Implementation of aVST3 plugin based on the geometrical modelling of Tonal Harmony with the Spiral Array model within the Juce framework for intelligent lighting engineering purposes

Alessio Sopranzi, Simone Priori

Abstract

This project is mostly devoted to the implementation in C++ language of a plugin within the Juce framework whose main goal has been to automatize lighting on a stage in a real-time fashion. It is based on the incoming and processing of MIDI events which are processed in real time, and the extracted tension metrics can control the tuning of brightness and change of patterns in stage lights. Demonstration of this has been carried out in Festivalle electronic music stage in Valle dei Templi, 9 and 10 August 2019.

1 The spiral array model and application to MIDI data

1.1 Introduction

Perceptually speaking, the term "tension", as employed and modelled here, can refer both to sensory dissonance and cognitive dissonance or instability. Often tension is referred to how musical scales come up, how musical keys work for our perception, how to create expectation in harmony, rhythm etc.... According to Mary Farbood in her research on a temporal, parametric model of tension [1], "increasing tension is a feeling of rising intensity or impending climax, while decreasing tension can be described as a feeling of relaxation or resolution".

There are many way to create musical tension breaking the expectation-resolution "law" and surprising the listener in a controlled-wise breaking-the-rule approach. For example, introducing dissonant tones resolving into consonant ones is maybe one of the easiest ways to create musical tension. Modulation to other tonal contexts or a resolution within a certain key that's a long time coming can increase tension too. Tension can be achieved in other ways such as an increase in harmonic rhythm (starting with one harmony per bar, then two, then a further increasing occurring). There are also forms of other non-harmonic tensions such as syncopation, which is a common one, and last, but not least, even unexpected silence can create enormous tension in some music genera such as rock music or in an orchestra.

However, tension is a complex characteristic, which makes it very hard to measure in a quantitative way. It does worth precising that the problem of quantifying such a difficult definition is here addressed in the context of tonal music. *Tonal music* has one pitch as

its primary focus, a *tonal* center, that serves as a reference point for all the other chords and melody notes in the piece and *Harmonic function* of the chords composing a certain key describes their relationship within the tonal center itself. As a result, the relative stability of the chord changes in relation to the tonal center [2].

The key of a musical piece identifies the most stable pitch, the tonic, and its hierarchical relations to all other pitches in that pitch set. The tonic is sometimes referred to as the tonal center and when the key changes, each pitch assumes a different role in relation to the new tonal center.

1.2 About the spiral array model

Musical tension can be divided in three categories: melodic tension (melody versus chord), harmonic tension (chord versus key), and chord tension (simultaneous tones). An example of quantitative theory of tonal tension had been proposed by Lerdahl [3] and his model was based on four components: a representation of hierarchical (prolongational) event structure, a pitch-space model and all distances within it, a treatment of surface (largely psychoacoustic) dissonance, and a model of voice-leading (melodic) attractions. These components combine to predict the rise and fall in tension in the course of listening to a tonal passage or piece. This theory has also a strong empirical evidence with a variety of musical excerpts [4]. Just to make a comparison, while Lerdahl's method is more focused on harmonic tension, Chew's spiral array method implemented here is related to chord tension. Chords (or arpeggios) tend to «resolve» locally to some specific notes and not to others, and the harmonic tension of the chord is determined by the intervallic structures formed by its notes [5].

The spiral Array model is a geometrical spatial construct representing the interrelations among musical pitches. Essentially, it is a three-dimensional realization of the Harmonic Network [6] (an arrangement of pitch classes on a lattice separated by perfect fifth intervals along the x-axis) following from its "rolling up" to eliminate redundancy. The result of this transformation is the pitch class spiral of the Spiral Array. The model is generative and designed to provide a concise description of the characteristics of tonality representing pitches, intervals, chords and keys in the same spatial framework. Distance in the model corresponds to perceived closeness in tonal music [7]. The model is versatile if we try to apply it to several fundamental problems in the cognition and analysis of western tonal music.

Pitch classes are represented by spatial coordinates along a spiral and each pitch class is indexed by its number of perfect fifths from an arbitrarily chosen reference pitch, C (set at position [0,1,0]). An increment in the index results in a quarter turn along the spiral. Four quarter turns locate pitch classes related by a major third interval in vertical alignment with each other as shown in Figure 1. The spiral form is assumed so to preserve the symmetry in the distances among pitch entities.

1.2.1 Representation of interval, pitch classes, chords and keys

In the Spiral Array model, adjacent pitch classes in the circle of fifth representations are positioned at each quarter turn of the spiral, that is, following along the ascending spiral, neighboring pitches are a perfect fifth apart. In such a way, tonal representations in the spiral array mirror close tonal relationships between the entities.

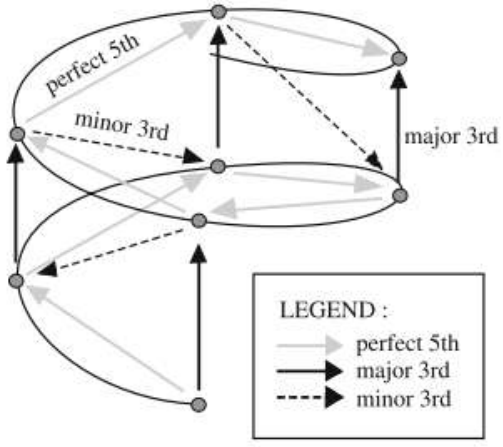

Figure 1 - Perfect fifth, major third, and minor third interval representations in the Spiral Array [7]

In the Spiral Array, interval relations correspond to Euclidean distances between pitch representations so that each pair of pitch positions is separated by some spatial distance which represents different interval distances.

As we can see, in this framework, the overall generating principle in the Spiral Array is that higher level representations of musical entities can be represented unambiguously as convex combinations of their lower level components. Geometrically speaking, this 3D configuration allows higher level musical entities to be defined and embedded in the interior of the structure as a form of nested spirals. The outermost helix represents pitch classes and inner helices represent higher level tonal constructs such as chords (major and minor) and keys (major and minor), as shown in Figure 2.

