

SoundPaint – Painting Music

Jürgen Reuter
Karlsruhe
Germany
reuter@ipd.uka.de

Abstract

We present a paradigm for synthesizing electronic music by graphical composing. The problem of mapping colors to sounds is studied in detail from a mathematical as well as a pragmatic point of view. We show how to map colors to sounds in a user-definable, topology preserving manner. We demonstrate the usefulness of our approach on our prototype implementation of a graphical composing tool.

Keywords

electronic music, sound collages, graphical composing, color-to-sound mapping

1 Introduction

Before the advent of electronic music, the western music production process was clearly divided into three stages: Instrument craftsmen designed musical instruments, thereby playing a key role in sound engineering. Composers provided music in notational form. Performers realized the music by applying the notational form on instruments. The diatonic or chromatic scale served as commonly agreed interface between all participants. The separation of the production process into smaller stages clearly has the advantage of reducing the overall complexity of music creation. Having a standard set of instruments also enhances efficiency of composing, since experience from previous compositions can be reused.

The introduction of electro-acoustic instruments widened the spectrum of available instruments and sounds, but in principle did not change the production process. With the introduction of electronic music in the middle of the 20th century however, the process changed fundamentally. Emphasis shifted from note-level composing and harmonics towards sound engineering and creating sound collages. As a result, composers started becoming sound engineers, taking over the instrument crafts men's job. Often, a composition could not be notated with traditional notation, or, even worse,

the composition was strongly bound to a very particular technical setup of electronic devices. Consequently, the composer easily became the only person capable of performing the composition, thereby often eliminating the traditional distinction of production stages. At least, new notational concepts were developed to alleviate the problem of notating electronic music.

The introduction of MIDI in the early 80s was in some sense a step back to electro-acoustic, keyed instruments music, since MIDI is based on a chromatic scale and a simple note on/off paradigm. Basically, MIDI supports any instrument that can produce a stream of note on/off events on a chromatic scale, like keyed instruments, wind instruments, and others. Also, it supports many expressive features of non-keyed instruments like vibrato, portamento or breath control. Still, in practice, mostly keyboards with their limited expressive capabilities are used for note entry.

The idea of our work is to break these limitations in expressivity and tonality. With our approach, the composer creates sound collages by visually arranging graphical components to an image, closely following basic principles of graphical notation. While the graphical shapes in the image determine the musical content of the sound collage, the sound itself is controlled by color. Since in our approach the mapping from colors to actual sounds is user-definable for each image, the sound engineering process is independent from the musical content of the collage. Thus, we resurrect the traditional separation of sound engineering and composing. The performance itself is done mechanically by computation, though. Still, the expressive power of graphics is straightly translated into musical expression.

The remainder of this paper is organized as follows: Section 2 gives a short sketch of image-to-audio transformation. To understand the role of colors in a musical environment, Section

3 presents a short survey on the traditional use of color in music history. Next, we present and discuss in detail our approach of mapping colors to sounds (Section 4). Then, we extend our mapping to aspects beyond pure sound creation (Section 5). A prototype implementation of our approach is presented in Section 6. We already gained first experience with our prototype, as described in Section 7. Our generic approach is open to multiple extensions and enhancements, as discussed in Section 8. In Section 9, we compare our approach with recent work in related fields and finally summarize the results of our work (Section 10).

2 Graphical Notation Framework

In order to respect the experience of traditionally trained musicians, our approach tries to stick to traditional notation as far as possible. This means, when interpreting an image as sound collage, the horizontal axis represents time, running from the left edge of the image to the right, while the vertical axis denotes the pitch (frequency) of sounds, with the highest pitch located at the top of the image. The vertical pitch ordinate is exponential with respect to the frequency, such that equidistant pitches result in equidistant musical intervals. Each pixel row represents a (generally changing) sound of a particular frequency. Both axes can be scaled by the user with a positive linear factor. The color of each pixel is used to select a sound. The problem of how to map colors to sounds is discussed later on.

3 Color in Musical Notation History

The use of color in musical notation has a long tradition. We give a short historical survey in order to show the manifold applications of color and provide a sense for the effect of using colors.

