OM-Frak Documentation

Alexander Prähauser

February 13, 2022

Chapter 1

Summary

The OM-Frak fractal music generator for OM# is based on Tom Leinster's conception of fractals (see [1] for a readable overview) and seeks to transfer this theory into the musical realm. It starts with a list of basic shapes, which are musical motifs with specified insertion intervals, then recursively inserts the given motifs into each other along the specified insertion intervals, after doing some scaling and applying some given functions, until a given depth is reached.

Chapter 2

Input

OM-Frak is built for versatility and treats rhythm and pitch information separately. As a tradeoff, the input is somewhat complicated (see also the exemplary patch of the Mahlstrom composition). It consists of (in order)

1. a list of *input contours* containing the rhythm and insertion information,
2. a list of *normalized pitch sequence*[1],
3. the *recursion depth*,
4. optional *modification functions* that further modify the behavior.

2.1 Input Contours

The bulk of input information for OM-Frak is given in the input contours. An input contour consists of (in order)

1. *insertion data*,
2. the *rhythm sequence* of the contour,
3. the *pitch sequence number* of the contour,
4. the *height* of the shape.

2.1.1 Insertion Data

The insertion data of a shape is primarily given by *insertion datums*. Due to the requirement that each voice only have one melodic line however, these insertion datums themselves are organized into *insertion lines*. An insertion line is a list

[1]The assumption throughout this document is that OM-Frak is used to calculate rhythms and pitches. However, it can be used to calculate all kinds of numeric information. See also the section on Csound score calculation.

of insertion datums, subject to the requirement that the intersection of any of the *insertion intervals* of its insertion datums is either empty or one point (so the end point of an insertion interval can be the starting point of the next, but the overlap cannot be any larger), and ordered by starting points of insertion intervals.

Insertion Datum

An insertion datum is given by

1. the insertion interval, which is given by a bracketed tuple of two numbers, the first of which is the *starting point* of the insertion or, more precisely, the length between the beginning of the shape and the beginning of insertion[2], while the second number describes the length of the insertion interval. For instance, (2 1) describes the interval of an insertion that starts after two whole notes and has the length of one whole note.
2. the *CP section*.

CP section The CP section of an insertion datum consists of

1. the *contour number* of the base contour that is to be inserted.
2. the *pitch datum* of the insertion.

Pitch Datum A pitch datum consists of

1. the number of the normalized pitch sequence in the list of normalized pitch sequences.
2. the *height* of the insertion at its starting point. The pitch center of the normalized pitch sequence will be transposed by this much at its starting point when it gets inserted.
3. the height of the insertion at its end point. The *pitch center* of the normalized pitch sequence will be transposed by this much at its end point when it gets inserted. The pitch center of the pitch sequence between its start and end is calculated based on the *curvature function*, with the simplest case being a linear curve. See also the sections about normalized pitch sequence and the curvature function for more information.
4. the *multiplier* of the insertion at its starting point. The distance of the pitches of the normalized pitch sequence to its pitch center will be multiplied by this amount at the start of the insertion. Usually, this will be a number smaller or equal to 1.

[2]Generally, points of an interval will be identified with their distance from the beginning of the interval

5. the multiplier of the insertion at its end point. Again, this determines the scaling of the pitch sequence, this time at the end of the insertion. The pitch multiplier between the start and end of the insertion is calculated based on the input *multiplier function*, with the simplest case being a linear curve. See also the sections about normalized pitch sequence and the multiplier function for more information.

Example of Insertion Data

(((3/4 1/3) (0 (0 900 700 1 1))))

is the insertion data of one insertion line containing one insertion datum, which occupies an interval that starts after one quarter note and is one third of a whole note long. In this interval, the rhythm sequence of the first base contour is inserted (keep in mind that the enumeration of lisp lists starts at zero, so that the first entry is actually enumerated by 0) and given the first normalized pitch sequence, transposed by 900 midicents at the beginning and 700 midicents at the end, with trivial scaling. This behavior is however subject to optional customisation given by modification functions. See there for more information.

