William I. Hallahan

DECtalk Software: Text-to-Speech Technology and Implementation

DECtalk is a mature text-to-speech synthesis technology that Digital has sold as a series of hardware products for more than ten years. Originally developed by Digital's Assistive Technology Group (ATG) as an alternative to a character-cell terminal and for telephony applications, today DECtalk also provides visually handicapped people access to information. DECtalk uses a digital formant synthesizer to simulate the human vocal tract. Before the advent of the Alpha processor, the computational demands of this synthesizer placed an extreme load on a workstation. DECtalk Software has an application programming interface (API) that is supported on multiple platforms and multiple operating systems. This paper describes the various text-to-speech technologies, the DECtalk Software architecture, and the API. The paper also reports our experience in porting the DECtalk code base from the previous hardware platform.

During the past ten years, advances in computer power have created opportunities for voice input and output. Many major corporations, including Digital, provide database access through the telephone. The advent of Digital's Alpha processor has changed the economics of speech synthesis. Instead of an expensive, dedicated circuit card that supports only a single channel of synthesis, system developers can use an Alpha-based workstation to support many channels simultaneously. In addition, since text-to-speech conversion is a light load for an Alpha processor, application developers can freely integrate text to speech into their products.

Digital's DECtalk Software provides natural-sounding, highly intelligible text-to-speech synthesis. It is available for the Digital UNIX operating system on Digital's Alpha-based platforms and for Microsoft's Windows NT operating system on both Alpha and Intel processors. DECtalk Software provides an easy-to-use application programming interface (API) that is fully integrated with the computer's audio subsystem. The text-to-speech code was ported from the software for the DECtalk PC card, a hardware product made by Digital's Assistive Technology Group. This software constitutes over 30 man years of development effort and contains approximately 160,000 lines of C programming language code.

This paper begins by discussing the features of DECtalk Software and briefly describing the various text-to-speech technologies. It then presents a description of the DECtalk Software architecture and the API. Finally, the paper relates our experience in porting the DECtalk code base.

Features of DECtalk Software

The DECtalk Software development kit consists of a shared library (a dynamic link library on Windows NT), a link library, a header file that defines the symbols and functions used by DECtalk Software, sample applications, and sample source code that demonstrates the API.

DECtalk Software supports nine preprogrammed voices: four male, four female, and one child's voice. Both the API and in-line text commands can control the voice, the speaking rate, and the audio volume. The volume command supports stereo by providing independent control of the left and right channels. Other in-line commands play wave audio files, generate single tones, or generate dual-tone multiple-frequency (DTMF) signals for telephony applications.

Using the text-to-speech API, applications can play speech through the computer's audio system, write the speech samples to a wave audio file, or write the speech samples to buffers supplied by the application. DECtalk Software produces speech in 3 audio formats: 16-bit pulse code modulation (PCM) samples at an 11,025-hertz (Hz) sample rate, 8-bit PCM samples at an 11,025-Hz sample rate, and μ-law encoded 8-bit samples at an 8,000-Hz sample rate. The first two formats are standard multimedia audio formats for personal computers (PCs). The last format is the standard encoding and rate used for telephony applications.

The API can also load a user-generated dictionary that defines the pronunciation of application-specific words. The development kit provides a window-based tool to generate these dictionaries. The kit also contains a window-based application to speak text and an electronic mail-notification program. Sample source code includes a simple window-based application that speaks text, a command line application to speak text, and a speech-to-memory sample program.

The version of DECtalk Software for Windows NT also provides a text-to-speech dynamic data exchange (DDE) server. This server integrates with other applications such as Microsoft Word. Users can select text in a Word document and then proofread the text merely by clicking a button. This paper was proofread using DECtalk Software running a native version of Microsoft Word on an AlphaStation workstation.

Speech Terms and DECtalk Software

Human speech is produced by the vocal cords in the larynx, the trachea, the nasal cavity, the oral cavity, the tongue, and the lips. Figure 1 shows the human speech organs. The glottis is the space between the vocal cords. For voiced sounds such as vowels, the vocal cords produce a series of pulses of air. The pulse repetition frequency is called the glottal pitch. The pulse train is referred to as the glottal waveform. The rest of the articulatory organs filter this waveform.[1] The trachea, in conjunction with the oral cavity, the tongue, and the lips, acts like a cascade of resonant tubes of varying widths. The pulse energy reflects backward and forward in these organs, which causes energy to propagate best at certain frequencies. These are called the formant frequencies.

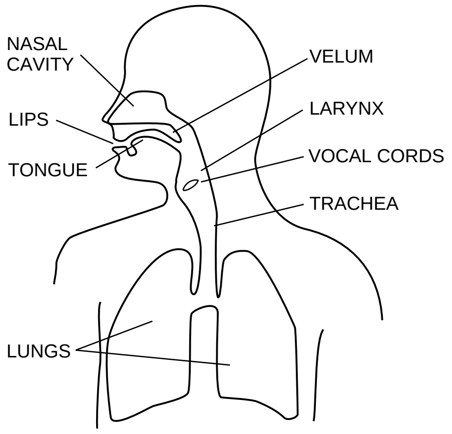


Figure 1
The Speech Organs

The primary discrimination cues for different vowel sounds are the values of the first and second formant frequency. Vowels are either front, mid, or back vowels, depending on the place of articulation. They are either rounded or unrounded, depending on the position of the lips. American English has 12 vowel sounds. Diphthongs are sounds that change smoothly from one vowel to another, such as in *boy, bow,* and *bay.* Other voiced sounds include the nasals *m, n,* and *ng* (as in *ing*). To produce nasals, a person opens the velar flap, which connects the throat to the nasal cavity. Liquids are the vowel-like sounds *l* and *r.* Glides are the sounds *y* (as in *you*) and *w* (as in *we*).

Breath passing through a constriction creates turbulence and produces unvoiced sounds. *f* and *s* are unvoiced sounds called fricatives. A stop (also called a plosive) is a momentary blocking of the breath stream followed by a sudden release. The consonants *p, b, t, d, k,* and *g* are stop consonants. Opening the mouth and exhaling rapidly produces the consonant *h.* The *h* sound is called an aspirate. Other consonants such as *p, t,* and *k* frequently end in aspiration, especially when they start a word. An affricative is a stop immediately followed by a fricative. The English sounds *ch* (as in *chew* and *j* (as in *jar*) are affricates.

These sounds are all American English phonemes. Phonemes are the smallest units of speech that distinguish one utterance from another in a particular language.[2] An allophone is an acoustic manifestation of a phoneme. A particular phoneme may have many allophones, but each allophone (in context) will sound like the same phoneme to a speaker of the language that defines the phoneme. Another way of saying this is, if two sounds have different acoustic manifestations, but the use of either one does not change the meaning of an utterance, then by definition, they are the same phoneme.

Phones are the sets of all phonemes and allophones for all languages. Linguists have developed an international phonetic alphabet (IPA) that has symbols for almost all phones. This alphabet uses many Greek letters that are difficult to represent on a computer. American linguists have developed the Arpabet phoneme alphabet to represent American English phonemes using normal ASCII characters. DECtalk Software supports both the IPA symbols for American English and the Arpabet alphabet. Extra symbols are provided that either combine certain phonemes or specify certain allophones to allow the control of fine speech features. Table 1 gives the DECtalk Software phonemic symbols.

Speech researchers often use the short-term spectrum to represent the acoustic manifestation of a sound. The short-term spectrum is a measure of the frequency content of a windowed (time-limited) portion of a signal. For speech, the time window is typically between 5 milliseconds and 25 milliseconds, and the pitch frequency of voiced sounds varies from 80 Hz to 280 Hz. As a result, the time window ranges from slightly less than one pitch period to several pitch periods. The glottal pitch frequency changes very little in this interval. The other articulatory organs move so little over this time that their filtering effects do not change appreciably. A speech signal is said to be stationary over this interval.

