Chapter 5

Shapes

Most Important Things to Learn

Shapes are the primary data object in HMSL. They can contain notes, parameters for equations, samples or whatever you like. Shapes are multi-dimensional. A typical melodic shape might have 3 dimensions: time, pitch and loudness. Shapes have special methods for editing and manipulating their contents. Some useful manipulation methods are **RANDOMIZE:** , **REVERSE:** , **SCALE:** , **SCRAMBLE:** , **TRANSPOSE:** and **INVERT:**. Shapes can calculate statistics, minima, maxima and mean, for each dimension. Shapes must have memory allocated by **NEW:** before using them. Data is typically added to shapes using the **ADD:** method. You can access Shape data by the methods **GET:** , **PUT:** , **ED.AT:** and **ED.TO:** from the **OB.ELMNTS** class. Users often extend this basic class to implement their favorite transforms and data manipulation techniques.

Introduction to Shapes

Shapes are the primary data structure of HMSL. Think of shapes as an ordered list of points in some N-dimensional space. The points are called *elements*. If the dimensions were latitude and longitude then they could be points on a map. A 3 dimensional "space" consisting of time, pitch, and loudness might represent a melody. Shapes can have as many dimensions and as many elements as needed.

Shapes are "abstract" data objects. They do not know what their data means. The same shape could mean different things to different parts of the system at the same time. Shapes are used by HMSL to contain melodies, tuning tables, audio samples, coefficients for a compositional algorithm, etc. HMSL has a graphical Shape Editor that can be used to edit any shape regardless of its "meaning." Thus part of an Amiga sample can be reversed using the same button that does a retrograde inversion of a melody. More on this editor later.

Shapes can also analyse their contents. The MIN, MAX, and average value of any dimension can be queried. Optional high and low limits for each dimension can also be set. For example, one might want to limit a MIDI velocity dimension to 0-127 when editing a shape with the Shape Editor.

Tutorial 1: Using Shapes

This tutorial will illustrate how to use the basic data object in HMSL, the *shape*. Rather than describe what a shape is in mathematical terms, let's start playing with one and experience it more directly. First we need to create, or *instantiate* a shape to play with. Please do not enter this tutorial in a file. Enter it directly at the keyboard:

```
OB.SHAPE   SH-1
```

Now we create some room in the shape for data. We'll make room for up to 32 rows, or "elements," with 3 columns, or "dimensions" in each. (For those of you who are up on ODE, a shape is a subclass of OB.ELMNTS). Enter:

```
32   3  NEW: SH-1
```

Now that there's room, let's put in some data. Enter:

```
10  17  100   ADD: SH-1
```

```
10   9   80   ADD: SH-1
PRINT: SH-1
```

The first thing that PRINT: does for shapes is print out some statistics about the data, namely the minimum, maximum, and mean for each dimension. It also prints out the data. You should be able to see your data in rows just as you entered it. Notice that the dimensions are numbered starting with 0.

This data can be used in a number of ways. One common way is to treat the second dimension, dimension 1, as note values and the third dimension as velocities. This allows us to play the data as a melody using MIDI. The one missing ingredient here is timing. The first dimension is, therefore, commonly used to specify the time between elements and thus determines the note durations (to be a bit more specific, the duration is considered to be the difference in *start-times* between two elements of a shape). To hear this shape as a melody, enter:

```
CLEAR: SHAPE-HOLDER     ( we will explain this later )
SH-1  HMSL.EDIT.PLAY    ( play a shape while editing )
```

A window will pop up and the two notes, 17 and 9 will start repeating. (The name SH-1 should be printed directly below the drawing. If not, click on the Shape Selector UP-ARROW until you see SH-1 then click on the box with the name SH-1.) Using the mouse, click on the UP-ARROW in the grid with the word DIM above it. This switches you between dimensions. Click on this until you are on dimension 1, the note (pitch) dimension. Now go up and click on "Insert" in the "Set Mode" grid. Now click in the big graphic window to insert notes. You should hear the melody get longer. Now click on "Replace" and then click in the graphic window to replace the existing values with new ones. Change dimensions as before and try editing the durations in dimension 0 and the velocities in dimension 2. (For more information on how to use the Shape Editor refer to the section later in this chapter.)

You may be wondering why we cleared the shape-holder. The shape-holder contains a list of all the shapes available to the Shape Editor. We cleared it so that there wouldn't be other shapes available which might be confusing.

