Towards Music-Driven Procedural Animation

Carlos Brito
Departamento de Informática
Universidade do Minho, Portugal
cfabrito@gmail.com

António Ramires Fernandes
Centro Algoritmi
Universidade do Minho, Portugal
arf@di.uminho.pt

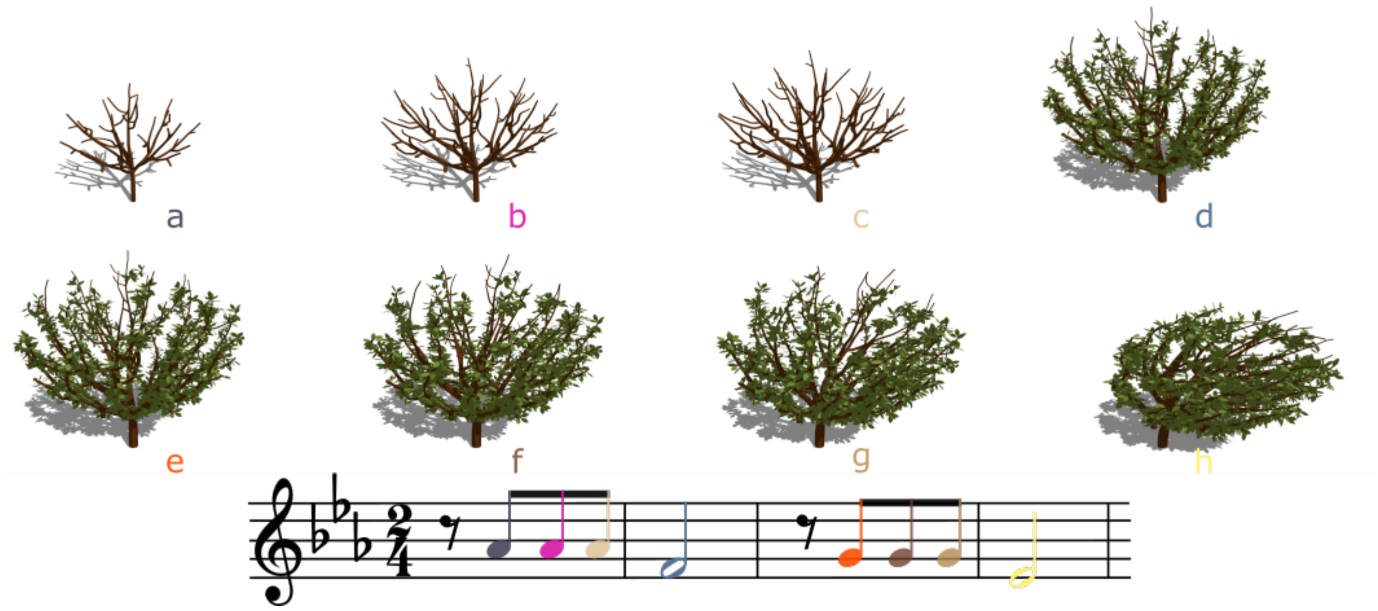

Figure 1: (a) Initial tree; (b, c) Tree grows and thickens with every note; (d, e) Tree develops leaves upon reaching the half-note; (f, g, h) Each note increases the wind intensity.

***Abstract*—We present our approach towards the development of a framework for the creation of music-driven procedural animations. We intend to explore the potential that elementary musical features hold for driving engaging audio-visual animations. To do so, we bring forward an integrated environment where real-time musical information is available and may be flexibly used for manipulating different aspects of a dynamic animation.**

In general terms, our approach consists of developing a virtual scene, populated by controllable entities, termed *actors*, and using scripting to define how these actors' behaviour or appearance change in response to musical information. Scripting operates by establishing associations, or mappings, between musical events, such as the ringing of notes or chords, or sound information, such as the frequency spectrum, and changes in the animation.

The scenario we chose to explore is comprised of two main actors: trees and wind. Trees grow in an iterative process, and may develop leaves, while swaying in response to the wind field. The wind is represented as a vector field whose configuration and strength can be altered in real-time. Scripting then allows for synchronising these changes with musical events, providing a natural sense of harmony with the accompanying music.

By having real-time access to musical information, as well as control over a reactive animation we believe to have taken a first step towards exploring a novel interdisciplinary concept with vast expressive potential.

***Keywords*—audio-visual, procedural animation, tree generation, physical simulation**

I. Introduction

The creation of audio-visual compositions is an interdisciplinary pursuit which joins together art, science and engineering. The interplay between vision and music not only provides a vibrant means of artistic expression but may also be used for aiding in the understanding of music. A significant amount of academic research takes the latter approach and often leads to frameworks that display various theoretical aspects of music in graphical form (e.g. [1], [2], [3], [4]). Other software exists that strives for a purely aesthetic visual accompaniment of music which rely on information obtained from the audio file. These are usually referred to as music visualiser software. Two of the most prominent music visualisers are Milkdrop2 [5] and G-force [6]. These two applications produce abstract psychedelic imagery which often illustrates musical aspects since the underlying generation algorithms use waveform and spectral information obtained from the audio signal. Music, however, contains far richer structural information than what may be readily extracted from its corresponding audio representation, namely information found in sheet music. This higher level information is accessible in symbolic representations such as MIDI [7] and MusicXML[8] and comprises, for instance, note onset times, rests, and time and key signatures. A prominent example of using this information for animation is the work of the Animusic Company [9]. These two contrasting approaches are representative of using features from distinct representations of music. On one hand, visualiser software uses low-level audio features, such as the frequency spectrum. On the other Animusic uses low-level musical features (which can arguably be regarded as high-level audio features). Each of these representations reflects certain aspects of a musical object, but no single representation encompasses all its properties [10].