Color was perhaps first applied as a purely notational feature by GUIDO VON AREZZO, who invented colored staff lines in the 11th century, using yellow and red colors for the do and fa lines, respectively. During the *Ars Nova* period (14th century), note heads were printed with black and red color to indicate changes between binary and ternary meters (Apel, 1962). While in medieval manuscripts color had been widely applied in complex, colored ornaments, with the new printing techniques rising up in the early 16th century (most notably PETRUCCI's *Odhecaton* in 1501), extensive use of colors in printed music was hardly feasible or just too expen-

sive and thus became seldom. MOZART wrote a manuscript of his horn concert K495 with colored note heads, serving as a joke to irritate the hornist LEUTGEB – a good friend of him (Wiese, 2002). In liturgical music, red color as contrasted to black color remained playing an extraordinary role by marking sections performed by the priest as contrasted to those performed by the community or just as a means of readability (black notes on red staff lines). Slightly more deliberate application of color in music printings emerged in the 20th century with technological advances in printing techniques: The advent of electronic music stimulated the development of graphical notation (cp. e.g. STOCKHAUSEN's *Studie II* (Stockhausen, 1956) for the first electronic music to be published (Simeone, 2001)), and WEHINGER uses colors in an aural score (Wehinger, 1970) for LIGETI's *Articulation* to differentiate between several classes of sounds. For educational purposes, some authors use colored note heads in introductory courses into musical notation (Neuhäuser et al., 1974). There is even a method for training absolute hearing based on colored notes (Taneda and Taneda, 1993). Only very recently, the use of computer graphics in conjunction with electronic music has led to efforts in formally mapping colors to sounds (for a more detailed discussion, see the Related Work Section 9).

While Wehinger's aural score is one of the very few notational examples of mapping colors to sounds, music researchers started much earlier to study relationships between musical and aural content. Especially with the upcoming psychological research in the late 19th century, the synesthetic relationship between hearing and viewing was studied more extensively. WELLEK gives a comprehensive overview over this field of research (Wellek, 1954), including systems of mapping colors to keys and pitches. Painters started trying to embed musical structures into their work (e.g. KLEE's *Fugue in Red*). Similarly, composers tried to paint images, as in MUSSORGSKY's *Pictures at an Exhibition*. In Jazz music, synesthesia is represented by coinciding emotional mood from acoustic and visual stimuli, known as the blue notes in blues music.

4 Mapping Colors to Sounds

We now discuss how colors are mapped to sounds in our approach.

For the remainder of this discussion, we define

a *sound* to be a 2π -periodic, continuous function $s : \mathbf{R} \mapsto \mathbf{R}, t \rightarrow s(t)$. This definition meets the real-world characteristic of oscillators as the most usual natural generators of sounds and the fact that our ear is trained to recognize periodic signals. Non-periodic natural sources of sounds such as bells are out of scope of this discussion. We assume normalization of the periodic function to 2π periodicity in order to abstract from a particular frequency. According to this definition, the set of all possible sounds – the *sound space* – is represented by the set of all 2π -periodic functions.

Next, we define the *color space* \mathbf{C} following the standard RGB (red, green, blue) model: the set of colors is defined by a three-dimensional real vector space \mathbf{R}^3 , or, more precisely, a subset thereof: assuming, that the valid range of the red, green and blue color components is $[0.0, 1.0]$, the color space is the subset of \mathbf{R}^3 that is defined by the cube with the edges $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. Note that the color space is not a vector space since it is not closed with respect to addition and multiplication by scalar. However, this is not an issue as long as we do not apply operations that result in vectors outside of the cube. Also note that there are other possibilities to model the color space, such as the HSB (hue, saturation, brightness) model, which we will discuss later.

Ideally, for a useful mapping of colors to sounds, we would like to fulfill the following constraints:

- **Injectivity.** Different colors should map to different sounds in order to utilize the color space as much as possible.
- **Surjectivity.** With a painting, we want to be able to address as many different sounds as possible – ideally, all sounds.
- **Topology preservation.** Most important, similar colors should map to similar sounds. For example, when there is a color gradation in the painting, it should result in a sound gradation. There should be no discontinuity effect in the mapping. Also, we want to avoid noticeable hysteresis effects in order to preserve reproducibility of the mapping across the painting.
- **User-definable mapping.** The actual mapping should be user-definable, as research has shown that there is no general mapping that applies uniquely well to all individual humans.

