

# Chromatic and Harmonic Scales for Intuitive Sonographic Compositions

Vincent Lesbros

GAIV - La Ferrassie

lesbros@gmail.com

## Abstract

Historically, sonographic representations of sounds are based on two types of scales: linear or logarithmic (Xenakis, 1971; Roads, 1995; Lesbros, 1995, 1996; Xiaoying, 2008). These two kinds of scales introduce beats between sounds produced by adjacent lines of the image. In a linear scale, for example, of 1 Hz by pixel, a beat of 1 Hz will appear if any horizontal line has a thickness of more than 1 pixel. In a logarithmic scale, for example, of 192 ppo (pixel per octave), a beat of one 16<sup>th</sup> of semitone will spread. Here, we introduce a new kind of scale in order to use hand drawn sonographs, avoiding this particular beat effect. We extend the usual additive method to frequency modulation and add a way to spacialize sonographs in stereo channels. We propose different kinds of filters to control rhythm and harmony, and finally, we expose a way to compose music tightly linked with a video by creating sonographs directly from video frames.

## 1. introduction

If music and painting pull at your heartstrings, you may like to create a correspondence between the two, turning a simple brush stroke into a flight of lyricism, letting colour set the tone and the power behind the brush bring new vivacity to a piece of music. While so doing, you will want to know all the ins and outs, understand the underlying physical structure, the theory of light (QED: The Strange Theory of Light and Matter; Richard Feynman, 1985) and the fundamentals of sound synthesis. UPIC (Unité Polyagogique Informatique du CEMAMU), *Phonogramme* (Vincent Lesbros 1991) and *Sonographe* (Vincent Lesbros 2004) have paved the way. *Cyclonium* is the fruit of more advanced research, aiming toward a faithful reproduction of spontaneity and in-depth understanding.

This article focuses on three points developed in our new framework *Cyclonium*: frequency scales with regard to timbre, the geometry of rhythm associated with mathematical series and potential cohesion between musical composition and the temporal progression of animated images, such as those in videos or in films.

The standard use of *Sonographe* consists in the design of a two dimensional image, structured from left to right in time, and from bottom to top for the bass frequencies up to the treble ones. The image is thus translated into sound by a logical computer process: a formula in line with physics, taking place in a discrete world, where the real numbers are reduced to artificial and incomplete representations<sup>1</sup>.

The game consists in transmitting, in the best possible way, the author's creative energy during the rendering of the final result, by transforming the initial trace left on paper into speaker vibration.

The choice of language is paramount in what it is possible to express (1996, Claude Hagège, *L'homme de parole*). « *Composer c'est programmer, programmer c'est composer* » wrote Giuseppe Englert. (Giuseppe G. Englert, 1889).

*Cyclonium* builds a bridge between the efficiency of the C language and the flexibility of Smalltalk with a constant concern to help visualize the manipulated elements and processes of transformations and calculations. The latest version automatically generates the C code and the interface for the programs expressed by the user in the Smalltalk environment by drawing graphical schemes.

By composing a piece of music on paper with a brush, you will be able to obtain a comprehensive, heuristic and intuitive vision, with no holds barred as to your creative gesture. Translating the strokes, the colors and the textures, takes you closer to the physical structure of the image and sound. If you do not actually reach their physical structures, their IT ersatz is at least created.

*Cyclonium* offers a discrete space-time at the level of the sample in the sound domain and at pixel level in the graphical field, as well as special tools for linking visual and sound aspects, opening the door to any new creative metaphor significant in these spaces. Images can thus be translated into sounds through the programming activity. Moreover, it is also a way of creating first-hand what moves us, namely images and sounds.

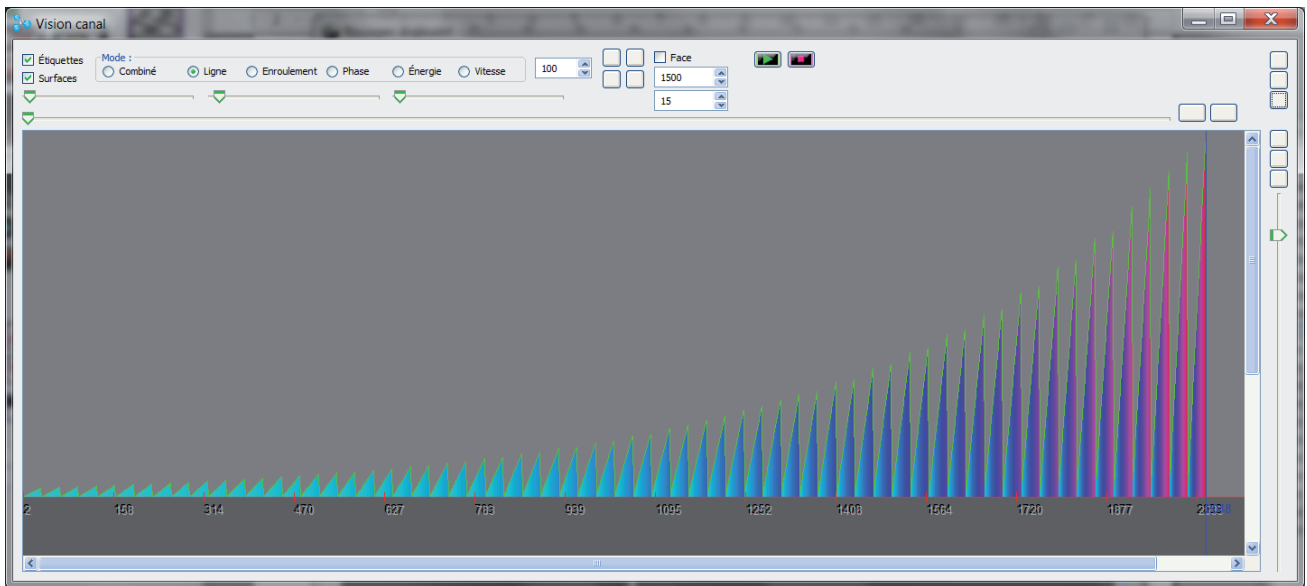
In conclusion, a simple rule in the programming space can give rise to complex phenomena, usable in the artistic effervescence, and yet to be described. *Cyclonium* is an exploratory tool to reveal these phenomena, raw material for creation.