We hypothesise that due to music being the result of artistic expression, often regarded as the language of emotion [11], the creation of meaningful visual accompaniment should not be built exclusively by algorithmic processes. Instead, we believe that algorithms can provide configurable and reactive building blocks from which complex animations can be built. Additionally, research into human perception has found statistical evidence to indicate that there exist intuitive auditory-visual mappings common among the general population [12].

We present a framework for procedurally generating three-dimensional animations strongly choreographed to music. Our framework offers a semi-automatic approach that allows for building animations by specifying how particular events in music can influence or determine the behaviour of *actors* in a virtual world. The example we selected to animate is comprised of two *actors*: a growing tree and a controllable wind field. The wind field directs the motion of tree branches, which allows for an intuitive control over their motion.

This approach can be regarded as a trade-off between control and automation. In this instance, a procedural algorithm is responsible for the synthesis of tree geometry, a physical simulation allows for its realistic response to wind, and other continuous changes are modelled through interpolation.

Each animation is directed by a script that operates while having access to both audio and symbolic musical information and is responsible for defining the scene and establishing the choreography in the form of audio-visual mappings.

This method provides a layer of abstraction from manually specifying changes to animation elements. Instead, we define controllable actors and a set of commands that will modify their behaviour or appearance. When we synchronise these commands to perceptual cues in music, such as the ringing of notes or chords, we found it possible to achieve a natural sense of harmony, which can be built upon to produce complex visual accompaniments to music.

What we hope to demonstrate is the potential to unlock artistic possibilities and empower creativity by supplying a high-level interface for dealing with a real-time stream of musical information and by allowing it to be projected onto several aspects of an animation.

In our case study, we found this approach to be successful. Although music structure is thoroughly described by a formal theory, most of its fundamental aspects are perceived in an intuitive way, which allows for them to be used in order to derive an animation while simultaneously emphasising the accompanying music.

II. Background and Related Work

Establishing connections between sound and vision has been a subject of active exploration with origins dating back to ancient Greece [13][14][15]. Sir Isaac Newton and others have proposed different connections of sound and vision through the wave properties of colour in light and pitch in music [16]. Arguably the first efforts to use technology to aid this form of expression came with a series of experimental devices from the eighteenth through the twentieth centuries generally referred to as "colour organs". These instruments, modelled on both the piano and the pipe organ, were built to produce flashes of differently coloured light simultaneously with notes played [17]. Due to the vast and interdisciplinary nature of this subject, we focus this section exclusively on technological approaches that allow for the creation of visual effects driven by musical information.

The second half of the twentieth century was marked by the rapid development of electronic devices, such as video synthesizers, some of which were designed with audio-visual exploration in mind [18]. During the 1970s one of the earliest examples of music visualisers were developed, often referred to as "light organs". These devices operated by converting audio signals into rhythmic lighting effects. Most of these operated by decomposing the audio signal into several frequency bands, whose intensity then regulates the intensity of differently coloured lights. This decade was also the stage for the emergence of laser light shows, another example of joining vision to music [19].

The software music visualisation age began in the mid-1990s. From then on, several music players began incorporating visualiser software and the concept became widespread. In this time it became common to use low-level audio features, such as spectral features to produce abstract imagery [20]. Possibly the earliest visualiser using this approach was Cthugha, initially released in 1993 [21]. Milkdrop2 and G-force, two prominent visualisers, were both initially released in 2001 and employ similar methods, having later been adopted as the default visualisers for the WinAmp and iTunes media players respectively [5]. Milkdrop2 has since then been open sourced and re-implemented in OpenGL[22] and is able to generate a myriad of fluid psychedelic effects. It operates with preset files, which consist of code to describe how the imagery evolves over time and how it responds to the input audio signal. These files can describe effects ranging from simple particles to intricate fluid and fractal-like shapes and terrain. Due to its versatility, Milkdrop2 remains popular to this day with a dedicated community who continues to re-invent presets, exploring its endless possibilities [5].

Approaches which use only audio information have been largely successful and remain popular among current music visualiser applications. Beat detection algorithms can be implemented using this information and often provide accurate results. However, the audio-visual connection is limited by the fact that only low-level features of sound are used, which do not always bear a direct relationship to perceptual experiences [12]. Despite the importance of music, music processing is still a relatively young discipline [10].

