Procedural Audio for Video Games:

Are we there yet ?

Nicolas Fournel – Principal Audio Programmer

Sony Computer Entertainment Europe

Overview

- What is procedural audio ?
- How can we implement it in games ?
  - Pre-production
  - Design
  - Implementation
  - Quality Assurance

What is Procedural Audio ?

First, a couple of definitions…

Procedural

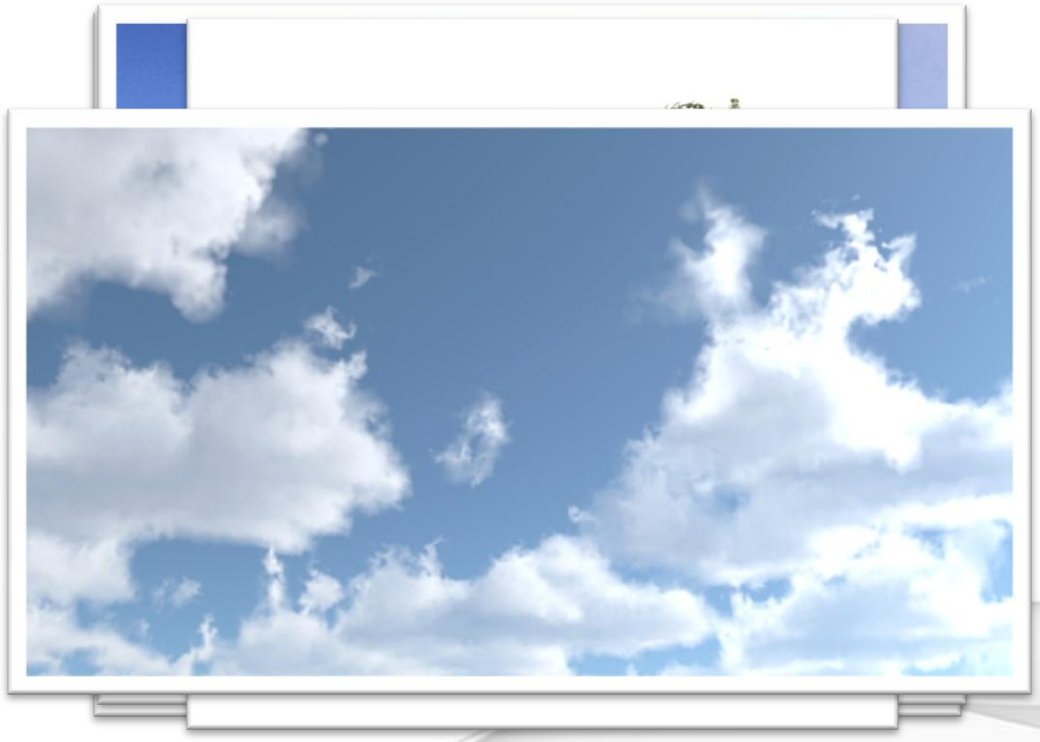
refers to the process that computes a particular function

Procedural content generation

generating content by computing functions

Procedural techniques in other domains

Landscape generation

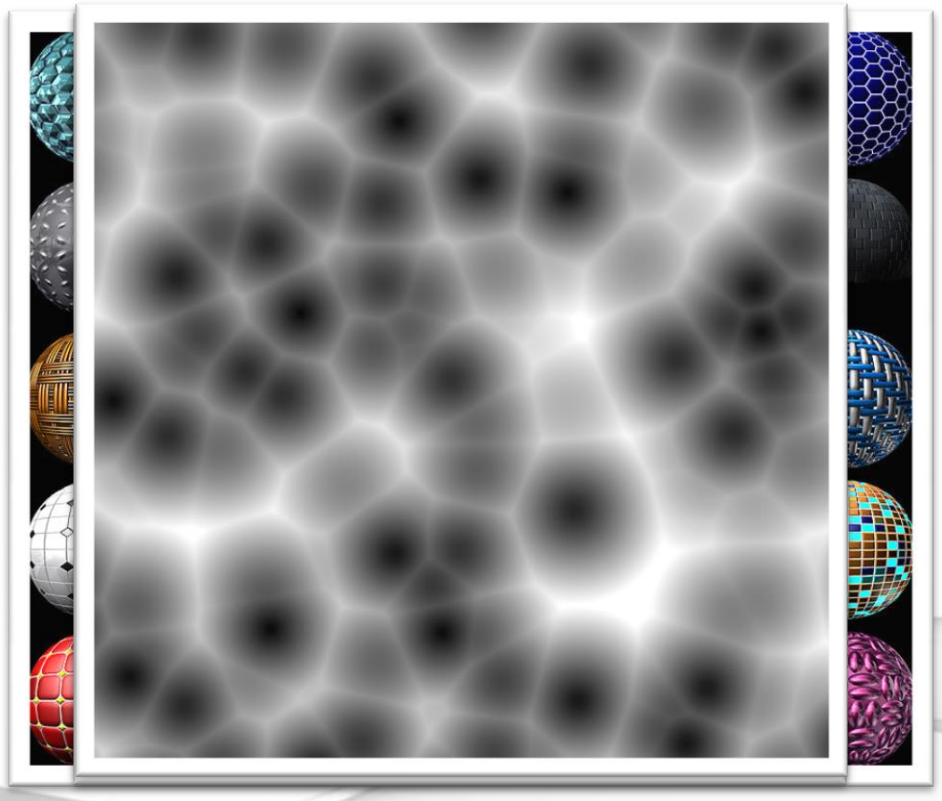


- Fractals (terrain)
- L-systems (plants)
- Perlin noise (clouds)