2.1.2 Rhythm Sequence

The rhythm sequence R of a contour is just the list of durations of its notes and breaks (breaks being marked by minus signs, as usual). No more complex rhythm trees are allowed, nor would they be useful, since measures lose their meaning after an insertion has taken place.

2.1.3 Pitch Sequence Number

The pitch information of each contour that is newly built by OM-Sharp is already specified by the pitch data of the insertion datums used to build the contour. Since however this is not possible for the base contours, their corresponding pitch sequence has to be specified manually by giving its list number. Several contours can share the same pitch sequence, but the pitch sequence of a base contour has to have as many pitches as its rhythm sequence has non-break elements.

2.1.4 Height

The height of a shape is the amount the pitches of the whole shape are transposed once all recursive insertions have taken place.

2.1.5 Example of a contour

((((3/4 1/3) (0 (0 900 700 1 1)))) (1/2 1/4 -1/3 1/4) 0 6000)

is an example of a base contour for a shape, with the insertion data from before with a rhythm sequence, such that the insertion interval of the only

insertion datum of the contour is precisely over a a break of 1/3 in the rhythm sequence of the contour. To this base contour is associated the first normalized pitch sequence in the list of pitch sequences, and at the end of the algorithm, the pitches of each created contour will be transposed by 6000 midicents.

2.2 Normalized Pitch sequence

A normalized pitch sequence is just a sequence of pitches, except that a pitch center has been chosen and all pitches are given by their distance from this pitch center. For instance, (0 900 -100 0) is the normalized pitch sequence of a motif that starts by sounding its pitch center, followed by the note 900 midicents above its pitch center, then the note 100 midicents below the pitch center and finally the pitch center again. Thus the pitch center functions as the central axis of the normalized pitch sequence. In an insertion, this central axis is transposed by the starting height of the insertion at its starting point and by the final height of the insertion at its end point. In between, the pitch center is calculated by the curvature function. Then the normalized pitch sequence is multiplied by a multiplier, so that multipliers with absolute values less or equal to one correspond to shrinking the normalized pitch sequence toward its center, multipliers with values larger than one correspond to stretching it and negative values correspond to inverting the pitches around the pitch center. The value of the multiplier at the start and end of the insertion is given by the insertion, the values in between are calculated by the multiplier function.

Example of a normalized Pitch Sequence (-100, 0, 100) is an example of a normalized pitch sequence that would be suitable for the contour example above. It consists of three semisteps, and its pitch center is the middle pitch.

2.3 Recursion Depth n

This is the number of times new contours are built by the OM-Frak-Core algorithm. Furthermore, the *iteration number* is derived from the recursion number and used to count the depths at which an insertion is made: each time a *new insertion datum* is built, it is enumerated by the amount of recursions that have passed once the contour built from the insertion is used as an *acute contour*. So the insertion datums provided as input get the number 1 and the insertion datums from which the last batch of contours is built get the number n, since the last iteration of OM-Frak-Core doesn't create any new contours. To keep the building process uniform, the input contours provided as input get insertion number 0.

2.4 Modifying Functions

The remaining inputs of OM-Frak are given by functions that allow for finer control its behavior. All of this functions have defaults and are thus optional.

2.4.1 Rhythm Modifying Function

The *rhythm modifying function* $mr(R, i)$ is a function that takes as inputs a normalized rhythm sequence R and a number i. It is applied to the rhythms of the base contours after they are extracted and before they are aligned into their insertions to create a *new contour*, and the number of the next iteration, and is generally to be thought of as a family of functions parameterized by the iteration number i. Usually these functions will be recursive in the sense that $mr_{i+1}(R) = mr_i(mr_1(R))$ (please note that mr_0 is never applied since the function modifies the creation of new contours and 0 is the number of the acute contours at the beginning of the algorithm, which are given as input). The intended use is to apply a permutation to the rhythm sequence, so that, for instance, mr_1 rotates the rhythm sequence by one, but it could also be an automorphism of the unit interval that stretches and squeezes it at some places. However, it is important that the output rhythm sequence $mr_i(R)$ is again of unit length. The default for this input is the function $mr(R, i) = R$, which leaves the rhythm sequence unchanged.

