Audio Software for the Intel® Play™ Computer Sound Morpher

Scott Boss, Smart Toy Lab (Connected Products Division), Intel Corp.

Index words: Consumer Products, Extended PC, Smart Toys, Signal Processing, Audio

ABSTRACT

The Intel® Play™ Computer Sound Morpher (CSM) is a hand-held audio recording device paired with easy-to-use, PC-based, sound-editing software. It is targeted at children aged eight to twelve and allows them to explore the world of sound in fun and creative ways. As is true of all Intel® Play™ toys, the CSM consists of a bundled hardware device and software suite with its own unique set of implementation challenges. This paper outlines those challenges and presents the implementation details and approaches involved in building this audio-based Smart Toy.

From early concept prototypes to final product, the CSM software evolved from a collection of toy designer dreams, user interface sketches, and audio technologies into a unified multimedia application for children. Several of the original prototypes and experiments are presented to set the stage for the final product. Both the hardware and software evolved through stages of prototyping in order to find the best mix between technical capabilities and design requirements.

This paper focuses on the audio software technologies needed to provide a full software feature set and to make those features usable for children. This behind-the-scenes look at the CSM software reveals approaches to and implementation details of several features including audio tone detection, energy detection, singing text to speech voices, visual representations of audio, and audio effects filters. The technical details of the audio components developed at the Smart Toy Lab (STL) are presented along with the integration effort needed to tie them and other third-party solutions into a software application. This paper also presents the architectural decisions made in order to balance the tug of war between required software features and the time and resources available to make them part of the product.

In the end, the audio software for CSM was feature packed, consisting of several third-party vendor components as well as several Smart Toy Lab-developed audio components. Together, these software pieces combined to present a novel approach to exploring and manipulating sound using a personal computer. When combined with the CSM hardware, the software becomes part of a complete PC play experience that brings new uses and a new perspective to the PC.

INTRODUCTION

As the hardware for the Computer Sound Morpher (CSM) took shape and became simpler in its function, the need to add significant play value with the PC became evident. The onus fell on the software to deliver a broad, open-ended play experience to help deliver the Intel Play brand promise. In this paper, we describe the solutions that were put in place to deliver a large feature set on a tight schedule.

The CSM software consists of a collection of audio technologies bound together to present a rich feature set within a child's multimedia application. The audio technologies were developed both in-house at the Smart Toy Lab and out-of-house by several external vendors. Some technologies were built from the ground up and others were licensed and integrated as-is from third parties. The combination of existing components and newly developed components presented a significant integration and validation challenge.

CHALLENGES

The challenges presented by the Computer Sound Morpher (CSM) software stemmed mainly from the breadth of the application. Individually, the sound technologies were well known; however, collectively they presented a large integration challenge. In particular, the main challenges were as follows:

- the coordination of several software development teams
- the integration of external software components
- delivering a robust feature set implementation

In total, there were eight separate software groups both within Intel and external to Intel contributing components to the CSM software effort. Three of these groups were actively developing new code as the project progressed. Mapping out and coordinating the software development of these groups involved substantial planning and monitoring. Deliverables were tracked, integrated, and evaluated continuously as the project progressed.

The end product required a seamless presentation of the various software features. The application developer was presented with the large task of integrating these components. To aid in this effort, the Smart Toy Lab (STL) invested time and resources in developing a library to combine the third-party components and present them to the application developer as a single unit. This library not only simplified the integration of components into the application, but it also served as a vital tap into the application software stack for debugging and problem-solving purposes.

Delivering a robust feature set was a broad effort including the use and development of several in-house and out-of-house components. Each component was individually validated and tested before being integrated into the application. Some of the third-party components were currently shipping products that were simply integrated as-is and only required some code to drive them from the application. Making sure these components were feature-complete and functioned without error was an involved task that was met by the STL integration team.

FROM SNOOPERS TO PHASERS

Early development of the Computer Sound Morpher (CSM) resulted in many hardware and software prototypes, each playing a critical part in the evolution of this toy. By taking a look at this history, the reader can better understand the objectives of this project and the tradeoffs made in order to produce the end product.