Procedural techniques in other domains

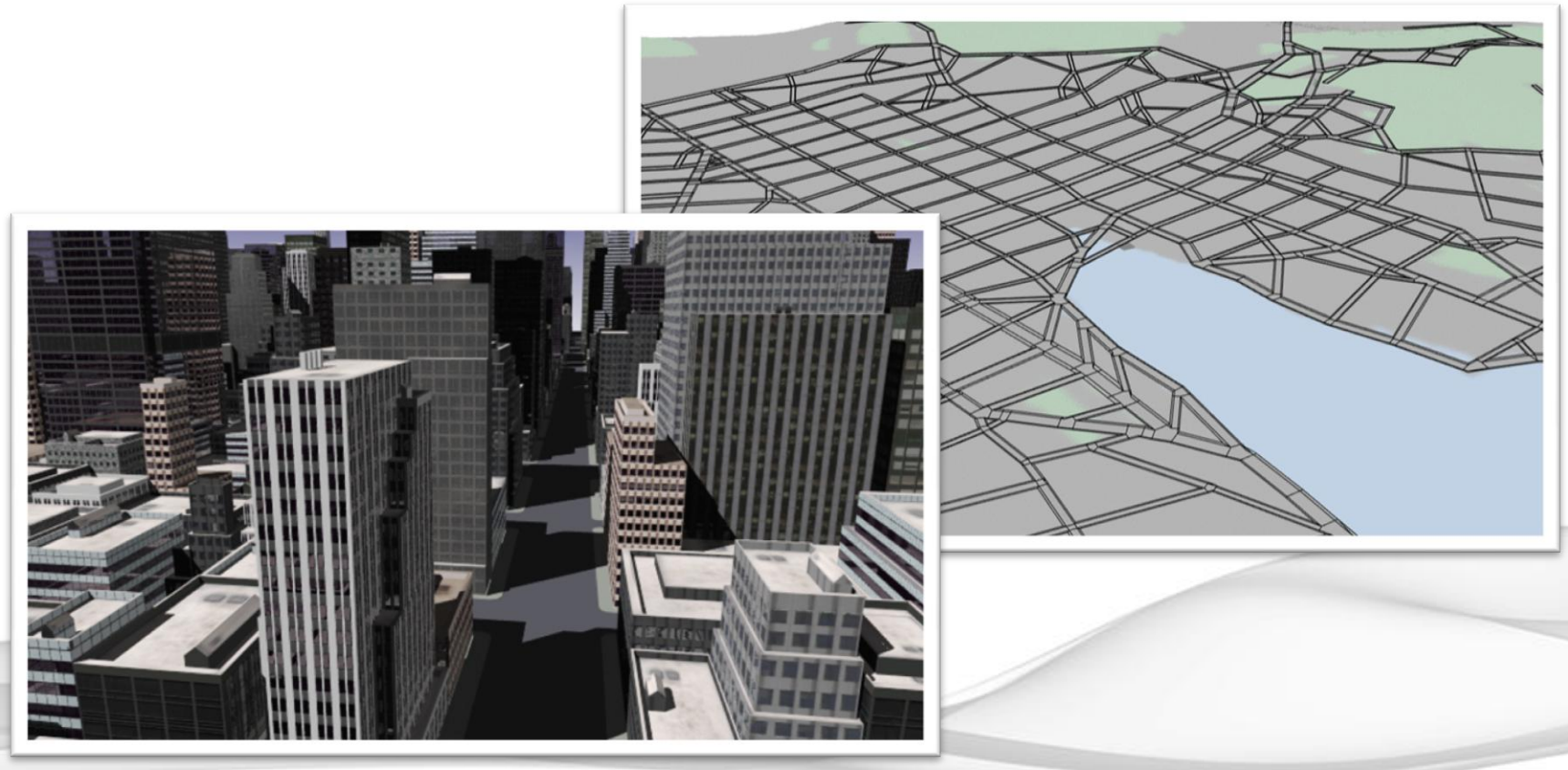
Texture generation

- Perlin noise
- Voronoi diagrams



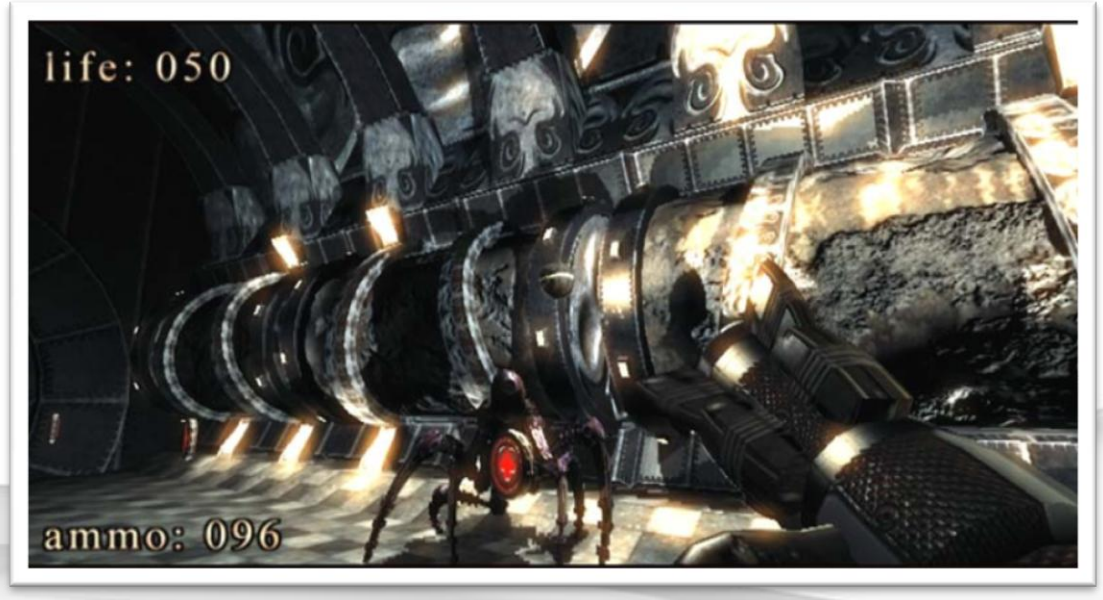
Procedural techniques in other domains

City creation (e.g. CityEngine)



Procedural techniques in other domains

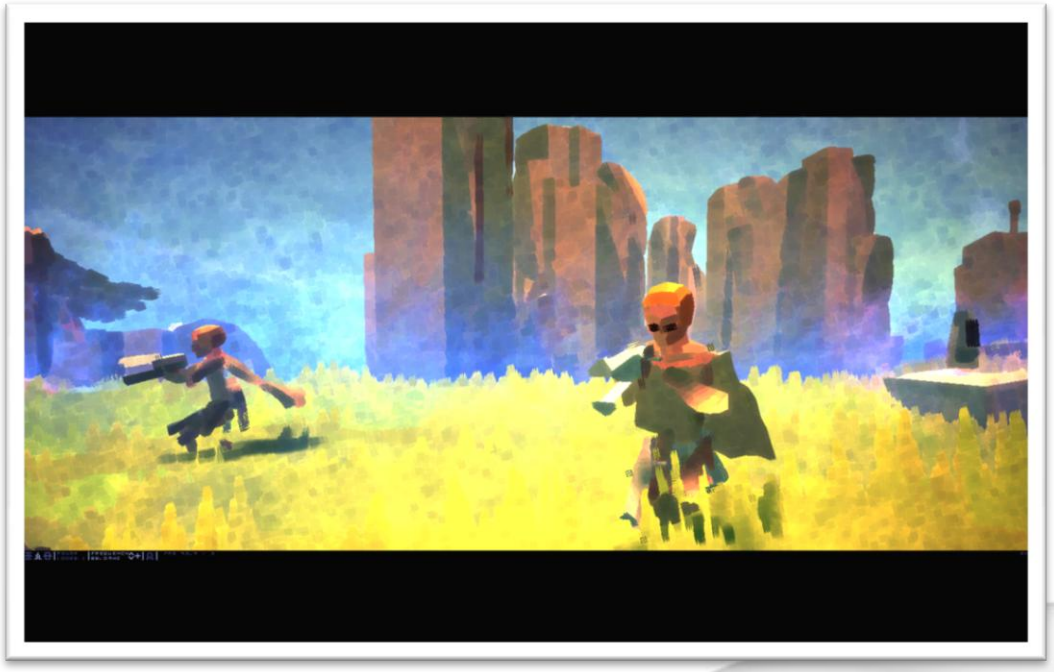
- Demo scene: 64 Kb / 4Kb / 1 Kb intros
- .kkrieger: 3D first person shooter in 96K from Farbrausch



Procedural content in games

A few examples:

- Sentinel
- Elite
- DEFCON
- Spore
- Love



Present in some form or another in a lot of games

What does that teach us ?

Procedural content generation is used:

- due to memory constraints or other technology limitations
- when there is too much content to create
- when we need variations of the same asset
- when the asset changes depending on the game context

What does that teach us ?

- Data is created at run-time
- Is based on a set of rules
- Is controllable by the game engine

