Interactive Visual Music with Csound and HTML5

Michael Gogins[1]

Irreducible Productions
michael.gogins@gmail.com

Abstract. This paper discusses aspects of writing and performing interactive visual music, where the artist controls, in real time, a computerized process that simultaneously generates both visuals and music. An example piece based on Csound and HTML5 is presented.

Keywords: Visual music, generative art, algorithmic composition, computer music, Csound, HTML5

1 Introduction

This paper presents an approach to writing interactive visual music using Csound [1] [2] [3] with HTML5. Artistic and technical problems are discussed, and some solutions are presented in the context of an example piece [4] that runs on csound.node [5] [6], currently the most stable and highest-peforming HTML5 environment for Csound. These techniques also work in Csound for Android [7] [8], PNaCl [9] [10], Emscripten [11] [12], and WebAssembly [14] [6]. In all these, Csound appears in the JavaScript context as a `csound` object that exposes the Csound API [15]. *Note*: By 2018 or so browsers are expected to have deprecated all other "native" execution environments in favor of WebAssembly.

HTML5 is the programming environment of current Web browsers. It is a world standard [16] with vast capabilities [17] that are programmable in JavaScript [18] [19]. In addition to HTML and JavaScript for defining user interfaces, features relevant to visual music include three-dimensional, animated computer graphics (WebGL) and high-resolution audio (WebAudio).

Visual music can mean (a) purely visual displays having a music-like evolution in time; (b) visualizations of music; or (c) a hybrid art in which the author, perhaps using software or other automatic processes, generates both visual and musical forms. This last type is the subject of this paper — in particular, where one process is performed interactively, in an improvisational way, to generate both visuals and sounds. In any case, visual music tends to abstraction, otherwise it would be no different from music videos.

The pioneers of visual music were the typical lot of visionary outsiders [20]. Walt Disney's ***Fantasia*** [21] showed stellar examples. Later, experimental filmmakers created some visual music [22], still later *light shows* [23] became standard at concerts of psychedelic music, and from the late 1970s the *demoscene*

[24] [25] showed programmed animations with musical accompaniment, some of high quality. Visual music became notable in computer music starting perhaps with ***Circles and Rounds*** by Dennis Miller at the 2006 ICMC [26].

2 Technical Problems (and Some Solutions)

Artistic issues are more important than technical ones, but technology must be discussed first as it is the foundation for the art. Historically the technical issues with visual music have been expensive tools, expensive labor, and incompatible standards. As computer power has doubled and redoubled every year or so, the problem of expensive tools has been mooted, the problem of expensive labor has not changed, and the problem of incompatible standards has perhaps worsened.

Visual music started with painting both images and sounds by hand on movie film. Then of course hand-drawn animation became highly developed, followed by all kinds of tricks used in experimental film. Eventually these were built into hardware for editing film or video with "effects," and then into software for computer animation. Today these technologies are collected in commercial applications such as 3ds Studio Max [27], open source software such as Blender [28] and Processing [29], and game engines such as Unreal [30] and Unity [31]. But this profusion of tools has not solved the problem of labor – see the amazing list of credits for any blockbuster animated film – and also has contributed to the problem of incompatible standards. As in other computer-based arts, there is now a Babel of applications and languages that tends to fragment the field as different artists choose different software for different reasons, and thus lose the ability to understand each other on a *technical* level.

All the above-mentioned software packages are just *different containers* for the *same algorithms*: time lines, scene graphs, texture mappings, shaders, convolvers and filters, software synthesizers, etc. And these are *all* present in the JavaScript context of standard Web browsers where they run at high speed by virtue of SIMD, GLSL, and expertly written C++. In short, HTML5 offers a viable solution to the problems of expensive tools and incompatible standards.

Blender, Processing, and the Unity engine offer similar solutions but, as Csound is arguably the most powerful software synthesizer and can be used directly in HTML5 via csound.node or Csound for WebAssembly, Csound in HTML5 offers a standards-based, high-performance platform for visual music.

Technical issues arise not only from the platform, but also from the artistic objectives. These are considered in the next section.

But first, here is an overview of the architecture of the example piece [4] running in csound.node (similar designs would work in any other Csound/HTML5 environment). The piece itself is one Web page with all code, including Csound itself and the Csound orchestra [4, lines 87-239], either embedded in the page or loaded locally. The user interface consists of sliders and key bindings with JavaScript event handlers, defined using the dat.gui library [32] [4, lines 1172-1219]. An embedded style sheet formats the elements [4, lines 7-52]. The generating process is a real-time GLSL [33] animation, a "shader toy" [34] adapted from the work of lomateron [35], which computes an animated fractal at exceptionally high speed on the massively parallel, dedicated GPU of the computer [4, lines 240-297]. JavaScript code samples pixels from the drawing buffer in real time [4, lines 994-1030] and translates them into musical notes [4, lines 974-992] that are sent to a running instance of Csound, which is called by the WebAudio driver to play the sound.