Table 1
DECtalk Software Phonemic Symbols

Consonants		**Vowels and Diphthongs**	
b	*b*et	aa	B*o*b
ch	*ch*in	ae	b*a*t
d	*d*ebt	ah	b*u*t
dh	*th*is	ao	b*ough*t
el	bott*le*	aw	b*ou*t
en	butt*on*	ax	*a*bout
f	*f*in	ay	b*i*te
g	*g*uess	eh	b*e*
hx	*h*ead	ey	b*a*ke
jh	*g*in	ih	b*i*t
k	*K*en	ix	kiss*es*
l	*l*et	iy	b*ea*t
m	*m*et	ow	b*oa*t
n	*n*et	oy	b*oy*
nx	si*ng*	rr	b*ir*d
p	*p*et	uh	b*oo*k
r	*r*ed	uw	l*u*te
s	*s*it	yu	c*u*te
sh	*sh*in	**Allophones**	
t	*t*est	dx	ri*d*er
th	*th*in	lx	e*l*ectric
v	*v*est	q	we eat
w	*w*et	rx	*o*ration
yx	*y*et	tx	La*t*in
z	*z*oo	**Silence**	
zh	a*z*ure	_ (underscore)	

The spectrum has two components for each frequency measured, a magnitude and a phase shift. Empirical tests show that sounds that have identical spectral magnitudes sound similar. The relative phase of the individual frequency components plays a lesser role in perception. Typically, we perceive phase differences only at the start of low frequencies and only occasionally at the end of a sound. Matching the spectral magnitude of a synthesized phoneme (allophone) with the spectral magnitude of the desired phoneme (taken from human speech recordings) always improves intelligibility.[3] This is the synthesizer calibration technique used for DECtalk Software.

A spectrogram is a plot of spectral magnitude slices, with frequency on the *y* axis and time on the *x* axis. The spectral magnitudes are specified either by color or by saturation for two-color plots. Depending on the time interval of the spectrum window, either the pitch frequency harmonics or the formant structure of speech may be viewed. It is even possible to ascertain what is said from a spectrogram. Figure 2 shows spectrograms of both synthetic and human speech for the same phrase. The formant frequencies are the dark regions that move up and down as the speech organs change position. Fricatives and aspiration are characterized by the presence of high frequencies and usually have much less energy than the formants.

The bandwidth of speech signals extends to over 10 kilohertz (kHz) although most of the energy is confined below 1,500 Hz. The minimum intelligible bandwidth for speech is about 3 kHz, but using this bandwidth, the quality is poor. A telephone's bandwidth is 3.2 kHz. The DECtalk PC product has a speech bandwidth just under 5 kHz, which is the same as the audio bandwidth of an AM broadcast station. The sample rate of a digital speech system must be at least twice the signal bandwidth (and might have to be higher if the signal is a bandpass signal), so the DECtalk PC uses a 10-kHz sample rate. This bandwidth represents a trade-off between speech quality and the amount of calculation (or CPU loading). The DECtalk Software synthesizer rate is 11,025 Hz, which is a standard PC sample rate. An 8-kHz rate is provided to support telephony applications.

People often perceive acoustic events that have different short-term spectral magnitudes as the same phoneme. For example, the *k* sound in the words *kill*

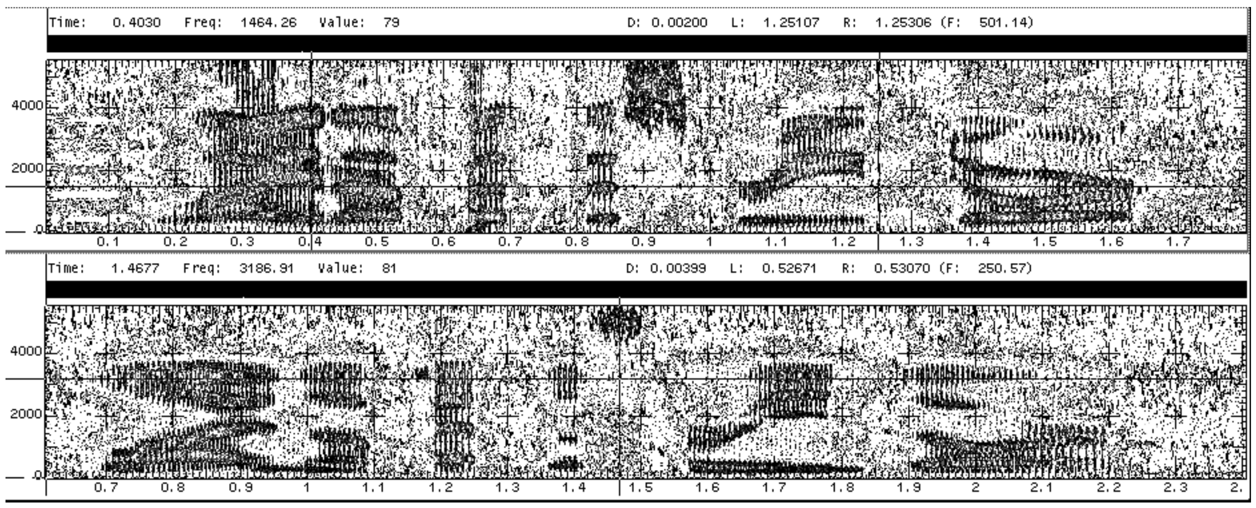


Figure 2
Two Spectrograms of the Utterance "Line up at the screen door." The upper spectrogram is the author's speech. The lower spectrogram is synthetic speech produced by DECtalk Software.

and *cool* have very different magnitude spectra. An American perceives the two spectra as the same sound; however, the sounds are very different to someone from Saudi Arabia. A Japanese person does not perceive any difference between the words *car* and *call.* To an English speaker, the *r* and the *l* sound different even though they have nearly identical magnitude spectra. The *l* sounds in the words *call* and *leaf* are different spectrally (acoustically) but have the same sound. Thus they are the same phoneme in English.

Several allophones are required to represent the *k* phoneme. Most consonant phonemes require several different allophones because the vowel sounds next to them change their acoustic manifestations. This effect, called coarticulation, occurs because it is often unnecessary for the articulatory organs to reach the final position used to generate a phoneme; they merely need to gesture toward the final position. Another type of coarticulation is part of the grammar of a language. For example, the phrase *don't you* is often pronounced *doan choo.*

All allophones that represent the phoneme *k* are produced by closing the velum and then suddenly opening it and releasing the breath stream. Speakers of the English language perceive all these allophones as the same sound, which suggests that synthesis may be modeled by an articulatory model of speech production. This model would presumably handle coarticulation effects that are not due to grammar. It is currently not known how to consistently determine speech organ positions (or control strategies) directly from acoustic speech data, so articulatory models have had little success for text-to-speech synthesis.[4]

For English, the voicing pitch provides cues to clause boundaries and meaning. Changing the frequency of the vibration of the vocal cords varies the pitch. Intonation is the shape of the pitch variation across a clause. The sentence "Tim is leaving." is pronounced differently than "Tim is leaving?" The latter form requires different intonation, depending on whether the intent is to emphasize that it is "Tim" who is leaving, or that "leaving" is what Tim is to do. A word or phrase is stressed by increasing its pitch, amplitude, or duration, or some combination of these. Intonation includes pitch changes due to stress and normal pitch variation across a clause. Correct intonation is not always possible because it requires speech understanding. DECtalk Software performs an analysis of clause structure that includes the form classes of both words and punctuation and then applies a pitch contour to a clause. The form class definitions include symbols for the parts of speech (article, adjective, adverb, conjunction, noun, preposition, verb, etc.) and symbols to indicate if the word is a number, an abbreviation, a homograph, or a special word (requiring special proprietary processing). For the sentence, "Tim is leaving?" the question mark causes DECtalk Software to raise the final pitch, but no stress is put on "Tim" or "leaving." Neutral intonation sometimes sounds boring, but at least it does not sound foolish.