Chords and keys are represented by points in space corresponding to mathematical sum of its lower level weighted components. Spatial coordinates representing each key are generated from weighted averages of its I, V and IV chord representations, which were in turn generated from each chord's root, fifth and third pitch representations. Chord representations inherit the same periodicity property of making one full turn in four steps as the pitch classes do, so that the fourth step lines the current chord vertically above the starting one. Note also graphically from Figure 4 that having three triangles that "touching" in two points means having 9-2=7 notes identifying that scale.

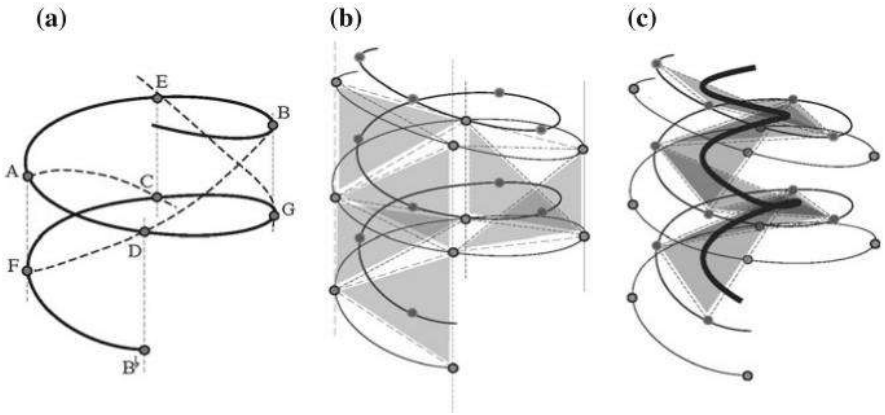

Figure 2 - The Spiral Array model: successive generation of major chords from pitches, and major keys from major chords. a Pitch class representations. b Major chord representations. c Major key representations [7]

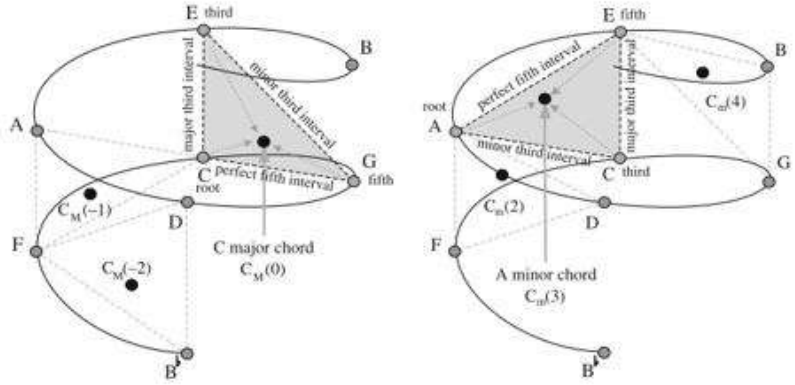

Figure 3 - Major (left) and minor (right) chord representations [7]

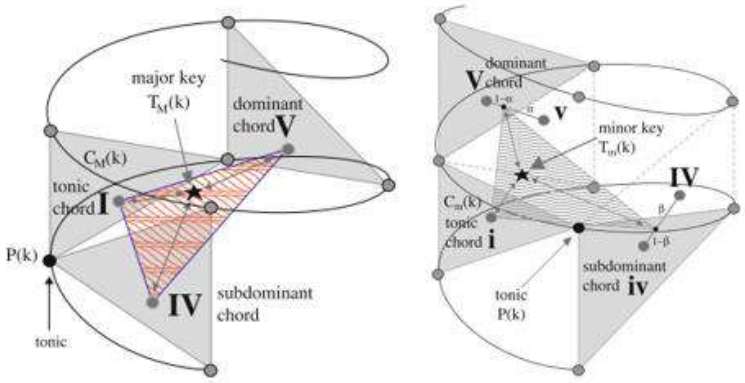

Figure 4 - Geometric representation of a major key, the composite effect of its I, V, and IV chords (left) and geometric representation of a minor key, a composite of its tonic (i), dominant (V/v), and subdominant (iv/IV) chords (right)

The representation of a major key is a combination of its I, V, and IV chord representations where the center one is the tonic (I), the one a quarter turn up the spiral (right neighbor of the I chord in Figure 3) the dominant (V), and the one a quarter turn down the spiral (left neighbor of the I chord in Figure 3) the subdominant (IV). A minor key reflects the same consideration and geometry but is a bit more complex. In the context of tonal music, we can have a natural minor key, harmonic minor key, and melodic minor key with the tonic triad being always minor in each case. However, the dominant triad is minor in the case of the natural, major in the case of the harmonic, and could be either major or minor in the case of the melodic minor mode. The subdominant triad is minor in the case of the natural, minor in the case of the harmonic, and major or minor in the case of the melodic minor key. This aspect will have effect on the mathematical representation of the minor key as explained in 1.2.3.

1.2.2 The center of effect CE

Any element in the space can generate a higher level construct, modeled, in the same space, as the centroid of its lower level members. The effect of time will be considered later. Thus, every collection of pitch representations can be appropriately weighted and summed to generate a barycenter in the interior of the helix that represents their composite effect called "center of effect" (CE), whereby any sequence of notes maps to a point in the interior of the model. Musical information can be condensed and summarized by 3D coordinates in this way. In other words, pitch classes of the input stream are mapped to their corresponding pitch positions and weighted by their proportional cumulative durations to get the CE of the input.

This is valid for any set of points in the spiral array and this concept is in fact exploited in the CEG algorithm, as described in Chapter 4 of the book [7], and briefly described later too. Because the Spiral Array model uses distance as a measure of perceived closeness, algorithms such the CEG uses the Spiral Array to reframe the problem of key recognition as a computationally simple one of finding a distance-minimizing key. For a given number of key changes, the computational complexity of the algorithm is polynomial in the number of pitch events. [8].

