Other recording issues

Presenting data

- . Printed 10 sentences per page

Splitting data

- . Use a 7khz beep

Sessions

- . Same time of day, days close together.

CSTR

Processing the data

- Pitchmark generation
 - From EGG signal or the waveform
- MFCCs
 - For labelling and join cost
- Waveform and MFCC normalisation
 - To make the data more consistent
- LPCs for final synthesis

CSTR

Labelling the data

Automatic vs. hand labelling

- Accuracy vs. consistency

We are not actually interested in the phone boundaries anyway!

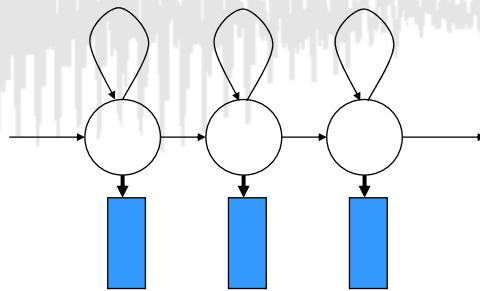
Suggest an HTK forced alignment approach

CSTR

Forced alignment

Speech recognition task, but we know what the phone sequence is!

- We model each phone with 3 state hidden Markov model (HMM)



CSTR

Forced alignment process

1. Train models from flat start
2. Realign labels
3. Retrain *better* models
4. Realign labels
5. Increase mixtures + retrain models
6. Final alignment

All nicely scripted.

CSTR

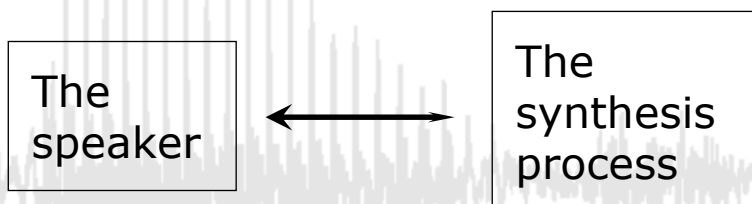
Script requirements

- The text for the sentences
 - Festival is used to create initial label file.
- A list of phones
- A list of allowed phone substitutions
- MFCCs for the speech

CSTR

Labelling issues

Competing requirements:



Matching the speaker is more important

CSTR

Labelling issues

We allow phone substitution

- To try to match what the speaker actually said

We label the closures of stops and affricates

- We use the boundary as the join point

We add an optional *short pause* at the end of each word

We add silence at beginning and end of utterance

We substitute some phone symbols, e.g. "?"

"Staff duplication..."

sil s t_cl t aa f sp d_cl d y p_cl p

CSTR

Building the utterances

Needed so we can compare target diphones to candidates

We automatically:

- Part synthesise each sentence (again!)
- Match the segments with our labels and extract the end times
 - Taking into account the pauses, closures, substitutions etc.
 - Add a few features to mark certain phones as being problematic.

CSTR

Defining the voice

Final stage in building a voice is to prepare a scheme definition file for the voice



CSTR

Likely problems with a new voice

Labelling problems

- Speaker did not follow the script
- Speaker pronounces things differently
- Random alignment failure

Other problems

- Bad pitchmarking

CSTR