Text-to-Speech Synthesis Techniques

Early attempts at text-to-speech synthesis assembled clauses by concatenating recorded words. This technique produces extremely unnatural-sounding speech.

In continuous speech, word durations are often shortened and coarticulation effects can occur between adjacent words. There is also no way to adjust the intonation of recorded words. A huge word database is required, and words that are not in the database cannot be pronounced. The resulting speech sounds choppy.

Another word concatenation technique uses recordings of the formant patterns of words. A formant synthesizer smoothes formant transitions at the word boundaries. A variation of this technique uses linear predictive coded (LPC) words. An advantage of the formant synthesizer is that the pitch and duration of words may be varied. Unfortunately, since the phoneme boundaries within a word are difficult to determine, the pitch and duration of the individual phonemes cannot be changed. This technique also requires a large database. Again, a word can be spoken only if it is in the database. In general, the quality is poor, although this technique has been used with some success to speak numbers.

A popular technique today is to store actual speech segments that contain phonemes and phoneme pairs. These speech segments, known as diphones, are obtained from recordings of human speech. They contain all coarticulation effects that occur for a particular language. Diphones are concatenated to produce words and sentences. This solves the coarticulation problem, but it is impossible to accurately modify the pitch of any segment. The intonation across a clause is generally incorrect. Even worse, the pitch varies from segment to segment within a word. The resulting speech sounds unnatural, unless the system is speaking a phrase that the diphones came from (this is a devious marketing ploy). Nevertheless, diphone synthesis produces speech that is fairly intelligible. Diphone synthesis requires relatively little compute power, but it is memory intensive. American English requires approximately 1,500 diphones; diphone synthesis would have to provide a large database of approximately 3 megabytes for each voice included by the system.

DECtalk Software uses a digital formant synthesizer. The synthesizer input is derived from phonemic symbols instead of stored formant patterns as in a conventional formant synthesizer. Intonation is based on clause structure. Phonetic rules determine coarticulation effects. The synthesizer requires only two tables, one for each gender, to map allophonic variations of each phoneme to acoustic events. Modification of vocal tract parameters in the synthesizer allows the system to generate multiple voices without a significant increase in storage requirements. (The DECtalk code and data occupy less than 1.5 megabytes.)

Poor-quality speech is difficult to understand and causes fatigue. Linguists use standard phoneme recognition tests and comprehension tests to measure the intelligibility of synthetic speech. The DECtalk family of products achieves the highest test scores of all text-to-speech systems on the market.[5] Visually handicapped individuals prefer DECtalk over all other text-to-speech systems.

How DECtalk Software Works

DECtalk Software consists of eight processing threads: (1) the text-queuing thread, (2) the command parser, (3) the letter-to-sound converter, (4) the phonetic and prosodic processor, (5) the vocal tract model (VTM) thread, (6) the audio thread, (7) the synchronization thread, and (8) the timer thread. The text, VTM, audio, synchronization, and timer threads are not part of the DECtalk PC software (the DECtalk PC VTM is on a special Digital Signal Processor) and have been added to DECtalk Software. The audio thread creates the timer thread when the text-to-speech system is initialized. Since the audio thread does not usually open the audio device until a sufficient number of audio samples are queued, the timer thread serves to force the audio to play in case any samples have been in the queue too long. The DECtalk Software threads perform serial processing of data as shown in Figure 3.

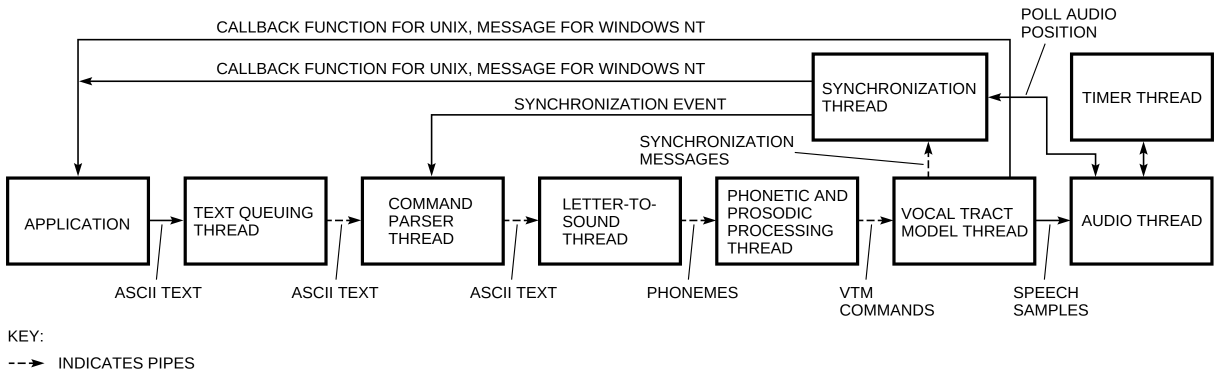


Figure 3
The DECtalk Software Architecture for Windows NT

Multithreading allows a simple and efficient means of throttling data in multistage, real-time systems. Each thread passes its output to the next thread through pipes. Each thread has access to two pipe handles, one for input and one for output. Most threads consist of a main loop that has one or more calls to a read_pipe function followed by one or more calls to a write_pipe function. The write_pipe function will block processing and suspend the thread if the specified pipe does not have enough free space to receive the specified amount of data. The read_pipe function will block processing and suspend the thread if the specified pipe does not contain the requested amount of data. Thus an active thread will eventually become idle, either because there is not enough input data, or because there is no place to store its output.

The pipes are implemented as ring buffers. The ring buffer item count is protected by mutual-exclusion objects on the Digital UNIX operating system and by critical sections on the Windows NT operating system. The pipes are created at text-to-speech initialization and destroyed during shutdown. The DECtalk Software team implemented these pipes because the pipe calls supplied with the Digital UNIX and Windows NT operating systems are for interprocess communication and are not as efficient as our pipes.

The DECtalk Software threads all used different amounts of CPU time. The data bandwidth increases at the output of every thread between the command thread and the VTM thread. Since the VTM produces audio samples at a rate exceeding 11,025 samples per second, it is no surprise that the VTM uses the most CPU time of all threads. Table 2 gives the percentage of the total application time used by each thread when the Windows NT sample application "say" is continuously speaking a large text file on an Alpha AXP 150 PC product. The output sample rate is 11,025 Hz. Note that the "say" program main thread blocks and uses virtually no CPU time after queuing the text block. These percentages have been calculated from times obtained using the Windows NT performance monitor tool.

Because the data bandwidth increases at the output of successive threads, it is desirable to adjust the size of each of the pipes ring buffers. If one imagines that all the pipes had an infinite length (and the audio queue was infinite) and that the operating system switched thread context only when the active thread yielded, then the text thread would process all the ASCII text data before the letter-to-sound thread would run. Likewise, each successive thread would run to completion before the next thread became active. The system latency would be very high, but the thread switching would be minimized. The system would use 100 percent of the CPU until all the text was converted to audio, and then the CPU usage would become very low as the audio played out at a fixed rate. Alternatively, if all the pipes are made very short, the system latency is low. In this case, all but one of the threads will become blocked in a very short time and the startup transient in the CPU loading will be minimized. Unfortunately, the threads will constantly switch, resulting in poor efficiency. What is needed is a trade-off between these two extremes.