While there were several hardware prototypes developed, the common theme throughout the early exploration stage was capturing "fun" sounds. The first attempts focused on capturing sounds from a distance in order to deliver an exciting play pattern, i.e., listening to sounds not easily heard with the naked ear. The prototype delivered on this promise with a slick "deflector dish" design that collected sounds from a range of 50 to 80 feet in an extremely directional manner. The sound quality and sound-collection ability of the prototype was tested by wiring the microphone and dish assembly to an evaluation sound-recording board. Sounds were recorded, transferred to the PC, and then analyzed for sound quality.

Figure 1: First functional prototype

The first prototype (Figure 1) provided solid data for determining desired sound quality and sound-collection ability. Unfortunately, its form factor was much too large. The next prototype faced the challenge of meeting two significant constraints.

- a tight material cost budget
- a small form factor

Figure 2a: Collapsible dish prototype–closed

The collapsible dish prototype (Figure 2a and 2b) incorporated the sound-capturing capability of the first prototype, but added a folding petal feature to reduce the form factor. Sounds could still be collected from a long distance when the petals were extended and the toy could easily fit in a small package when the petals were collapsed. Its shortcoming, however, was cost of manufacturing and complexity of design. The folding petals were not easily constructed and required complexities in manufacturing that were error prone and expensive.

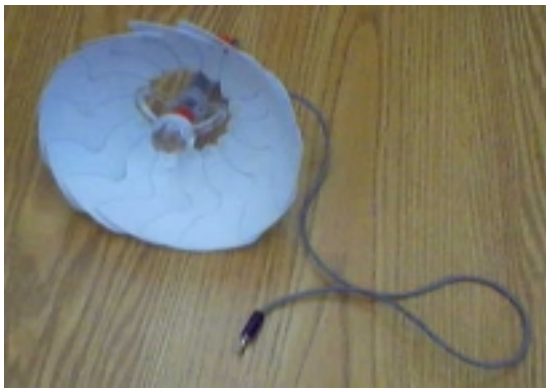

Figure 2b: Collapsible dish prototype–open

Thus far, each successive prototype was a refinement on the previous one: each one maintained the ability to capture sounds at a distance and at the same time improved on the form factor by virtue of it being more compact. In fact, the prototypes were victims of their own success when long-distance sound capture became a stumbling block due to its snooping and eavesdropping connotations. The final prototypes and eventual end product took on a sleek, high-tech look that emphasized the compact lines and minimized manufacturing complexities and cost.

The final engineering prototype (Figure 3) was constructed to validate the audio recording components in terms of form fit and functionality. This final prototype was fully functional as a recording device and used final production parts. As such, this prototype helped the toy designer in constructing a final form factor for the toy and helped the team determine the final cost of materials for the toy.

Figure 3: Engineering final prototype

Figure 4: Intel® Play™ Computer Sound Morpher

Intel Play is a registered trademark or trademark of Intel Corporation or its subsidiaries in the United States and other countries.

After the completion of the hardware prototyping, the final result was the CSM product that appears on store shelves today (Figure 4). It is the culmination of all the learning discovered during the prototyping and design stage. Ultimately, the expensive, long-distance sound capture device was traded for a low-cost, compact listening device that is now the Computer Sound Morpher.

The software exploration and prototyping was conducted in parallel with the hardware prototype development. The purpose behind the software prototypes was to explore, evaluate, and gauge the complexity of proposed software features. For the CSM, these features included the live voice changer, sound filters, synthetic voices, and audio streaming infrastructure. The CSM prototype integrated these features into a simple demonstration that was used for the following purposes:

- to allow software engineers to evaluate third-party software components
- to foster a dialog between engineering and design on the feasibility of features
- to help define sound filters by being used as a tool by designers
- to validate STL-developed components later in the project.