One of the earliest academic efforts of using musical features for the generation of imagery is presented in [23], where the authors developed software for generating images of notes, chords, and chord progressions, based on a set of artist-defined associations between shape and colours, and pitches of musical scales. In [24], MIDI information and forward kinematics were used to automatically animate a virtual human drummer. It was found necessary to allow for manual customisation of the generated motions, to better match the emotional aspects of the music played. [25] presents an approach to edit motion curves with cues from musical information based on using preexisting motions, created by an animator or resulting from motion capture, which are then algorithmically emphasised according to extracted musical cues, such as detected beats, chord changes or variations in loudness. [26] describes a system built to enhance the experience of live musical performances by having a virtual character react in real-time to the music being played. This was constructed upon the Animus system described in [27]. [28] defines an approach for synthesising dance motion matched to input music, based on approximating the emotional aspects of dance performance. A database of motion captured dance movements is used and processed. After segmentation of both the audio signal and the available motions, a sophisticated matching is made between the two, assuring connectivity and producing remarkable results. In [29], the *Soma* platform is described. This system uses three inter-operating applications in order to produce visual imagery from music. A live input application gathers and processes audio and control data from multiple instruments, from which several features are extracted and transmitted. These features are received by a second application which configures the mapping of audio to visual attributes. The results of the mapping processes are then sent to the final application which synthesises visual imagery using simultaneous Processing [30] *sketches*. The authors in [31] have developed a system for audio-visual mapping for the accompaniment of live orchestral performances. A score-following interface was implemented to account for tempo deviations during performance, allowing as well for manual adjustments in real-time by the conductor. The final projected visualisation was generated by Processing [30].

The Animusic Company [9] produces musical short films where intricate virtual instruments are animated to appear to play themselves. This is made possible by the company's internal software, MIDImotion, which generates motion curves from MIDI data [25]. Ubiquitously mentioned in music visualisation literature is Stephen Malinkowski's *Music Animation Machine* [32]. His work consists in what he refers to as "animated scores", that is, real-time animations of a musical sheet synchronised with a live performance. His animations often resemble the piano-roll representation of music, having note transitions inspired by the musical piece in question and colours based on musical structure. A vast library of animations has been produced and is available on Stephen Malinkowski's YouTube channel [33]. Max/MSP [34] is an advanced tool for the creation of interactive multimedia productions. It employs the visual programming paradigm which allows for an intuitive manipulation of real-time data. In 2003 the package *Jitter* was added which provides real-time video and 3D graphics processing. It has since then been employed for the creation of visual music [35] and is popular among visual and performing artists [36]. The original developer Miller Puckette, has since then authored Pure Data, often regarded as the open-source counterpart of Max/MSP [37].

Our framework shares some goals with the Max/MSP family, particularly the creation of audio-visual compositions. However, Max is a general purpose development tool and the access it provides to 3D graphics is low-level, as it involves the direct manipulation of matrices and OpenGL objects. Our approach intends to abstract the user away from all implementation details and focus solely on the mappings between musical events and changes in the animation.

The reported approaches demonstrate successful results for their purposes, however, none offers a generalised and extensible framework for the creation of audio-visual animations. Furthermore, none explores the possibilities of using real-time procedural generation or physical simulation techniques to produce visual imagery.

III. Towards Procedural Music-Driven Animation

The problem we approach is allowing for the flexible creation of animations strongly choreographed to music. In order to make this possible, controllable algorithms for procedural generation are necessary, as well as input for musical information. To join these two parts a versatile strategy to create mappings is also required.

Since any predefined mapping of musical to visual attributes would be arbitrary, what we strive to accomplish is to design a framework which allows for the definition of any mapping within the established scope. For this, we found that the use of a scripting system provided not only flexibility but also the possibility to create further abstractions, which aid in specifying increasingly complex behaviours.

In order to bring the concept of script-based music-driven procedural animation to life, we have developed a proof-of-concept demonstration. We selected trees swaying in the wind as our scene for three main reasons. Firstly, the procedural generation of trees is a well-studied field, having had contributions from both computer graphics and botany [38]. Second, trees possess several mutable aspects, such as topology, growth rate, texture and colour, which can be used as a "canvas" for projecting different musical aspects and third, by modifying the wind field we are able to create a variety of swaying motions which resemble dancing.

Our framework allows for the real-time processing of the various aspects which will produce our animations namely audio and MIDI processing, tree growth and response to wind, script execution and rendering. The system can be conceptually divided into the host application, where the actors are implemented, and the *Lua* scripting environment, where the audio-visual mappings are described.

A. System Architecture

In this subsection, we present a broad overview of our framework's internal organisation as an introduction to the various challenges it tackles. In particular, we describe the main modules which constitute our system, as well as how we structured their interaction. Figure 2 offers a visual representation of the different stages of data processing. We distinguish between what we have implemented from planned future work with dashed lines, which also serves to illustrate how the system can be extended.

Our system was designed to accept various representations of the music being played, in particular, audio data and MIDI events. These representations are fundamentally different since an audio stream consists of a continuous sequence of numbers (*audio samples*) whereas a MIDI stream is a sequence of discrete events.

The results of the processing stages can also be divided into continuous and discrete data. Audio data represents a continuous signal, and, for instance, the application of the *Short Time Fourier Transform* also yields continuous numerical data, which can be used directly for audio reactivity.