Unfortunately, there is no mapping between the function space of 2π -periodic functions and \mathbf{R}^3 that fulfills all of the three constraints. Pragmatically, we drop surjectivity in order to find a mapping that fulfills the other constraints. Indeed, dropping the surjectivity constraint does not hurt too much, if we assume that the mapping is user-definable individually for each painting and that a *single* painting does not need to address *all* possible sounds: rather than mapping colors to the full sound space, we let the user select a three-dimensional subspace \mathbf{S} of the full sound space. This approach also leverages the limitation of our mapping not being surjective: since for each painting, a different sound subspace can be defined by the composer, effectively, the whole space of sounds is still addressable, thus retaining surjectivity in a limited sense.

Dropping the surjectivity constraint, we now focus on finding a proper mapping from color space to a three-dimensional subset of the sound space. Since we do not want to bother the composer with mathematics, we just require the basis of a three-dimensional sound space to be defined. This can be achieved by the user simply defining three different sounds, that span a three-dimensional sound space. Given the three-dimensional color space \mathbf{C} and a three-dimensional subspace \mathbf{S} of the full sound space, a bijective, topology preserving mapping can be easily achieved by a linear mapping via a matrix multiplication,

$$M : \mathbf{C} \mapsto \mathbf{S}, x \rightarrow y = Ax, x \in \mathbf{C}, y \in \mathbf{S} \quad (1)$$

with A being a 3×3 matrix specifying the actual mapping. In practice, the composer would not need to specify this vector space homomorphism M by explicitly entering some matrix A . Rather, given the three basis vectors of the color space \mathbf{C} , i.e. the colors red, green, and blue, the composer just defines a sound individually for each of these three basis colors. Since each other color can be expressed as a linear combination of the three basis colors, the scalars of this linear combination can be used to linearly combine the three basis sounds that the user has defined.

5 Generalizing the Mapping

As excitingly this approach may sound at first, as disillusioning we are thrown back to reality: pure linear combination of sounds results in nothing else but cross-fading waveforms, which

quickly turns out to be too limited for serious composing. However, what we can still do is to extend the linear combination of sounds onto further parameters that influence the sound in a non-linear manner. Most notably, we can apply non-linear features on sounds such as vibrato, noise content, resonance, reverb, echo, hall, detune, disharmonic content, and others. Still, also linear aspects as panning or frequency-dependent filtering may improve the overall capabilities of the color-to-sound mapping. In general, any scalar parameter, that represents some operation which is applicable on arbitrary sounds, can be used for adding new capabilities. Of course, with respect to our topology preservation constraint, all such parameters should respect continuity of their effect, i.e. there should no remarkable discontinuity arise when slowly changing such a parameter.

Again, we do not want to burden the composer with explicitly defining a mapping function. Instead, we extend the possibilities of defining the three basis sounds by adding scalar parameters, e.g. in a graphical user interface by providing sliders in a widget for sound definition.

So far, we assumed colors red, green and blue to serve as basis vectors for our color space. More generally, one could allow to accept any three colors, as long as they form a basis of the color space. Changing the basis of the color space can be compensated by adding a basis change matrix to our mapping M :

$$M' : \mathbf{C}' \mapsto \mathbf{S}, x \mapsto y = A\phi_{C' \rightarrow C}x = A'x, \quad (2)$$

assuming that $\phi_{C' \rightarrow C}$ is the basis change matrix that converts x from space \mathbf{C}' to space \mathbf{C} .

Specifically, composers may want to prefer the HSB model over the RGB model: traditionally, music is notated with black or colored notes on white paper. An empty, white paper is therefore naturally associated with silence, while a sheet of paper heavily filled with numerous musical symbols typically reminds of terse music. Probably more important, when mixing colors, most people think in terms of subtractive rather than additive mixing. Conversion between HSB and RGB is just another basis change of the color space.

When changing the basis of the color space, care must be taken with respect to the range of the vector components. As previously mentioned, the subset of the \mathbf{R}^3 , that forms the color space, is not a vector space, since the sub-

set is not closed with respect to addition and multiplication by scalar. By changing the basis in \mathbf{R}^3 , the cubic shape of the RGB color space in the first octant generally transforms into a different shape that possibly covers different octants, thereby changing the valid range of the vector components. Therefore, when operating with a different basis, vectors must be carefully checked for correct range.

6 SoundPaint Prototype Implementation

In order to demonstrate that our approach works, a prototype has been implemented in C++. The code currently runs under Linux, using wxWidgets (Roebing et al., 2005) as GUI library. The GUI of the current implementation mainly focuses on providing a graphical front-end for specifying an image, and parameterizing and running the transformation process, which synthesizes an audio file from the image file. An integrated simple audio file player can be used to perform the sound collage after transformation.