Now select "Quit" from the pull down HMSL menu to get back to Forth. Print your shape using the PRINT: command as before, and notice how the data has been changed. We can also change the data using program commands. Enter:

```
3 SET.MANY: SH-1
PRINT: SH-1
```

This sets the number of elements in the shape to 3. There is still room for 30 elements but there are now only 3 that are defined. This can be a quick way to get rid of extra data or to make it appear as if data has been added. Now enter:

```
5  14  95    1  PUT: SH-1
PRINT: SH-1
```

The old values in element 1 have been replaced with the new values. Since you may be getting tired of typing "PRINT: SH-1" , let's make a convenient word to use instead.

```
: PS  ( -- )  PRINT: SH-1 ;
PS
```

Now try these other commands:

```
20   7   60   2  INSERT: SH-1     ( sticks in another element )
PS

1   REMOVE: SH-1    ( takes one out )
PS
```

These commands have all acted on entire elements. Let's look at ways to change values in dimensions. Enter:

```
10  0  FILL.DIM: SH-1    ( sets entire dimension to 10 )
PS
```

A good way to think about data in a shape is that the element is the "row," and the dimension is the "column."

Shapes provide a number of methods useful for manipulating data in a given dimension. When using the methods, it is often necessary to know how many elements are in a shape. Enter:

```
MANY: SH-1   .
```

Keep in mind that although there are 10 elements in the shape, they are numbered 0-9. You can't "fetch" element number 10!

Let's try a method that reverses the order of the values in a dimension. The stack diagrams for REVERSE: is:

`REVERSE:    ( start end dimension -- , reverse order of values )`

Enter the following:

```
PS
0  MANY: SH-1  1- .S  ( 1- needed, 5 elmnts. would be numbered 0
to 4)
1  REVERSE: SH-1      ( reverse order in dimension 1 )
PS
```

Another useful method is TRANSPOSE: which adds a constant to the values in a dimension. The stack diagram for TRANSPOSE: is:

`TRANSPOSE:  ( offset start end dimension -- , add offset to values )`

Enter:

```
10   0   MANY: SH-1  1- ( the "end" value is numbered "many-1" )
1  TRANSPOSE: SH-1
PS
```

If you want to access individual values in the shape, use ED.TO: and ED.AT: . The "ED" stands for Element and Dimension. Enter:

```
77  1  2  ED.TO: SH-1 ( change value in element 1, dimension 2 )
PS
```

To play the result of your changes, enter again:

```
SH-1    HMSL.EDIT.PLAY
```

Then exit using "Quit" from the pull down HMSL menu.

Tutorial 2: Building Shapes

The following is an example of building a simple shape. It uses the random number generator, CHOOSE, to generate "reasonable" values for the shape. Here is the stack diagram for CHOOSE in case you haven't seen it before.

CHOOSE (N -- RANDOM , number between 0 and N-1)

You may want to type this definition into a file so you can experiment with it. Note that SHAPE-1 is a stock morph that is predefined for HMSL. It is often used in these examples.

```
ANEW TASK-SHTUT
: BUILD1  ( -- , build SHAPE-1 )
   5 3 NEW: SHAPE-1    ( Use a stock shape. )
   5 0 DO
      2 CHOOSE 1+ 8 * ( random #: either 8 or 16 )
      24 CHOOSE ( random number between 0-23 )
      64 ( stack:  8|16  0-23  64 -- )
      ADD: SHAPE-1 ( put three values into SHAPE-1 )
   LOOP
;
```

Now compile the file and enter directly:

```
BUILD1
PRINT: SHAPE-1
```

Column 0 gets values of either 8 or 16. Column 1 gets values between 0 and 23. Column 2 always gets the value of 64.

The word above works fine but it only works with one shape, SHAPE-1. It would be nice to have a word that would "build" any shape. To do that we will need *late binding*. From the ODE chapter we learned that late binding gives us a way to send messages to whatever object is on top of the stack. Here's an alternative to the preceding word using late-binding and *local variables*. Note the stack diagram uses curly brackets which signifies the use of local variables. All you need to know here is that the local variable "shape" has a number passed to it on the stack at entry to the word, and that it returns its value simply by its "name" being called. (See the section in the JForth manual or the Macintosh Supplement for a complete explanation of Locals.) Enter in a file:

```
: BUILD.SHAPE { shape --, shape must be 3 wide... }
   10 3 SHAPE  NEW: []
   10  0
   DO 2 CHOOSE 1+ 8 *  ( dim 0 = 8 or 16 )
      24 CHOOSE   ( dim 1 = 0-23 )
      64    ( dim 2 = 64 )
      SHAPE  ADD: []
   LOOP
;
```

Now compile the file and enter:

```
SHAPE-1  BUILD.SHAPE
PRINT: SHAPE-1
SHAPE-2  BUILD.SHAPE  ( BUILD.SHAPE can be used with many
shapes )
PRINT: SHAPE-2
```

Shape — subclass of OB.MORPH, OB.ELMNTS

Theoretically, shapes are ordered sets of n-tuples of any length. They can also be thought of as sets of points in some N-dimensional space. In addition, they have certain statistical information for each dimension, including a name, minimum, maximum, and mean.