## 2. Enhancing Sonographe frequency scales

The initial purpose of this research was to compose music for films and one of the aims was to be able to harmonically combine real instruments and artificially synthesized sounds. The synthesized sounds should then match the chromatic scale in semitones of classical instruments and natural or usual perception. So we first divide the height of the image in a logarithmic scale but inside each step of this scale, corresponding to a semitone, we introduce a harmonic sub-scale in order that two adjacent lines in a step will be a harmonic of the other:

```
[ :i | | h dt |  
  h := (i - 1) \\ 32.  
  dt := (i - 1) // 32.  
  (440 / 64) * (2 raisedTo: (dt / 12)) * (1 + h) ]
```

<sup>1</sup> See standard IEEE, (David Goldberg 1991). (Bruce Dawson 2014, Intel underestimates Error Bounds by 1.3 quintillion <https://randomascii.wordpress.com/2014/10/09/intel-underestimates-error-bounds-by-1-3-quintillion/>).



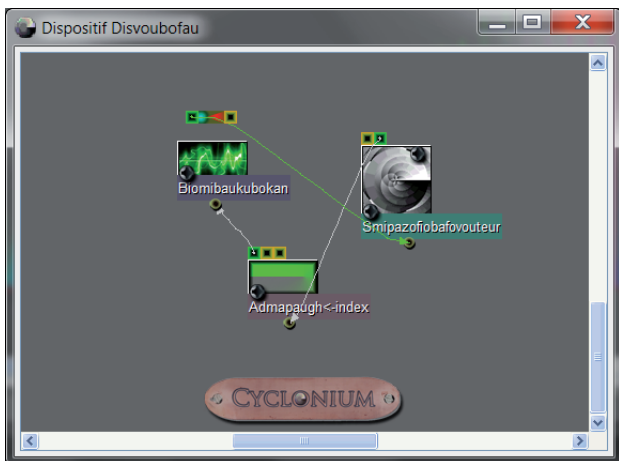
**Figure 1.** A chromatic and harmonic scale of frequencies.

The above code is the formula to compute a scale with 32 pixels by semitone with 32 harmonics by semitone. "i" has to be the index of the line of the image, ranging from 1 to the height of the image, "h" is the rank of harmonic, "dt" is the semitone number. The base line starts at 440.0 Hz / 2<sup>6</sup>, six octaves below the A 440.0 Hz. "\\" denotes modulo and "/" the integer division.

A graphic of the result is shown in figure 1.

Our framework called Cyclonium is developed in Smalltalk and C languages. The graphical interface is used to connect elements and blocs of Smalltalk code may be used to add any new function.

Figure 2 presents the graphical interface to construct the scale described above.



**Figure 2.** The graphical interface in Cyclonium to construct the scale: the element named *Biomibaukubokan* is a channel with one source, its source *Smipazofioabafovouteur* is a generic operator described by a Smalltalk block that takes in input the index of channel calculated by the *Admapaugh* element.

In this case, we choose semitones and harmonic scale, but any micro-tonal and inharmonic partials can be used as any random set of frequencies. The code module used to compute the sonograph is written in C language or, in the basic cases, uses the *OpenGL* library to delegate computing to the graphic card.

The user gives the image to be interpreted, a duration in milliseconds, and at least one frequency scale. By default a sine waveform is used for the additive synthesis.

If the user gives a second frequency scale, the system will interpolate frequencies between the first and the second along the whole time of the image for each line independently. This enables the user to describe frequency scales varying linearly from one scale to another along the x axis. A third option is provided to precisely control frequency of each dot of the image: the user gives a second image of the same size whose values are used to calculate the frequency of each point by using its value as a weight between the first and the second scale. This may be used, for example, to introduce vibrato or temperament variations locally.

In short, the interface of the Sonographe element in Cyclonium takes: an image, a duration, the first frequency scale, optionally a frequency control image, a second frequency scale, optionally a waveform (by default a sine wave), eventually a frequency modulation element (see below), and eventually an image used to spacialize the sound in a pair of stereo channels.

Figure 3 shows the interface for a Sonograph in Cyclonium<sup>2</sup>.

