

Synthetic game audio with Puredata (Summary)

Andy James Farnell
andy@obiwannabe.co.uk

Key points

- Efficient sound effects for real-time game audio.
- Spectral and physical analysis of real world sources.
- Devise concise synthetic models.
- Implement DSP in dataflow (Puredata).
- Parameterise, discrete & continuous world events.
- Test, OSC connection to running game.
- Code generation, embedding with Faust.

Why use synthetic sound? Existing situation using sample data has polynomial growth, worst case factorial. Large virtual worlds need automatic population through default object properties based on physics. Require highly interactive content but a shortage of sound designers and audio programmers makes ad hoc design limited.

Advantages of synthetic audio Very compact, space efficient code can generate many different sounds. Versatile parameterisation makes synthetic sounds seem "alive", unlike repeated samples shuffled randomly. Correct mapping makes parameterisation automatic. Dynamic level of detail with variable cost is possible. Speedy revision of audio assets possible.

Disadvantages of synthetic audio Must be introduced alongside existing data based methods. No commercial games "audio engines" capable of arbitrary real-time reconfigurable DSP graphs, so algorithms must still be hand coded in C/C++ as plug-ins. Development of tool-chains required to allow non-programmers to manipulate synthetic sound objects.

Why Puredata? It is a relatively efficient and easily understandable audio DSP prototyping tool. Abstractions allow reuse and good modularity. Fast to use, productivity is high. Can use "externals", pre-compiled C units to implement more expensive HRTF, Doppler, convolution reverb and localisation functions, as well as STK [4] and waveguide models

from Perry Cook or J.O.Smiths implementations. Potential to be a complete "game audio engine" replacing FMOD/Wwise with some additional work. Additionally we have access to OSC (Open Sound Control) protocol for easy binding to game world events.

Examples Nearly 100 examples for common game audio completed. Some are stand-alone "ambient keypoints", others are dynamic actors for real-time parameterisation.

- Fire
- Water
- Footsteps and rigid body impacts
- Engines and vehicles
- Weather
- Animals
- Weapons and explosions

Parameterisation Much work focuses on driving the models with real-time data. Weather systems correctly attenuating N-Wavelets[7] through atmospheric refraction gives realistic distance effects for thunder, rainfall is linked to wind speed to produce gusty squalls. Efficient footstep modelling is based on analysis of pressure curves in bipedal movement approximated by piecewise polynomial curves. Parameterised by actor weight, incline, speed and surface texture to automatically produce correct sounds[6]. Animals obtained by modelling physiological features, such as birds based on [8], create forests full of versatile virtual creature sounds.

Example - Fire Component analysis from the physics of combustion identifies 9 distinct processes, shown in 1, each of which is separately modelled. Layering these in combinations that reflect real world causal relationships produces the psychoacoustic cues to strongly suggest fire. Although the model components are extremely simple their correct relationship creates a powerful effect. For example, crackling density increases following fuel settling, increased outgassing produces an upsurge in roaring of flames.

Implementation Shown in 2 is fire using the three most dominant components. With each varied separately a wide range of fire textures are available from one small piece of code.

Example - Water Analysis of fluids reveals a sonic structure based on resonances within temporary cavities formed during surface movement. Simple models of bubbles based on decay of emerging spheres [1] are extended to a general model parameterised by depth, rate of flow and impedance to approximate surface turbulence using feedback FM or filtered noise as control signals. Taking the square of the first difference of a low frequency noisy signal, $(\frac{dF}{dt})^2$, with bilinear exponential distribution around 20Hz and modulating sine waves in the 1kHz to 3kHz region produces a very efficient water flow approximation [2]. This work was carried out independently, and in ignorance of the work of Van den Doel [5] while studying GG STokes “Turbulence of a sphere in Newtonian fluid” to obtain singing bubble sounds, but rather happily accords well with Van den Doels paper on fluid sound synthesis. Further, by modulating the center of noise distributions while applying formant filters in correct ratios poured fluids in a vessel have been synthesised³. In one example I demonstrate the complete process of making a cup of tea including filling and boiling the kettle then pouring out the liquid [3].

Machines Man made objects are modelled following their function and structure. Small clicks and bounces for switches and levers build up into complex clockwork sounds when arranged in a synchronous causally ordered fashion. Composite objects like an engine combine explosive sounds from chirp impulse and variable delays with pipe models to create an exhaust system with standing waves. More complex objects like a helicopter comprise an engine and two propellers which can interact to produce different sounds according to the position of the vehicle and listener.