2.4.2 Pitch Modifying Function

The *pitch modifying function* $mp(P, i)$ is a function that takes as inputs a normalized pitch sequence P and a number i. It is applied to the normalized pitch sequence before they are associated to their contours, and the iteration number of the contour they will be associated to. As with the rhythm modification function, it should be thought of as a family of functions $mp_i(P)$ parameterized by the iteration number i, and will usually be recursive. A typical example is a permutation of the pitch sequence or a transposition of all pitches by a constant amount.

2.4.3 Curvature Function

The *curvature function* $c(x, y, i, p)$ is a function that takes as input three numbers and one number between 0 and 1. It calculates the height h_p of the pitch center at a position p of the interval of an insertion based on the heights of the insertion at its start x and end y, the iteration number of the insertion i and the relative position $\frac{p}{l}$ within the insertion interval, standardized by dividing it by its length l. It is applied at two different places: first, whenever the insertion data of a new contour is created, it is used to calculate the starting and ending heights of the acute insertion that the new insertion is based on, which is one summand of those heights[3]. Second, at the very end, when the

[3]Subject to modification by the summation modification function, see below

pitch sequence for each contour is created, it is used to calculate the height of the pitch center at the starting point of each non-break element of the insertion. The default value of this input is the function $c_l(x, y, i, p) := x + (y - x)p$ which calculates the pitch center as a straight line, other reasonable choices include $c_p(x, y, i, p) := x + (y - x)p^2$, which calculates the center as a parabola or $c_{sin^2} := x + \frac{2sin(p)^2}{\pi}y$, which has a fairly natural shape, since its slope is 0 at both the start and end of the interval so that it stays smooth at the corners. This input is mainly thought for large-scale structures, the difference for small intervals might be hard to perceive.

2.4.4 Multiplier Function

The *multiplier function* $m(x, y, i, p)$ is a function that takes as input three numbers and one number between 0 and 1. It is generally similar to the curvature function, except that doesn't calculate the height at any given point, but the multiplier, and is used at the same places. With it, for instance, trumpet-like shapes can be created. The default value is $c_l(x, y, i, p) := x + (y - x)p$.

2.4.5 Insertion Permutation Function

The *insertion permutation function* $pi(\iota, \rho, \pi, i)$ takes as input three lists and is applied to

1. the list of insertion intervals of the insertions of an insertion line
2. the list of contour numbers of the insertions of an insertion line
3. the list of pitch datums of an insertion line
4. the iteration number of the current iteration.

The insertion permutation function is applied before a new contour is created from the insertion datums of an insertion line. Like the other functions, it should generally be thought of as a family of functions $pi_i(\iota, \rho, \pi)$ parameterized by the iteration number i and is usually recursive. A typical application is to permute the rhythms and pitches that are inserted into an insertion interval. Please note however that the contour number of an insertion does not just control the inserted rhythm but also the insertion data that is built for future insertions. Also, due to the fact that apply currently only has one possible output, the insertion permutation function has to restore the bracketing of the insertion data, so has to rebracket its output into a specific form. Please see the the insertion permutation function of the *Mahlstrom* exemplary composition for a usable example.

2.4.6 Summation Modification Function

The summation modification function takes as input three numbers and is applied at two steps of the algorithm:

1. It is applied each time the height of a new insertion datum is calculated from the height and multiplier of the acute insertion the new insertion is based on and the height of the insertion from the building contour that is to be modified.

2. It is applied at the very end to calculate the pitch of a contour at a given position from the height of the contour at that position, its multiplier at that position and the entry of the normalized pitch sequence at that position.