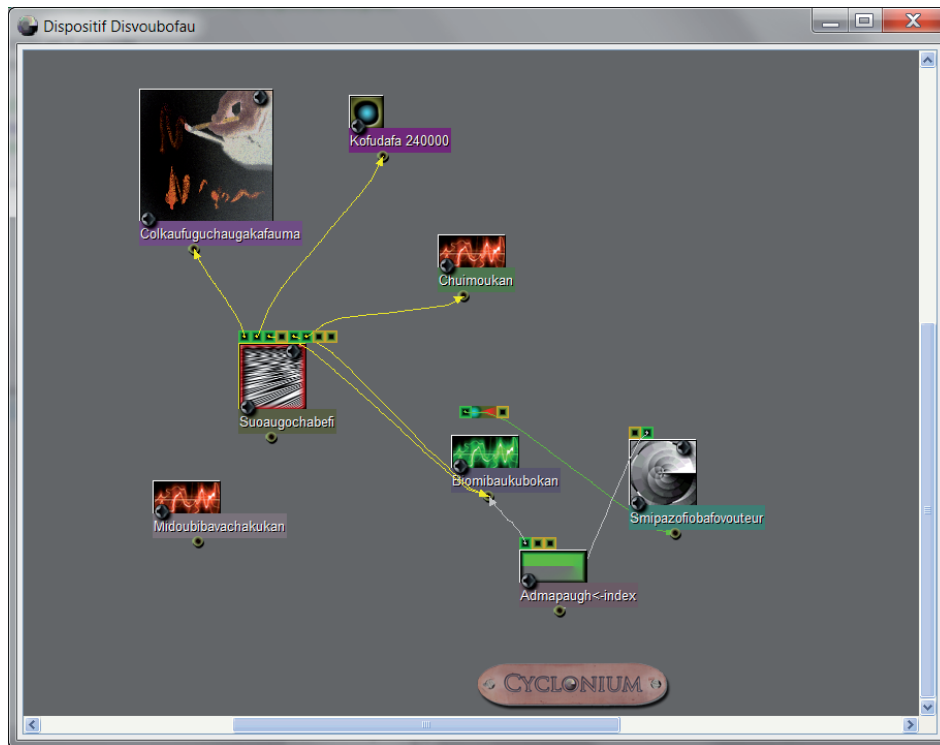
## 2.1 Benefits of the chromatic and harmonic scale

As we are drawing sonographs by hand, it is not easy to know how a drawing will sound. Brushstrokes are generally more than one pixel thick and with a classical scale they induce characteristic beats. In our case, strokes may cross semitone boundaries, but the thickness of the brush corresponds more to the harmonic richness than to a cluster of close frequencies.

Timbre in additive synthesis is controlled by partial amplitudes. As partials are regularly dispatched following a modulo rule it is easy to build an image that can be used as a filter by multiplying it with the source image in order to control amplitude of partials. For example, to generate a filter of 2336 pixels high, for 32 harmonic amplitudes linearly decreasing the following code block is connected to a *Dessinateur* element:

```
[ :p | 32 - ((2335 - p y) \ 32) ]
```

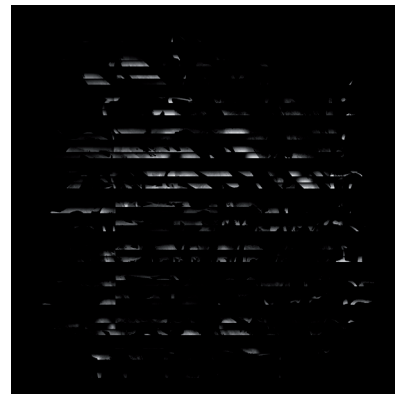
<sup>2</sup> The whole scheme may be saved in XML format for later use. Each element has a unique name automatically generated, but the user can rename and describe them as he wants. Hyperlinks to web pages may be included in full description of elements for their documentation.



**Figure 3.** *Colkaufuguchaugakafauma*: Imported colored image (bmp or jpeg). *Kofudafa*: Constant expressing the duration in milliseconds. *Chuimouka*: Automatically generated sine wave for the waveform. *Suoaugochabefi*: Sonographe element. *Biomibaukubokan*: Frequency scale used two times, from start to end, because it has to be constant in this case. *Midoubibavachakukan*: Mono sound channel resulting of the computation of this Sonographe. The signal is then saved in any sound file format managed by the Libsndfile from Erik de Castro Lopo (<http://en.wikipedia.org/wiki/Libsndfile>).



**Figure 4.** Example of a chord filter image build with 3 four-voice chords: 0 3 6 10, 0 4 7 11 and 0 4 7 10.



**Figure 5.** Image of figure 6 filtered by harmonic and chord filter of figure 4.

The resulting image is then multiplied by the source image to produce a new input to the *Sonographe* element (see figure 4 and 5). The filter may be used in an external application like GIMP or Photoshop and used as a layer mask or the filtered image may be directly produced with a binary block taking in parameters the image and each point of the image:

```
[ :i :p | (i lecture: p) *
(32 - ((2047 - p y) \ 32)) ]
```

Operators in Cyclonium may operate directly on images. To multiply two images, we just have to connect a product operator on the two images and compute it.

## 2.2 Chords and modal filtering

At this point, we can compute music in a chromatic scale and have a graphic control on pitch, dynamic and timbre. A simple set of new filters may be used to control chords. If we had to, or if we want the result to be constrained by harmonic rules or modal scales, we just have to multiply image by a new filter. As our filters are expressed in gray value scale image, the chord progression may be smooth or harsh. As chords and/or modes are expressed in the 2D space of time and frequency, we can draw *retards*, *anticipation*, and even mix different chords at the same time for different frequency ranges.



**Figure 6.** Original image, painted with watercolor and filtered in three steps: a negative filter is applied to convert the white background into black, then as all the colors have changed to negative, we apply a rotation of 180° on the hue component to restore tint. The last step is to apply a level filter in order to enhance contrast and eventually cut off the noise induce by the grain of paper.

For example, this is the block of code to build a simple four-voice chord of degree VII major or the second degree minor:

```
[ :p | (#(0 3 6 10) includes: ((4095 - p y) // 32) \\ 12)
  ifTrue: [1] ifFalse: [0] ]
```

0 3 6 10 are the intervals in semitone from the fundamental. 4095 is one less than the desired height of the image and 32 is the number of pixels by semitone.

Once we have built all the desired chords, modes and scales, we combine them in an external graphic editor and then multiply the image by the resulting filter.