1.2.3 Pitch class and pitch spelling

Each node of the outermost spiral represents a pitch class, i.e., a note is not just a note in the middle octave of a keyboard, but all different octaves above and below it so that spiral Array model assumes octave equivalence.

On the other hand, the model takes pitch spelling into account, meaning that enharmonically equivalent (but differently-spelled) pitches, such as F# and Gb have different spatial counterpart in this geometric framework. It is important to keep the spiral in the cylindrical configuration, hence distinguishing between F# and Gb, to preserve this tonally critical information. Strict enharmonic equivalence would require that the spiral be turned into a toroid.

Pitch spelling is the process of assigning appropriate pitch names that are consistent with the key context to numeric representations of pitch, such as MIDI or pitch class numbers. Center of effect (CE), approximates and tracks the key context for the purpose of pitch spelling. The appropriate letter name is assigned to each pitch through a nearest-neighbor search in the Spiral Array space and this method refers to this geometrical construct without having to first discriminate the key we are playing in that instant of time.

The problem of pitch spelling is an artifact of equal temperament tuning in Western tonal music whereby several pitches are approximated by the same frequency. Pitches of the same frequency but denoted by different pitch names are said to be enharmonically equivalent. In MIDI format and many other numeric music representations, enharmonically equivalent pitches are represented by the same number indicating the note's ostensible frequency and not its letter name. Note that, in the spiral Array enharmonic triplets are always in the form of <index−12, index, index+12 >, where index represents the k used before, which means translating rigidly in height in the Spiral array model of three cycles in pitch position.

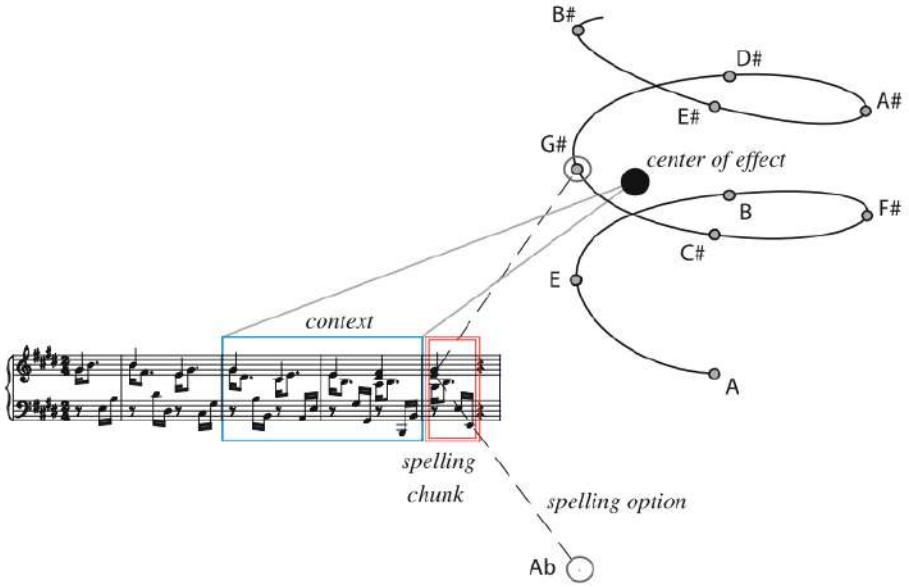

Figure 5 - Pitch-name assignment using the Spiral Array [7]

The local key context determines the pitch spelling; in turn, the name of the pitch determines the notation and serves as a clue to the key context. Here, the problems of pitch spelling and key-finding are decoupled by creating an algorithm that does not require explicit knowledge of the key context to determine the appropriate spelling. Using the Spiral Array, any musical fragment that has been correctly spelled will generate a center of effect (CE) that is closest to—and hence can act as a proxy for—the key.

Thanks to the CE method for generating a center of effect inside the Spiral Array, we use the Spiral Array model to determine spellings for unassigned pitches. Pitches in each key occupy a compact space in the Spiral Array model. Thus, the problem of finding the

best pitch spelling corresponds to finding the pitch representation that is nearest to the current key context. Each plausible spelling of an unassigned pitch is measured against the current CE, and the pitch that satisfies the nearest-neighbor criteria is selected to be the appropriate pitch name.

Assuming that $\mathbf{c}_j$ is the CE mathematically defined in 1.2.4 and representing the context of chunk j, with candidates for spelling at our disposal, the index that is consistent with the key context is:

$$I^*_{i,j}(\mathbf{c}_j) = \arg\min\{\|\mathbf{P}(I_{i,j} - 12) - \mathbf{c}_j\|, \|\mathbf{P}(I_{i,j}) - \mathbf{c}_j\|, \|\mathbf{P}(I_{i,j} + 12) - \mathbf{c}_j\|\}$$

The algorithm thus gives back the index that minimizes the function from $I_{i,j} - 12$, $I_{i,j}$ and $I_{i,j} + 12$.

1.2.4 Mathematical representation of spiral array entities

Two parameters, the radius of the cylinder (which constitute the outer envelope of the spiral), r, and the height gain per quarter rotation, h, uniquely define the position of a pitch representation as shown in Figure 6, which can be described as:

$$\mathbf{P}\ k\ = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} r\sin k\dfrac{\pi}{2} \\ r\cos k\dfrac{\pi}{2} \\ kh \end{bmatrix}$$

This notation stands for a point of three coordinates on the spiral representing a pitch of index k.