Defining Procedural Audio

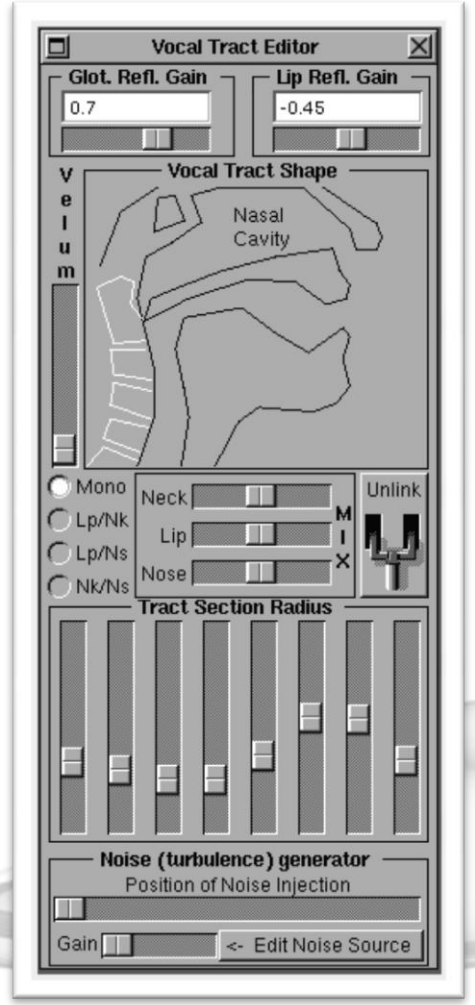
For sound effects:

- Real-time sound synthesis
- With exposed control parameters
- Examples of existing systems:
  - Staccato Systems: racing and footsteps
  - WWISE SoundSeed (Impact and Wind / Whoosh)
  - AudioGaming

Defining Procedural Audio

For dialogue:

- real-time speech synthesis
  e.g. Phonetic Arts, SPASM
- voice manipulation systems
  e.g. gender change, mood etc…

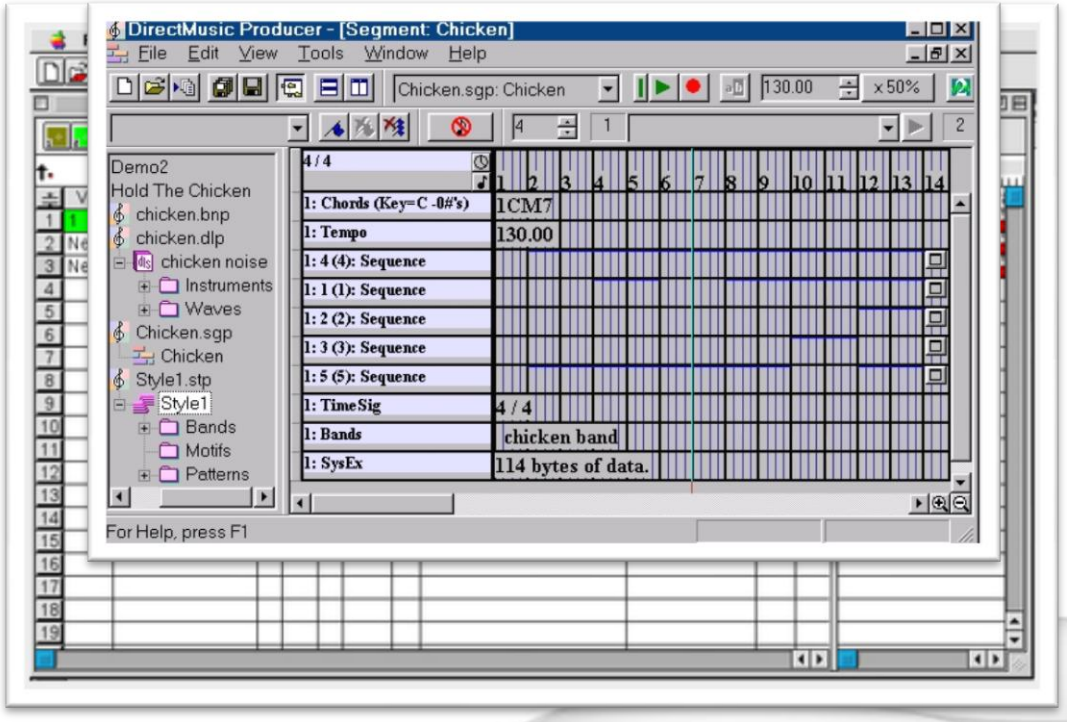


Defining Procedural Audio

For music:

- Interactive music / adaptive music
- Algorithmic composition

SSEYO Koan, Direct Music



Early forms of Procedural Audio

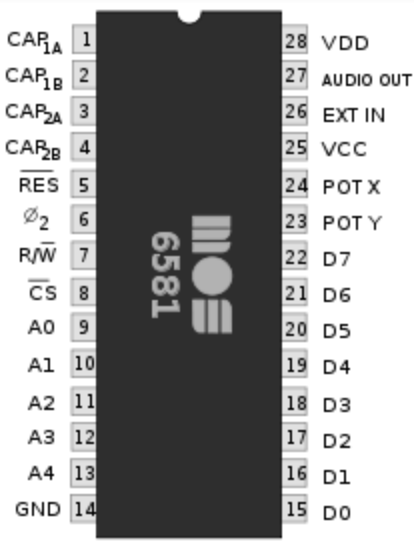
The very first games were already using PA !

- Texas Instrument SN76489
  3 square oscillators + white noise
  (BBC Micro, ColecoVision, Mega drive & Sega Genesis)

- General Instrument AY-3-8910
  (Intellivision, Vectrex, MSX, Atari ST, Oric 1)

- MOS SID (Commodore 64)
  3 oscillators with 4 waveforms + filter + 3 ADSR + 3 ring modulators etc…

- Yamaha OPL2 / OPL3 (Sound Blaster) : FM synthesis



Pre-Production

When to use PA ?