AUDIO CORNUCOPIA

The software feature set of this toy is packed with many fun activities centered on capturing, creating, and manipulating audio. These activities allow the child to explore the world of sound and include the following:

- **Sound download**–Detect and segment sound recordings from the hardware and store on hard drive.
- **Visual displays**–Present static waveform plots of captured audio for editing, and display live visual animations, as audio is played back.
- **Animated talking head**–Take any audio from the software library and play back the audio synchronized with an animated talking head.
- **Audio cut and paste**–Remove words or audio chunks from any recording for placement into a second recording.
- **Live voice changer**–Modify live audio input, i.e., the child's voice, from the toy hardware with sound filters and route the modified audio back out the speakers.
- **Sound filters**–Transform and manipulate any recorded or live audio stream. This included standard filters such as echo, reverb, and chorus as well as specialized filters such as 3D sound and noise reduction.
- **Synthetic voices**–Generate audio in one of several voices from text typed in by the user.
- **E-mail**–E-mail a sound recording by itself or connect it to an animated face.
- **Sound clips**–This is a built-in library of canned sound clips included on the software CD.

Behind each one of these features is some sort of audio technology to bring the feature to life. These technologies range from signal-processing algorithms and text-to-speech engines to waveform plotters and visual-effect generating libraries. At the core of all these audio technologies is a streaming infrastructure to tie everything together and route audio where it needs to go.

SOFTWARE ARCHITECTURE

The various audio technologies were bound together and presented as a seamless multimedia application. This section describes each of the pieces of this software puzzle and how all the pieces of the puzzle fit together. With all the pieces in place, it becomes clear how each piece met a specific need of the toy feature set (Figure 5).

The Computer Sound Morpher (CSM) software architecture followed the typical Smart Toy Lab (STL) product approach. The software was divided into three main categories: the application, the middleware, and the drivers. Fortunately, for the CSM, off-the-shelf audio drivers were used since the device connected to the PC via a standard audio input jack on the existing PC sound card. This left the middleware and application layers to be resolved by the STL.

The application layer, including the user interface and framework, was outsourced to a third-party multimedia application development house. This development house was experienced in developing rich multimedia applications for children, but lacked some of the specific technology needed for the CSM feature set. To fill in the missing pieces, other third-party packages were licensed including:

- text-to-speech engine
- audio streaming library
- audio sound effects library
- visual display library