HMSL users have experimented and composed using other, more high-level meanings for shapes. For example, shapes may contain indices of degrees of complexity of some process, or lists of metric values relating the similarity between other morphs. Values in shapes can describe the "trajectory" of distances from a given source morph to another, or what David Rosenboom has called "concept

spaces." For more information on these and other ideas, see the references at the back of this manual under Polansky and Rosenboom. Shapes have also been used as waveforms, and as lists of parameters for real-time digital signal processors.

Shapes can be stored to disk via the DUMP.SOURCE: method. This will create the source code, in a file, necessary to generate the shape in its current form.

Shape Methods

Method	Stack diagram
CALC.DIM.STATS:	(dim -- , calculate dimension statistics)
CALC.STATS:	(-- , calculate all stats)
CLIP.ED.TO:	(value elmnt dim -- , clip to limits then ED.TO:)
CLONE:	(target-shape -- , copy contents to target)
DIFFERENTIATE:	(sum dimension -- , differentiate values)
DUMP.SOURCE:	(-- , save shape to disk file)
EXTEND:	(n -- , allocates n new items)
FREE:	(-- , frees shape and its dimension names)
GET.DIM.LIMITS:	(dimension -- low high , get editor limits)
GET.DIM.MAX:	(dim -- max , maximum value of dimension)
GET.DIM.MEAN:	(dim -- mean , mean value of dimension)
GET.DIM.MIN:	(dim -- min , minimum value of dimension)
GET.DIM.NAME:	(dimension -- $name | 0, get name of dimension)
INIT:	(-- , initialize shape)
INTEGRATE:	(dimension -- sum , integrate values in dimension)
INVERT:	(value start end dim -- , reflect values in shape)
NEW:	(max_index number_of_dimensions -- , declare space)
PREFAB:	(-- , fabricate reasonable contents)
PRINT.STATS:	(-- , print statistics)
PUT.DIM.LIMITS:	(low high dimension -- , set limits for editor)
PUT.DIM.NAME:	($name dim -- , place pointer in name)
RANDOMIZE:	(minimum maximum start end dimension --)
REVERSE:	(start end dimension -- , reverse order of values)
SCALE:	(numerator denom. start end dim. -- , scale values)
SCRAMBLE:	(start end dimension -- , randomize order of data)
SEARCH.BACK:	(value dimension -- index, of next highest value)
SWAP:	(element1 element2 dimension -- , swaps 2 values)
TRANSPOSE:	(value start end dim -- , augment dimension data)

Table 6-4. Shape Methods

CALC.DIM.STATS: (dim -- , calculate dimension stats)

Calculates the statistics (minimum, maximum, and mean) for the dimension specified. Do this before using GET.DIM.MIN: , GET.DIM.MAX: or GET.DIM.MEAN: . This is used by PRINT: , and by the Shape Editor. It is valuable for use inside productions, etc.

Related Methods: CALC.STATS: , GET.DIM.MIN: , GET.DIM.MAX: , GET.DIM.SUM:

CALC.STATS: (-- , calculate all stats)

Does CALC.DIM.STATS: on all the dimensions.

Related Methods: CALC.DIM.STATS: , PRINT.STATS:

CLIP.ED.TO: (value element dimension -- , clip then set value)

Clip the value on the stack to the current dimension limits set by PUT.DIM.LIMITS: then use ED.TO: to put the value in the given dimension of an element. Used by Shape Editor.

CLONE: (target-shape -- , copy contents to target)

NEW: target shape if required, then copy data, limits, and name from one shape to another. For example:

```
PREFAB: SHAPE-1 ( fake some data )
PRINT: SHAPE-1
SHAPE-2 CLONE: SHAPE-1 ( copy to shape-2 )
PRINT: SHAPE-2
```

DIFFERENTIATE: (sum dimension -- , differentiate values)

Convert each value to the difference between itself and the next value. The last value is set to the SUM minus the last original value. For an initial set of values:

```
0 , 10 , 30 , 40
```

Calling DIFFERENTIATE: with a sum of 45 would yield:

```
10 , 20 , 10 , 5
```

This is useful for converting "absolute" times recorded from the MIDI parser to durations for more typical use. See INTEGRATE: and the utilities SH.COMPRESS.NOTES

As an example, suppose you are differentiating absolute times in dimension zero of a shape. Let's also assume that dimension 3 contains on times which is the length of time the note is ON. To calculate the sum to pass to differentiate we should use the following equation:

SUM = LAST_START_TIME + LAST_ONTIME - FIRST_START_TIME

Thus the SUM is the time span between the start of the first note and the end of the last note. Here is a Forth example that assumes notes have been recorded in SH1:

```
LAST: SH1 1- ( -- lasttime pitch vel ontime )
NIP NIP ( -- lasttime ontime )
+ ( -- lasttime+ontime )
0 0 ED.AT: SH1 + ( -- sum , add absolute time of first note )
0 DIFFERENTIATE: SH1 ( dimension 0, using sum )
```

DUMP.SOURCE: (-- , print source code that fills shape)

When used with the Forth LOGTO and LOGEND words, DUMP.SOURCE: will create a file that can be used later to restore the shapes values.