### 2.3 Image encoding adapted to sound control parameters

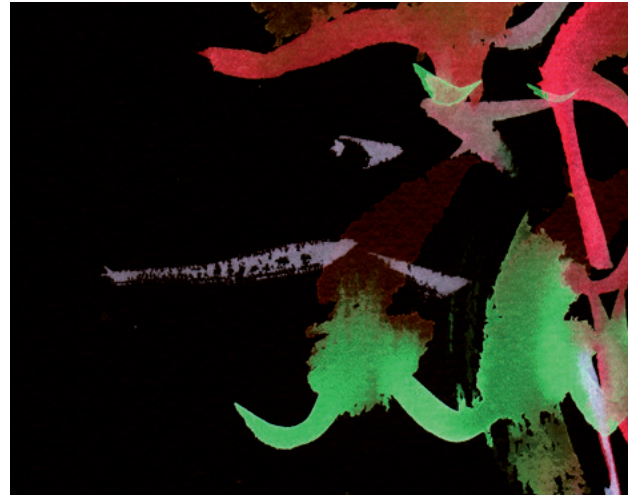
Usual painting, image tools and image file formats use only 8 bits by component to store value of pixels and 8 bits means only 256 different levels. In our case of using images to control intensities, frequency variations, and any sound parameter, we need more than 8 bit resolution. In our framework, we choose 16 bits to store gray levels for each pixel of an image. This leads to 65536 levels. In order to keep compatibility with external softwares, we use the following method to encode 16 bits gray level images into 24 bits colored images:

The first four high bits are triplicated in each first nibble of the RGB components, then, the lower weight bits are shuffled in the remaining space, alternatively in R, G and B. Figure 8 shows this wrapping.

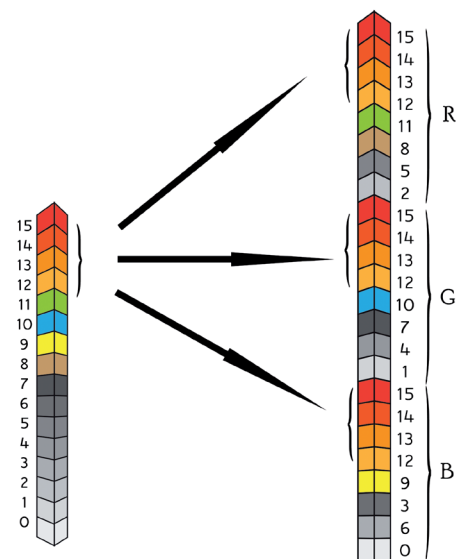
For the user, resulting images in RGB are close to the original gray value image and, for the computer, the 16 bits original data can be used. The framework leaves the possibility to save a calculated image either in RGB encoded bmp, lossless, or to export it in standard 8 bit gray value bmp with a loss of data.

Figures 9 and 10 show the same image with the two encoding format construct with the code:

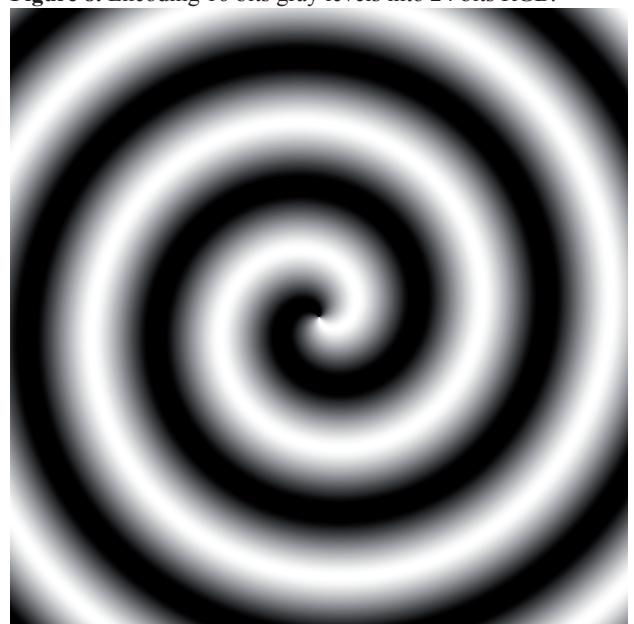
```
[ :theta :r |
  (theta + (r * 15)) sin ]
```



**Figure 7.** A detail of the image figure 6.

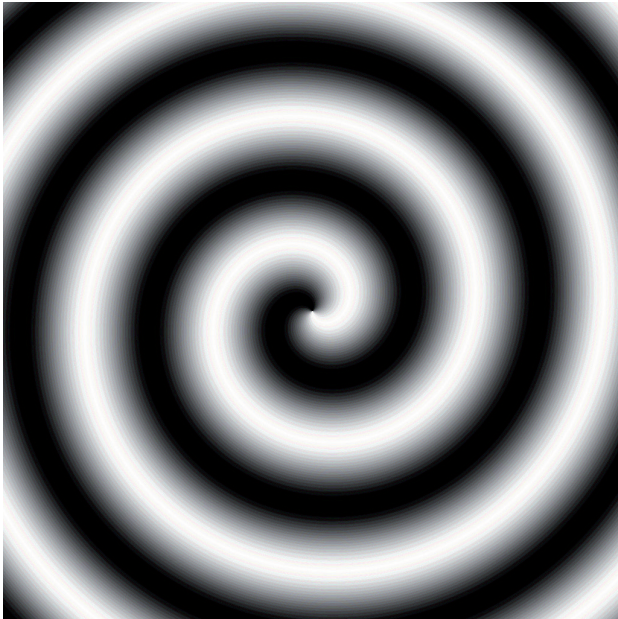


**Figure 8.** Encoding 16 bits gray levels into 24 bits RGB.



**Figure 9.** 8 bits standard gray level image.





**Figure 10.** 16 bits gray levels image encoded in RGB. Notice that this image looks very similar than the figure 9 image, this is expected, but the data encoded are on 16 bits instead of 8.