Table 2
DECtalk Software Thread Loading

Thread	Percentage of Total Application CPU Time
Application (say.exe)	1.0
Text queue	0.2
Command parser	1.4
Letter-to-sound processing	2.4
Prosodic and phonetic processing	18.3
Vocal tract model	71.9
Audio	2.9
Synchronization	0.0
Timer	0.0
System	1.9

For a specified latency, the optimum pipe sizes that minimize memory usage for a given efficiency are in a ratio such that each pipe contains the same temporal amount of data. For example, let us assume that 64 text characters (requiring 64 bytes) are in the command thread. They produce approximately 100 phonemes (requiring 1,600 bytes) at the output of the letter-to-sound thread and approximately 750 VTM control commands (requiring 15,000 bytes) at the output of the prosodic and phonetics thread. In such a case, the size of the input pipes for the command, letter-to-sound, and prosodic and phonetic threads could be made 64, 1,600, and 15,000 bytes, respectively, to minimize pipe memory usage for the specified latency. (All numbers are hypothetical.) The pipe sizes in DECtalk Software actually increase at a slightly faster rate than necessary. We chose the faster rate because memory usage is not critical since all the pipes are small relative to other data structures. The size of the VTM input pipe is the most critical: it is the largest pipe because it supports the largest data bandwidth.

The Text Thread

The text thread's only purpose is to buffer text so the application is not blocked during text processing. An application using text-to-speech services calls the TextToSpeechSpeak API function to queue a nullterminated text string to the system. This API function copies the text to a buffer and passes the buffer (using a special message structure) to the text thread. This is done using the operating system's PostMessage function for Windows NT and a thread-safe linked list for Digital UNIX. After the text thread pipes the entire text stream to the command thread, it frees the text buffer and the message structure.

The Command Processing Thread

The command processing thread parses in-line text commands. These commands control the text-to-speech system voice selection, speaking rate, and audio volume, and adjust many other system state parameters. For DECtalk, most of these commands are of the form [: command <parameters>]. The string "[:" specifies that a command string follows. The string "]" ends a command. The following string illustrates several in-line commands.

```
[:nb][:ra 200] My name is Betty.
[:play audio.wav]
[:dial 555-1212][:tone 700 1,000]
```

This text will select the speaker voice for "Betty," select a speaking rate of 200 words per minute, speak the text "My name is Betty." and then play a wave audio file named "audio.wav." Finally, the DTMF tones for the number 555-1212 are played followed by a 700-Hz tone for 1,000 milliseconds.

Because the text-to-speech system may be speaking while simultaneously processing text in the command thread, it is necessary to synchronize the command processing with the audio. The DECtalk PC product (from which we ported the code) did not perform synchronization unless the application placed a special string before the volume command. For DECtalk Software, asynchronous control of all functions provided by the in-line commands is already available through the text-to-speech API calls. For this reason, the DECtalk Software in-line commands are all synchronous.

The DECtalk command [:volume set 70] will set the audio volume level to 70. Synchronization is performed by inserting a synchronization symbol in the text stream. This symbol is passed through the system until it reaches the VTM thread. When the VTM thread receives a synchronization symbol, it pipes a message to the synchronization thread. This message causes the synchronization thread to signal an event as soon as all audio (that was queued before the message) has been played. The volume control code in the command thread is blocked until this event is signaled. The synchronization thread also handles commands of the form [:index mark 17]. Index mark commands may be used to send a message value (in this case 17) back to an application when the text up to the index mark command has been spoken.

The command thread passes control messages such as voice selection and speaking rate to the letter-to-sound and the prosodic and phonetic processing threads, respectively. Tone commands, index mark commands, and synchronization symbols are formatted into messages and passed to the letter-to-sound thread. The command thread also pipes the input text string, with the bracketed command strings removed, to the letter-to-sound thread.

The Letter-to-Sound Thread

The letter-to-sound (LTS) thread converts ASCII text sequences to phoneme sequences. This is done using a rule-based system and a dictionary for exceptions. It is the single most complicated piece of code in all of DECtalk Software. Pronunciation of English language words is complex. Consider the different pronunciations of the string *ough* in the words *rough, through, bough, thought, dough, cough,* and *hiccough*.[6] Even though the LTS thread has more than 1,500 pronunciation rules, it requires an exception dictionary with over 15,000 words.

Each phoneme is actually represented by a structure that contains a phonemic symbol and phonemic attributes that include duration, stress, and other proprietary tags that control phoneme synthesis. This is how allophonic variations of a phoneme are handled. In the descriptions that follow, the term phoneme refers either to this structure or to the particular phone specified by the phonemic symbol in this structure.

The LTS thread first separates the text stream into clauses. Clause separation occurs in speech both to encapsulate a thought and because of our limited lung capacity. Speech run together with no breaks causes the listener (and the speaker) to become fatigued. Correct clause separation is important to achieve natural intonation. Clauses are delineated by commas, periods, exclamation marks, question marks, and special words. Clause separation requires simultaneous analysis of the text stream. For example, an abbreviated word does not end a clause even though the abbreviation ends in a period. If the text stream is sufficiently long and no clause delimiter is encountered, an artificial clause boundary is inserted into the text stream.

After clause separation, the LTS thread performs text normalization. For this, the LTS thread provides special processing rules for numbers, monetary amounts, abbreviations, times, in-line phonemic sequences, and even proper names. Text normalization usually refers to text replacement, but in many cases the LTS thread actually inserts the desired phoneme sequence directly into its output phoneme stream instead of replacing the text.

The LTS thread converts the remaining unprocessed words to phonemes by using either the exception dictionary or a rule-based "morph" lexicon. (The term *morph* is derived from morpheme, the minimum unit of meaning for a language.) By combining these two approaches, memory utilization is minimized. A user-definable dictionary may also be loaded to define application-specific terms. During this conversion, the LTS thread assigns one or more form classes to each word. As mentioned previously, form class definitions include symbols for abbreviations and homographs. A homograph is a word that has more than one pronunciation, such as *alternate* or *console.* DECtalk Software pronounces most abbreviations and homographs correctly in context. An alternate pronunciation of a homograph may be forced by inserting the in-line command [:pron alt] in front of the word. DECtalk Software speaks the phrase "Dr. Smith lives on Smith Dr." correctly, as "Doctor Smith lives on Smith Drive." It uses the correct pronunciation of the homograph *lives.*

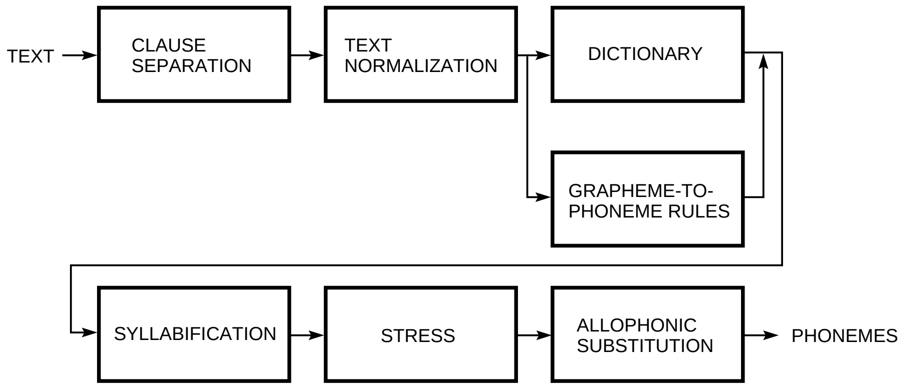
Before applying rules, the LTS thread performs a dictionary lookup for each unprocessed word in a clause. If the lookup is successful, the word's form classes and a stored phoneme sequence are extracted from the dictionary. Otherwise, the word is tested for an English suffix, using a suffix table. If a suffix is found, sometimes the form class of the word can be inferred. Suffix rules are applied, and the dictionary lookup is repeated with the new suffix-stripped word. For example, the word *testing* requires the rule, locate the suffix *ing* and remove it; whereas the word *analyzing* requires the rule, locate the suffix *ing* and replace it with *e.* The suffix rules and the dictionary lookup are recursive to handle words that end in multiple suffixes such as *endlessly.*

If the word is not in the dictionary, the LTS thread performs a decomposition of the word using morphs. DECtalk uses a morph table to look up the phonemic representation of portions of words. A morph always maps onto one or more English words and can be represented by a letter string. Morphs generally consist of one or more roots that may contain affixes and suffixes. Although new words may frequently be added to a language, new morphs are rarely added. They are essentially sound groupings that make up many of the words of a language. DECtalk contains a table with hundreds of morphs and their phonemic representations. Either a single character or a set of characters that results in a single phoneme is referred to as a grapheme. Thus this portion of the letter-to-sound conversion is referred to as the grapheme-to-phoneme translator. Figure 4 shows the architecture of the LTS thread.

