Carnegie Mellon University
Language Technologies Institute
Alan W Black (awblack@cs.cmu.edu)

http://cmuir.ore

a small run-time synthesizer

PILT:
- no Scheme, no interpreter, no GC
- no loading (copying) of data, all constant
- thread safe
- time to speak: less than 0.250 seconds (on iMac)
- options to be half (and half again)
- less than 4 Meg (lite+lex+desktop voice)

Options get "compiled" into static structures:

- C for size and portability
- not a replacement, but companion
- Festival "compatible"
- small, fast, portable run-time synthesizer

Festival - Festival-lite
can components be simplified – what’s really necessary in a synthesizer engine – trade-off between size/quality/speed

Research questions:

– helps to have built previous systems
– need to understand task be a good programmer

… “we’ll put a team of 20 programmers on it…”

… “But it’s just an engineering problem…”

… Speech technology has to be ubiquitous

… synthesizers advocate allows no excuses

… “OK, but does it scale…”

… Festival is too slow, to big, and not portable:

**Title: motivation**
when a component can be easily added, it will be.

these are also for the speech/dialog researcher.

No, not exclusively.

But these are "just" commercial uses:

- multi-channel telephone dialogue systems
- server systems
- toys and games
- digital radios, books
- local rendering of speech on PDA
- embedded systems: PDA

Target audiences
Each goes to distinct Lib:

- voice: waveforms/mcep/lpc, index (from testvox)
- lexicon: words, plus l's
- language models: text analysis, prosody etc
- □ LAng

□ Phile Library: with CST, "C Speech Tools"

Phile components
Some general points

- Avoid non-const globals

- Except interface definition

- Therefore no C allocation required

- Dynamically allocated vars linking into ut

- Scheme code used to do conversion

- Cart trees, waveforms, etc converted C structures

- Static contexts where possible?
and appropriate models, duration etc
- (e.g. appropriate text analysis, F0 model functions
  - voice linked to synthesis functions
  - voice is global (const)
    File: Link voice with each utterance
    (other globals too)
    - used by each utterance
    - set as global
      : Rest trap: uses notion of "current voice"

Non-Global current voices
CST: "C Speech Tools"
CST II: A lite

- Class support (digital/unit selection) to be completed
- Prosody modification
- Telephone (PC synthesized)

Voice structure, phonesets
- Generic synth functions (voice parameterizable)

- Public Lite functions

Voice synthesis

- Lite rule interpreter
- Lexicon index support

/lexicon

wavetsynth

/specs (PC synthesized)
US English feature functions

- Hand written cart and converted

  Indicating:

  - autoconverted CART trees from Festivaal voices
  - Prosody:

  Pretty basic at the moment

  - Number/Symbol expansion

  - Text analysis:

  US English specific models

/lang/usenglish/
(((NB)))
    (BB))
(0 :: name (is :: 0))
(((NB))
    ((BB))
("" :: Token.parent.punc :: is :: "")
    ((BB))
("" :: Token.parent.punc :: is :: ",")
    (0 :: Token.name (is :: 0)),
set | phrase-cart-tree
```csharp
};

const cshtmlTag Table

} = const us-phrase-node
cost cart us-phrase-cart

} = null

"n-name",
"R:token.parent.punc",
"R:token.name",
"R:token.name",

STATIC CONST VAL STRING ( Val = 0000, "0")

;{ { 255, CST-CART-DP-NONE, 0, 0
}; 255, CST-CART-DP-NONE, 0, 0;
{ 255, CST-CART-DP-NONE, 0, 0;
{ 255, CST-CART-DP-NONE, 0, 0;
{ 255, CST-CART-DP-NONE, 0, 0;
{ 255, CST-CART-DP-NONE, 0, 0;
{ 255, CST-CART-DP-NONE, 0, 0;
{ 255, CST-CART-DP-NONE, 0, 0;
{ 0, CST-CART-DP-NONE, 0, 0;
} = [ ]
```
can we remove infrequent words – the word list/phone list is well over 2,9M for complex

comment: □

– syllabifier
– rules
– word list
– addenda (simple list)
– lexicon: □

minimised packed its decision & graphs (black/leuko 98)

– from word to pronunciation or “use it”
– pronunciations: □

/Jan8/complex
2.5M code, 20M run-time, 10 times faster than real time

Faster than 10MHz per port

thus about 10MHz per port

20 mins for speech takes 20 seconds to synthesize.

- On 500MHz PIII, about 60 times faster than real time
- Speed:

Runtime RAM requirements:

- uint8 2.1M
- complex 2.9M
- unencoded 25K
- After code 50K

With current 8KHz diphone voice:

(third safe after short initialization of voice)

no leaks

Current sizes
- reduce mean footprint to 0.25 (at least)
- time to first noise, less than 95%

**F1**ite advantages would be

but waveform regeneration can be streamed
- must do (fill table) text, phrase, prosodies first
- play/send as its built
- don’t build full waveform

**Streaming synthesis:**

- process full utt, and make full waveform
- utterance by utterance
- **F1**ite (and restival) synthesis

**Improvement: Streaming synthesis**
Improvement: Unit database compression

- e.g. all diaphones into stop can be shared
  - re-use units
  - Unit selection:
    - (could use spike excited for very small footprint)
    - Quantizing coefficients and/or residual
    - number of coefficients
    - Optimise size vs quality:
      - Residual (small dynamic range)
      - Requires LPC coefficients (pitch synchronous)

LPC Resynthesiser:
- 16KHz speech
- FM
- Medium: ☐
- lex etc as with Festival
- telephone quality speech
- 2.5M
- Small: ☐

(?) spike excited LPC (quantised ?)
- streaming synthesis
- compressed minimal lexicon
- all less than 1M

Very small (will be poorer quality): ☐

Target footprints
Todo...

- PAO, isy etc

Some key demonstration projects:

- Documentation

- Fixed point version

Should really be using the NSV models:

- Text analysis:

  - Should be fast and much smaller
  - build process from Rest/Text format

- DB will always be larger than diphone

- Support clients/JSON
- (i.e. about twice the size I'd like it to be)

- at about the 4M level
  - probable size: □
  - 8 KHz dictaphone voice
  - printed lex based on complex
  - plus basic US English
  - core file library
    - Release will include: □

- intended release end of Feb '01

First Release