3 Artistic Problems (and Some Solutions)

In visual music, usually the music comes first and then the visuals ("light shows"), or the visuals come first and then the music (experimental films or demos with derivative music). Either way, one half of the visual music equation usually suffers by comparison with the other half. In theory, this imbalance can be righted by generating both visuals and music from the same process:

1. Both visuals and music are generated by the same underlying, more abstract process. This is rare, as usually the processes have already been designed for one purpose or the other.
2. The music generator is sampled or processed to also generate the visuals. This is quite common.
3. The visual generator is sampled or processed to also generate the music. This is not so common, but not unknown.

Understanding the tradeoffs of the second and third options requires analysis.

3.1 Bandwidth and Format Disparities

Both visuals and music can be digitally processed at different levels of abstraction. For visuals, the highest level of abstraction consists of scenes of geometric objects or meshes covered with textures, illuminated by lights, and viewed by a virtual camera; the lowest level of abstraction is a screen of pixels, a thousand or so wide and high, presented at up to 60 or so frames per second.

A perspective rendering of three dimensions is very common, and virtual realities that immerse the viewer in a stereoscopic perspective view are becoming more common. But for the purposes of visual music, the perspective rendering and the stereoscopic rendering are the same: a three-dimensional scene.

For music, the highest level of abstraction is the score, which consists of notes assigned to instruments, which produce actual streams of audio that are further processed and mixed. There are usually a few to a few dozen discrete notes per second. (There can be an intermediate level of abstraction not considered here, grains of sound that stream at a rate of hundreds or thousands per second.) The lowest level of abstraction is 44,100 to 96,000 frames per second of stereo (or, increasingly, multi-channel) audio samples.

There are obvious disparities of data formats and rates between visuals and music. At the highest level of abstraction, dozens to thousands of visual objects are moving, but no more than a dozen or so musical notes are moving. At the lowest level of abstraction, for uncompressed raw data, the bandwidth of high-definition video is on the order of 3,732,480,000 bits per second, whereas the bandwidth of uncompressed high-definition stereo audio is on the order of 4,608,000 bits per second. In reality visual data is more redundant than audio data; a compressed stream of video runs at about 30,000,000 bits per second, whereas a compressed stream of audio runs at about 500,000 bits per second. Hence visual bandwidth runs about 60 times audio bandwidth.

Finally, the visuals are not always computed as objects in a scene; they may be computed directly at the pixel level. This is attractive, because HTML5 environments can execute runtime-compiled "shader" programs, which operate directly on pixels, on the graphics processing unit (GPU) at *much* higher speeds than on the general purpose central processing unit (CPU).

The much greater data bandwidth of visuals is one reason it makes sense to derive the music from the visuals, instead of the other way round. But then it also becomes necessary not only to map the visual data to musical parameters, but also to filter or reduce the density of data – while still preserving a perceptible relationship between the visuals and the music.

3.2 Mapping, Triggering, and Filtering

"Mapping" actually involves *dimensional mapping*, *filtering* to reduce the data bandwidth, and *triggering* musical events. Triggered events may in addition be post-processed, e.g. to tie overlapping notes, or to fit into a harmony.

Dimensional Mapping Mapping visual *objects* to music is complex, and must be considered case by case. Such a mapping amounts to using the visuals as a sort of score for the music. A minimal set of dimensions for visual objects might be the following. Lower-case letters stand for visual attributes, and upper-case letters for musical attributes.

t	Time	Real	Seconds from beginning of performance.
x	Horizontal Cartesian coordinate	Real	Arbitrary units
y	Vertical Cartesian coordinate	Real	Arbitrary units
z	Depthwise Cartesian coordinate	Real	Arbitrary units

For mapping visual objects to musical events, musical attributes can be computed from, or even attached to, the objects in the scene, thus reinforcing its dual role as a score. Mapping visual pixels is more straightforward, as there is the following fixed set of dimensions:

t	Time	Real	Seconds from beginning of performance
x	Horizontal Cartesian coordinate	Integer	0 to image width
y	Vertical Cartesian coordinate	Integer	0 to image heght
h	Hue	Real	0 through 1
s	Saturation	Real	0 through 1
v	Value (brightness)	Real	0 through 1

The dimensions of notes, at a useful minimum, are:

T	Time	Real	Seconds from beginning of performance
I	Instrument	Integer	0 to I_{Max}
K	MIDI key	Real	K_{Min} through K_{Max}
V	MIDI velocity	Real	V_{Min} through V_{Max}
P	Stereo pan	Real	-1 through 1

Given the dimensional units with their minima and maxima, the actual mappings are obvious. For animated visuals [4, lines 922-992]:

$$T = t$$
$$I = \lfloor 1 + x/x_{Max} \rfloor$$
$$K = \lfloor (y/y_{Max})(K_{Max} - K_{Min}) + K_{Min} \rfloor$$
$$V = v(V_{Max} - V_{Min}) + V_{Min}$$
$$P = s2 - 1$$

Such mappings are necessary but not sufficient. Visual bandwidth is still far greater than musical bandwidth, so the number of events must be cut down even further without breaking the perceptible relation between visuals and music.

To accomplish this, musical events can be triggered only from the most salient visual events. Then, the triggered events can be filtered to further cut down the number of events.

