The Sound Synthesis System “O’kinshi”: Its Data Structure and Graphical User Interface

Naotoshi Osaka, Ken-Ichi Sakakibara, Takafumi Hikichi

NTT Communication Science laboratories
email: {Osaka, kis, hikichi}@brl.ntt.co.jp

Abstract

We are studying on Windows a sound synthesis system called “O’kinshi” that has functions of sound generation, modification and performance. This paper describes new sound-data-structure and GUI (Graphical User Interface) concepts for the system. Existing music generation software is based on either a sequencer which deals with common music notation or sound synthesis. In O’kinshi, the “sound object” covers universally all concepts from music to sound. That is, a sound object is represented in terms of other lower-level sound objects and operation objects. Moreover, the operation history is implemented in a single-sound object, which is the lowest-level object in the hierarchy. It can also be expressed as command script and is editable as well as programmable. This hierarchical sound object structure and duality of operation program execution allows flexible manipulation of sound and music at various temporal rates. Successful music creation and performance have been achieved using the new system.

1. Introduction

Several hardware/software systems are available for computer music creation. These can be widely classified into sequencers that synthesize music based on common music notation using MIDI, and sound synthesis systems that use signal processing. There are several commercial sequencers available, while MAX/MSP (Zicarelli 1998) and Kyma (Hebel and Scalaletti 1993) are well known real-time sound synthesis systems. The design concepts of the two system types seem to be completely different. Sequencer places great importance on a musician’s ability to manipulate notes and rhythm in the same manner as when they face a score. Not much attention is paid to timbre control or the GUI for signal processing since the sound source is mainly under MIDI control. On the other hand, in sound synthesis systems, such as MAX/MSP or Kyma, sophisticated GUI uses operational object boxes or a Unit Generator and patch code connection. Also, in a wave editor, operations are done mainly using pull down menus. These user interfaces are far from score manipulation.

In computer music creation, particularly in timbre-based music, composers have to have experience in many processes such as sound material synthesis, composition in a common music notation level, and sound mixing, and recording. An important point is that the composers do not perform these processes in some straightforward sequence, but goes back and forth. A score may be altered after listening to newly synthesized sound material, and composers might want to resynthesize the sound material after listening to the mixed music. Such feedback is intrinsic to the composition process, and composers must change their mind in different temporal rates very quickly; from millisecond-order signal processing, several hertz frequency vibrato rates, music tempo, to some minutes of recording time. Therefore, a consistent data structure and its GUI is desirable for music and sound data. O’kinshi uses a new GUI for sound synthesis that unifies music and sound. Figure 1 shows the data structure adopted in O’kinshi and that used in previous systems. In O’kinshi, sound objects are defined hierarchically using lower-level sound objects. As a result, the top level of music is written in a multi-track music passage description, and the bottom level of sound is represented in multi-channel sound waves.

We have been working on an off-line sound synthesis, modification, and performance system, “O’kinshi”, which runs on Windows (Osaka and

![Fig. 1 Comparison of music and sound representation](image)
Hikichi 1999). (Its name means a synthesis system for both sound and speech in Japanese.) The system has been modified so that the GUI satisfies the new requirement discussed above. This paper describes the GUI focusing on the sound object and operation command script.

2. System Configuration

Fig. 2 outlines the system configuration. The system is divided into two sections: sound synthesis and sound performance. The sound performance part has a panel containing buttons corresponding to sound files. The sound synthesis section is based on an offline sound modifier and includes the functions of a general wave editor, sinusoidal-model-based sound manipulation, such as sound morphing and sound hybridization, and other sound effects, including granular synthesis.