An example of its use follows:

```
: SAVE.MY-SHAPE ( <filename> -- )
   LOGTO   DUMP.SOURCE: MY-SHAPE   LOGEND
;
```

Use of "< >" enclosing a word, means that the word in question is not passed on the stack, but is a string that will follow the word being defined (see the example below). In this case, LOGTO "eats" that filename. By putting LOGTO DUMP.SOURCE: and LOGEND together into one word (instead of the more conventional LOGTO FILENAME in interpretive mode), you can avoid getting "OK" messages in your file. These next examples are intended to be used OUTSIDE of colon definitions.

For the Amiga:

```
SAVE.MY-SHAPE RAM:MY_FILE
COPY RAM:MY_FILE HW:MY_FILE ( Writing first to RAM: saves
time)
```

For the Macintosh:

```
SAVE.MY-SHAPE MY_FILE
```

This will put the necessary source code for filling MY-SHAPE, with its current data, in the file MY_FILE . Dimensions names will not be saved in the file. Users can then edit this text file as they wish. The shape itself needs to be instantiated outside this file.

A handy technique is to have one source code file that defines everything about a piece except the shape data. This file can INCLUDE another file created by DUMP.SOURCE: that fills the shapes. The shape data can be edited using a text editor or the Shape Editor and then saved without affecting the other aspects of the piece. One could even have one main source file and several alternate shape data files for different versions of one piece.

GET.DIM.LIMITS: (dimension -- low high , get limits for editor)

Get values that editor will clip to.

EXTEND: (n -- , allocates n new items)

Enlarge shape. Leave data intact.

FREE: (-- , frees shape and dimension names)

Free the shape (deallocates memory, as in all morphs). Also deallocate storage for the dimension names.

GET.DIM.MAX: (dim -- max , maximum value of dimension)

CALC.DIM.STATS: must be called first.

GET.DIM.MEAN: (dim -- mean , mean value of dimension)

CALC.DIM.STATS: must be called first.

GET.DIM.MIN: (dim -- min , minimum value of dimension)

CALC.DIM.STATS: must be called first.

GET.DIM.NAME: (dimension -- $name , get name of dimension)

Get value that was stored with that dimension. Return the address of the name. You can then use $. to print the name.

Related Method: PUT.DIM.NAME:

GET.DIM.SUM: (dim -- sum , sum of values of dimension)

CALC.DIM.STATS: must be called first.

INIT: (--)

Initialize the shape as it would any morph. Also set the dimension names to a null value. It does not add the shape to the SHAPE-HOLDER (you must do this manually, for example, MY-SHAPE ADD: SHAPE-HOLDER). INIT: is done automatically when shape is instantiated.

INTEGRATE: (dimension -- sum , integrate values in dimension)

The values in the dimension will be converted to the sum of the previous values. For example, if the dimension contained:

```
10 , 10 , 10 , 10
```

then integration would result in:

```
0 , 10 , 20 , 30
```

with a resulting sum of 40. This is useful when you want to convert durations, or "delta times," in a shape to absolute times. See DIFFERENTIATE:

INVERT: (value start end dimension -- , reflect values in shape)

Does an inversion about "value," like reflection about an axis. This is used in the Shape Editor.

NEW: (max_num_elements num_dimensions -- , declare space)

Does NEW: as for a morph. Also allocates space for the new instance variables. Clears itself.

Example:

```
8 5 NEW: SHAPE-1
```

Now your PRINT: command will give a table which is 5 wide (from 0 to 4).

This must be done before values can be added to a shape. See ADD:

PREFAB: (-- , fabricate reasonable contents)

This is useful for quick tests. It will generate "quarter and half" notes based on a random walk, assuming the shape represents a melody. Try:

```
PREFAB: SHAPE-1
SHEP
```

PRINT.STATS: (-- , print statistics)

Does CALC.STATS: , then prints results. Prints minimums, maximums and means for all dimensions.