Its arguments are in order (height of the building contour/value of the normalized pitch sequence, multiplier, height of the acute contour). The default for this function is $f(x, y, z) = x * y + z$, multiplying the height of the building contour with the multiplier and adding the height of the acute contour. Using this function, this behaviour can be modified, as is sometimes reasonable or even necessary, in particular when OM-Frak is used not to calculate pitches but other values for Csound. For instance, one reasonable choice is $f(x, y, x) = (x - z) * y + z$, translating the acute height by the vector from the acute height to the building height scaled by the multiplier. If x, y, z are between 0 and 1, this punction will ensure that the result will be too, which can be important for some Csound variables.

Chapter 3

Algorithm

The OM-Frak algorithm can roughly be divided into four parts: the pre-calculation preparation, the calculation of new contours, the calculation of of the *pitch sequence* for each contour and the final preparation of the output.

3.1 Initial Preparation

At this step, the input data is prepared and and forwarded to OM-Frak-Core.

1. The *total length* the rhythm sequence of each contour is calculated by summing the absolute value of its elements, and extracted for use during the final preparation.
2. Each input contour is normalized by dividing its rhythm sequence and insertion intervals through the total length of the rhythm sequence.
3. To each insertion pitch datum is appended a *length number*, which is the length of its insertion interval and the iteration number 1.
4. The shape height of each shape is extracted and will be used in the final preparation.
5. The last two numbers of each input contour are replaced by a *contour pitch data list* containing one contour pitch datum of the form (k 0 0 1 1 0), where k is the pitch sequence number of the contour.
6. The input contours are fed to OM-Frak-Core in two ways:
   (a) As *base contours* that will be used to build new contours from the insertion data the acute contours at each iteration.
   (b) As the first acute contours of each shape.
7. A list is created that contains an empty list for each shape, which will used to store the *processed contours*.

All other input is passed through, except for the recursion number, which is duplicated and fed to OM-Frak-Core in two ways, once as the *acute recursion number*, which will be reduced by one in each iteration of OM-Frak-Core, and once as the *total recursion number*, which remains constant. The iteration number is then calculated as the difference between the total recursion number and the acute recursion number.

3.2 OM-Frak-Core

OM-Frak-Core first checks if the recursion depth is zero. If the recursion depth is zero, the processed contours are finished. If it is not zero, then new contours are created from the acute contours received by OM-Frak-Core.

3.2.1 Creating New Contours

Contour creation consists roughly of the following steps:

1. To the insertion datums of every insertion line of every acute contour of every shape, the insertion permutation function is applied.
2. For each insertion datum of an insertion line, the base contour specified by its contour number is taken, its contour pitch datum is replaced by the insertion pitch datum of the insertion, and it is used as a *building contour*.
3. Of each building contour, the rhythm sequence is extracted and modified in the following way:
   (a) The rhythm modification function is applied.
   (b) The new rhythm sequence is multiplied by the length of the insertion interval it will be inserted in.
   (c) If the insertion interval x into which the new rhythm sequence is to be inserted does not start at zero, a break with length of the interval between the ending point of the previous insertion (or the start of the contour, if x is the first insertion of the insertion line) and the starting point of x is inserted.
4. From the insertion data of each building contour, new insertion data is built by modifying it in the following ways:
   (a) Each new insertion interval of the new insertion data of each building contour is multiplied by the length of the acute insertion interval that the contour will be inserted in.
   (b) Each pitch datum is modified by
      i. replacing the length number by the length of the insertion interval,

ii. multiplying the starting/end multiplier of the new pitch datum with the multiplier of the acute insertion datum the new contour is inserted in at the starting/end point of the new insertion, as calculated by the multiplier function,

iii. calculating the starting/end height of the new pitch datum from the starting/end height of the building datum, and the multiplier and height of the acute pitch datum at the starting/end point of the new insertion, as calculated by the multiplier function and curvature function, through the summation modification function.

iv. replacing the iteration number of the pitch datum by the current iteration number plus two, which is the iteration when the contours constructed by the new pitch data will become acute.