Morphemes are abstract grammatical units and were originally defined to describe words that can be segmented, such as *tall, taller,* and *tallest.* The word *tallest* is made from the morphemes *tall* and *est.* The word *went* decomposes into the morphemes *go* and PAST. Thus a morpheme does not necessarily map directly onto a derived word. Many of the pronunciation rules are based on the morphemic representations of words.

Many morphs have multiple phonemic representations that can depend on either word or phonemic context. The correct phonemic symbols are determined by morphophonemic rules. For example, plural words that end in the morpheme *s* are spoken by appending either the *s,* the *z,* or the *eh z* plural morphemes (expressed as Arpabet phonemic symbols) at the end of the word.[7] Which allomorph is used depends on the final phoneme of the word. Allomorphs are morphemes with alternate phonetic forms. For another example requiring a morphophonemic rule, consider the final phoneme of the word *the* when pronouncing "the apple," and "the boy."

After applying many morphophonemic rules to the phonemes, the LTS thread performs syllabification, applies stress to certain syllables, and performs allophonic recoding of the phoneme stream. The LTS



Note that the grapheme-to-phoneme rules are used only if the dictionary lookup fails.

Figure 4
Block Diagram of the Letter-to-Sound Processing Thread

thread groups phonemes into syllables, using tables of legal phoneme clusters and special rules. The syllabification must be accurate, because the LTS thread applies stress between syllable boundaries.

The LTS thread then assigns either primary stress, secondary stress, or no stress to each syllable. The stress rules are applied in order. They assign stress only to syllables that have not had stress previously assigned. These rules take into account the number of syllables in a word and the positions of affixes that were found during morph decomposition of a word.

Allophonic rules are the last rules the LTS thread applies to the phoneme stream. These are really phonetic rules. Most allophonic rules are described as follows: "if phoneme A is followed by phoneme B, then modify (or delete) phoneme A (or B)." Most allophonic rules are not applied across morpheme boundaries. These rules handle many specific cases; for example, the *p* in the word *spit* is aspirated, whereas the *p* in the word *pit* is not. The *s* phoneme modifies the articulation of the *p.* The *s* phoneme is different in the words *stop* and *street* because the *r* sound is anticipated and modifies the *s* in the word *street.* This last example is called distant assimilation.

The LTS thread passes the phonemes that include durations and lexical information to the prosodic and phonetic processing thread. Tone, dial, index mark, and synchronization messages are passed unmodified through the LTS thread.

The Phonetic and Prosodic Processing Thread

The phonetic and prosodic processing (PH) thread, shown in Figure 5, converts the phoneme stream to a series of vocal tract control commands. Both prosodic rules and additional phonetic rules are applied to the input phoneme stream.[8] Prosody refers to clause-based stress, intonation, and voice quality in speech. Words are stressed to add meaning to a clause. Stress is achieved by increasing one or more of either the pitch, the duration, or the amplitude of an utterance. The phonetic rules handle coarticulation effects and adjust phoneme durations based on the form class, the clause position, and the speaking rate. One example is a rule that increases the duration of the final stressed phoneme in a clause. Additional context-dependent phonetic coarticulation rules can adjust the durations of phonemes or delete them.

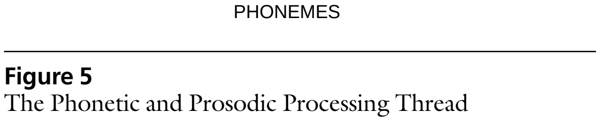


Figure 5
The Phonetic and Prosodic Processing Thread

The correct application of stress, like intonation, requires understanding, so DECtalk Software generally applies syllabic stress only as part of an intonation contour across a clause. Intonation contours are generated by fixed rules. In most clauses, the pitch rises at the start of the clause and falls at the end of the clause. This basic form is changed for questions, prepositional phrases, exclamations, compound nouns, and numbers. This intonation is also changed based on the syllabic stress assigned by the LTS thread. The PH thread can also process pitch control symbols that are placed in-line with text. These pitch commands are parsed in the command thread and pass through the LTS thread.

The PH thread uses each phoneme symbol and its context to generate any allophonic variation of the phoneme. The resulting allophone symbol indexes into one of two tables, one table for each gender. Each allophone symbol indexes a set of parameters that includes voicing source amplitude, noise source amplitude, formant frequencies, and formant bandwidths. These, along with voicing source pitch and a number of fixed speaker-dependent parameters, make up the VTM parameters. A new set of parameters is generated for every 6.4 milliseconds of speech. The VTM thread uses these parameters, which are collectively called a voice packet, to generate the speech waveform.

In addition to sending voice packets to the VTM thread, the PH thread can send a speaker packet to select a new speaking voice. The voice is selected either by an in-line text command or by the application calling a specific API function. The PH thread has fixed tables of parameters for each voice. There are many voice parameters, but some of the more interesting ones include the gender, the average pitch, the pitch range, the assertiveness, the breathiness, and the formant scale factor. The gender is used by some of the PH rules and by the PH thread to select the table used to generate voice packets. The average pitch and the pitch range are used by the PH thread to set the pitch characteristics for the VTM's voicing source. The assertiveness parameter sets the rate of fall of the pitch at the end of a clause. A high assertiveness factor results in an emphatic voice. The breathiness parameter sets the amount of noise that is mixed with the voiced path signal. The formant scale factor effectively scales the size of the speaker's trachea.

Tone, dial, index mark, and synchronization messages are passed unmodified through the PH thread.

The Vocal Tract Model Thread

The Vocal Tract Model (VTM) thread processes speaker packets, voice packets, tone messages, and synchronization messages. Speaker packets set the speaker-voice-dependent parameters of the VTM.

One of these, the formant scale factor, is multiplied by the first, second, and third formant frequencies in each voice packet. Other parameters include the values for the frequencies and bandwidths of the fourth and fifth formants, the gains for the voiced path of the VTM, the frication gain for the unvoiced path of the VTM, the speaker breathiness gain, and the speaker aspiration gain.

Each voice packet produces one speech frame of data. The output sample rate for DECtalk Software is either 8,000 Hz or 11,025 Hz. For each of these sample rates, a frame is 51 and 71 samples respectively. Each voice packet includes frequencies and bandwidths for the first, second, and third formants, the nasal antiresonator frequency, the voicing source gain, and gains for each of the parallel resonators. Figure 6 shows the basic architecture of the VTM.[9] The VTM, in conjunction with the PH rules, simulates the speech organs.

The VTM consists of two major paths, a voiced path and an unvoiced path. The voiced path is excited by a pulse generator that simulates the vocal cords. A number of resonant filters in series simulate the trachea. These cascaded resonators simulate a cascade of tubes of varying widths.[10] A nasal filter in series with the resonant tube model simulates the dominant resonance and antiresonance of the nasal cavity.[11] The cascade resonators and the nasal filter complete the "voiced" path of the VTM.

Unvoiced sounds occur as a result of chaotic turbulence produced when breath passes through a constriction. This turbulence is difficult to model. In our approach, the VTM matches the spectral magnitude of filtered noise with the spectral magnitude of the desired unvoiced phoneme (allophone). The noise source is realized by filtering the output of a uniform-distribution random number generator. Unvoiced sounds contain both resonances and antiresonances.