PUT.DIM.LIMITS: (low high dimension -- , set limits for editor)

Set minimum and maximum limits that the editor and shape transformation methods (like RANDOMIZE: and SCALE:) can generate. The default limits are 0 and the highest possible integer. An example of its use would be to set the MIDI velocity dimension limits in a shape to 0 and 127. Then you can edit without worrying about generating illegal velocities! For example, if you are using dimension 2 of SHAPE-1 for velocity, then enter:

0 127 2 PUT.DIM.LIMITS: SHAPE-1

PUT.DIM.NAME: ($name dim -- , place pointer in name)

User routine for setting the name of a dimension for a shape. These names will appear in the Shape Editor when you are editing that dimension. Any text string will work. Note however, that this should be used INSIDE a colon definition.

Example:

```
: NAME.SHAPE-1 ( create a routine which names dimensions )
   " Duration " 0 PUT.DIM.NAME: SHAPE-1
   " Pitch " 1 PUT.DIM.NAME: SHAPE-1
   " Timbre " 2 PUT.DIM.NAME: SHAPE-1
;
NAME.SHAPE-1 ( execute that routine )
SHAPE-1  HMSL.EDIT.PLAY  ( notice names under graphics
display)
```

The reason that this procedure is necessary is that the shape only stores the address of the string. Strings created outside a colon definition reside in a temporary area called the PAD that can be overwritten. Thus the shape would only have the address of where the string used to be! Note also that this can only be done after using NEW: for the shape. Otherwise there are no dimensions to name.

Note that these names have nothing to do with the shape's eventual interpretation. That is, one could call a dimension LOUDNESS, but have one player interpret it as pitch, and another as loudness.

Related Method: GET.DIM.NAME:

RANDOMIZE: (minimum maximum start end dimension --)

Uses CHOOSE to set the values within a given shape, in a specified dimension, and within a given range, to random values. This is used by the Shape Editor.

Minimum and maximum are the parameters for the random number generator. Start and end are the limits of which elements will be affected, and dimension determines which column will be affected.

Example:

```
5 10 2 4 1 RANDOMIZE: SHAPE-1
```

will randomize rows 2, 3, and 4 in column 1 to values 5 <= x <= 9.

REVERSE: (start end dimension -- , reverse order of values)

Retrogrades the values from start to end in the specified dimension, within the specified range (that is, ordinal range of values). *If the dimension parameter is -1, all dimensions will be reversed.*

Example: if column 1 in SHAPE-1 contains: 8, 16, 5, 2, 9 and you say:

```
1 4 1 REVERSE: SHAPE-1,
```

then column 1 of SHAPE-1 will contain: 8, 9, 2, 5, 16.

SCALE: (numerator denominator start end dimension -- , scale values)

Multiply the shape values in the specified dimension by the numerator and then divide them by the denominator. As an example, to scale the durations in the last half of a 20 note shape by 2/3, you might enter:

```
2   3  10  19  0  SCALE:  SHAPE-1
```

SCRAMBLE: (start end dimension -- , randomize order of data)

Reorder the values in a shape. This is similar to shuffling a deck of cards. The value of the cards is not changed, just their sequence. Again (like in REVERSE:), a negative dimension parameter will cause all dimensions to be scrambled.

SEARCH.BACK: (value dim -- index , of next highest value)

This will search starting from the end of a shape. It returns the index of the element with the next higher value in that dimension. This is very useful for insertion sorts. For most applications we will be adding very near the end so this is reasonably efficient. Here is an example of insertion sorting notes.

```
: INSERT.SORT  ( time note vel -- )
   2 PICK  ( get time )
   0 SEARCH.BACK:  SHAPE-1
   INSERT: SHAPE-1
;
32 3 NEW: SHAPE-1
30  7  80  INSERT.SORT
10  5  80  INSERT.SORT
80  9  80  INSERT.SORT
45  2  80  INSERT.SORT
PRINT: SHAPE-1
100 0 DIFFERENTIATE: SHAPE-1
PRINT: SHAPE-1
```

SWAP: (element1 element2 dimension -- , swap values in shape)

Swaps the values in a shape. As in REVERSE: and SCALE: , a negative dimension parameter will cause entire elements to be swapped.

TRANSPOSE: (value start end dimension -- , augment dimension data)

Adds value to the numbers in the columns from start to end in the specified dimension. This is a simple transpose, used by the Shape Editor.

Example: If column 2 of SHAPE-1 contains: 7, 23, 10, 20, 16, and the user enters:

```
10 1 3 2 TRANSPOSE: SHAPE-1
```

then column 2 of SHAPE-1 contains: 7, 33, 20, 30, 16.

Note that TRANSPOSE: may operate on any dimension in a shape.

Example: if one dimension is being interpreted via a given player/instrument as duration, TRANSPOSE: will produce a ritard or accelerando.