### 2.3.1 Colored images

We can also directly use a colored image as the input. In this case, the color value coded in RGB is divided by 16777215 to get a value between 0 and 1. As the red component has a factor of  $256 \times 256$ , the green component 256 and the blue component only 1, this is not a correct nor conventional transcription of a color into an intensity but it is useful in practice to avoid conversions and results are still in conjunction with our perception of the image.

### 2.3.2 Virtual images

Images may be seen as function of  $x$  and  $y$  coordinates. When we can express them by a formula, we don't have to store every pixel value in memory. Instead, we only give a block to be evaluated each time the value is needed. This kind of image has no frame, neither height nor width.

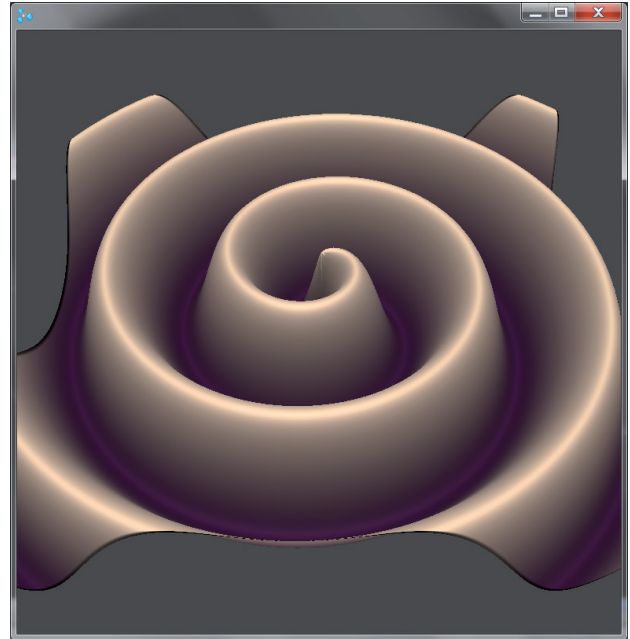
### 2.4 Sonographe with frequency modulation

As mentioned above, a Sonographe element can be connected to a Modulator element. The Modulator element like the Sonographe element takes as input an image to control amplitudes, one or two frequency scales and an optional frequency control image to control the modulation frequencies with the same rules, an optional wave form and finally an element giving the modulation indices for each line of the image.

## 3. Time and rhythms aspects in Sonographe

Actually, the time in sonographs is linear and constant for a given computation. It is expressed in milliseconds by pixel, or in an alternative way, we give a duration in milliseconds for an image or a sequence of images and the sample frequency, then, the value of number of sample by pixel is deduced.

For all the parameters of sonographs controlled by an image, the procedure includes automatically a maximal variation of the parameter from one pixel to the next in the time axis. By default the maximum variation is one divided by the number of samples in one pixel, but we can set any coefficient to this value in the setting of the *Sonographe* element, independently for increasing and decreasing values. For example, we can set different values



**Figure 11.** 3D view of the levels.

for the attack and the release, or for ascending and descending portamenti when an image controls the frequencies. In the same way, we can control the maximum speed of the pan to the left or to the right. The limitation is that this control is global on the image or the sequence.

### 3.1 Geometric progression and rhythms

We now focus on rhythms and we want to go beyond the mechanical constraint of linear time given in milliseconds per pixel by constructing rhythm filters (or time stretching patterns) for our sonograph images.

A time stretching pattern is an image where the gray levels are normalized abscissa of points to be taken in the source image. The time identity function is a pattern with a shade of gray from black on left to white on right, and will result in no change, just copy the source image.

For example, we want to build a rhythmic cell that will be repeated with a geometric progression, accelerating, but with the aim that, at the end of a sequence, the sum of  $k + 1$  cells gets the same duration as the first one. The problem is the same for decelerating by reversing the  $x$  axis orientation.

We are looking for the common ratio  $q$  of a geometrical sequence  $U_0, U_1, \dots, U_n$  of  $n + 1$  terms, such that the sum of  $k + 1$  last terms be equal to the first term.

We must have :  $n \geq 2, q \neq 1, k < n$ .

In general for any geometrical progression we have:

$$U_p + U_{p+1} + \dots + U_n = U_p * (1 - q^{(n-p+1)}) / (1 - q).$$

If  $p = n - k$  we have

$$U_{n-k} + U_{n-k+1} + \dots + U_n = U_{n-k} * (1 - q^{(n-k+1)}) / (1 - q)$$

and, as  $U_{n-k}$  is a term of the progression,

$$U_{n-k} = U_0 * q^{(n-k)}$$

and we want that :  $U_0 = U_{n-k} + U_{n-k+1} + \dots + U_n$

So :

$$U_0 = U_0 * q^{(n-k)} * (1 - q^{(k+1)}) / (1 - q)$$

$$1 = (q^{(n-k)} - q^{(n+1)}) / (1 - q)$$

$$1 - q = q^{(n-k)} - q^{(n+1)}$$

We get the general formula :

$$q^{(n+1)} = q^{(n-k)} + q - 1$$

A well known particular case:

If  $n = 2$  and  $k = 1$ , there are 3 terms and the sum of the last two is equal to the first.

$$q^3 = q^1 + q - 1 = 2q - 1$$

We find that the reciprocal of the golden ratio : 0.6180339887... is a solution.

As the general formula is of high degree, we find the value of the ratio  $q$  by approximations.