Good candidates:

- Repetitive (e.g. footstep, impacts)
- Large memory footprint (e.g. wind, ocean waves)
- Require a lot of control (e.g. car engine, creature vocalizations)
- Highly dependent on the game physics (e.g. rolling ball, sounds driven by motion controller)
- Just too many of them to be designed (vast universe, user-defined content...)

Obstacles

- No model is available
  - don’t know how to do it !
  - not realistic enough !
  - not enough time to develop one !
- Cost of model is too high and/or not linear
- Lack of skills / tools
  - no synthesis-savvy sound designer / coder
  - no adequate tool chain

Obstacles

- Fear factor / Industry inertia
  - It will replace me !
  - It won't sound good !
  - If it's not broken, don't fix it
- Citation effect required
- Legal issues
  - synthesis techniques patented

    (e.g. waveguides / CCRMA and before that FM synthesis)

Design

Two approaches to Procedural Audio

Bottom-Up:

- examine how the sounds are physically produced
- write a system recreating them

Top-Down

- analyse examples of the sound we want to create
- find the adequate synthesis system to emulate them

Or using fancy words…

- Teleological Modelling

  process of modelling something using physics laws

  (bottom – up approach)

- Ontogenetic Modelling

  process of modelling something based on how it appears / sounds (top –down approach)

Which one to choose ?

Bottom-up approach requirements:

- Knowledge of synthesis
- Knowledge of sound production mechanisms (physics, mechanics, animal anatomy etc…)
- Extra support from programmers

Top-down approach usually more suitable for real-time:

- Less CPU resources
- Less specialized knowledge needed

Ultimately depends on your team skills

Which one to choose ?

Importance of using audio analysis / visualisation software

Basic method:

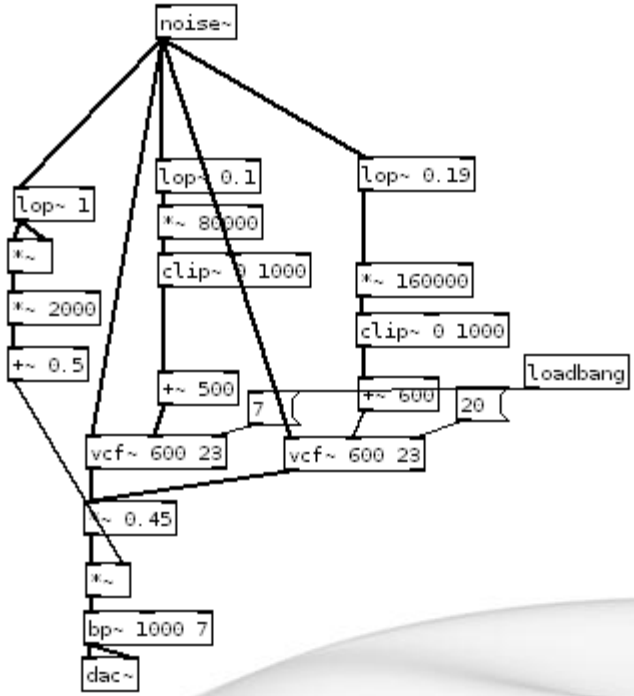
- Select a set of similar samples
- Analyse their defining audio characteristics
- Choose a synthesis model (or combination of models) allowing you to recreate these sounds

Procedural Model Example : Wind

Good example of bottom-up versus top-down design

- Computational fluid dynamics to generate aerodynamic sound (Dobashi / Yamamoto / Nishita)

- Noise generator and bandpass filters (Subtractive synthesis)

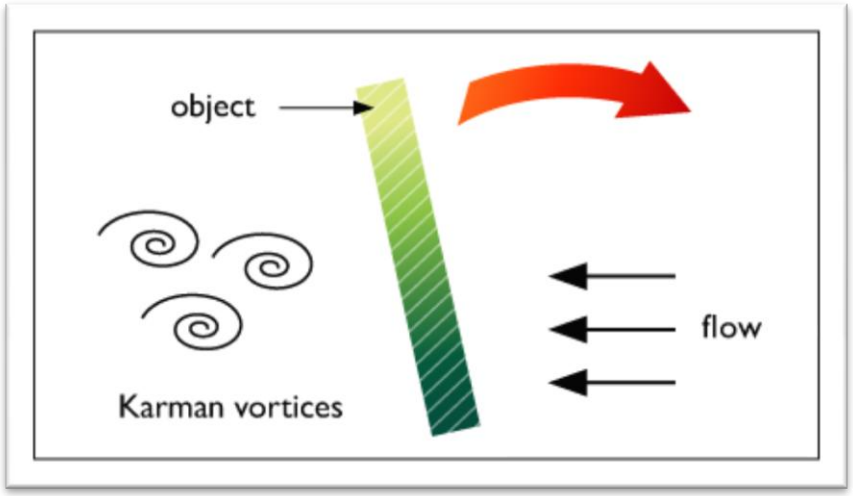


Wind Demo


Procedural Audio Models
Bubbles
Wind
Footsteps
Impact
Monster
Speed
20
Gustiness
0.1000
Variation
0.5000
Whistling
0.2500
Play
Capture
Load Preset
Save Preset
Spark