Workflow Data is applied to prototypes through OSC links over UDP sockets from game hooks. Once an object is tweaked to perform correctly it must be converted to C++ code unless a Max/Pd interpreter is run as part of the game code. Data-flow DSP is converted to an intermediate functional signal flow algebra, Faust[9]. We work upwards from the bottom, in reverse from output to signal sources. Faust automatically creates highly optimised C++ code to embed in the game. Objects are controlled at runtime from game world events. Work is needed to

fully automate this process for a faster development environment.

Conclusion and extensions Real-time synthetic effects for native client-side execution are possible and offer many advantages over sample based methods. Pure-data offers an excellent environment to prototype algorithms. The OSC protocol is a very useful way to connect external synthesis code to game world events during testing. Until game audio engines incorporate a Max/Pd interpreter to directly run data-flows code must be hand converted. Faust is an important tool for converting data-flows to optimised C++. For the future, work will continue on extending the range of effects and their efficiency. Small games projects where sound samples may be replaced by synthesis are sought. Future considerations are exploiting parallelism, creating code to run on GPU devices, thinking about network replication for multi-player games, and how the MPEG-4-SA protocol may be extended as a way to deliver synthetic content as “just in time” dynamically built DSP graphs.

Process name	Physical cause	Synthetic production
Crackling	Small scale fuel explosions	Narrow F noise band decays
Hissing	Regular outgassing	White noise
Roaring	Turbulent gaseous combustion at flame front	Cascade biquad 10-25Hz
Creaking	Stress within fuel	Wooden formants on quasi-noise
Bubbling	Liquid phase, resins, water	Swept sine bands in 300Hz range
Whining	Periodic outgassing	Relaxation oscillator model
Fizzing	Aerial conflagration of particles	Very short HF noise grains
Clattering	Settling of fuel	Impact impulses to formant for fuel
Woofing	LFO oxygen intake cycle	LF modulation of all processes

Figure 1: Nine components of synthesised fire.

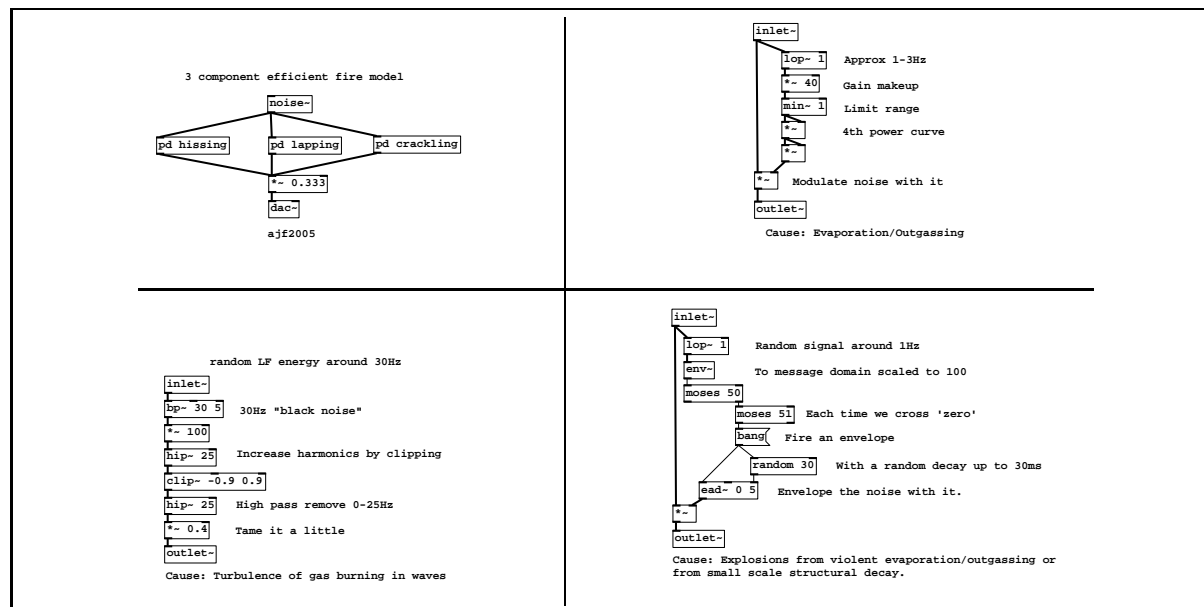


Figure 2: Simplified fire implementation in Pd

References

- [1] Farnell A. *Practical synthetic sound design - Synthesising Bubbles.*
http://www.obiwannabe.co.uk/tutorials/html/tutorial_bubbles.html, 2006.