The chord's representation is generated by a linear combination of its three component pitch positions, the root, the fifth and the third, which can be denoted, according to the geometry of the spiral itself, as $\mathbf{P}\ k\ ,\ \mathbf{P}\ k+1\ $ and $\mathbf{P}\ k+4\ $ for major triads and $\mathbf{P}\ k-3\ $ for minor triads. The mathematical representations of a minor and major chord are:

$$\mathbf{C}_M\ k\ \stackrel{\text{def}}{=} w_1\mathbf{P}\ k\ + w_2\mathbf{P}\ k+1\ + w_3\mathbf{P}\ k+4$$

$$\mathbf{C}_m\ k\ \stackrel{\text{def}}{=} u_1\mathbf{P}\ k\ + u_2\mathbf{P}\ k+1\ + u_3\mathbf{P}\ k-3$$

with $\mathbf{C}_M\ k\ $ and $\mathbf{C}_m\ k\ $ being the major and minor chord respectively and with the weights w_i (or u_i) set monotonically decreasing from the root, to the fifth, to the third. In this notation, the k in round brackets stands for the pitch index of the root of the chord and the subscript in capital or small letter denotes whether the chord is major or minor respectively.

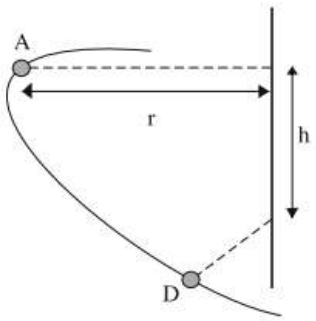

Figure 6 - The two parameters that uniquely identify pitch position: the radius (r), and vertical step (h) [7]

An important property of the Spiral Array is that representations of pitches in a given key still occupy a compact neighborhood. In the mathematical weighted averaging tonic, dominant and subdominant triad to compose a major key, the I chord is given the largest weight, followed by that of the V chord, then the IV chord:

$$\mathbf{T}_M\ k\ \stackrel{\text{def}}{=} \omega_1\mathbf{C}_M\ k\ + \omega_2\mathbf{C}_M\ k+1\ + \omega_3\mathbf{C}_M\ k-1\ ,$$

with $\omega_1 \geq \omega_2 \geq \omega_3$ and $\sum_{i=1}^{3}\omega_i = 1$. The minor key representation is generated by a convex combination of its tonic, dominant (major and minor), and subdominant (minor and major):

$$\begin{aligned}\mathbf{T}_m\ k\ \stackrel{\text{def}}{=} v_1\mathbf{C}_m\ k\ + v_2[\alpha\mathbf{C}_M\ k+1\ -\ 1-\alpha\ \mathbf{C}_m\ k+1\ ] \\ + v_3[\beta\mathbf{C}_m\ k-1\ +\ 1-\beta\ \mathbf{C}_M\ k-1\ ]\end{aligned}$$

with $v_1 \geq v_2 \geq v_3$, $\sum_{i=1}^{3} v_i = 1$ and $0 \leq \alpha \leq 1,\ 0 \leq \beta \leq 1$. We can notice that, when the minor key representation is modeled using the tonic minor, the dominant major and the subdominant minor, setting $\alpha = \beta = 1$ generates the effect of a harmonic minor key, while $\alpha = 0, \beta = 1$ corresponds to the melodic minor.

About the generic definition of center of effect, we can weight each pitch position by its duration since pitch duration is one of the factors that contribute to the cognition of pitch importance. If the i^{th} note is represented in the Spiral Array by pitch position p_i and has duration d_i, then the aggregate center at the i^{th} pitch event pitch is defined as

$$c_i \stackrel{\text{def}}{=} \frac{1}{D_i}\sum_{j=1}^{i} d_j p_j$$

where $D_i = \sum_{j=1}^{i} d_j$. A CE is generated at each pitch event (temporal step: each of the pitches from the beginning to the present is weighted (multiplied) by its proportional duration, and the center of effect is generated by the sum of these weighted pitch positions.

In the Spiral Array model, any collection of notes generates a CE. In particular, for a sequence of music time series data, the CE is a point in the interior of the Spiral Array that is the convex combination of the pitch positions weighted by their respective durations. If music information is divided in equal time slices called "chunks", we can define $\mathbf{c}_{a,b}$ to be

$$\mathbf{c}_{a,b} = \sum_{j=a}^{b} \sum_{i=1}^{n_j} \frac{d_{i,j}}{D_{a,b}} \cdot \mathbf{p}_{i,j}, \qquad with\ D_{a,b} = \sum_{j=1}^{b} \sum_{i=1}^{n_j} d_{i,j}$$

with formally n_j being the number of active pitches in time chunk j, $\mathbf{p}_{i,j}$ is the geometric Spiral Array position of the i^{th} pitch in chunk j and $d_{i,j}$ is the duration of the i^{th} pitch in chunk j. Basically, in a certain time instant, the various pitches played in that moment are summed together.

Specializing definitions in the context of MIDI data

Note that the previous definitions are general one in which we must define a step to be a pitch event. In the context of MIDI stream of data, the previous definition can be re-arranged in a slightly different way considering the Note On event as a step.

For example, if a piece of music is segmented by time into short time slices, n_j being the number of active pitches in time chunk j, $\mathbf{p}_{i,j}$ is the geometric Spiral Array position of the i^{th} pitch in chunk j then the evenly weighted CE of all pitch events in the window $a, b]$, then:

$$\mathbf{c}_{a,b} = \frac{1}{b-a} \sum_{j=a+1}^{b} \frac{1}{n_j} \sum_{i=1}^{n_j} \mathbf{p}_{i,j}$$

This CE definition used in our MIDI stream of data is conforming with the one in Chapt.9 used by E. Chew for the implementation of the MuSA.RT algorithm [7]:

$$CE_\alpha\ t\ = \alpha \cdot \mathbf{c}_{t-1,t} +\ 1-\alpha\ \cdot CE_\alpha\ t-1$$

This is basically a time series acting as a linear combination of the present and the past. This maintains two different CEs, one macro-level CE and one micro-level CE. Qualitatively, α determines the balance between the local and global contexts.