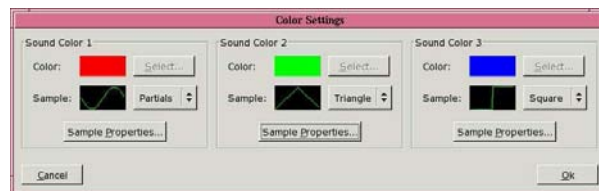


Figure 1: Mapping Colors to Sounds

Currently, only the RGB color space is supported with the three basis vectors red, green, and blue. The user defines a color-to-sound mapping by simply defining three sounds to be associated with the three basis colors. Figure 1 shows the color-to-sound mapping dialog. A generic type of wave form can be selected from a list of predefined choices and further parameterized, as shown in Figure 2 for the type of triangle waves. All parameters that go beyond manipulating the core wave form – namely pan, vibrato depth and rate, and noise content – are common to all types of wave forms, such that they can be linearly interpolated between different types. Parameters such as the duty cycle however only affect a particular wave form and thus need not be present for other types of wave forms.

Some more details of the transformation are worth mentioning. When applying the core transformation as described in Section 2, the

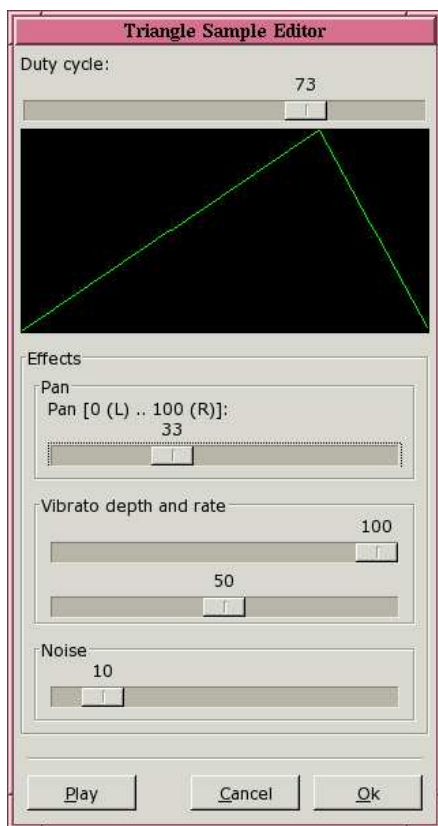


Figure 2: Parameterizing a Triangle Wave

resulting audio file will typically contain many crackling sounds. These annoying noises arise from sudden color or brightness changes at pixel borders: a sudden change in sound produces high-frequency peaks. To alleviate these noises, pixel borders have to be smoothed along the time axis. As a very simple method of anti-aliasing, SoundPaint horizontally divides each image pixel into sub-pixels down to audio resolution and applies a deep path filter along the sub-pixels. The filter characteristics can be controlled by the user via the **Synthesize Options** widget, ranging from a plain overall sound with clearly noticeable clicks to a smoothed, almost reverb-like sound.

Best results are achieved when painting only a few colored structures onto the image and leaving the keeping the remaining pixels in the color that will produce silence (i.e., in the RGB model, black). For performance optimization, it is therefore useful to handle these silent pixels separately, rather than computing a complex sound with an amplitude of 0. Since, as an effect of the before mentioned pixel smoothing, often only very few pixels are exactly 0, SoundPaint simply assumes an amplitude of 0, if the am-

plitude level falls below a threshold value. This threshold value can be controlled via the **gate** parameter in the **Synthesize Options** widget.

7 Preliminary Experience

SoundPaint was first publically presented in a workshop during the last *Stadtgeburtstag* (city's birthday celebrations) of the city *Karlsruhe* (Sta, 2004). Roughly 30 random visitors were given the chance to use SoundPaint for a 30 minutes slot. A short introduction was presented to them with emphasis on the basic concepts from a composer's point of view and basic use of the program. They were instructed to paint on black background and keep the painting structurally simple for achieving best results. For the actual process of painting, XPaint (as default) and Gimp (for advanced users) were provided as external programs.

Almost all users were immediately able to produce sound collages, some of them with very interesting results. What turned out to be most irritating for many users is the additive interpretation of mixed colors. Also, some users started with a dark gray rather than black image background, such that SoundPaint's optimization code for silence regions could not be applied, resulting in much slower conversion. These observations strongly suggest to introduce HSB color space in SoundPaint.