Here follows an array with the first values of  $n$  and  $k$ :

n	k	q
2	1	0.6180339887
3	1	0.7548776662
3	2	0.5436890127
4	1	0.8191725134
4	2	0.6823278038
4	3	0.5187900637
5	1	0.8566748839
5	2	0.7548776662
5	3	0.6518234538
5	4	0.5086603916
6	1	0.8812714616
6	2	0.8000949944
6	3	0.724491959
6	4	0.6368829168
6	5	0.5041382584
7	1	0.8986537126
7	2	0.8311059462
7	3	0.7713274516
7	4	0.708133776
7	5	0.6289286919
7	6	0.5020170552

Example:

Once we get the formula for a particular case, for example,  $n = 6$  and  $k = 2$

$$q^{(n+1)} = q^{(n-k)} + q - 1$$

$$q^7 = q^4 + q - 1$$

we get an expression in high degree, with no simple way to find the roots. The approximate value of  $q$  is 0.8000949944 in this case. Then we compute the series  $U_0, U_1, \dots, U_n$  and sum the series:

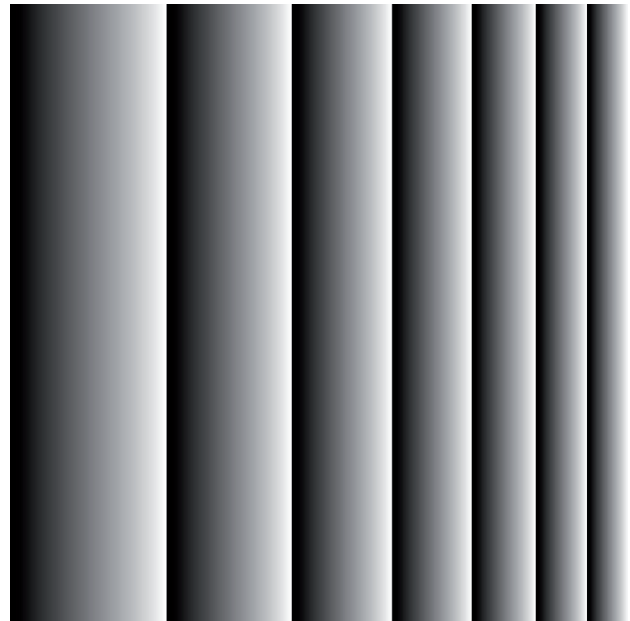
$0, U_0, U_0 + U_1, U_0 + U_1 + U_2, \dots$ , up to the width of the destination image. We associate these accumulations to the source coordinates to form the control points of a spline, and finally approximate the spline to a polyline with a sufficient precision (see figure 12).

#### 4. Sounds and music production from video

We propose here two ways to create sounds and music from video using the time / frequency representation of sonographs and another method, using trajectories of virtual mobiles (Lesbros, 1999). All these methods give results that are tightly coupled to the video images along time.

##### 4.1 Sonographe sequence

The first one is to select a set of video frames, taken at regular time intervals and to chain the sonographic transformation of



**Figure 12.** Time stretching pattern for the case  $n = 6$  and  $k = 2$ . We get 7 bands, and the width of the first one is equal to the sum of the width of the three last. This image is later used to transform coordinates of the real rhythmic pattern to be used in the composition. Regular time pulsations in the source image will be accelerated regularly (in geometrical progression), and respecting the fact that the three last occurrences will have the same duration than the first one.

each selected frame. For example, we choose a time interval of three seconds, we select one frame of the video every three seconds, and on each frame we compute a sonograph of three seconds.

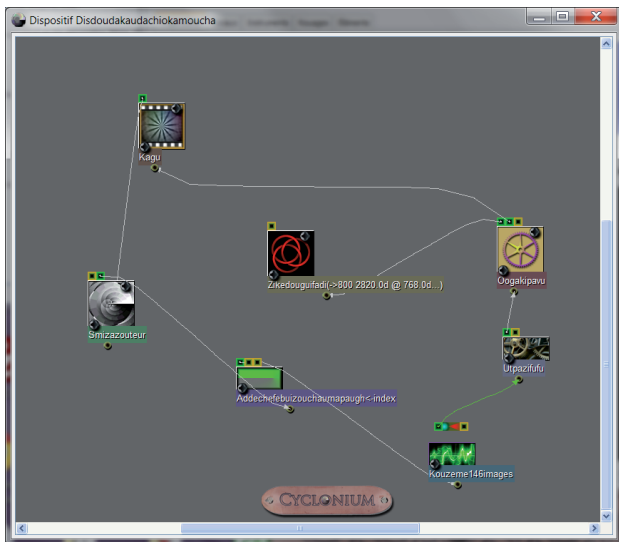
Our function library enables us to chain computation on successive images<sup>3</sup> by returning all current temporary values to the caller at the end of each step. For example the current envelope value, the current frequency, the current pan value, etc. for each line. This yields a continuous sound in spite of frame boundary crossings.

As in the case of a unique image, the user gives the total duration in milliseconds. This duration is divided by the number of images in the sequence. The system calculates an entire number of samples by image.

##### 4.2 Trajectories

The sequence of image element can point to a computation chain that calculates the index of the image to use at any step of a computation. If we are computing a sound channel, a step of computation will correspond to one sample of the produced signal. If we are computing an image (with a *Dessinateur* element), a step will correspond to one pixel. Thanks to these properties, we can produce sounds whose parameters, or even direct sample values, are taken from any frame of the sequence. If it is ensured that the index of the frame is calculated in relation to the time, the produced sounds will fit to the evolution of the movie or film.

<sup>3</sup> The user interface representing a sequence of images needs only to know the folder where images are stored. All the images have to be of the same size and they are computed in alphabetical order, but they don't have to be taken from a video, these sequences may be constructed by any other source.