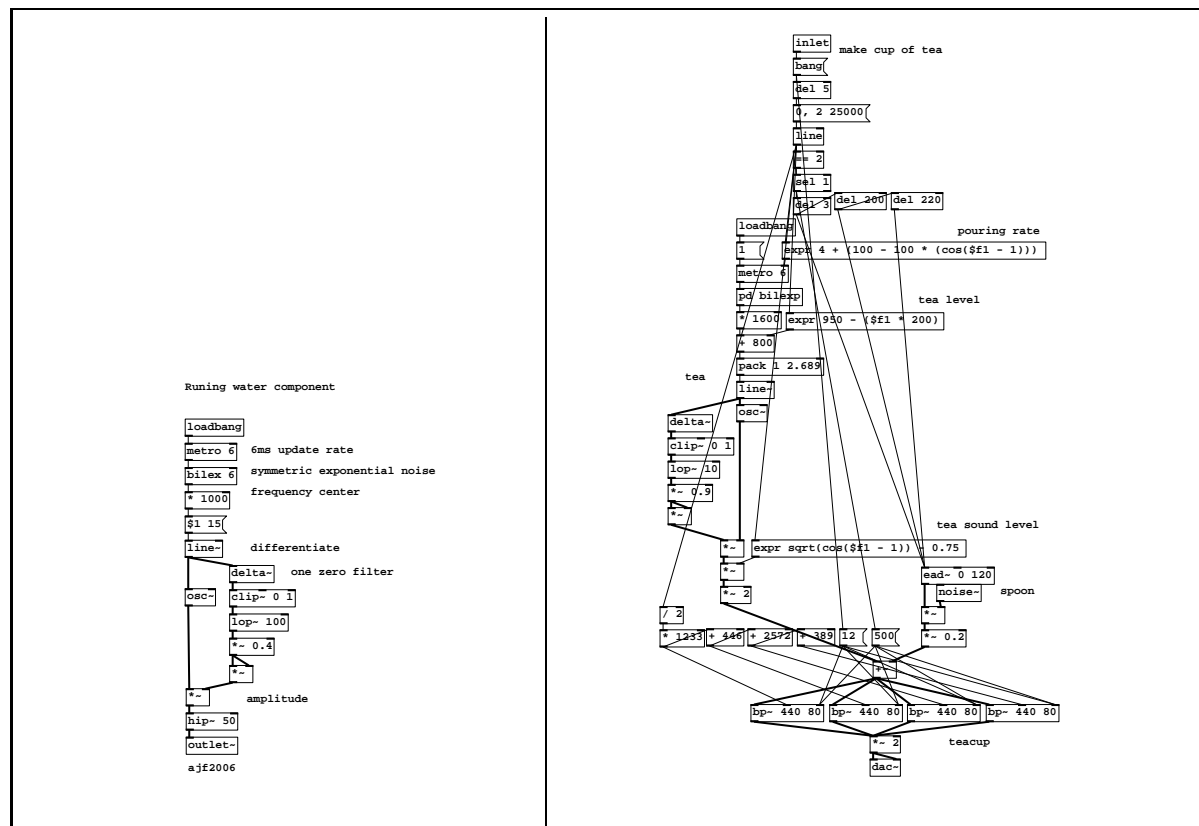


Figure 3: Flowing and poured liquids.

- [2] Farnell A. *Practical synthetic sound design - Synthesising flowing stream*. http://www.obiwannabe.co.uk/tutorials/html/tutorial_stream.html, 2006.
- [3] Farnell A. *Practical synthetic sound design - Synthesising poured liquids*. http://www.obiwannabe.co.uk/tutorials/html/tutorial_tea.html, 2006.
- [4] Perry Cook. *Real sound synthesis for interactive applications*. AK Peters, Ltd ISBN: 1-56881-168-3, 2002.
- [5] Van den Doel K. Physically based models for fluid sounds. *Department of Computer Science, University of British Columbia, Vancouver, Canada*, 2004.
- [6] A. Farnell. Procedural synthetic footsteps for video games and animation. *Proc. Pdcon2007, Monteval, Quebec, Canada.*, 2007.
- [7] D. Ribner, H. S.; Roy. Thunder from tortuous lightning - a computer model made audible. *American Institute of Aeronautics and Astronautics, 7th Aeroacoustics Conference*, 1981.
- [8] Julius O. III Smyth, Tamara; Smith. The syrinx: Nature's hybrid wind instrument. *The Journal of the Acoustical Society of America*, 112:2240-2240, 2002.
- [9] D. Fober Y. Orlarey and S. Letz. An algebra for block diagram languages. *Proceedings of International Computer Music Conference*, page 542547, 2002.