Example: if that dimension is being interpreted as loudness, TRANSPOSE: will produce crescendi/decrescendi.

The user can experiment with many more non-traditional interpretations of these transformations on shape data: for example, retrograding or transposing MIDI programs.

Different ways of representing data in shapes

Although Shapes are abstract data objects, they are often used to represent melodies. There are things we do with these melodic shapes, like recording, saving the data in MIDIFiles, analyzing them, etc., that require different ways of representing the notes. The way in which data is stored can have a profound impact on the way a program operates. Since this is a subject of much debate among programmers, we'll describe how this is done in HMSL. There are at least 5 different commonly used types of shape data formats, each with their own advantages and disadvantages. Note that this is just a conceptual guideline, there is in fact only object called shape. The different types that we might categorize are:

Type 0: Raw. Abstract data. "Pure" Shapes.

These shapes may contain algorithmic parameters, MIDI control information, or other complex data. Their actual format varies greatly.

Type 1: Compact Notes. Monophonic.

These shapes have one 3-dimensional element per note. The 3 dimensions are time, note index, and velocity. The only time specified is the start time of the note. There is no indication of how long the note will sound and, therefore, no easy way to arbitrarily overlap notes. This type of shape is best suited for monphonic melodies.

In the following example of a type 1 shape, the time values are delta times until the inception of the next element. This example might represent a quarter note followed by a half note.

```
Time Note Velocity
 12    5   80
  6    7   80
```

Type 2: Expanded Notes. Separate ON and OFF elements.

Each note is represented by two 3-dimensional elements, an ON element and a separate OFF element. OFF elements are indicated by a zero velocity. It is very easy to represent polyphony with this type of shape. It is also very easy to convert to a MIDI File since MIDI Files also have separate ON and OFF events. When recording MIDI input, the data is first stored in this raw form. A disadvantage of this

type is that it is harder to edit and analyze because the ON and OFF elements have to be matched. Notes can be played together for chords, etc., by using a zero time difference between their ON events

```
Time Note Velocity
 10    5   80  ( turn note 5 on )
  2    5    0  ( then off )
  0    7   90  ( play notes 7 & 11 together )
  8   11   85
  0    7    0  ( turn them off )
  4   11    0
```

Type 3: Compressed Notes. Notes with ON dimension.

This type of shape stores notes as one 4-dimensional element. The dimensions are time, note index, velocity, and ON-TIME. The ON-TIME is the time between the MIDI.NOTEON and the MIDI.NOTEOFF, ie. how long the note is "on". Typically this would allow the composer to control the amount of legato or staccato, but also allows polyphony. It is easy to edit and analyse because it is compact. The following example is musically equivalent to the example for the Format 2 shape.

```
Time Note Velo OnTime
 12    5   80   10 ( on for 10/12 )
  0    7   90    8 ( play 7 & 11 together )
 12   11   90    8
```

Type 4: MIDI Data Stream.

This type contains all forms of MIDI data, pitch bends, preset changes, etc. These are most useful for capturing a MIDI performance verbatim. These are not very common in HMSL.

Type 5: Control Shapes.

These shapes contain morphs or functions to be executed at scheduled times. These could be used to implement Cue Lists for a soundtrack.

Shape Conversion Utilities - SH.COMPRESS.NOTES

These routines copy data from one shape in one format to another shape in another format. **The times must be stored as *absolute* times for these to work.**

`SH.COMPRESS.NOTES  ( source-shape target-shape -- )`

The source shape is a type 2, "Expanded Notes" shape.

The target shape is a type 3, "Compressed Notes," shape. For example:

```
OB.SHAPE  EXP-SHAPE
OB.SHAPE  COM-SHAPE
:  TEST1  ( -- convert shape data )
   30  3  NEW:  EXP-SHAPE
   STUFF{
\  Time Note Velocity
   10    5   80  ( turn note 5 on )
    2    5    0  ( then off )
    0    7   90  ( play notes 7 & 11 together )
    8   11   85
    0    7    0  ( turn them off )
    4   11    0
   }STUFF:  EXP-SHAPE
\
\ Convert RELATIVE times to ABSOLUTE
```

```
        0 INTEGRATE: EXP-SHAPE  ( -- sum )
        PRINT: EXP-SHAPE
        \ Now convert to Compressed form.
           EXP-SHAPE  COM-SHAPE  SH.COMPRESS.NOTES
           PRINT:  COM-SHAPE
        \ Convert ABSOLUTE TIMES back to RELATIVE using SUM on stack
           0 DIFFERENTIATE: COM-SHAPE
        ;
        TEST1
```

SH.EXPAND.NOTES (source-shape target-shape --)

The source shape is a type 3, "Compressed Notes" shape.