Procedural Model Example : Whoosh

- Karman vortices are periodically generated behind the object (primary frequency of the aerodynamic sound)
- Using classic subtractive synthesis is cheaper
- Ideal candidate for motion controllers



Procedural Model Example :Whoosh

Heavenly Sword:

- about 30 Mb of whooshes on disk
- about 3 Mb in memory at all times

Recorded whooshes

Subtractive synthesis (SoundSeed)

Aerodynamics computations

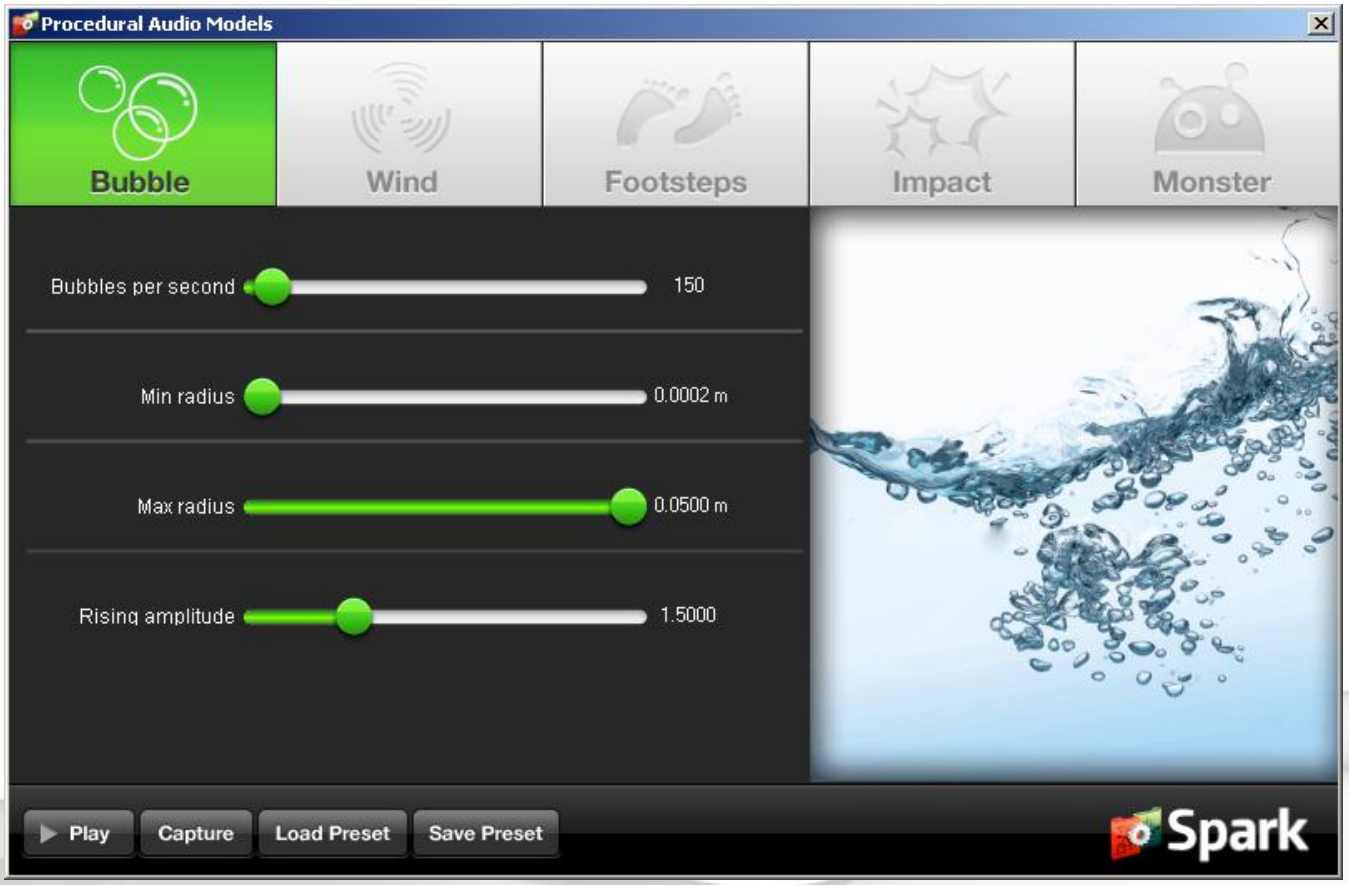
Procedural Model Example
Water / Bubbles