- noise reduction library
- 3D sound algorithm

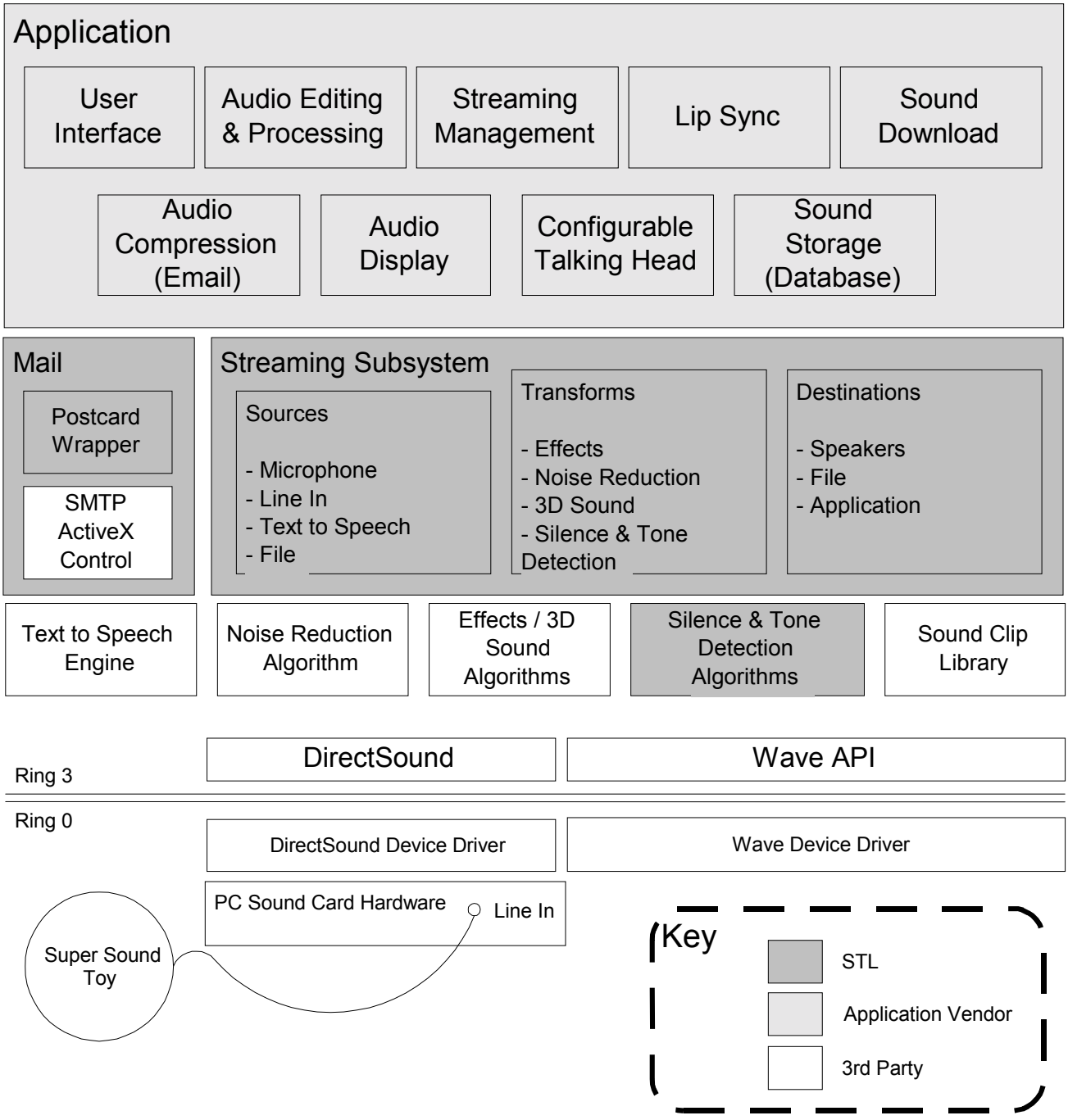

Figure 5: Computer Sound Morpher software architecture

Because of the variety and number of additional third-party pieces involved, a middleware component was developed at the Smart Toy Lab to unify all third-party pieces and present them as a single unit to the application developer. This "glue" provided a single point of access and simplified interface to the application vendor for all the technologies involved.

In addition to tying together third-party components, the Smart Toy Lab also provided some signal processing algorithms to implement involved features such as segmenting audio, recognizing hardware inserted tones, detecting audio feedback, and changing audio pitch.

Finally, there were other software pieces borrowed and modified from another software peer group within the Consumer Products Division. The Create & Share Camera team provided technologies to configure audio settings for playback and recording volumes as well as libraries to automatically connect to the Internet and send mail.

EYE CANDY

Putting a user interface on an audio application is an interesting task. This was accomplished by creating visual representations of sounds, which were both static and dynamic in nature. Whenever an individual sound was downloaded, edited, or created, a waveform plot was displayed (Figure 6). This waveform plot provided a visual representation for the audio that could be manipulated and examined.

Figure 6: Static waveform plot

In addition to the static waveform, another third-party solution was licensed to provide dynamic visualizations of the audio (Figure 7). As audio was played back from file, or routed by the live voice changer, or created by the text-to-speech engine, it was also displayed as an animated graphic. The third-party software included several different visualizations, all of which provided an active on-screen experience while audio was being manipulated.

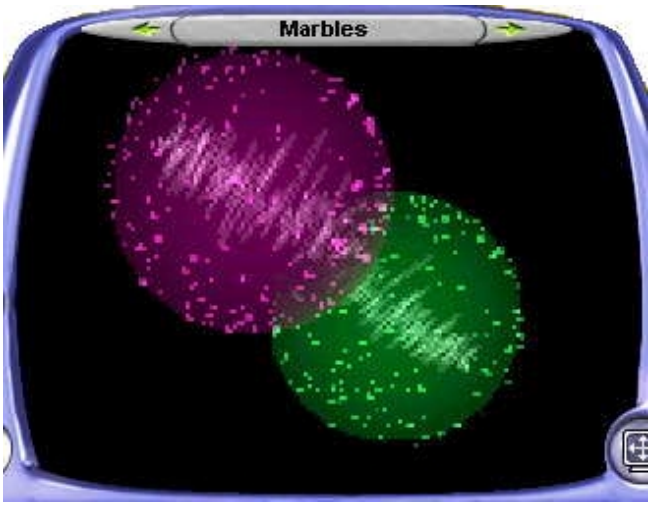

Figure 7: Visual display

CREATING ALIENS AND MONSTERS

With its main purpose as a sound-creating and sound-editing tool, the software naturally required several technologies to analyze and manipulate audio. Features such as downloading sounds from the toy, cutting up sounds into pieces, streaming live audio from the toy to the speakers, and applying special effects to sounds all required direct analysis and manipulation of the audio. Specifically, the following technologies were used in order to implement the feature set listed previously:

- **Frequency Shift Keying data modulation**–Used to download and delineate multiple captured sounds from the hardware.
- **Edge detection**–Used to identify and extract portions of audio from captured sounds.
- **Digital signal processing**–Used for special effects such as echo, reverb, chorus, and pitch as well as 3D sound and noise reduction.
- **Feedback detection**–Used to detect feedback during live voice loop back from the toy to the speakers.

Bring It Down

Downloading sounds from the hardware was constrained by the one-way analog connection from the toy to the PC via an existing sound card. There is no mechanism to "communicate" with the hardware from the PC due to this analog connection. Starting and stopping sound download, as well as segmenting individually captured sounds, required a unique communication mechanism. Existing and well-known modem technology was applied to overcome this limitation.

The hardware used Frequency Shift Keying data (FSK) modulation in order to encode information about the captured sounds into the analog data stream flowing from the toy to the PC. The encoded information describes the data on the toy in enough detail for the software to successfully start and stop the download of sounds to the PC as well as to check for aborted downloads by the toy. The information includes the number of sounds and the total length of all sounds on the hardware as well as sound delimiters to mark the beginnings and ends of sounds. This information was stored on the hardware and was readily accessible by the firmware. The user initiates the transfer of sounds from the toy to the PC by entering the sound download screen in the host software and then pressing the download button on the toy (Figure 8).

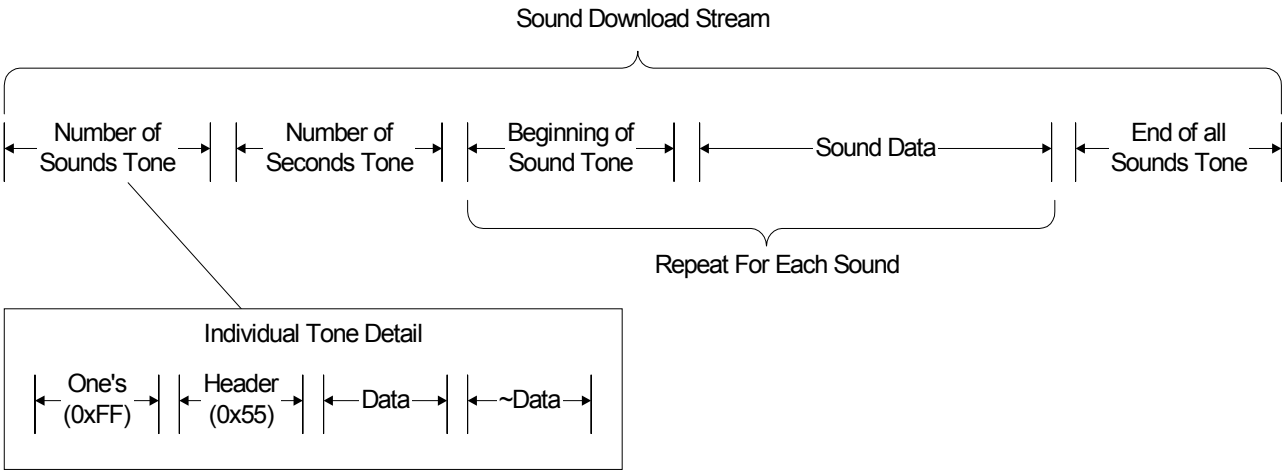

Figure 8: FSK download stream format

By entering the sound download screen, the user indicates to the software that a download will possibly follow. The software then opens an active input channel to the hardware via the PC sound card and monitors this channel for the FSK tones from the hardware. After the download button on the toy is pressed, the hardware encodes the number of sounds and the total length of all sounds in seconds as FSK tones in the analog output going to the computer.