8 Future Work

Originally stemming from a command-line tool, SoundPaint still focuses on converting image files into audio files. SoundPaint's GUI mostly serves as a convenient interface for specifying conversion parameters. This approach is, from a software engineering point of view, a good basis for a clean software architecture, and can be easily extended e.g. with scripting purposes in mind. A composer, however, may prefer a sound collage in a more interactive way rather than creating a painting in an external application and repeatedly converting it into an audio file in a batch-style manner. Hence, SoundPaint undoubtedly would benefit from integrating painting facilities into the application itself.

Going a step further, with embedded painting facilities, SoundPaint could be extended to support live performances. The performer would simply paint objects *ahead* of the cursor of SoundPaint's built-in player, assuming that the image-to-audio conversion can be performed in real-time. For Minimal Music like perfor-

mances, the player could be extended to play in loop mode, with integrated painting facilities allowing for modifying the painting for the next loop. Inserting or deleting multiple objects following predefined rhythmical patterns with a single action could be a challenging feature.

Assembling audio files generated from multiple images files into a single sound collage is desired when the surjectivity of our mapping is an issue. Adding this feature to SoundPaint would ultimately turn the software into a multi-track composing tool. Having a multi-track tool, integration with other notation approaches seems nearby. For example, recent development of LilyPond's(Nienhuys and Nieuwenhuizen, 2005) GNOME back-end suggests to integrate traditional notation in separate tracks into SoundPaint. The overall user interface of such a multi-track tool finally could look similar to the arrange view of standard sequencer software, but augmented by graphical notation tracks.

9 Related Work

Graphical notation of music has a rather long history. While the idea of graphical composing as the reverse process is near at hand, practically usable tools for off-the-shelf computers emerged only recently. The most notably tools are presented below.

Maybe IANNIS XENAKIS was the first one who started designing a system for converting images into sounds in the 1950's, but it took him decades to present the first implementation of his UPIC system in 1978(Xenakis, 1978). Like SoundPaint, Xenakis uses the coordinate axes following the metaphor of scores. While SoundPaint uses a pixel-based conversion that can be applied on any image data, the UPIC system assumes line drawings with each graphical line being converted into a melody line.

Makesound(Burrell, 2001) uses the following mapping for a sinusoidal synthesis with noise content and optional phase shift:

x position	phase
y position	temporal position
hue	frequency
saturation	clarity (inv. noise content)
luminosity	intensity (amplitude)

In Makesound, each pixel represents a section of a sine wave, thereby somewhat following the idea of a spectrogram rather than graphical notation. Color has no effect on the wave shape itself.

EE/CS 107b(Suen, 2004) uses a 2D FFT of

each of the RGB layers of the image as basis for a transformation. Unfortunately, the relation between the image and the resulting sound is not at all obvious.

Coagula(Ekman, 2003) uses a synthesis method that can be viewed as a special case of SoundPaint's synthesis with a particular set of color to sound mappings. Coagula uses a sinusoidal synthesis, using x and y coordinates as time and frequency axis, respectively. Noise content is controlled by the image's blue color layer. Red and green control stereo sound panning. Following Coagula's documentation, SoundPaint should show a very similar behavior when assigning 100% noise to blue, and pure sine waves to colors red and green, with setting red color's pan to left and green color's pan to right.

Just like Coagula, MetaSynth(Wenger and Spiegel, 2005) maps red and green to stereo panning, while blue is ignored.

Small Fish(Furukawa et al., 1999), presented by the ZKM(ZKM, 2005), is an illustrated booklet and a CD with 15 art games for controlling animated objects on the computer screen. Interaction of the objects creates polytonal sequences of tones in real-time. Each game defines its own particular rules for creating the tone sequences from object interaction. The tone sequences are created as MIDI events and can be played on any MIDI compliant tone generator. Small Fish focuses on the conversion of movements of objects into polytonal sequences of tones rather than on graphical notation; still, shape and color of the animated objects in some of the games map to particular sounds, thereby translating basic concepts of graphical notation into an animated real-time environment.

The PDP(Schouten, 2004) extension for the Pure Data(Puckette, 2005) real-time system follows a different approach in that it provides a framework for general image or video data processing and producing data streams by serialization of visual data. The resulting data stream can be used as input source for audio processing.

Finally, it is worth mentioning that the visualization of acoustic signals, i.e. the opposite conversion from audio to image or video, is frequently used in many systems, among them Winamp(Nullsoft, 2004) and Max/MSP/Jitter(Cycling '74, 2005). Still, these species of visualization, which are often implemented as real-time systems, typically

work on the audio signal level rather than on the level of musical structures.