Physics of a bubble is well-known

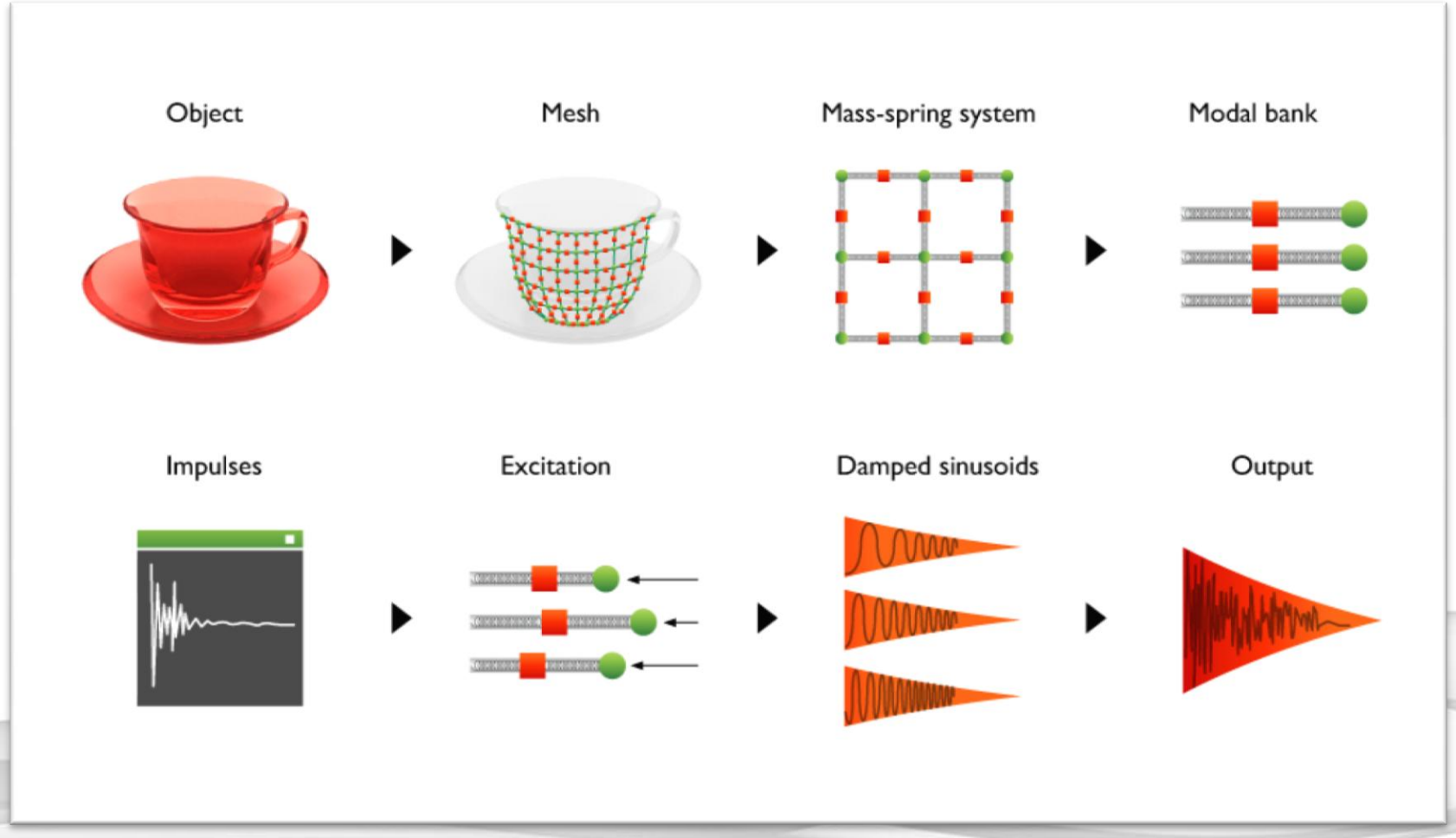
- Impulse response = damped sinusoid
- resonance frequency based on radius
- Energy loss based on simple thermodynamic laws
- Statistical distributions used to generate streams / rain
- Impacts on various surfaces can be simulated

Bubbles generated with procedural audio

Bubbles Demo



Procedural Model Example : Solids



Procedural Model Example : Solids

Other solutions for the analysis part:

- LPC analysis

  Source – Filter separation

- Spectral Analysis

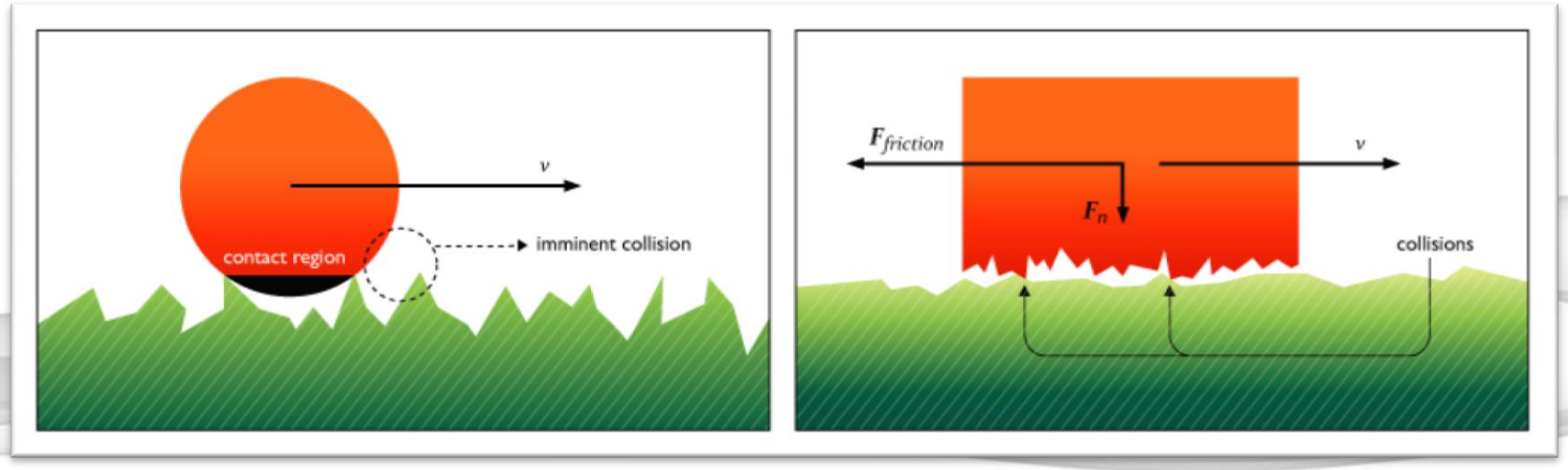
  Track modes, calculate their frequency, amplitude and damping