**Figure 13.** *Kagu*: a sequence of 146 images.  
*Smizazouteur*: the image selector receiving the index of a sample and producing the index of a frame in the sequence with the following block of code:  

$$[:i | (((i - 1) * 146) / 6912000) + 1 ]$$
*"i"* is the sample index (from 1 to 6912000), 146 is the number of frames.  
*Zikedougufadi*: a trajectory.  
*Oogakipavu*: a cog used to link our instrument *Utpazifufu* with the trajectory and the image sequence.  
*Kouzeme146images*: The resulting channel being calculated. It's length is 6912000 samples corresponding to the 144 seconds of sounds at 48000 Hz (the standard sample rate for video).

In this example (figure 13), reduced to the minimum, the trajectory is fixed and the frequency of the instrument is fixed too. There is no amplitude control or frequency modulation. But the framework provides elements to build calculation chains to modify and combine trajectories, to sample images along these trajectories and to use these values for controlling frequencies, amplitudes, frequency modulation, etc., in fact, any value in the scheme. Elements are also provided to compute elastic coupling in conjunction with trajectories. See (Rainer, 2004, Lesbros 1999) for quick synthesis algorithms.

### 4.3 Building a summary of a video

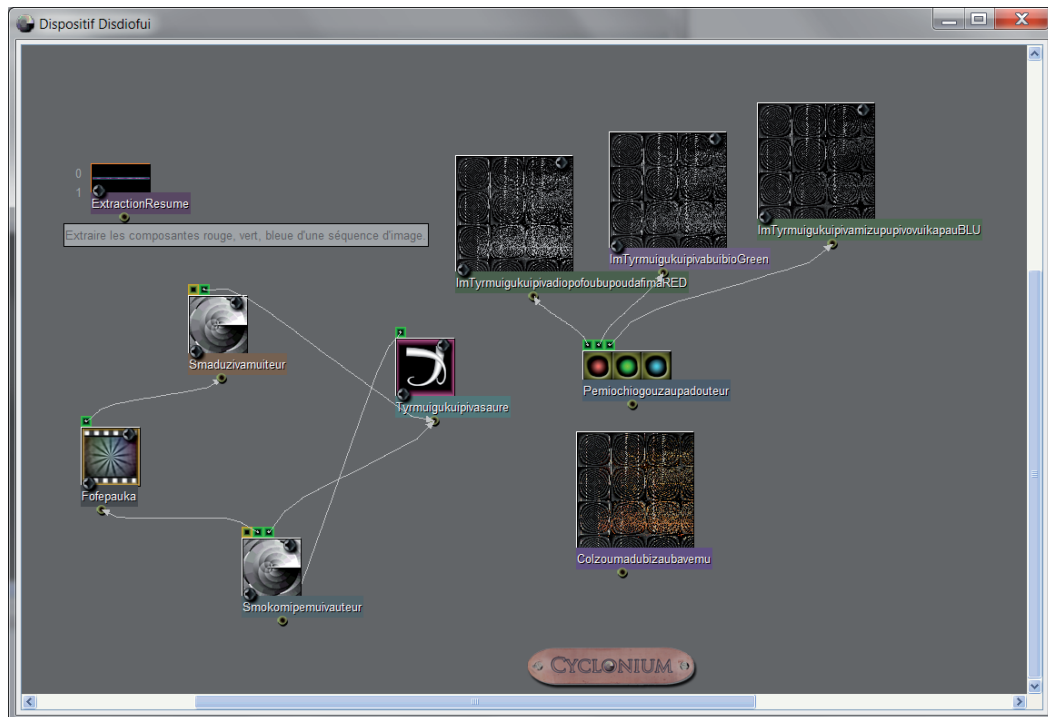
With the same two principal elements; a sequence of image coming from a video and a selector, we can build an image that acts as a summary of the video on which we can compute the sonograph. Prior to computing the sonograph, we can apply all the filtering methods we have seen above; harmonic filtering, chord filtering, etc.

A *Dessinateur* element is used to create the new image. Each step of computation consists in determining the value of a given pixel. The *Dessinateur* element provides the coordinates of the point to be calculated, and its input should respond with the value to store in the produced image at that location.

The heart of this example (figure 14) is the element that gives the value to the *Dessinateur*. For the extraction of the green component this element uses the following code:

```
[:i :p | | aRate aColor |
  aRate := (p y / 2335) * 436.32 + ((2335 - p y) / 2335).
  aColor := i couleur: ((p x * 3504 * aRate / 7272) @ p y).
  aColor green ]
```

The block receives the image sequence, "i", and the *Dessinateur* current coordinate "p" as parameters. "aRate" is