We also define a set of events to represent higher-level musical features. Events can be extracted from either input, such as detected chords from MIDI or detected beats from audio. All this information is made available to our scripting environment which is the central connecting element of our framework.

The `Script Environment` has access to all available actors. As such, it allows to instantiate actors, configure how they interact and define how they react to incoming musical events. Once again, actors may react in a continuous or a discrete way. An example of a continuous change would be adjusting the size of an object based on the frequency spectrum's average. A discrete change would be to trigger branch growth at the start of a musical note. However, we still wish to produce smooth continuous motion from discrete changes. To this effect, we have used *Tweens* as a versatile interpolation strategy.

Regarding our actors, we have defined both `Tree` and `Wind`. We model trees as hierarchical structures composed of several connected segments, which are individually accessible from the script environment. Furthermore, it is possible to arbitrarily select subsets of the tree and manipulate them independently, which provides yet another degree of flexibility for the creation of visual effects, such as growing part of the tree or adding leaves selectively.

We represent wind as a velocity field built from various *flow primitives*, namely uniform, sink and source flow. The script environment is able to freely create wind fields by combining these primitives and adjusting their parameters in real-time, such as strength or position.

IV. Musical Input

Our system has access to sample data as music is played. From this data, we can extract audio features, such as the frequency spectrum which we obtain by using the *Short Time Fourier Transform* (STFT). The result of this transform can be interpreted as a histogram where each bin corresponds to a frequency interval and its corresponding height corresponds to those frequency's intensities over the sampling period. As previously mentioned, this is one of the most common features used in music visualisation.

We make this numeric information available to the scripting environment, which is then able to use it to manipulate any defined attribute. Although more sophisticated analysis techniques exist, we found this low-level feature to already produce interesting results, in particular when applied to subtle effects in the animation, such as scaling the tree skeleton or the leaves.

In audio representations high-level musical information such as note onset times, pitch or duration are not given explicitly. Extracting this information remains a difficult problem, still under active investigation [10]. We did not approach this problem and instead bypassed it by using musical symbolic formats, namely MIDI. By doing so we have direct access to higher level musical features that can be used directly in animations. For instance, it becomes trivial to implement chord detection with MIDI since individual notes are readily available. However, this does require that the MIDI information must closely match the audio representation.

Our music processing stage results in a new set of events. These events can originate directly from MIDI, such as the beginning and ending of notes, or they can be the result of pre-processing, such as the chord event.

In general, we found that events are very useful as triggers to noticeable changes in the animation, while continuous data is usually more usable by modulating numerical attributes such as sizes or wind strengths. We also found that the choice of interpolation strategy has a significant impact when attempting to translate musical aspects into visual form.

The ideal scenario for an animation to be generated is having both an audio file as well as a synchronised MIDI file describing every instrument. This combines the best of both worlds, as music analysis tasks can be performed on MIDI data whereas dynamics and timbre information is more accessible on the audio data.

V. Scripting

As mentioned in section III, an animation in our system is configured through scripting, establishing the scene and determining how the animation evolves as music progresses. The usage of scripting is both flexible and straightforward, since it allows for new actors to be added having only to

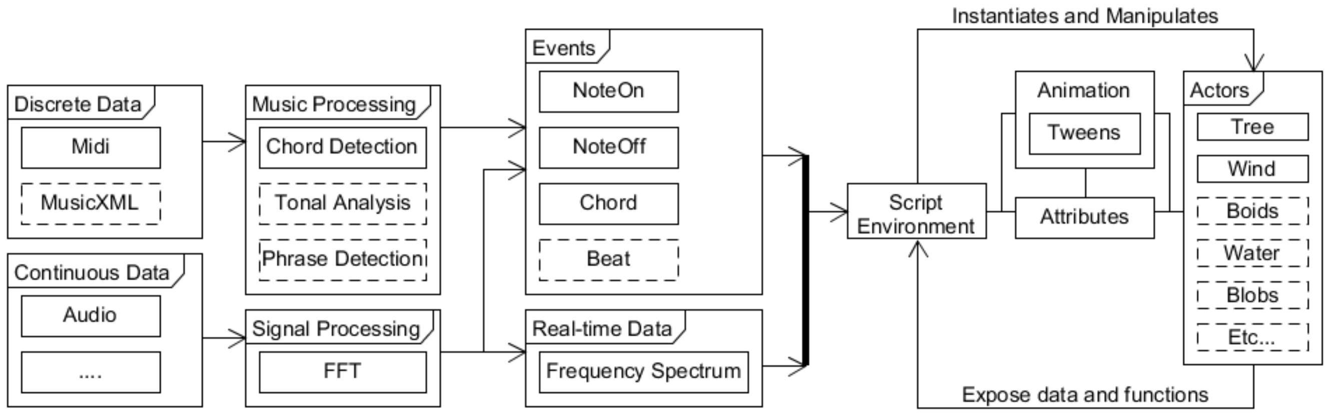

Figure 2: Overview of our system's architecture.

register their properties and commands in order to become usable.