<table>
<thead>
<tr>
<th>Sound synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Special functions</td>
</tr>
<tr>
<td>Vibrateo control</td>
</tr>
<tr>
<td>Morphing:</td>
</tr>
<tr>
<td>physical model/signal model</td>
</tr>
<tr>
<td>Granular synthesis</td>
</tr>
<tr>
<td>• Sound editor functions</td>
</tr>
<tr>
<td>Sound wave generations,</td>
</tr>
<tr>
<td>Recording, playback</td>
</tr>
<tr>
<td>Sound wave modification</td>
</tr>
<tr>
<td>• Special editor functions</td>
</tr>
<tr>
<td>Various spectral display</td>
</tr>
<tr>
<td>Decomposition of sound into the collection of partials</td>
</tr>
<tr>
<td>Selection, deletion of arbitrary partials</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-delay sound playback with button press</td>
</tr>
<tr>
<td>Live performance use</td>
</tr>
<tr>
<td>MIDI control</td>
</tr>
</tbody>
</table>

Windows PC
Fig. 2 System configuration

2.1 Sound hybridization

Both sinusoidal-model-based and physical-model-based timbre morphing were implemented in the previous version. This version newly implements a couple of sound hybridization functions. Firstly, sinusoidal-model-based vibrato addition and subtraction were functions newly added. This enables the flute sound to have a soprano vibrato and a trumpet sound to have violin vibrato.

3. Overview of various objects used in the system

In satisfying the GUI requirements stated in section 1, object definitions are of great importance. These are summarized in Table 1. The common feature of the GUI for these objects is that at the top layer (layer 0), an object is simply displayed as an icon. By double clicking the icon, the system changes to layer 1: a panel appears and displays more information relevant to the specific object. In layer 1, the contents of the panel for all the objects are editable.

<table>
<thead>
<tr>
<th>Object type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound objects</td>
<td>Single-sound object</td>
</tr>
<tr>
<td></td>
<td>Multiple-sounds object</td>
</tr>
<tr>
<td>Operation objects</td>
<td>Filtering, pitch conversion, level conversion, etc.</td>
</tr>
<tr>
<td>Sound attribute objects</td>
<td>Pitch, spectral envelope data</td>
</tr>
<tr>
<td>Sound model objects (analysis/synthesis)</td>
<td>sinusoidal model packet (chunks)</td>
</tr>
<tr>
<td></td>
<td>granular synthesis representation</td>
</tr>
<tr>
<td>Command script object</td>
<td>Sequential description of operation in text (see Fig.3)</td>
</tr>
</tbody>
</table>

3.1 Sound objects

We defined two types of sound objects: a single-sound-object and a multiple-sounds object. Figures 3 and 4 depict the panels of layer 1 for them. For both, sound playback is done with a right single click at layer 0, where only an icon is shown. This is called pilot play and similar to previewing a photo, which shows the outline of the objects. The panels of layer 1 are reached by double clicking an icon. Layer 0 again appears when the close button is pressed. The sound object is the most important concept of the system, and is described in the next section.

3.2 Operation objects

Operation objects define the generation or modification function of the sound objects. They are so called Unit Generators. At the top layer, simply a button with an icon is displayed. Execution is done either by right clicking an the operation icon, or dragging the icon to the receptive area of the operation history on the single-sound object. An example of operation history is seen in the upper right of Fig.3. By double clicking the icon, we can reach the second layer and set the operation parameters in detail. The object includes various filters, modulations and wave generations.

3.3 Sound attributes objects

Pitch, envelope, and vibrato objects are in this category at present. These objects lack an ability to resynthesize the original sounds by themselves, because they represent only one aspect of a sound. These objects are used for general analysis and to input particular effects. To add vibrato, for instance, the vibrato object icon is dragged into the vibrato control panel.
3.4 Sound model parameter objects

These objects have the ability to resynthesize the original sound. But the resynthesized sound is not always the same as the original in terms of sound quality. Therefore, the sound object expressed by a certain model is distinguished from the original sound object. At present, a sinusoidal model packet (chunks) and a granular synthesis representation are the model-expressed objects.

3.5 Command script object

The procedure history is automatically described as an ascii command sequence. The procedure history display using icons is simply the visual representation of this command sequence. This object is a text panel in layer 1, and editable.

4. System features

This section explains the two system features on the GUI: 1) the ability to execute operation program both by GUI and command script (text) and 2) the hierarchical structure of the sound objects.

4.1 Single-sound object

A single-sound object represents sound primitives and is defined in terms of wave data and the past history of the process. This modification is rather an inner process and pipeline processing, such as filtering and level change, and no other sound object is involved. In layer 1 of a single-sound object (Fig. 3), visual monitoring of both the wave and spectrum and acoustic monitoring are possible. In this layer, sound modification is possible. The operation history is shown by operation icons. This provides a genealogy of the sound and programming notations. Editing the sequence of icons allows musicians to redo the procedure and stop at any point as long as the stoppers are appropriately positioned.