Triggering In the retina, a neural network specializes in detecting edges. Here, edges can similarly be used to detect easily perceptible events. (An actual computerized neural network could perhaps be used to identify these features.)

For animated visuals, an edge occurs when, for a single pixel, the color at frame f_t changes at time f_{t+i}. When the value of a pixel changes from a level below a threshold, to a level at or above that threshold, a note on event is triggered [4, lines 1017-1020]; and when the value changes from a level at or above that threshold, to a level below that threshold, a note off event is triggered [4, lines 1021-1024].

This already reduces the visual bandwidth by a considerable amount, as no new musical events are generated at a pixel while its color remains stable from frame to frame, or its value does not cross the threshold.

However, for animated visuals, this still creates too many musical events per second. Additional filtering or sampling must also be used.

Filtering and Sampling The kind of filtering or sampling required obviously differs between objects and pixels. In the case of objects, for example, only the centers of volume could be considered, or even only a certain level of ramification in the tree of objects in the scene graph. In the case of pixels, which is considered here, the actual grid of pixels can be sampled at some modulus of its width and height [4, lines 1003-1007], or along radii from the center, or so on. If only every 1000th pixel is sampled, the artistic intelligibility of the relationship between the visuals and the music may suffer; this must be judged case by case. To restore intelligibility, the sizes of visual features can perhaps be increased.

Experience shows that the correlation between visual and musical events need by no means be constant. If there is some perceptible synchrony every bar or so, that seems to do the job.

4 Improvisational Control

The whole purpose of the approach to visual music discussed here is to put on a show: to improvise a work of visual music involving both visual and audible forms.

For a single performer to do this, the controls must be manageable. The computer mouse and keyboard provide a limited palette of control gestures. But if the process generating the visual music is a fractal or recursive process controlled by a few numerical parameters, a few gestures suffice for playing [4, lines 1312-1346]. A few key combinations can also be dedicated to changing the arrangement or tonality of the music, for example by applying voice-leading transformations that generate chord progressions in the music [4, lines 1279-1309].

References

1. Csound home page, `http://csound.github.io`
2. Lazzarini, V. et al.: Csound: A Sound and Music Computing System. Springer (2016)
3. Boulanger, R. (ed.): The Csound Book. The MIT Press (2000)
4. AuthorA: Csound Visual Music Example, `https://AuthorA.github.io://csound/csound_visual_music.html`. This piece requires NW.js and csound.node to run.
5. NW.js, `https://nwjs.io/`.
6. Gogins, M.: csound.node, `https://github.com/csound/csound/tree/develop/frontends/nwjs`.
7. Yi, S. and Lazzarini, V.: Csound for Android. In: Proceedings of the Linux Audio Conference 2012, pp. 29-34. Stanford (2012).
8. Csound for Android App, `https://play.google.com/store/apps/details?id=com.csounds.Csound6&hl=en`.
9. NaCl and PNaCl, `https://developer.chrome.com/native-client/nacl-and-pnacl`.
10. Csound for PNaCl, `https://github.com/csound/csound/tree/develop/nacl`.
11. emscripten, `http://kripken.github.io/emscripten-site/`.

12. Csound for Emscripten, `https://github.com/csound/csound/tree/develop/emscripten`.
13. WebAssembly, `http://webassembly.org/`.
14. Csound for Emscripten (for WebAssembly see `build-wasm.sh`). `https://github.com/csound/csound/tree/develop/emscripten`.
15. Csound API, `http://csound.github.io/docs/api/index.html`.
16. HTML5, `https://www.w3.org/TR/2016/REC-html51-20161101/`.
17. HTML 5 Test, `https://html5test.com/`.
18. ECMAScript 2016 Language Specification, `http://www.ecma-international.org/ecma-262/7.0/index.html`.
19. Flanagan, D.: JavaScript: The Definitive Guide (6th ed.). O'Reilly (2011).
20. Brougher, K. et al.: Visual Music. Thames & Hudson (2005).
21. Culhane, J. and Walt Disney Productions: Walt Disney's Fantasia. H.N.Abrams (1983).
22. Rees, A.L.: A History of Experimental Film and Video (2nd ed.). British Film Institute (2011).
23. The Joshua Light Show, `http://www.joshualightshow.com/about`.
24. Demoscene Research, `http://www.kameli.net/demoresearch2/`.
25. Demoscene Portal, `http://www.demoscene.info/the-demoscene/`.
26. International Computer Music Association: International Computer Music Conference (2006).
27. 3ds Studio Max, `https://www.autodesk.com/products/3ds-max/overview`.
28. Blender, `https://www.blender.org/`.
29. Processing, `https://www.processing.org/`.
30. Unreal Engine, `https://www.unrealengine.com/`.
31. Unity, `https://unity3d.com/`.
32. dat.gui, `https://github.com/dataarts/dat.gui`.
33. Kessenich, J.: The OpenGL Shading Language: Language Version 4.50. The Khronos Group (2016).
34. Shadertoy, `https://www.shadertoy.com/`.
35. lomateron: Lights pattern generator, `https://www.shadertoy.com/view/Xl3SzB`.
36. Tasajarvi, L: Demoscene: The Art of Real Time. Even Lake Studios (2004).