The scene and choreography are defined on the script's initialisation function, which is where actors are instanced and configured.

A. *Handling events*

The approach we took to define event audio-visual mappings consists of having each script specify how particular musical events translate to a visual modification. The general form of the structure that stores these mappings is illustrated in listing 1.

```
handlers =
{
  [track_number] =
  {
    event_type =
    {
    {predicate, action}
    {predicate, action}
    ...
    }
    ...
  }
  ...
}
```

Listing 1: `handler` structure example.

The *predicate* and *action* function pair is the fundamental element in this structure. The *predicate* function evaluates the incoming event and determines whether the respective *action* function should be called, while the *action* function specifies the changes that happens within the animation in response to the event. For instance, the *predicate* function may determine that the *action* function will be called if a particular set of pitches is played, or if a note is played with a particular intensity. The *action* function may, for instance, trigger a growth iteration of a tree, or an increase in wind strength.

Additionally, each event has an index in the MIDI file and a duration, which we use for choreographing the various sections of the song. Part of the handler structure from the example in figure 1 is shown in listing 2.

```
handlers =
{
  [9]= { note_on = {
      -- Grow 2 iterations with notes 1 to 4
      { pred = note_range(1, 4),
        func = note_on_full_grow(tree,
               "easeOutQuad", 2) },
      -- Swell the trunk on every note
      { pred = note_range(1, 4),
        func = note_on_displacement(tree,
               "easeOutQuad", 2, 0.26) },

      -- Increase lateral wind strength
      { pred = note_range(5, 7),
        func = note_on_empower_wind(100) },

      ...
    }
    ...
  }
}
```

Listing 2: Part of the `handler` structure from the Beethoven's 5th example.

This particular excerpt of code specifies the mapping of the first seven `NoteOn` events in track of index 9, where the main melody is present, commanding the tree to grow and thicken its branches on the first four notes, and altering wind strength on the latest three.

B. *Handling Continuous Data*

In order to associate real-time information, such as frequency intensities, to arbitrary attributes, we defined a simple abstraction called *DataLink*. This structure is composed of an *extractor* function which is responsible for obtaining a value, and an *applicator* function which may transform that value and assign it to an attribute.

An example usage is that of adjusting the size of leaves according to the frequency spectrum's average. The code that achieves this can be found in listing 3. The `leaf_scale` is determined based on the output from the extractor function.

```
local pulseLink =
  DataLink.create("leaf_size_link",
    function ()
      return AudioAnalyser["spect_average"]
    end,
    function (v)
      tree:set("leaf_scale", [0.3 + v/80])
    end)
```

Listing 3: An example of a `DataLink`

Lastly, it's also possible to configure real-time actor behaviour, including their interaction, with each other, which can be achieved through an exchange of data structures. This can be implemented using a *DataLink* or by using the built-in `onUpdate` function, as seen in listing 4.

```
function onUpdate(dt)
  t = t + dt
  tree["vector_field"]:set(
        wind:get("vector_field"))
end
```

Listing 4: Computing tree motion from the wind's vector field.

VI. Tree Modelling and Simulation

Our system represents a tree model as a hierarchically organised structure. Each node on this structure is a possible branching point and its connections to both parent and child branching points establish small branch segments, often referred to as *internodes* in botany. Each chain of consecutive vertices represents a full branch.

Regarding tree generation, we adopted the *Space Colonization Algorithm* (SCA) as described in [39]. As the name implies, SCA operates by simulating the growth of a branching structure based on a description of available space.

The elegance of this algorithm lies in its simplicity, as available space is signalled by a set of points, referred to as *attraction points*. This name results from each point's behaviour of attracting the tree structure, promoting branch growth in its vicinity. Points are removed when approached by branch segments, which ensures no self-intersections occur in the final structure. Consequently, the point cloud effectively defines the tree crown's shape.

The algorithm takes a small number of input parameters, namely the segment length, the radius of influence, and the kill distance, as well as the set of attraction points. The radius of influence defines the size of the spherical region surrounding every attraction point which triggers the spawn of new nodes. Conversely, the kill distance defines the minimum distance between a tree node and an attraction point, that once reached causes the latter to be removed.

We found this algorithm to be particularly suitable for our purposes as it provides intuitive control over the shape of the generated trees. The input parameters also correspond to visually relevant characteristics, offering convenient control of branch shape and density. Furthermore, the tree is produced with a natural base-to-tips order which makes it adequate for animation.

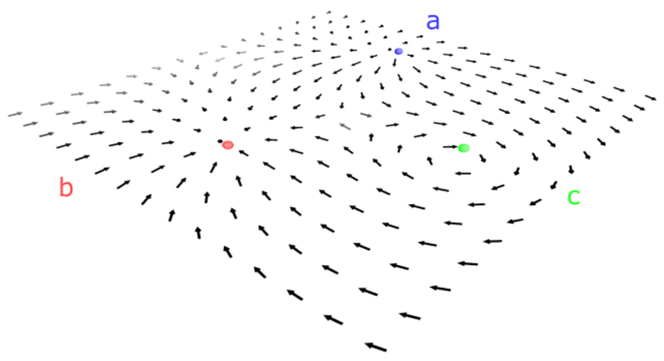

Figure 3: Three different flow primitives combined: (a) Source flow; (b) Sink flow; (c) Vortex flow.