10 Conclusions

We presented SoundPaint, a tool for creating sound collages based on transforming image data into audio data. The transformation follows to some extent the idea of graphical notation, using x and y axis for time and pitch, respectively. We showed how to deal with the color-to-sound mapping problem by introducing a vector space homomorphism between color space and sound subspace. Our tool mostly hides mathematical details of the transformation from the user without imposing restrictions in the choice of parameterizing the transformation. First experience with random users during the city's birthday celebrations demonstrated the usefulness of our tool. The result of our work is available as open source at <http://www.ipd.uka.de/~reuter/soundpaint/>.

11 Acknowledgments

The author would like to thank the Faculty of Computer Science of the University of Karlsruhe for providing the infrastructure for developing the SoundPaint software, and the department for technical infrastructure (ATIS) and Tatjana Rauch for their valuable help in organizing and conducting the workshop at the city's birthday celebrations.

References

- Willi Apel. 1962. *Die Notation der polyphonen Musik 900-1600*. Breitkopf & Härtel, Wiesbaden.
- Michael Burrell. 2001. Makesound, June. URL: <ftp://mikpos.dyndns.org/pub/src/>.
- Cycling '74. 2005. Max/MSP/Jitter. URL: <http://www.cycling74.com/>.
- Rasmus Ekman. 2003. Coagula. URL: <http://hem.passagen.se/rasmuse/Coagula.htm>.
- Kiyoshi Furukawa, Masaki Fujihata, and Wolfgang Münch. 1999. *Small fish: Kammermusik mit Bildern für Computer und Spieler*, volume 3 of *Digital arts edition*. Cantz, Ostfildern, Germany. 56 S. : Ill. + CD-ROM.
- Meinolf Neuhäuser, Hans Sabel, and Richard Rudolf Klein. 1974. *Bunte Zaubernoten. Schulwerk für den ganzheitlichen Musikunterricht in der Grundschule*. Diesterweg, Frankfurt am Main, Germany.
- Han-Wen Nienhuys and Jan Nieuwenhuizen. 2005. LilyPond, music notation for everyone. URL: <http://lilypond.org/>.
- Nullsoft. 2004. Winamp. URL: <http://www.winamp.com/>.
- Miller Puckette. 2005. Pure Data. URL: <http://www.puredata.org/>.
- Robert Roebing, Vadim Zeitlin, Stefan Cosmor, Julian Smart, Vaclav Slavik, and Robin Dunn. 2005. wxwidgets. URL: <http://www.wxwidgets.org/>.
- Tom Schouten. 2004. Pure Data Packet. URL: <http://zwizwa.fartit.com/pd/pdp/overview.html>.
- Nigel Simeone. 2001. Universal edition history. 2004. Stadtgeburtstag Karlsruhe, June. URL: <http://www.stadtgeburtstag.de/>.
- Karl-Heinz Stockhausen. 1956. Studie II.
- Jessie Suen. 2004. EE/CS 107b. URL: <http://www.its.caltech.edu/~chia/EE107/>.
- Naoyuki Taneda and Ruth Taneda. 1993. *Erziehung zum absoluten Gehör. Ein neuer Weg am Klavier*. Edition Schott, 7894. B. Schott's Söhne, Mainz, Germany.
- Rainer Wehinger. 1970. *Ligeti, Gyorgy: Articulation. An aural score by Rainer Wehinger*. Edition Schott, 6378. B. Schott's Söhne, Mainz, Germany.
- Albert Wellek. 1954. Farbenhören. *MGG – Musik in Geschichte und Gegenwart*, 4:1804–1811.
- Eric Wenger and Edward Spiegel. 2005. Methasynth 4, January. URL: http://www.uisoftware.com/DOCS_PUBLIC/MS4.Tutorials.pdf.
- Henrik Wiese. 2002. *Preface to Concert for Horn and Orchestra No. 4, E flat major, K495*. Edition Henle, HN 704. G. Henle Verlag, München, Germany. URL: <http://www.henle.de/katalog/Vorwort/0704.pdf>.
- Iannis Xenakis. 1978. The UPIC system. URL: <http://membres.lycos.fr/musicand/INSTRUMENT/DIGITAL/UPIC/UPIC.htm>.
2005. Zentrum für Kunst und Medientechnologie. URL: <http://www.zkm.de/>.