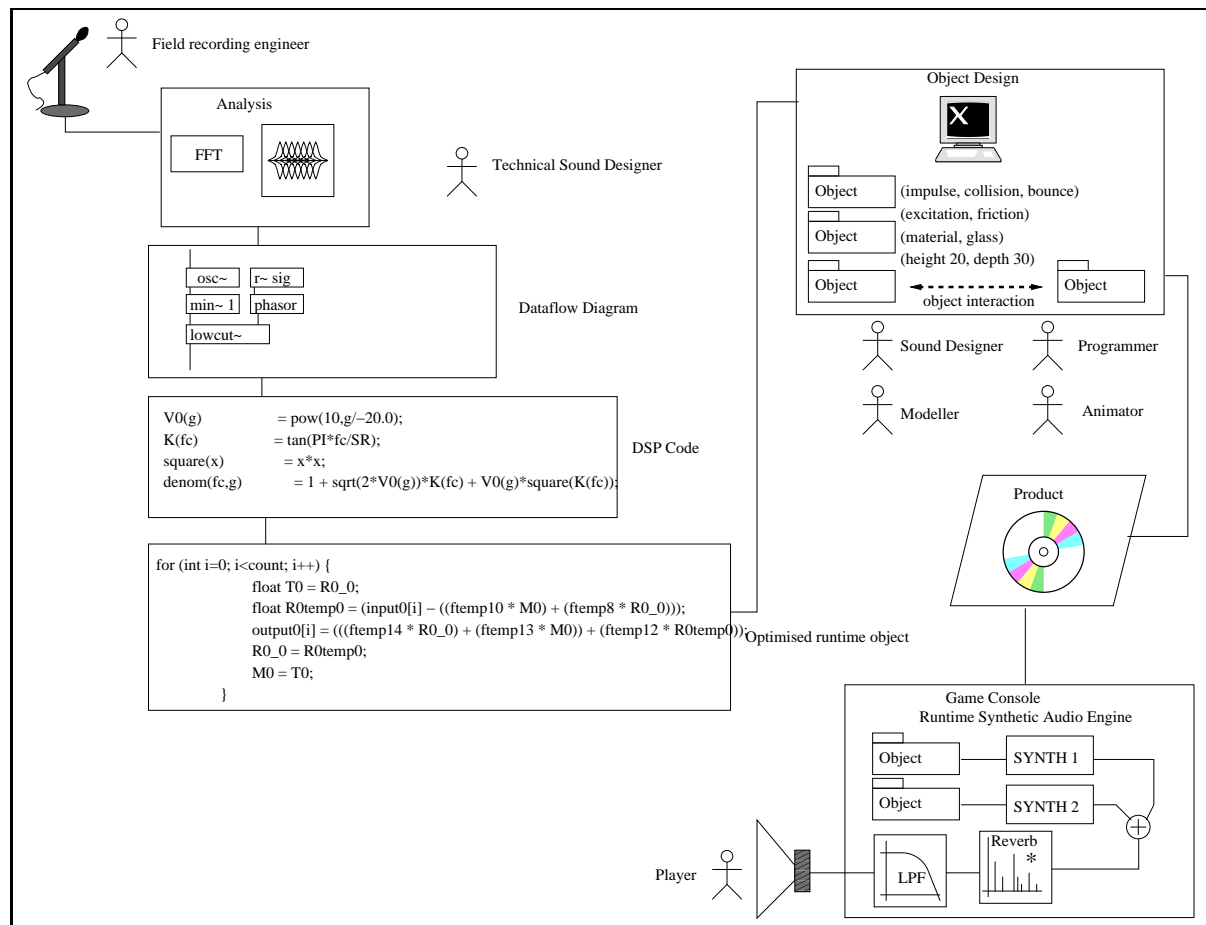


Figure 4: Pd + Faust in game development