Lastly, we compute the tree's reaction to the wind's vector field based on the method described in [40]. The method consists in simulating the movement of individual segments and then recursively combining the results. This is achieved by defining a set of forces which act simultaneously upon the tree, leading to a final equation which is continuously integrated in order to produce the resulting motion.

A. *Wind Field Construction*

Wind is the motion of air, and as such, it can be simulated as a moving fluid. Mathematically, the state of a fluid at a given instant of time can be modelled as a velocity vector field.

A wind field can be described by the Navier-Stokes equations which can be solved using a variety of algorithms. Recent work in tree-wind simulation such as [41] and [42] each use different approaches. In [41] the authors use Smoothed Particle Hydrodynamics, while [42] employ a discrete solver as described in [43]. Both these approaches offer some degree of control regarding the wind field. [41] allows for the arbitrary placement of "wind emitters", which are finite planes from where wind flow originates and [42] allows for control by exerting external forces upon the wind field.

In our system, we greatly simplify the original Navier-Stokes equations by using linearised fluid flow, as described in [44]. This simplification arises from the assumptions that the fluid is inviscid, irrotational and incompressible and allows for defining a small set of flow primitives which can be combined to produce complex flows. Each primitive is described by a linear equation which can be solved analytically and allows for their combination to be expressed through simple addition. By using a set of flow primitives, this approach allows for an intuitive construction of vector fields.

These primitives correspond to building blocks of flow: uniform (Figures 3a, 4a), which flows in a single direction; sink/source(Figures 3a,3b), which flows towards/away from a given point and lastly vortex (Figures 3c, 4b), which flows around a given axis in a circular trajectory.

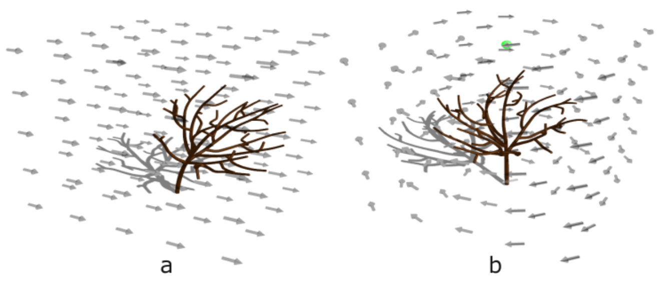

Figure 4: (a) Uniform Wind; (b) Vortex Wind.

All attributes that characterise wind primitives, for instance, strength (which is common to all primitives), can be modified in real-time from the script environment. This approach fulfils our requirements, as it provides sufficient movement credibility, while simultaneously allowing for a wide range of configurable animation effects, see Figure 5.

For instance, placing directional upwards wind leads to an animation of a tree with its branches closing, whereas the reverse direction will cause its branches to open. On the other hand, placing a vortex primitive with an axis along the main tree trunk will cause the tree branches to curl. A sequence where the vortex wind's axis is periodically inverted, for instance, will resemble a dancer swinging their arms as can be seen of Figure 5b.

VII. Discussion and Conclusions

In this project, we explored the challenge of creating a framework for deriving animations from musical information. In general, we found that this can be made possible by creating a set of configurable actors, a system that accounts for their interaction and finally, a set of rules which describe how their behaviour or appearance changes according to musical input. This concept can be summarised as a semi-automatic construction of visual tracks for music.

We found that imagining visual metaphors for music comes naturally to most people since casual discussions regarding this application often brought with them fresh ideas for possible associations between our tree's reaction to moments in songs. The flow of ideas seems particularly abundant from those with greater affinity for music, and even more so from those who study it. This was both an exciting and motivating result since in this project we have barely scratched the surface of what could be accomplished by extending this concept in different directions. For instance by extracting more features from audio and by modelling different phenomena. Some examples would be flock and swarm behaviour, fluids, kinematics, fractals.

Furthermore, in any context where there is abundant real-time data, this approach can lower the effort required to translate any such data into a meaningful visualisation, as such we believe this concept could extend beyond music. Any sort of data could be used as input instead, leading to many distinct usage possibilities.

Cinematographic and theatrical concepts and techniques should be considered. Incorporating even the most basic structural elements from these areas such as the notion of scenes and acts, as well as camera and lighting, could yield significant improvements regarding the expressiveness of the resulting animations.

In terms of usability, we became increasingly aware that scripting, although powerful, is not intuitive and has a steep learning curve. We believe that an adequate interface for our system would be the visual programming paradigm, as in Max/MSP and PureData.

By creating this proof-of-concept, we believe to have put forth a new way in which technology may assist in the creation of audiovisual artwork. To the best of our knowledge, the connection between musical performance and procedural/physically-based animation has not been previously explored. Many ideas remain delightfully open for future research, exploration and experimentation. We hope that this project will be able to incite further study and development.

We believe to have stumbled upon what seems to be a largely unexplored concept, a union between the seemingly ever-growing processing power of modern technology and human expressiveness. "Extraordinary possibilities remain untried, unknown, even barely imaginable"[45].