The number of sounds and total length of all sounds in seconds designate the beginning of a sound download from the hardware. The software uses these two pieces of information to determine whether the following sound data constitutes a valid download. Following the initial information, each individual sound is preceded by a beginning-of-sound tone, and the last sound is followed by an end-of-all-sounds tone. Individual sound lengths are not encoded in the stream; however, the length of an individual sound is determined by the beginning-of-sounds and end-of-all-sounds tone markers.

The software detects and decodes each of the tones in order to maintain a download state machine. Each of the tones mentioned above must be present in order to complete a download. In addition to detecting tones, the software also overwrites the tones with silence before sending the audio to the speakers. Each individual sound is extracted from the live audio stream and written to a unique file.

The individual tones themselves each consist of a sequence of four characters or bytes of information. The length of the characters is determined by the baud rate of the modem, which is approximately 302 bits per second (bps). Each character consists of 10 bits (first bit = 1, 8 bits of data, last bit = 0) and, with the given baud rate, results in a character length of 33 milliseconds. To ensure that random audio data does not get interpreted as a valid tone, the tone format was defined as follows:

- Character 1 consists of a sequence of 10 bits all set to one.
- Character 2 consists of the tone header with data part equal to 0x55 (0x1010101010).
- Character 3 consists of the data for the tone.
- Character 4 consists of the opposite of the data in Character 3 (each bit is flipped).

Encoding each tone as four characters with a header, data, and opposite data values makes it extremely

unlikely to detect random tones inside the valid audio data.

Cut It Up

The "cut and paste" feature of the software required analysis of sounds for "boundaries" or changes in energy level. Once identified, these boundaries defined portions or chunks of audio that could then be extracted from one piece of audio and placed into another piece of audio. An edge-detection algorithm was developed at the Smart Toy Lab to implement this feature. Targeted mainly at detecting words within spoken sentences, this algorithm also served the purpose of breaking up generic sounds into segments (Figure 9).

Figure 9: Segmented audio display

The algorithm was designed to work in two stages. The first stage walks the entire buffer of audio samples and calculates energy levels to be used during the second stage. The second stage then uses the pre-calculated energy levels to identify and return sound "boundaries" to the application. A sound "boundary" is a drop in sound energy with specific constraints and can be controlled by the application through a minimum feature size parameter. This parameter allows the application to generate several "cut up" versions of the audio and present them to the user on demand. This proved to be very useful for breaking up speech into phrases, words, and even syllables.

Morph It

Audio manipulation was the name of the game for the CSM software. To bring this "morphing" to life, a wide array of special filters was provided for sound manipulation. These filters were used in several of the features, including the live voice changer, the synthetic voices, and the sound-editing area. To do this, the filters were integrated into an audio-streaming system that provided for application of filters to live audio from the sound card, to audio output from the text-to-speech engine, and to audio from a file.

The majority of the special filters was provided by a third-party software vendor and included standard audio algorithms such as echo, reverb, chorus, phaser, and flange. The algorithms were combined and configured in order to create unique effects, which were presented to the user as a list of voices. For example, a voice simulating a group of aliens consists of a combination of chorus and echo algorithms to generate multiple space-like voices. The third-party software provided the infrastructure for combining the algorithms, which proved to be a powerful and flexible framework for creating new effects on the fly.

To add to the fun, a special third-party algorithm was licensed to provide a 3D audio effect. This algorithm was packaged and presented to the user in the identical way the core algorithms listed above were presented; however, this algorithm was unique in its complexity. It exposed parameters to allow for positioning of the audio in space, which the application could manipulate over time to simulate the sound moving from left to right or from in front of the user to behind the user.

In addition to the algorithms provided by the third-party vendors, a couple more were added at the Smart Toy Lab to allow for more audio manipulation. These included pitch and volume algorithms, which allowed for the creation of chipmunk- and monster-like voices as well as for the adjustment of the volume level of individual recordings.

Finally, the ability to clean up audio recordings was provided via a noise reduction algorithm developed in the Intel® Architecture Lab. This noise reduction was again presented as another option in the list of effects or voices and allowed the user to remove unwanted hum or hiss from captured recordings. Because of the portable nature of the CSM hardware, just about any type of recording could be made in any type of recording environment. With this in mind, the noise-reduction algorithm gave users a tool to remove an unwanted audio from their captured recordings.