5. A new contour is created by aligning the rhythm sequences and insertion data of the modified building contours.

   (a) The rhythm sequences of each contour are strung together and a final break is inserted that brings the rhythm sequence to unit length.

   (b) The contour pitch datums of all building contours are put into the contour pitch data list of the new contour.

   (c) The insertion lines of the new contour are created by, for i from 1 to the maximum number of insertion lines of any building contour, for every building contour, taking the ith insertion line of the contour if it exists, adding the length of the sum of the rhythm sequences of previous contours to the starting point of the insertion interval of each insertion of the insertion line, and appending it to the ith insertion lines of previous building contours. This way, the lowest possible number of new insertion lines is created.

6. After the new contours are created, for each shape, the acute contours of this recursion are added to the processed insertion,

7. the acute recursion number is reduced by one,

8. and a new instance of OM-Frak-Core is instantiated, which is given the new contours of this iteration as acute contours, the lowered recursion number as recursion number, the enriched stack of processed contours as processed contours, and all other inputs are passed through.

3.2.2 Finishing contours

If the acute recursion number given to OM-Frak-Core is 0, then, for each shape, for each contour,

1. the rhythm sequence is extracted,

2. for each contour pitch datum, the corresponding normalized pitch sequence is extracted and for each item in the sequence is calculated a pitch number from the item, the multiplier at the position corresponding to the item as calculated by the multiplier function, and the curvature at the position as calculated by the curvature function, by the summation modification function. Then the calculated pitch sequence are strung together.

Then each rhythm sequence and its corresponding pitch sequence are put into a two-item list and passed on to final preparation.

3.3 Final Preparation

Finally, all rhythm sequences in a shape are descaled by multiplying them with the length of the input contour of that shape, and all pitch sequences of a shape are transposed by the height of that shape, and rounded to the nearest integer.

3.4 Output

The output of OM-Frak consists of a list of shapes that are each filled by a list of finished contours, that consist of a two-item tuple of a rhythm sequence and a pitch sequence. All rhythm sequences are of the same total length, which is the total length of rhythm sequence of the input contour. The pitch sequences that are given as output are *not* in integers and usually have to be rounded. This is because OM-Frak can also be used to calculate Csound variables, and some Csound variables aren't useful if they are rounded, see the section on Csound score calculation for more information. The omfrakrp patch can be used to round the pitches of the OM-Frak output.

Chapter 4

Techniques

4.1 Csound Score Calculation

OM-Frak is a powerful fractal generator with the ability to compute the pitch value of generated notes up to midicent level. Given the many configuration options it exhibits, in particular the ability to choose non-constant multipliers and differing start and ending insertion heights, the output cannot generally be assumed to be within 12-TET or one of its multiples. This raises issues concerning the playback of OM-Frak-generated scores. One possibility is to use a synthesizer within OM#, such as Csound. However, such synthesizers often require more input than simply note-length and pitch values, like amplitude. However, OM-Frak currently only produces two output lists per shape, assumed to correspond to amplitude and pitch. But the algorithm used to construct the second list does not depend on being applied to pitch lists and can, with appropriate values, similarly produce lists determining amplitude or any other chosen value. So one possible technique to obtain the values needed for a satisfactory Csound performance is by running the algorithm twice, once with the sequences and datums to calculate pitch values, and once with sequences and datums to calculate amplitudes. Of course, this procedure can be repeated to obtain other values too, if needed. See the Charybdis-Csound patch for a complete example.

4.2 Fading

Sometimes it might be desirable to have a motif a fade in gradually, over the course of several iterations. This can be simulated by a sequence of motifs $a_i, 0 < i \leq n$ such that $a_i \subseteq a_i + 1$, $a_n = a$ and each subsequent motif a_{i+1} is inserted into a_i with the insertion interval of a_{i+1} being the whole of a_i.

Bibliography

[1] T. Leinster, 'General self-similarity: an overview', https://arxiv.org/abs/math/0411343