Acknowledgments

This work has been supported by national funds through FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2019.

References

[1] A. Graves, C. Hand, and A. Hugill, “Interactive visualisation of musical form using vrml,” in *Proc. Fourth UK VR-SIG Conference*, 1997, pp. 98–109.

[2] E. Chew and A. R. Francois, “Interactive multi-scale visualizations of tonal evolution in musa. rt opus 2,” *Computers in Entertainment (CIE)*, vol. 3, no. 4, pp. 1–16, 2005.

[3] T. Bergstrom, K. Karahalios, and J. C. Hart, “Isochords: visualizing structure in music,” in *Proceedings of Graphics Interface 2007*. ACM, 2007, pp. 297–304.

[4] G. D. Cantareira, L. G. Nonato, and F. V. Paulovich, “Moshviz: A detail overview approach to visualize music elements,” *IEEE Transactions on Multimedia*, vol. 18, p. 2238–2246, 2016.

[5] R. Geiss. Geisswerks - about milkdrop. [Online]. Available: http://www.geisswerks.com/milkdrop/

[6] A. O'Meara. Soundspectrum news - g-force concert visuals, updates, shows. [Online]. Available: http://www.soundspectrum.com/news.html

[7] A. Billias, “The midi association (tma).” [Online]. Available: https://www.midi.org/about

[8] “Musicxml for exchanging digital sheet music.” [Online]. Available: http://www.musicxml.com/

[9] “Software.” [Online]. Available: http://animusic.com/company/software.php

[10] M. Muller, *Fundamentals of music processing: audio, analysis, algorithms, applications*. Springer, 2015.

[11] P. N. Juslin, “What does music express? basic emotions and beyond,” *Frontiers in psychology*, vol. 4, p. 596, 2013.

[12] K. Giannakis, “A comparative evaluation of auditory-visual mappings for sound visualisation,” *Org. Sound*, vol. 11, no. 3, pp. 297–307, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1017/S1355771806001531

[13] J. L. Caivano, “Color and sound: Physical and psychophysical relations,” *Color Research & Application*, vol. 19, no. 2, pp. 126–133, 1994.

[14] W. Moritz, “The dream of color music, and machines that made it possible,” *Animation World Magazine*, vol. 2, no. 1, 1997.

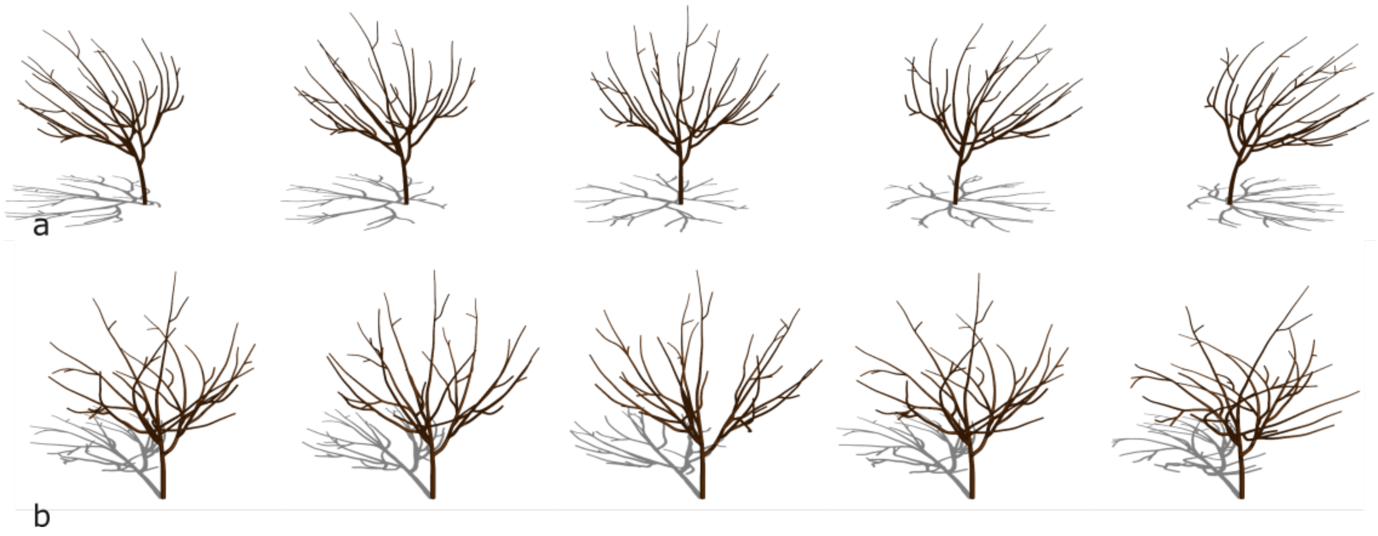

Figure 5: Frame sequence of tree reacting to (a) Uniform lateral wind and (b) Vortex wind.

[15] F. Collopy, “Color, form, and motion: Dimensions of a musical art of light,” *Leonardo*, vol. 33, no. 5, pp. 355–360, 2000.