Feed It Back

The live voice changer feature of the software allowed children to speak into the CSM hardware and then hear their voices come back out the speakers in some modified form. This live loop back of audio presented a unique set of challenges including the following:

- how to minimize audible delay from the microphone to the speakers
- how to maintain an uninterrupted audio stream
- how to detect and handle audio feedback

The audible delay and uninterrupted audio stream issues were at constant odds with each other. In order to minimize delay, small audio buffers are required so that

the first chunk of audio can be collected from the sound card and immediately sent to the speakers. Unfortunately, small audio buffers are much more likely to result in interruptions in the audio streaming due to system activity. If the audio thread does not receive processing time for a period greater than the audio buffer size, then the audio stream is interrupted and a pop or click results. The utilization of Direct Sound and its primary buffer support allowed for low latency in the audio stream and minimized the delay. However, much tuning and compatibility testing was required in order to find a consistent sound setting across all audio hardware.

For anyone familiar with audio feedback, it is evident that placing a live microphone in front of a pair of speakers is not always a good idea. High-pitched squealing and wailing is often the result. For the live voice changer feature, this was a problem. A dual approach was taken in order to minimize this problem since it was out of the scope of the project to completely eliminate it with a technology such as echo cancellation.

First, the user was guided through an audio set-up wizard upon installation of the software in order to select the "ideal" sound card input and output volume levels for the live voice changer. Once configured, these settings were used by the application and significantly reduced the chances of producing audio feedback while using the live voice changer. Second, an algorithm was written at the Smart Toy Lab to detect when audio feedback occurs. Once the feedback was detected, the application was notified and could make the necessary adjustments to help the user fix the problem. The application turned the volume down to zero and then instructed the user to slowly turn up the volume to an acceptable level without producing feedback.

The feedback-detection algorithm was very simplistic in its approach but proved to be effective. It measured feedback as a high-energy period in the live audio stream. Energy was measured in decibels, and if a certain decibel threshold was maintained for a minimum time period, then this was determined to be a feedback situation. It is possible to generate false feedback detections in this case, but this was deemed more desirable than allowing feedback to occur unchecked.

Talking PC

Creating a recording from typed-in text was yet another fun feature of the software. The user was presented with a list of voices to choose from which included all kinds of wacky characters. The text-to-speech engine provided the guts behind this feature and was extended and combined with special effects to create wacky custom voices. The text-to-speech engine was licensed from a third party and was selected for its ability to configure and create new text-to-speech voices as well as for its capability to provide raw audio buffers to the client.

The creation of text-to-speech voices involved a few different technical approaches. First, new voices were created simply by adjusting the built-in parameters of the text-to-speech engine itself. These parameters were adjustable using in-line text command sequences, which were interjected before the text typed in by the user. The types of voices generated using this approach include child voices, female, male, and high- and low-pitched voices.

Second, the output from the text-to-speech engine was sent through the various special effects algorithms to produce an even wider range of voices. This is where the voices started to take on the wacky form where space creatures, cockroaches, and monsters came to life.

Finally, a couple of advanced features of the text-to-speech engine were utilized to create "singing" voices. These voices allowed the user to type in text and hear the words output as a melody. For example, imagine the phrase "Singing words is big fun," sung back to the tune of "Happy Birthday To You."

The approach taken to allow for a generic phrase utilized the phoneme and pitch features of the text-to-speech engine. The engine was configured to generate a stream of phonemes from the provided text. Phonemes are parts of speech and can be thought of as portions of a syllable. This stream of phonemes was then broken down into sonorants and non-sonorants. The sonorants are the audible portions of words such as vowel sounds. Each sonorant was assigned a pitch and duration in order to generate the singing sequence. The entire phoneme stream was reconstructed as a new text sequence with the pitch and duration commands embedded in the sequence. This encoded phoneme stream of text was then fed back into the text-to-speech engine, which generated the actual "singing" audio.