Another approach to obtain an appropriate frequency characteristic is to filter the noise source signal using a series of parallel resonators. A consequence of putting resonators in parallel is to create antiresonances. The positions of these antiresonances are dependent on the parallel formant frequencies, but it has been empirically determined that this model provides more than enough degrees of freedom to closely match the spectral magnitude of any unvoiced sound. The noise source generates fricatives, such as *s,* plosives, such as *p,* and aspirates, such as *h.* The noise source also contributes to some voiced sounds, such as *b, g,* and *z.* The noise source output may also be added to the input of the voiced path to produce aspiration. To generate breathy vowels, the parallel formant frequencies are set equal to the cascade formant frequencies.[12]

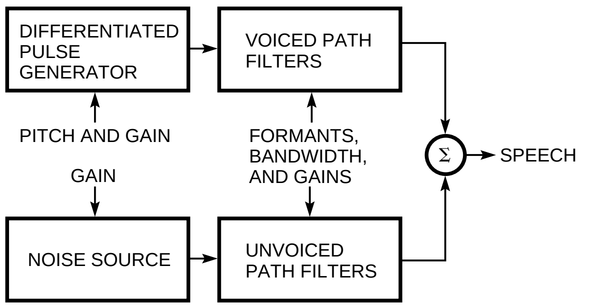


Figure 6
Basic Architecture of the Vocal Tract Model

The radiation characteristic of the lips approximates a differentiation (derivative) of the acoustic pressure wave. Since all the filters in the VTM are linear and time-invariant, the radiation effects can be incorporated in the signal sources instead of at the output. Therefore the glottal source (pulse source) produces differentiated pulses. The differentiated noise signal is the filtered first difference of a uniform-distribution random number generator.

The DECtalk Software VTM (also known as the Klatt Synthesizer) is shown in Figure 7. The italicized terms are either speaker-dependent parameters or constant values. All other parameters are updated every frame. Depending on the system mode, the audio samples generated for each frame are passed to the output routine and subsequently are either queued to the audio device, written to a wave audio file, or written to a buffer provided by the application. After generating a speech frame, the VTM code increases the audio sample count by the frame size. This count is sent to the synchronization thread whenever a synchronization symbol or an index mark is received by the VTM thread. The count is reset to zero at startup and whenever the text-to-speech system is reset.

Tone messages are processed by the VTM thread. Tone messages are for single tones or DTMF signals. Each tone message includes two frequencies, two amplitudes (one for each frequency), and one duration. For a single tone message, the amplitude for the second frequency is zero. Tone synthesis code generates tone frames and queues them to the output routine. The first 2 milliseconds and the last 2 milliseconds of a tone signal are multiplied by either a rising or a falling cosine-squared shaping function to limit the out-of-band pulse energy. Each tone sample is synthesized using a sinusoid look-up table.[13]

The Synchronization Thread

The synchronization thread is idle unless the VTM thread forwards a synchronization symbol message or an index mark message. Both messages contain the current audio sample count. The index mark message

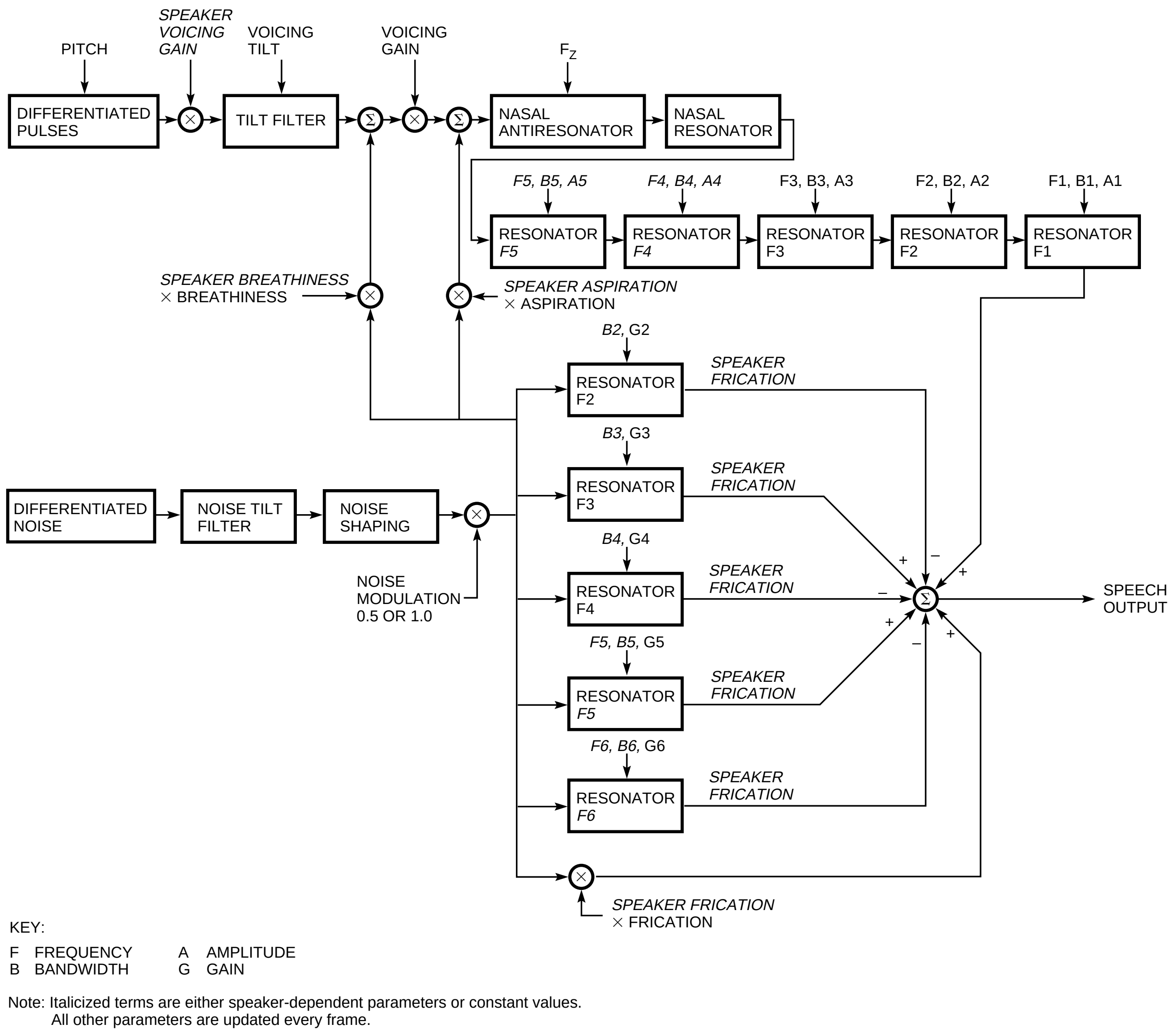


Figure 7
The DECtalk Software Vocal Tract Model (also known as the Klatt Synthesizer)

also contains an index mark number from 0 to 99. After receiving one of these messages, the synchronization thread periodically polls the audio thread until the indicated audio sample has been played. If the message contained a synchronization symbol, an event is set that unblocks the command thread. If it is an index mark message, the synchronization thread sends the index mark number back to the application. For the Digital UNIX operating system, this number is returned by calling a callback function that the application specifies when DECtalk Software is started. For the Windows NT operating system, the SendMessage function is used to return the index mark number to the application. The message is sent to a window procedure specified by the window handle that is provided when the text-to-speech system is started.