In the context of MIDI pitch events, labeled by a pitch number such as 67, that are assessed by the system, they need to be mapped to a pitch class name. This process of performing pitch spelling in order to map MIDI pitch numbers to the model is done in this way:

$$k^* = \arg\min_{\mathrm{k}} \|\mathbf{P}\ k\ - CE_\alpha\ t-1\ \|$$

where k is the index of an appropriately named pitch class. Once the proper pitch names have been assigned, the MIDI pitch events can be mapped to the Spiral Array to update both the local and long-term CEs. For each MIDI pitch event active at time t, amongst

all its plausible pitch class name assignments, our strategy is to choose the one that is closest to $CE_\alpha\ t$.

1.3 The tension model

In Herremans and Chew [9], the authors developed a model for tonal tension based on the spiral array [7]. The model represents tonal tension as captured by the geometry in the spiral array.

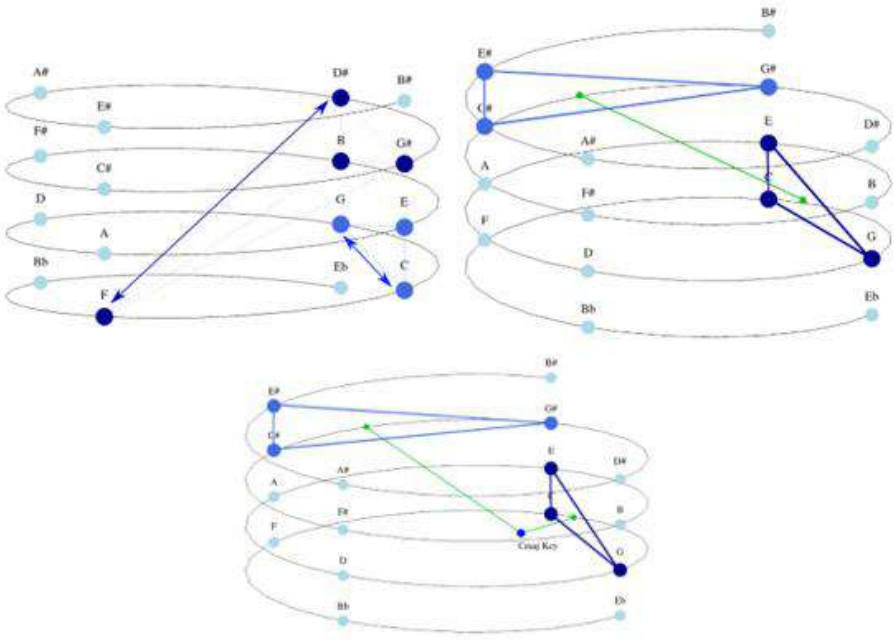

Figure 8 - (top-left) Cloud diameter of a C major chord (top-right) Cloud momentum ıord (bottom) Tensile strain of a C major and C# (small) versus the Tristan chord (large to a C# major chord [10]

Tension in musical pieces is a property that varies over time. Therefore, in order to calculate the tonal tension of a musical fragment with a sliding window approach, the musical piece is divided into equal length segments, that can be mapped to clouds of points in the spiral array. a musical piece is divided into equal length windows.

Based on these clouds, three measures of tonal tension can be computed:

- **Cloud diameter** captures the diameter of the cloud, that is, measures the dispersion of clusters of notes in tonal space. By looking at the largest Euclidean distance within the cloud we can basically identify the largest tonal distance in the spiral array model. When a chord or a cloud of notes contains intervals that are tonally far apart (i.e. dissonant), the distance between these pitches in the spiral array will be large.
- **Cloud momentum** reflects the movement in tonal space between two consecutive clouds of notes. The idea of cloud momentum is to quantify how large

the distance between the CEs of two clouds of points is, thus capturing the movement in tonality.

- **Tensile strain** captures the tonal distance between the CE of a cloud and the position of the CE of the global key in the spiral array[1]. This needs the implementing the key detection algorithm described in [7].

These three methods for quantifying aspects of tension are developed based on the spiral array model. From [10] an illustration of the three tension measures in the pitch class helix of the spiral array is illustrated in Figure 7.

The implemented system is able to display tension ribbons over the input musical scores, thus allowing for easy interpretation [9]. An analysis of existing pieces and a comparison with the Farbood empirical study [1] revealed that cloud diameter, cloud movement and tensile strain all contribute to capturing the composite feature humans refer to as.

2 Main code and technical implementation

2.1 How to handle MIDI data

MIDI essentially consists of sequences of serial instructions called "MIDI messages" corresponding to an event or change in a control parameter. A MIDI message has one or more bytes of data, categorized into two types: status bytes and data bytes. The latter always starts with a 1 bit, which means that each data byte has 7 bits free to specify values, with a resultant range of between 0 and 127. MIDI allows the use of up to sixteen different "channels" (numbered from 1 to 16) on which different types of messages can be sent and each channel is independent from the others.

There are two important classes of MIDI messages: "channel messages" and "system messages." The latter affect the MIDI system as a whole while the former influence the behavior of only a single channel and are the only type of messages that are relevant to this work. The two types of channel voice messages we are interested to in this work are the "Note On" and the "Control Change" one. A Note On messages instructs a synthesizer to begin playing a note that will keep on playing until a Note Off message is received corresponding to it. Note On messages have two data bytes. The first specifies pitch and the second specifies velocity: both of the byte ranges from 0 to 127. Pitch is numbered in semitone increments, with note 60 being designated as middle C (equal temperament tuning considered by default). Basically, Control Change messages affect the sound of notes that are played on a specified channel.

The Note On message is the one we intercept from the stream of data to compute the geometrical entities on the chunk processed by the Juce framework up to that point. On the other hand, a Control Change message is sent on a particular channel

[1] It essentially measures the distance between the local and global tonal context

to the input of the light stage automation software MadMapper[2] as shown in Figure 10 to control the brightness of lights on stage. In particular, a Control Change message is sent on channel 8 with an integer value for the velocity given by the harmonic tension value(s).