The target shape is a type 2, :Expanded Notes." shape. For example:

```
OB.SHAPE  EXP-SHAPE
OB.SHAPE  COM-SHAPE
:  TEST2  ( -- convert shape data )
   30  4  NEW:  COM-SHAPE
   STUFF{
\  Time Note Velo OnTime
   12    5   80   10 ( on for 10/12 )
    0    7   90    8 ( play 7 & 11 together )
   12   11   90    8
   }STUFF:  COM-SHAPE
   0 INTEGRATE: COM-SHAPE
   PRINT: COM-SHAPE
\ Now convert to Compressed form.
   COM-SHAPE  EXP-SHAPE  SH.EXPAND.NOTES
   PRINT:  EXP-SHAPE
   0 DIFFERENTIATE: COM-SHAPE
;
TEST2
```

The Shape Editor Screen

Introduction

The *Shape Editor* provides a way to graphically edit the contents of a shape. It will display any dimension of a shape as a graph of value versus index. The Shape Editor allows you to edit not only OB.SHAPE objects, but also any object which is a descendant of OB.SHAPE. In this way, the editor can be used to manipulate envelopes, waveforms, samples, and any user-defined subclasses of OB.SHAPE without modification.

For a shape to be edited, it must be ADD:ed to the list called SHAPE-HOLDER. For example:

```
OB.SHAPE MY-SHAPE
32 3 NEW: MY-SHAPE
MY-SHAPE ADD: SHAPE-HOLDER
HMSL
MY-SHAPE DELETE: SHAPE-HOLDER
```

If you forget these shapes, be sure to **CLEAR: SHAPE-HOLDER** , or use **DELETE:** as above, to remove invalid addresses from the holder.

Description of the Screen

The screen, when selected, displays a series of control grids and a Cartesian plane (area for drawing and displaying shapes). The grids displayed (counterclockwise from the top left) allow:

* setting the editing mode (SetMode)
* selecting the shape to be edited (SelectShape)
* selecting the dimension of the current shape to be edited (Dim)
* executing an operation (Operations)
* setting coefficients for scaling (ABC)
* setting modes (Options)

The shape display area allows the display and editing of a single dimension of a shape. The x-axis is the element index; the y-axis indicates the values of that element for the current dimension. The scale for the y-axis is dynamically modified by the system to suit the values of the dimension or portion of the dimension currently being displayed. When a new dimension is selected, the MIN and MAX values from that dimension are used to determine the range of values displayed along the y-axis. The y-axis is rescaled if you go outside the bounds when editing or when you change dimensions.

When you are editing a shape, the values you can enter are limited to the range of values specified using the shape method PUT.DIM.LIMITS: . This applies to INSERT , REPLACE , INVERT , RANDOMIZE , etc.

Above the shape Display axes (top left), the screen shows the x and y values of the point most recently changed or entered; directly below the x-axis the screen displays the name of the current shape and the name of the current dimension (for example, "Pitch," "Location"). If no name is given to the dimension (you use PUT.DIM.NAME: to do this), three dashes will be displayed instead of that name.

The SetMode grid includes the following options:

INSERT allows the insertion of points in the shape within the limits specified by PUT.DIM.LIMITS: .

DELETE allows the deletion of points from the shape.

REPLACE allows the modification of the value of a point.

SELECT allows the user to select a range of elements to operate on.

RUBBER allows you to drag a "rubber band" to specify values along a line. Click on RUBBER then click and drag in the graphic area.

DRAW allows you to draw values in freehand. Click on DRAW then click and drag in the graphic area.

SET Y allows the specification of a y value used by some of the operations.

RANDOMIZE allows the user to randomly set values for selected points.

The SelectShape grid allows the user to pick a shape to be edited. The name of the shape currently selected is shown in the center of the box; clicking the mouse button in the box with the "^" in it will select the next shape in the Shape Holder; clicking the button in the "v" box will select the previous shape in the holder. The selection wraps around; that is, selecting the previous shape when you're currently looking at the first shape in the Holder will select the last shape in the Holder, and vice versa. You must click on the name of the shape itself to select the shape!

Note that shapes may be defined in the system but not appear in the SelectSHAPE grid. Shapes must be placed in the Shape Holder (MY-SHAPE ADD: SHAPE-HOLDER) for them to appear in the Shape Editor Screen.

The Dimension grid allows the user to select the dimension of the current shape. Unlike the SelectShape grid, the central box displays the number of the dimension currently selected; the "^" box allows the selection of the next dimension, and the "v" box the previous one.

The Operations grid allows a number of editing operations, including the following:

CUT cuts a range of points from the shape and move them to temporary storage.

COPY copies a range of points to temporary storage.