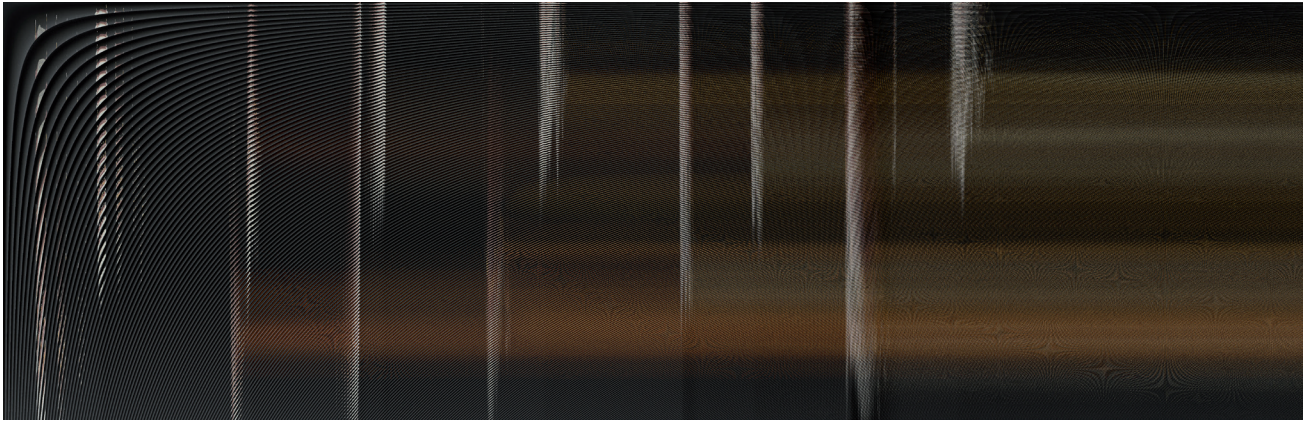
**Figure 14.** In this example, the sequence *Fofepauka* contains 3636 images extracted from a video of 145,44 seconds (at 25 frames by seconds) and we construct a summary of 7272 pixels wide. The selector *Smaduzivamuiteur* is a Smalltalk block of code:

```
[:p | (p x * 3636 / 7272) + 1 ]
```

where p given by the *Dessinateur* element, is the coordinate of the desired point in final image. The value returned is the index of frame in the sequence (between 1 and 3636).

The *Dessinateur* *Tyrmuigukuiivasaura* has been set with the following parameters and choices: 7272 x 3636 for the size of resulting image, unset automatic level to insure that values of components of the image will stay unchanged and process from left to right instead of top down to limit the number of time an image from the sequence will be loaded into memory.





**Figure 15.** The summary image of the video with variable scan rates according to the ordinate.

the rate at which we want to scan source images and we make it dependent of the ordinate which is linked to the frequency. We want low lines to be scanned at three times per second ( $145.440 * 3$  gives 436.32) and high line at only one scan for the whole piece. Then we compute the x and y coordinates (y is left unchanged) and ask for the color of the point with the #couleur: message. Finally we deliver the green component value, ranging from zero to one. We do the same for red and blue components and with the *Pemiochiogouzaupadouteur* element we are able to reassemble the three components into one image which can be seen as a summary of the video. The next step is to use this image to generate sounds evolving with the movie.

## 5. Conclusion

Our framework is still under development. A set of tools has been built in order to create images, sound and music from images. As the computations are not in real time, we can set the atomic level of a chain of computation at the sample level or even at a subsample level (Roads 2001, p3-5). This may enable, for example, to control envelopes with the same unit of time as the samples.

Using generic operators and the Smalltalk language keeps the framework open to new experimentation. For example, by using generic operators, one may combine elements used to generate MIDI files with a *Traceur* element to generate at the same time a MIDI score and an image usable in a Sonographe. Any method can be used to generate paintings used with any mapping in musical compositions (Guéret *et al.*, 2004; Mao *et al.*, 2002; Thiebaut *et al.*; 2008).

By introducing variable scales and frequency control through images, we enhance the possibilities of sonographic composition. Sonographs with frequency modulation and pan are also open to new experiments, combined with filters for harmonics, chords and/or rhythmic time stretching patterns.

Finally, we show how to use sequences of images extracted from a video in order to have any parameter of a generated sound matching the stream of the video.

## Annexe 1 sound samples

<https://soundcloud.com/vincent-lesbros>

## References

- Englert, Giuseppe G., 1981, Automated Composition and Composed Automation, dans : The Music Machine, édité par C. Roads MIT Press 1989 p-191, translated from french by Otto LASKI with Curtis ROADS. Musique : Composition Automatique, Automatique Composée, Informatique et Sciences Humaines 45, Sorbone, Paris, 1981.
- Goldberg, David, 1991, What Every Computer Scientist Should Know about Floating-Point Arithmetic, ACM Computing Surveys, vol. 23, no 1, mars 1991.
- Guéret, Christophe ; Monmarché, Nicolas ; Slimane, Mohamed, 2004, « Ants can play music » in : Conference: Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5 - 8, 2004, Proceedings
- Lesbros, Vincent, 1995, « Atelier Incrémentiel pour la musique expérimentale », Thèse de Doctorat en Intelligence artificielle, Paris, Université de Paris 8.
- Lesbros, Vincent, 1996, « From images to sounds: a dual representation », Computer Music Journal, 20(3):59-69.
- Lesbros, Vincent, 1999, « Phonograms, Elastic Couplings, and Trajectories », Computer Music Journal, 23(2):70-79.
- Mao, Xia; Chen, Bin; Zhu, Gang; Hoshino, Tsutomu, 2002, « Study on transforming from painting to music », VIPromCom-2002. 4th EURASIP - IEEE Region 8 International Symposium on Video Image Processing and Multimedia Communications, 16-19 June 2002, Zadar, Croatia.
- Rainer, Harald, 2003, « Smart sound generation for mobile phones », Diploma thesis at Graz university of Music and Dramatic Arts.
- Roads, Curtis, 1995, The computer music tutorial, MIT Press Cambridge, MA, ISBN:0262181584
- Roads, Curtis, 2001, Microsound (en). Cambridge: MIT Press. ISBN 0-262-18215-7
- Thiebaut, Jean-Baptiste ; Healey, Patrick G. T. ; Kinns Nick Bryan, 2008, « Drawing electroacoustic music », Interaction, Media and Communication, Queen Mary, University of London.
- Xenakis, Iannis, 1971, Formalized music, Bloomington, Indiana University Press.
- Xiaoying, Wu, and Ze-Nian, Li, 2008, « A study of image-based music composition », School of Computing Science, Simon Fraser University.