2.2 Implementation strategy and application in the context of Spiral array model

As far as the definition of CE and specific implementation of the CEG algorithm in the scenario of MIDI audio are concerned, refer to the end of 1.2.2 and 3.1.1 paragraphs. The core of the code is shown hereafter:

```
if (m.isNoteOn())
        {
        if (!bypassState) {
          midiNoteVal = labelNote(m.getMidiNoteName(m.getNoteNumber(), true, f
alse, 4), CE[0], CE[1], CE[2]);
          if (playposinfo.timeInSeconds - old_time > 0.05) {
            ++countMeasures;
            actualTime = playposinfo.timeInSeconds;

            if (countMeasures == 2) {
              c[0] = computeCE(playedNotes, noteTimes, countMeasures - 1, 0);
              c[1] = computeCE(playedNotes, noteTimes, countMeasures - 1, 1);
              c[2] = computeCE(playedNotes, noteTimes, countMeasures - 1, 2);

              CE_prev[0] = CE[0];
              CE_prev[1] = CE[1];
              CE_prev[2] = CE[2];

              CE[0] = alfa * c[0] + (1 - alfa) * CE_prev[0];
              CE[1] = alfa * c[1] + (1 - alfa) * CE_prev[1];
              CE[2] = alfa * c[2] + (1 - alfa) * CE_prev[2];

              cloudMomentum_prev = cloudMomentum;
              cloudMomentum = getDistance(CE, CE_prev);
```

From Note On messages, we calculate the CE after storing information about notes and their timestamps. This information is also important to handle the incoming of chords instead of single notes. The countMeasures represents the number of time chunks considered for the CE calculation. Also, the cloud momentum metric is computed according to its definition. The next snippet of code refers to the calculation of tensile strain, once the key is selected from the GUI:

```
if (selectedKey != 0 && selectedKey != 1) {
                if (btnState2) {
                  actualKey[0] = T2[0][selectedKey];
```

[2] https://madmapper.com/

```
        actualKey[1] = T2[1][selectedKey];
        actualKey[2] = T2[2][selectedKey];
        selectedTon = spiralArray[selectedKey + 4] + "m";
      }
      else {
        actualKey[0] = T1[0][selectedKey];
        actualKey[1] = T1[1][selectedKey];
        actualKey[2] = T1[2][selectedKey];
        selectedTon = spiralArray[selectedKey + 4];
      }
      tensileStrain = getDistance(actualKey, CE);
    }
    else {
      tensileStrain = 0.0;
      selectedTon = "not selected";
    }
```

Cloud diameter calculation is straightforward:

```
cloudDiameter_prev = cloudDiameter;
            cloudDiameter = getMaxDistance(playedNotes, old_time, noteTimes);
```

The last excerpt describes the averaging of the three tension metric extracted:

```
valueToSend = int(zeta*(100.0 + 6*cloudDiameter + 150*harmTension) + (1-zeta)*
(100.0 + 6*cloudDiameter_prev  + 150*harmTension_prev));
    if (valueToSend > 127)
      valueToSend = 127;

    auto messageDimmer = MidiMessage::controllerEvent(8, 0, uint8(valueToSend)
);
    controllerNumber = messageDimmer.getDescription();
    addMessageToBuffer(messageDimmer, midiBuf);
```

Then, this value is sent as a velocity parameter of a Control Change message. Figure 9 shows a screenshot of the vst3 running.

2.3 Choice of the parameters for the model

The parameters appearing in the model can be selected to reflect the cognitive interval relations and can be used to calibrate the model. A detailed discussion can be found in Appendix A of [7] where model constraints such as additional constraints on the aspect ratio $\frac{h}{r}$ and the parameter solution satisfying a set of principles of tonal cognition are highlighted.

The pitch weights for major chords and the chord weights for the major and minor keys are $[0.6025, 0.2121, 0.1145]$; and the pitch weights for minor chords are $[0.6011, 0.2121, 0.1868]$. This particular assignment assures that a half step interval should be interpreted as the leading tone (degree 7) and tonic (degree 1) of the major key and that the lower and upper pitches of a perfect fourth interval should be interpreted

as the dominant (5) and tonic (1) of the major key. Moreover, these weights satisfy perceived interval relations, such as, pitches related by a perfect fifths distance being closer than those a major thirds apart. In addition, these assignments also satisfy other two conditions: two pitches an interval of a half-step apart generates a center that is closest to the key of the upper pitch; and, the coordinates of a pitch class closest to the key representation of the same name. Basically, all the choice of weights results in a major chord being closest to its root, followed by its fifth, then its third; and, a minor chord being closest to its root, followed by its fifth, then its third.

Figure 9 – Example: Beethoven Sonata Op. 31 No. 3 in Eb major

The choice for the parameter $h = \sqrt{\frac{2}{15}}$ $r = 1$ is compliant with the fact that the closest distance between any two pitch positions must denote a perfect fifth, pitches a third apart are closer than those a second apart, a major interval is closer than a minor; and a tritone is more distant than any of the above. More generally, the aspect ratio h/r can be constrained so that $\frac{2}{15} < \frac{h^2}{r^2} < \frac{2}{7}$.

The parameter α (the one in the calculation of CE_α) has been experimentally set to 0.08, while the ones used for dealing with minor tonalities are $\alpha = 1$, $\beta = 0$.

2.4 Interfacing with real stage

We use virtual MIDI ports to interface the vst3 with a MIDI track and to let the vst3 communicate with a software, external to the DAW, in order to send the MIDI message related to light control. MIDI messagges are sent using loopMIDI[3] (for Windows functionality), a software for emulating MIDI ports.

Figure 10 - MadMapper for interfacing with led on stage

Figure 11 – The wooden stage structure before the beginning of the event

[3] https://www.tobias-erichsen.de/software/loopmidi.html

Figure 12 – Festivalle stage by night

The practical application of the plugin is shown in the above pictures.