TYING IT ALL TOGETHER

With so many audio components involved, an efficient and simple interface was needed to present the various technologies to the application developer. A single component was developed at the Smart Toy Lab to house the various audio technologies and present a single interface to the application. This audio component also managed the routing of audio to and from the various pieces, which included the necessary

multi-threading to allow for prioritization of audio over the visual displays and user interface code of the application.

The audio library implements the streaming of audio from several sources through a set of transforms, and then to one or more destinations. A third-party audio streaming infrastructure was used as a base to provide low-level direct sound, wave, and audio mixing support. This helped to isolate any driver specific support in a single place. On top of this streaming infrastructure, the audio library encapsulated the various technologies as follows:

- **Audio sources**–These are file sources, text-to-speech sources, live microphone sources, and memory sources.
- **Audio transforms**–These are basic effects algorithms (echo, chorus, reverb, etc.), Smart Toy Lab algorithms (FSK tone detection, edge detection, volume, pitch, feedback detection), special algorithms (3D sound, noise reduction).
- **Audio destinations**–These are file destinations, speaker destinations (via audio mixer), application destinations.

Audio can be routed from any source, through any transform, and to any destination either individually or simultaneously. Each routing is referred to as an audio stream, and the only limitation on the number of audio streams is CPU load and memory. The following list contains the example usages of the sound library by the application for various features.

- **Sound download–**The audio is streamed from the microphone source, through the FSK tone-detection algorithm, and routed to the speakers for playback, to the application for live video display, and to individual file destinations for storing the downloaded files.
- **Sound editing–**The sounds captured by the user are loaded into memory for editing and previewing. The user can also request that they be run through the edge-detection algorithm or audio special effects transforms. Finally, they are routed to the speakers for playback, to the application for visual display, and, optionally, back to file if users request a save for their edits.
- **Live voice changer–**The audio is streamed from the microphone source and routed through special effects transforms and the feedback-detection algorithm and then to the speakers for playback and to the application for visual display. Users also have the option of sending the audio to file if they want a live voice recording.
- **Synthetic voice generation–**The audio is generated by the text-to-speech source and routed through special effects transforms and then to the speakers for playback and to the application for visual display. Again the audio is also routed back to file if users request a save for their edits.
- **User interface sounds–**The user interface sounds are first read from file and stored entirely in memory, as they are small and need to be played back immediately. The sounds are then mixed with any other active audio and then sent to the speakers.

Creating the sound library involved a lot of plumbing, and required constant integration into the application along with constant debug and refinement. Many delivery dates and checkpoints were set up at the beginning of the project to support application development while this audio library came to life.

EXTENSIBILITY

Much of the software was designed to allow for content updates after the product shipped, without modifications to the application itself. This allows for new application content to be delivered, such as add-on packs or updates, after the toy is out on the market. The areas that are extensible include the following:

- audio special effects
- text-to-speech voices
- audio visualizations
- noise-reduction configurations

Each of these areas can be configured by adding or modifying entries in the Windows* registry.

SUMMARY

The Computer Sound Morpher (CSM) software brought together a large set of audio technologies into a single fun-packed child's application. A lot of effort from several sources went into the implementation of the feature set, which resulted in a rich and diverse final product. This paper focused on the audio-processing portion of the CSM, which was a large chunk of the application. However, the complete application involved more than just processing audio. Development of the user interface with various screens and features

*Other brands and names may be claimed as the property of others.

was very involved. Each of the features described in this paper were bolted up and exposed to the user in one form or another. In the end, the final software product was a successful integration of sound technologies, user interface components, and colorful artwork and animations.

ACKNOWLEDGMENTS

Mark Leavy was instrumental in designing and implementing several of the signal-processing portions of the Computer Sound Morpher. Specifically, the approaches behind the sound-download, edge- detection, and feedback-detection features were outlined and brought to life by Mark.

AUTHOR'S BIOGRAPHY

Scott Boss is a software engineer who has been with Intel for eight years and has been working at the Smart Toy Lab for the past two years. Before the Toy Lab, Scott worked in the Intel Architecture Labs on various multimedia projects ranging from audio infrastructures to 3D graphics. He received his B.S. degree in Computer Science from Valparaiso University in 1991 and his M.S. degree in Computer Science from Purdue University in 1993. His e-mail is scott.d.boss@intel.com.