PASTE copies a set of points from temporary storage to a point on the screen. This replaces any selected points

INVERT flips a range of points vertically about the Y mark.

UP1 and DOWN1 increment and decrement the values of a range of points.

REVERSE reverses the order of points within the range.

SCRAMBLE randomly "shuffles," or reorders, the values of a range of points without changing the values.

***A/B+C** performs a scaling operation on the selected data. The Coefficients A,B,C are taken from the ABC Numeric Control Grid. This is particularly useful for durations. Each value will have the following integer operation performed on it:

v' = (v*A)/B + C

Note that in integer math, v*(a/b) is not necessarily the same as (v*a)/b. For example, in integer math, "one divided by two" is zero (!), so "one hundred times (one divided by two)" is still zero, whereas "(one hundred times one) divided by two" is 50. In other words, on the shape editor screen, this function multiplies your y value by a, and then divides that result by b, and then adds c to that result.

-C*B/A performs the inverse of the above scaling operation. Because of roundoff errors, you may not always get back exactly the same values you started with. This function can be written as:

v' = ((v-c)*b)/a

or, subtract c from the y value, multiply that result by b, and then divide by a.

ZOOM expands the view of a range of points to fill the shape Display axes.

UNZOOM restores the view to the entire dimension.

<- PAN displays the section of a shape immediately to the left of those in a zoomed display.

PAN -> displays the section of a shape immediately to the right of those in a zoomed display.

The Options Grid

This grid allows you to select various options for the Shape Editor.

All/1 determines whether a REVERSE or SCRAMBLE operation will operate on ALL dimensions or just the one currently being displayed.

TRACK determines whether a vertical line will be displayed over the element currently being played. If the Event Buffering is on, the line will appear to precede the sound. This drawing takes a lot of time and can slow down a time critical piece so use this carefully. Try this with SHEP.

Tutorial 3: Customizing a Shape for the Shape Editor

Here's a simple example of defining a shape, setting some simple limits for its values, and putting some names for its dimensions that will show up in the shape editor.

```
OB.SHAPE MIDI-SHAPE
OB.PLAYER MIDI-PLAYER
: SETUP.MIDI-SHAPE
   10 3 new: midi-shape ( shape is 10 long and 3 wide )
   0 127 2 put.dim.limits: midi-shape
      ( limit velocity dimenension to legal values )
   " Duration " 0 put.dim.name: midi-shape
```

```
      " Pitch " 1 put.dim.name: midi-shape
      " Velocity " 2 put.dim.name: midi-shape
      1 new: midi-player
      ins-midi-instrument put.instrument: midi-player
      midi-shape add: midi-player
      midi-shape add: shape-holder
   ;

      MIDI-PLAYER HMSL.PLAY
```

Note that this shape will not let you edit in velocity values greater than the MIDI velocity limit of 127. Don't forget to CLEAR: or DELETE: this shape from the shape-holder when you're finished.

Advanced Use of the Shape Editor

Here are some advanced calls you can use with the Shape Editor.

SE.GET.DIM (-- dimension , currently selected)

SE.GET.SELECT (-- start end , of selected range)

If no range is selected then end < start.

SE.GET.SHAPE (-- shape , currently being edited)

SE.GET.YMARK (-- yvalue , as in "Set Y")

SE.UPDATE.DRAW (-- , redraw shape after editing)

SE.UPDATE.SHAPE (shape -- , update drawing of shape)

If your program is changing the values in a shape and you want these changes to appear in the Shape Editor, call this word. See HP:XFORMS for an example. It will only draw if the shape is currently being displayed.

SE.SET.CUSTOM ($text cfa -- , set custom function)

Specify a function to call if the the "Custom" part is hit on the Operations grid. You can use some of the other advanced calls in your custom function. As an example, here is the actual definition for the INVERT function.

```
: SE.INVERT  ( -- , invert data )
   se.get.select  2dup <
   IF se.get.ymark  -rot 1-
      se.get.dim
      se.get.shape invert: []
      se.update.draw
   ELSE 2drop
   THEN
;

: INSTALL.CUSTOM  ( -- , install custom function in SE )
   " MyInvert"  'c se.invert  se.set.custom
;
```

Cross Reference

* Shape data can be loaded using the Score Entry System word **SHAPEI{**. This makes for an easy interface between data described in conventional musical terms to shape data, and vice versa. See SES chapter.

* Shapes can be played using players. See Chapter 6, Players.

* Shapes can be dumped to MIDIFiles. See Chapter 13 on MIDI.

* Shapes can be recorded via MIDI, and then accessed in many ways throughout HMSL. See Chapter 17 on Recording and Sequencing. Shape data may also be "dumped" and loaded to the sequencer.