3 Limit of the model and future work

The central point of tonality is that the decisive factor in the tonal effect is the functional association with the tonic chord (emphasized by functional theory), not the mere link with a scale of notes, that is, adherence to the pitch set of a major or minor scale. Chords, depending on their relationships, can generate a key. At the same time, a key gives rise to certain chord relationships with specific functions within the key. The chords that define the key are given names, with respect to the key that reflects their function (tonic, dominant and subdominant). As stated in [9] the proposed three measures relate to perceived distance between notes in a cluster, between consecutive clusters of notes, and between the global and local tonal contexts. They fail, however, to consider tension that is caused by other kinds of expectation, such as that due to delay of cadential closure (as mentioned in the beginning), modeling this kind of tension is trickier because not all dominant-tonic pairs form cadence.

In the context of tonal music there are three types of minor modes: the natural minor, harmonic minor, and melodic minor. With this reasoning, the parameters α and β in the model represents the relative weight of the major versus the minor dominant chord and the relative weight assigned the minor versus the major subdominant chord. It would be interesting framing the problem of harmonic minor and melodic minor tonalities (and relative mapping to the Spiral Array) in the context of secondary dominants thus extending the model also to other “tonal” keys.

3.1.1 Implementation of the CEG algorithm

The idea of CE is crucial to the Spiral Array model and its associated algorithms. For example, the Center of Effect Generator (CEG) key-finding algorithm maps an input music stream to its pitch representations, with each note weighted by its duration to get a CE. Since keys are also defined as points in space, it is then simple to compute the distance between the CE and the key, and nearby keys, to

determine which key is closest to the CE. So, the key is identified by searching for the key representation nearest to the CE.

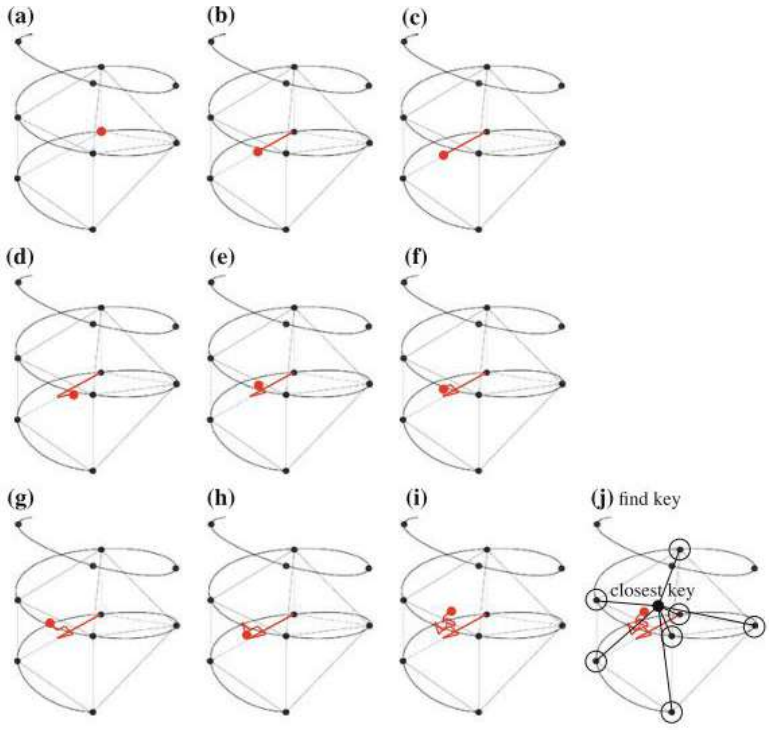

Figure 13 – Illustration of the CEG algorithm [7]

The CEG algorithm illustrated above is with a simple sequence of notes but simply generalizes to more complex music with simultaneous tones at any given time.

At any point in time, the CEG method can generate a CE, according to its mathematical definition in 1.2.2 that summarizes the tonal space generated by the pitches sounded thus far that serve as cues to the tonal space for that melody. Then, given this melody, the CEG algorithm successively generates CEs as each note event occurs, thus updating itself as it walks around the right key representation. As the number of the pitches increases, the algorithm reduces the pitch information down to a single point, the center of effect. The distance between CE and a key can both increase and decrease giving rise to a random oscillator behavior around it.

The CEG method can be applied to polyphonic music in MIDI using the Spiral Array model. The key defines the context of the music and can be detected most stably using long-term information. Hence, in the context of MIDI data, the the nearest-neighbor search for the closest key representation can be adapted to the long-term $CE_\alpha\ t$ defined in 1.2.4:

$$\mathbf{T}^* = \arg\min_{\mathrm{T}} \| \mathrm{T} - CE_\alpha\ t-1\ \|$$

where T is any major or minor key representation.

In the real-time spelling and key detection of MIDI captured from live performance the chunk size should be set to some unit of time. In this way, time would increase linearly, while, if we resort to the NoteOn message as the triggering event for the plugin reading in the input buffer of Juce Framework processor, we could be in a wrong temporal line. In other words, the time t that appears in the CE formula coincides with the NoteOn timestamp.

This, from an implementation point of view, would necessitate to handle another absolute temporal line in the whole structure of the code that is also able to take into account the pausing of the plugin with respect to the underlying piece. In fact, notes per sé are not equally distributed in time so, when we implement the CE low-pass filtering in time domain formula the code actually accounts simply for how many notes arrive in the input buffer with different NoteOn timestamps. The times t and $t-1$ in the formula coincide now with the filling and clearing time of the buffer of the processor while now the $c_{t-1,t}$ is the CE calculated on how many notes (not evenly distributed in time) are in the input buffer in that instant of time.

The implementation problem now is all about the aforementioned triggering the algorithm; we could have implemented between $t-1$ and t with an accuracy of 10^{-3}s, if we assume the chunk being 1s long. When the exact ms of time we have the first NoteOn event, we can calculate the summation outside the $\mathbf{c}_{a,b}$ so that the first NoteOn event is a and the second is b. The $c_{t-1,t}$ must account for pauses in between $t-1$ and t in the filtering process. To calculate $c_{t-1,t}$ within that interval the a is the only one that contains the information as notes are sounded only in that interval. We imagine that a solution for accounting also for the silence parts.