The Audio Thread

The audio thread manages all activities associated with playing audio through the computer's sound hardware. An audio API insulates DECtalk Software from the differences between operating systems. The audio API communicates with the audio thread. The VTM thread calls an audio API queuing function that writes samples to a ring buffer that is read only by the audio thread. The audio thread opens the audio device after approximately 0.8 seconds of audio samples have been queued and closes the audio device when there are no more samples to play. If the number of audio samples in the queue is too small to cause the audio device to be opened, and the flow rate (measured over a 100-millisecond interval) into the audio ring buffer is zero, the timer thread will send the audio thread a message that causes the audio device to open and start playing audio. When audio either starts or stops playing, a message is sent to the application.

For the Digital UNIX operating system, the audio thread is an interface to the low-level audio functions of the Multimedia Services for Digital UNIX (MMS) product. MMS provides a server to play audio and video.

For the Windows NT operating system, the implementation also uses the system low-level audio functions, but these functions interface directly with a system audio driver. The audio API provides capabilities to pause the audio, resume paused audio, stop audio from playing and cancel all queued audio, get the audio volume level, set the audio volume level, get the number of audio samples played, get the audio format, and set the audio format. An in-line play command can be used to play audio files. DECtalk Software uses the get format and set format audio capabilities to dynamically change the audio format so it can play an audio file that has a format different from the format generated by the VTM.

DECtalk Software API

In the mid-1980s, researchers at Digital's Cambridge Research Lab ported the DECtalk text-to-speech C language-based code to the ULTRIX operating system. The command, LTS, PH, and VTM portions of the system were different processes. The pipes were implemented using standard UNIX I/O handles, `stdin` and `stdout`. These, along with an audio driver process, were combined into a command procedure. This system lacked many of the rules and features found in DECtalk Software today, but it did demonstrate that real-time speech synthesis was possible on a workstation. Before this time, DECtalk required specialized Digital signal-processing hardware for real-time operation.[14] On a DECstation Model 5000/25 workstation, the text-to-speech implementation used 65 percent of the CPU. If the output sample rate of this system had been raised from 8,000 Hz to 11,025 Hz, the highest-quality rate provided by DECtalk Software, it would have loaded approximately 89 percent of the CPU. Workstation text-to-speech synthesis, while possible, was still very expensive.

The power of the Alpha CPU has changed this. Today, many copies of DECtalk Software can run simultaneously on Alpha-based systems. Speech synthesis is now a viable multimedia form. This change created the need for a text-to-speech API. Table 3 shows the DECtalk Software CPU load for various computers.

On Alpha systems, the performance of DECtalk Software depends primarily on the SPECmark rating of the computer. A lesser consideration is the secondary cache size. System bus bandwidth is not a limiting factor: The combined data rates for the text, phonemes, and audio are extremely low relative to modern bus speeds, even when running the maximum number of real-time text-to-speech processes that the processor can support.

The API we have developed is the result of collaboration between several organizations within Digital: the Light and Sound Group, the Assistive Technology Group, the Cambridge Research Lab, and the Voice

Table 3
DECtalk Software CPU Loading versus Processor SPECmarks

System	Clock (MHz)	Processor	Secondary Cache (MB)	SPECint92	SPECfp92	Audio Rate (kHz)	Total CPU Load (%)
Alpha AXP 150 PC	150	Alpha 21064	512	80.9	110.2	11,025	8
AlphaStation 250 4/266 workstation	266	Alpha 21064	2,048	198.6	262.5	11,025	2.4
DEC 3000 Model 800 workstation	200	Alpha 21064	2,048	138.4	188.6	11,025	5
DEC 3000 Model 900 workstation	275	Alpha 21064A	2,048	230.6	264.1	11,025	3
AlphaStation 400 4/233 workstation	233	Alpha 21064A	512	157.7	183.9	11,025	3
AlphaStation 600 5/266 workstation	266	Alpha 21164	2,048	288.6	428.6	8,000	1
XL 590 PC	90	Pentium	512	Unknown	N/A	11,025	24

and Telecom Engineering Group. We had two basic requirements: We wanted the API to be easy to use and to work with any text-to-speech system. While creating the API, we defined interfaces so that future improvements to the text-to-speech engine would not require any API calls to be changed. (Customers frown on product updates that require rewriting code.) Some decisions were controversial. Some contributors felt that the text-to-speech system should return speech samples only in memory buffers, and the application should shoulder the burden of interfacing to the workstation's audio subsystem. The other approach was to support the standard workstation audio (which is platform dependent) and to provide an API call that switched the system into a speech-to-memory mode. We selected the latter approach because it simplifies usage for most applications.

The API Functions

The core text-to-speech API functions are the TextToSpeechStartup function, the TextToSpeechSpeak function, and the TextToSpeechShutdown function. The simplest application might use only these three functions.

All applications using text-to-speech must call the TextToSpeechStartup function. This function creates all the DECtalk system threads and passes back a handle to the text-to-speech system. The handle is used in subsequent text-to-speech API calls. The startup function is the only API function that has different arguments for the Digital UNIX and the Windows NT operating systems. This is necessary because the asynchronous reporting mechanism is a callback function for Digital UNIX and is a system message for Windows NT. The TextToSpeechShutdown function frees all system resources and shuts down the threads. This would normally be called when closing the application.

The TextToSpeechSpeak function is used to queue text to the system. If an entire clause is not queued, no output will occur until the clause is completed by queuing additional text. A special TTS_FORCE parameter may be supplied in the function call to force a clause boundary. The TTS_FORCE parameter is necessary for applications that have no control over the text source and thus cannot guarantee that the final text forms a complete clause.

The text-to-speech API provides three audio output control functions. These pause the audio output (TextToSpeechPause), resume output after pausing (TextToSpeechResume), and reset the text-to-speech system (TextToSpeechReset). The reset function discards all queued text and stops all audio output.

The text-to-speech API also provides a special synchronization function (TextToSpeechSync) that blocks until all previously queued text has been spoken. This API call may not return for days if a sufficient amount of text is queued. (Index marks provide nonblocking synchronization.)

The API supplies functions to both load (TextToSpeechLoadUserDictionary) and unload (TextToSpeechUnloadUserDictionary) an application-defined dictionary. The dictionary contains words and their phonemic representations. The developer creates a dictionary using a window-based user-dictionary tool. This tool can speak words and their phonemic representations. It can also convert text sequences to phonemic sequences. This last feature frees the developer from having to memorize and use the DECtalk Software phonemic symbols.

Additional functions select the speaker voice, control the speaking rate, control the language, determine the system capabilities, and return status. The status API function can indicate if the system is currently speaking.

Special Text-to-Speech Modes

DECtalk Software has three special modes: the speech-to-wave file mode, the log-file mode, and the speech-to-memory mode. Each mode has two complementary calls, one to enter the mode and one to exit. When in the speech-to-wave file mode, the system writes all speech samples to a wave audio file. The file is closed when exiting this mode. This is useful on slower Intel systems that cannot perform real-time speech synthesis. The log-file mode causes the system to write the phonemic symbol output of the LTS thread to a file. The last mode is the speech-to-memory mode. After entering this mode, the application uses a special API call to supply the text-to-speech system with memory buffers. The text-to-speech system writes synthesized speech to these buffers and returns the buffer to the application. The buffers are returned using the same mechanism used for index marks, a callback function on the Digital UNIX operating system and a system message on the Windows NT operating system. These buffers may also return index marks and phonemic symbols and their durations. If the text-to-speech system is in speech-to-memory mode, calling the reset function causes all buffers to be returned to the application.

Porting DECtalk Software

The DECtalk PC code used a simple assembly language kernel to manage the threads. The existence of threads on our target platforms simplified porting the code. The thread functions, signals (such as conditions or events), and mutual exclusion objects are different for the Digital UNIX and the Windows NT operating systems. Since these functions occur mainly in the pipe code and the audio code, we maintain different versions of code for each system. The message-passing mechanism for Windows NT has no equivalent on Digital UNIX; therefore part of the API code had to be different. The command, LTS, and PH threads are all common code for Digital UNIX and Windows NT. Most of the VTM thread is also common code.