There are several differences between MAX patch and this history description. Firstly, the former is a procedure definition, while the latter is a (sound) data definition that has a finite duration, and uses sound icons and operation icons. Secondly, this system restricts the procedure to a sequence. There are no branches or loops, which are the controls necessary for programming language. Thirdly patch cord connections are completely avoided. While these differences make our GUI a little inconvenient from the programming point of view, they do make the GUI simpler.

4.2 Programming using Command script objects

The ability to edit the procedure history using a command script object compensates for the programmability disadvantage of the GUI.

We extended the system so that this script is written in the object-oriented language Ruby (Thomas and Hunt 2000) and can be redone under Ruby's control. Fig. 5 shows the command scripts which is automatically acquired after the operations depicted in Fig 3. Users can add some other control functions in Ruby, such a branch, jump, or loop, to the command script object, so that sound synthesis can be done under Ruby's control. Here, procedures are expanded to serial order and icons are repeatedly pasted to the procedure history part of layer 1 of the single sound-object over and over again, if a loop is executed in the command script.

```ruby
# Initialize the environment
require 'otkinshi'
okn = Otkinshi.new()
okn.Initfield()

# Operation history (sequence)
okn.OpenWave('C:\sop.wav')
okn.Level(6)
oknFadeIn(100)
oknFadeOut(100)
oknCutFFr.
oknBandPass(1000,2000)
okn.Level(6)
okn.Level(6)
okn.PitConv(5,4)
```

Fig.5 Command script of operation history of a single-sound object
4.3 Multiple-sounds object

A multiple-sounds object represents a higher and wider level manipulation of sound primitives. In contrast to a single-sound object, it defines the inter-sounds operation, such as multi-sound mixing, windowing, or the convolution. One multiple-sounds object can be recursively or hierarchically used in another multiple-sounds object. The most easily understandable usage of a multiple-sounds object is in multi-track mixings, an example of which is shown in Fig. 6.

At the upper left corner there is a top level icon that represents the whole music piece. Users can listen to the music by a right single click at this layer. Double clicking gives layer 1. This panel refers to multiple passages or sounds. Each of them points to lower level sound objects, and this relation is hierarchical. The terminators of the hierarchy are single-sound objects. There are two ways of drawing the sound in the layer 1 panel. One is file name printing as shown in Fig. 4. The other is common music notation. Users can choose either mode using a toggle switch. The horizontal direction represents time and the vertical direction represents track or channel. This fits well with the common music notation.

After evaluation, sound is mixed. Both monaural and stereo sounds are incorporated for input and output.

4.4 Sound object and the representation of operation

In a multiple-sounds object, not only the mixing operation, but other operations such as convolution and incorporation of operation object are possible. The graphical display (layer 1) corresponds to only a resultant sound placement, and does not depict all the procedures. This is different from the case of a single sound object. However, musicians can again make use of command script and define many operations. This implies the great possibility of timbre operation with a script base. Since we can define a single sound object as one symbol, we can compute new timbre in terms of a defined sound (or timbre) symbol and an operation object. An example of script

is given in Fig. 7. No operation is defined in the script. However, mixing operation is set by default.

5. Performance

The newer version of the system was provided to the students of Kunitachi College of Music and used for composition. A couple of professional composers have also given us positive feedback. However, a formal evaluation has not been done yet. In March 2001, a computer music concert was held, and two pieces using the system were premiered successfully.

6. Conclusion

We introduced a new version of the sound synthesis system “Okinshi”, in which new GUI concepts are proposed: 1) a hierarchical structure of sound objects, and 2) duality of operation program execution by both GUI and command script. The former allows a sound object penetrates from music to a tiny sound primitive in GUI.

Future work includes extending the utility operations and adding more modification functions for sound hybridization, pitch conversion, or sound stretching/compression. We are ready to distribute the system and expect tens of musicians to create artistic pieces using it.

Acknowledgments

The authors would like to express their gratitude to Dr. Hiroshi Murase for his enthusiastic support.

References