We suppose that this discrepancy in the evaluation of CE_α matters only in the key finding problem and not in the pitch spelling one that is seen to perform accurate in the present implementation. Probably this is due to the fact that, while pitch classes dwell in the outermost spiral, spatial representation of keys lives in the innermost spiral where distances among one element and another are smaller. For this reason, calculation of the Euclidean distance between CE_α and that particular point suffer from this lack of accuracy that becomes not negligible. Mathematically speaking, we think that this inconsistency is in the fact that, in the calculation of CE_α the term $c_{t-1,t}$ is used as a different thing with respect to $\mathbf{c}_{a,b}$. Moreover, we noticed that in key-detecting problem through the CEG algorithm the "hidden" error is systematic and after a very long time the correct information about the key is retrieved anyway.

For all the aforementioned reasons up to here, the tonality information used here for the calculation of the tensile strain metric can be put manually from the user in the graphic interface of the plugin.

3.1.2 Implementation of a graphical representation of the musical tension flow in tonal music using a piecewise parametric curve

Based on harmonic tension and chord tension, generally twenty four different chords belonging to the three chord families (tonic < subdominant < dominant) can be sorted. These families are known to have different degrees of tension with inherently different tension values.

This measuring of the tension in terms of a specific key can be exploited by interpolating the series of numerical tension values to construct a tension curve over time with a curve interpolation algorithm (e.g. B-spline can be one of the most suitable method to model a complex shape such as tension flow) as a visual tool that can be effectively used in several interesting applications: enhancing or weakening the overall feeling of tension in a whole song, the local control of tension in a specific region of music, the progressive transition of tension flow from source to target chord progressions, and natural connection of two songs with maintaining the smoothness of the tension flow [11]. The overall tension in original music can be enhanced by shifting the tension curve, however, melodic tension, that is, relation between melody and chord, must be taken into account otherwise new generated chord progression could not well match a given melody. The re-harmonization of the original chord progression can be done by geometrically editing the tension curve generated. This would give the possibility of controlling the perceptual factor (tension) in music by using numerical methods.

Real time modification of tension, with no expensive real time computation, could be applied in the context of interactive scenarios such as intelligent lighting engineering. A further investigation about real time implementation and effectiveness in these interactive scenarios could be interesting. Furthermore, this musical tension curve can be embedded, as a graphic tool, in the upper part of the vst3 plugin to show in real time the musical tension trend simultaneously with the music playing.

3.1.3 MorpheuS algorithm

If we think about situations in which music matching a specific narrative, and thus, emotion perceived is desired, it would be great to investigate how to exploit a generated and predefined tension profile of a piece and how to generate music having that particular tension profile.

An interesting research in this direction is the developing of the "MorpheuS" algorithm [10] where the target tension can be specified by the user, or calculated from a template piece using a computational model developed by the authors [9]. The purpose here is not to provide a formal description of the algorithm but only to provide the general aspects about how the algorithm works in order to motivate a future work for an efficient implementation and use of the algorithm in the context of live composition and stages. For further details the reader is referred to [10].

MorpheuS uses these three (weighted) tension characteristics described in 1.3 to evaluate the musical output and match it to given template tension profiles. Most of the existing music generation systems often lack long-term structure; this algorithm's novel

framework can generate polyphonic pieces with a given tension profile and long and short-term repeated pattern structures. Polyphonic music generation is dealt as an optimization problem which imposes the way the patterns repeat in the generated piece using hard constraints: an efficient optimization generates music by assigning pitches that best fit the prescribed tension profile to the template rhythm while hard constraining long-term structure through the detected patterns. Long term structure is that which generates coherence over larger time scales than a single note-to-note level.

The novelty consists in the inclusion of an original tension model and the integration of a state-of-the-art pattern detection algorithm which is used to find recurring patterns and themes in the template piece acting as hard-constraints during the generation process. The objective function of the optimization problem is based on the mathematical model for tonal tension. For example, structural elements such as these detected patterns consist of groups of notes that can recur transposed in different places throughout the piece. The aforementioned spiral array model described above, can be implemented in the MorpheuS system to quantify tension in an optimization context. In this perspective, the tension model in MorpheuS uses only the pitch class and the major and minor key helices.

References

[1] M. Farbood, "A Parametric, Temporal Model of Musical Tension," pp. 387–428, 2012.

[2] J. Mulholland and T. Hojnacki, *The Berklee book of Jazz Harmony*. Berklee College of Music, 2013.

[3] F. Lerdahl, "Tonal Pitch Space," *Music Percept.*, vol. 5, no. 3, pp. 315–349, 1988.

[4] F. Lerdahl and C. L. Krumhansl, "Modeling tonal tension," *Music Percept.*, vol. 24, no. 4, pp. 329–366, 2007.

[5] L. Balsach, *The fundations of harmonic tensions (The fundamentals of harmonic tensions)*. 2016.

[6] R. Cohn, "Neo-Riemannian Operations , Parsimonious Trichords , and Their " Tonnetz " Representations," *J. Music Theory*, vol. 41, no. 1, pp. 1–66, 1997.

[7] E. Chew, *Mathematical and Computational Modeling of Tonality*, Volume 204., vol. 204. Centre for Digital Music Queen Mary University of London, London, UK: Springer, 2014.

[8] E. Chew, "The spiral array: An algorithm for determining key boundaries," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2445, pp. 18–31, 2002.

[9] D. Herremans and E. Chew, "Tension ribbons: Quantifying and visualising tonal tension," *Proc. TENOR, Cambridge*, no. June, 2016.

[10] D. Herremans and E. Chew, "MorpheuS: generating structured music with constrained patterns and tension," *IEEE Trans. Affect. Comput.*, no. 99, 2017.

[11] M. J. Yoo and I. K. Lee, "Musical tension curves and its applications," *Int. Comput. Music Conf. ICMC 2006*, pp. 482–486, 2006.