Porting the code for each thread required putting conditional statements that define thread entry points into each module for each supported operating system. We also had to add special code to each thread to support our API call that resets the text-to-speech system. The reset is the most complicated API operation, because the data piped between threads is in the form of variable-length packets. During a reset, it is incorrect to simply discard data within a pipe because the thread that reads the pipe will lose data synchronization. Therefore a reset causes each thread to loop and discard all input data until all the pipes are empty. Then each thread's control and state variables are set to a known state. In many complicated systems, resetting and shutting down are the most complicated parts of a control architecture. System designers should incorporate mechanisms to simplify these functions.

The VTM code is much shorter and simpler than the code in either the LTS or the PH thread, but it is by far the largest CPU load in the system. The DECtalk PC hardware used a specialized Digital Signal Processor (DSP) for the VTM. The research VTM code (written in the C language) was rewritten to be sample-rate-independent. The filters were all made in-line macros. With this new VTM, the DECtalk Software system loaded an Alpha AXP 150 PC product 31 percent. After rewriting this code using floating-point arithmetic and then converting it to assembly language, DECtalk Software loaded the processor less than 8 percent. (Both tests were conducted at an 11,025-Hz output sample rate.)

There are several reasons a floating-point VTM runs faster than an integer VTM on an Alpha system. An integer VTM requires a separate gain for each filter to keep the output data within the filter's dynamic range. For a floating-point VTM, the gains of all cascaded filters are combined into one gain. The increased dynamic range allows combining parts of some filters to reduce computations. Also, floating-point operations do not require additional instructions to perform scaling. The processor achieves greater instruction throughput because it can dual issue floating-point instructions with integer instructions, which are used for pointers, indices, and some loop counters. Finally, the current generation of Alpha processors performs some floating-point operations with less pipeline latency than their equivalent integer operations (note the SPECfp92 and SPECint92 ratings of the current Alpha processors listed in Table 3).

The integer VTM is faster than the floating-point VTM on Intel processors, so we maintain two versions of the VTM. Both versions support multiple sample rates. The pitch of the glottal source and the frequencies and bandwidths of the filters are adjusted for the output sample rate. When necessary, the filter gains are adjusted. These extra calculations do not add much to the total time used by the VTM because they are performed only once per frame.

Possible Future Improvements to DECtalk Software

The Assistive Technology Group continues to improve the letter-to-sound rules, the prosodic rules, and the phonetic rules. Future implementations could use object-oriented techniques to represent the dictionaries, words, phonemes, and parts of the VTM. A larger dictionary with more syntactic information can be added. There has even been some discussion of combining the LTS and PH threads to make more efficient use of lexical knowledge in PH. The glottal waveform generator can be improved. Syntactic parsers might provide the information required for more accurate intonation. Someday, semantic parsing (text understanding) may provide a major improvement in synthetic speech intonation. Researchers both within and outside of Digital are investigating these and many other areas. It seems likely that the American English version of DECtalk Software will continue to improve over time.

Summary

DECtalk Software provides natural-sounding, highly intelligible text-to-speech synthesis. It was developed to perform on the Digital UNIX operating system on Digital's Alpha-based platforms and with Microsoft's Windows NT operating system on both Alpha and Intel processors. It is based on the mature DECtalk PC hardware product. DECtalk Software also provides an easy-to-use API that allows applications to use the workstation's audio subsystem, to create wave audio files, and to write the speech samples to application-supplied memory buffers. An Alpha-based workstation can run many copies of DECtalk Software simultaneously.

DECtalk Software uses a dictionary and linguistic rules to convert speech to phonemes. An application-supplied dictionary can override the default pronunciation of a word. Prosodic and phonetic rules modify the phoneme's attributes. A vocal tract model synthesizes each phoneme to produce a speech waveform. The result is the highest-quality text to speech. The Assistive Technology Group continues to improve the DECtalk text-to-speech algorithms.

Acknowledgments

I wish to acknowledge and thank all the members of the DECtalk Software project and additional support staff. Bernie Rozmovits, our engineering project leader, was the visionary for this entire effort. He contributed most of our sample applications on Windows NT, and he also wrote the text-to-speech DDE server. Krishna Mangipudi, Darrell Stam, and Hugh Enxing implemented DECtalk Software on the Digital UNIX operating system. Thanks to Bill Scarborough who did a great job on all of our documentation, particularly the on-line help. Special thanks to Dr. Tony Vitale and Ed Bruckert, from Digital's Assistive Technology Group. They both were instrumental in developing the DECtalk family of products and are continuing to improve them. Without their efforts and support, DECtalk Software could not exist. Tom Levergood and T. V. Raman at Digital's Cambridge Research Lab helped test DECtalk Software and provided many suggestions and improvements. Thanks also to the engineering manager for Graphics and Multimedia, Steve Seufert, who continues to support our efforts. Finally, we are all indebted to Dennis Klatt who was the creator of the DECtalk speech synthesizer and to all the other developers of the original DECtalk hardware products.

References

1. G. Fant, *Acoustic Theory of Speech Production* (The Netherlands: Mouton and Co. N.V., 1960).
2. C. Schmandt, *Voice Communication with Computers* (New York: Van Nostrand Reinhold, 1994).
3. J. Allen, M. Hunnicutt, and D. Klatt, *From Text to Speech: The MITalk System* (Cambridge, Mass.: Cambridge University Press, 1987).
4. J. Flanagan, *Speech Analysis, Synthesis, and Perception,* 2d ed. (New York: Springer-Verlag, 1972).
5. D. Pisoni, H. Nusbaum, and B. Greene, "Perception of Synthetic Speech Generated by Rule," *Proceedings of the IEEE,* vol. 73, no. 11 (1985): 1665–1676.
6. A. Vitale and M. Divay, "Algorithms for Grapheme-Phoneme Translation in French and English" (in preparation).
7. V. Fromkin and R. Rodman, *An Introduction to Language,* 2d ed. (New York: Holt, Rinehart, and Winston, 1978).
8. D. Klatt, "Review of Text-to-Speech Conversion for English," *Journal of the Acoustical Society of America,* vol. 82, no. 3 (1987): 737–793.
9. D. Klatt, "Software for a Cascade/Parallel Formant Synthesizer," *Journal of the Acoustical Society of America,* vol. 67 (1980): 971–975.
10. L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing* (London: Prentice Hall, 1975).
11. L. Rabiner and R. Schafer, *Digital Processing of Speech Signals* (London: Prentice Hall, 1978).
12. D. Klatt and L. Klatt, "Analysis, Synthesis, and Perception of Voice Quality Variations among Female and Male Talkers," *Journal of the Acoustical Society of America,* vol. 87, no. 2 (1990): 820–857.
13. J. Tierney, "Digital Frequency Synthesizers," Chapter V of *Frequency Synthesis: Techniques and Applications,* J. Gorski-Popel, ed. (New York: IEEE Press, 1975).
14. E. Bruckert, M. Minow, and W. Tetschner, "Three-Tiered Software and VLSI Aid Development System to Read Text Aloud," *Electronic* (April 21, 1983).

Biography

William I. Hallahan
Bill Hallahan is a member of the Light and Sound Group, part of Software Engineering for the Workstation Business Segment. Previously he worked in the Image, Voice, and Video Group on signal-processing algorithms and the rewriting of the DECtalk vocal tract model. Before joining Digital in 1992, he was employed at Sanders Associates for 12 years, where he developed and implemented algorithms that performed signal analysis, signal demodulation, and numerical methods. Bill received a B.S.E.E. from the University of New Hampshire in 1980. He is co-author of a patent application for a specific color-space conversion algorithm used in video multimedia applications.