[16] B. Alves, “Digital harmony of sound and light,” *Computer Music Journal*, vol. 29, no. 4, p. 45–54, 2005.

[17] M. Betancourt, “Making music with color,” Nov 2015. [Online]. Available: https://www.theatlantic.com/technology/archive/2015/11/color-organs/414460/

[18] N. Collins and J. d’Escriván, *The Cambridge companion to electronic music*. Cambridge University Press, 2017.

[19] P. Daukantas, “A short history of laser light shows,” *Optics and Photonics News*, vol. 21, no. 5, pp. 42–47, 2010.

[20] T. Kitahara, “Mid-level representations of musical audio signals for music information retrieval.” 2010.

[21] [Online]. Available: http://www.afn.org/~cthugha/

[22] C. Piccione, P. Sperl, A. Descartes, R. Dannenburg, M. Klumpp, and M. Spiegelmock. [Online]. Available: http://projectm-visualizer.github.io/projectm/

[23] J. B. Mitroo, N. Herman, and N. I. Badler, “Movies from music,” *ACM SIGGRAPH Computer Graphics*, vol. 13, no. 2, p. 218–225, Jan 1979.

[24] A. M. Wood-Gaines, “Modelling expressive movement of musicians,” Ph.D. dissertation, Applied Sciences: School of Computing Science, 1997.

[25] M. Cardle, L. Barthe, S. Brooks, and P. Robinson, “Music-driven motion editing: Local motion transformations guided by music analysis,” in *Eurographics UK Conference, 2002. Proceedings. The 20th*. IEEE, 2002, pp. 38–44.

[26] R. Taylor, D. Torres, and P. Boulanger, “Using music to interact with a virtual character,” in *Proceedings of the 2005 conference on New interfaces for musical expression*. National University of Singapore, 2005, pp. 220–223.

[27] D. Torres and P. Boulanger, “The animus project: a framework for the creation of interactive creatures in immersed environments,” in *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, 2003, pp. 91–99.

[28] T. Shiratori, A. Nakazawa, and K. Ikeuchi, “Dancing-to-music character animation,” in *Computer Graphics Forum*, vol. 25, no. 3. Wiley Online Library, 2006, pp. 449–458.

[29] I. Bergstrom, “Soma: live performance where congruent musical, visual, and proprioceptive stimuli fuse to form a combined aesthetic narrative,” Ph.D. dissertation, UCL (University College London), 2011.

[30] P. Foundation, “Processing.org.” [Online]. Available: https://processing.org/

[31] K. Ng, J. Armitage, and A. Mclean, “The colour of music: Real-time music visualisation with synaesthetic sound-colour mapping,” *Electronic Visualisation and the Arts (EVA 2014)*, Aug 2014.

[32] “Music Animation Machine.” [Online]. Available: http://www.musanim.com/

[33] smalin, “smalin.” [Online]. Available: https://www.youtube.com/channel/UC2zb5cQbLabj3U9l3tke1pg

[34] D. Zicarelli. Cycling ’74. [Online]. Available: https://cycling74.com/

[35] R. Jones and B. Nevile, “Creating visual music in jitter: Approaches and techniques,” *Computer Music Journal*, vol. 29, no. 4, pp. 55–70, 2005.

[36] F. Visnjic. (2016, Jan) Hexpixels – "c punks", a unit for realtime visual expression. [Online]. Available: http://www.creativeapplications.net/maxmsp/hexpixels-c-punks-a-unit-for-realtime-visual-expression/

[37] M. Puckette *et al.*, “Pure data: another integrated computer music environment,” *Proceedings of the second intercollege computer music concerts*, pp. 37–41, 1996.

[38] O. Deussen and B. Lintermann, *Digital design of nature: computer generated plants and organics*. Springer, 2005.

[39] A. Runions, B. Lane, and P. Prusinkiewicz, “Modeling trees with a space colonization algorithm,” in *Proceedings of the Third Eurographics Conference on Natural Phenomena*, ser. NPH’07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 63–70. [Online]. Available: http://dx.doi.org/10.2312/NPH/NPH07/063-070

[40] T. Sakaguchi and J. Ohya, “Modeling and animation of botanical trees for interactive virtual environments,” *Proceedings of the ACM symposium on Virtual reality software and technology - VRST ’99*, 1999.

[41] S. Pirk, T. Niese, T. Hädrich, B. Benes, and O. Deussen, “Windy trees,” *ACM Transactions on Graphics*, vol. 33, no. 6, p. 1–11, 2014.

[42] N. J. Oliapuram and S. Kumar, “Realtime forest animation in wind,” in *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*. ACM, 2010, pp. 197–204.

[43] J. Stam, “Stable fluids,” in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.

[44] J. Wejchert and D. Haumann, “Animation aerodynamics,” *SIGGRAPH Comput. Graph.*, vol. 25, no. 4, pp. 19–22, Jul. 1991. [Online]. Available: http://doi.acm.org/10.1145/127719.122719

[45] J. Whitney, “To paint on water: The audiovisual duet of complementarity,” *Computer Music Journal*, vol. 18, no. 3, pp. 45–52, 1994.