Procedural Model Example : Solids

Different excitation signals for:

- Impacts (hitting)
- Friction (scraping / rolling / sliding)

Interface with game physics engine / collision manager



Procedural Model Example : Solids

"Physics" bank for Little Big Planet on PSP:

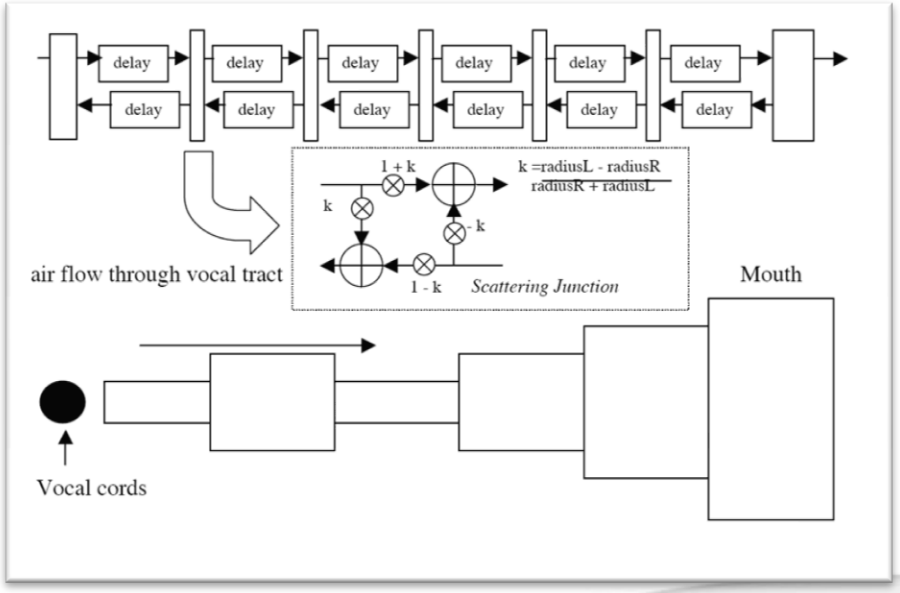
- 85 waveforms
- 60 relatively "complex" Scream scripts
- Extra layer of control with more patches (using with SCEA's Xfade tool)

Impacts generated by procedural audio

Impacts Demo

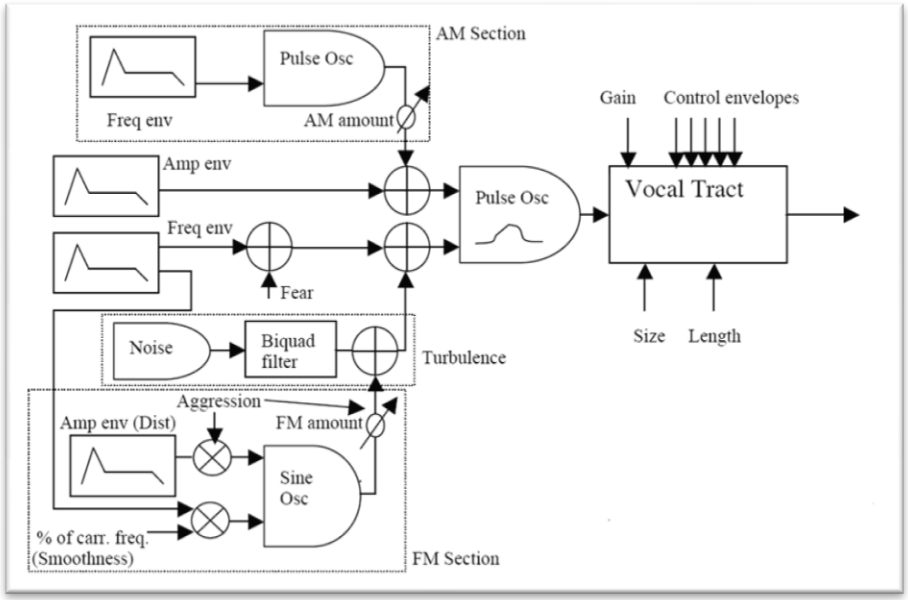
Procedural Model Example : Creature

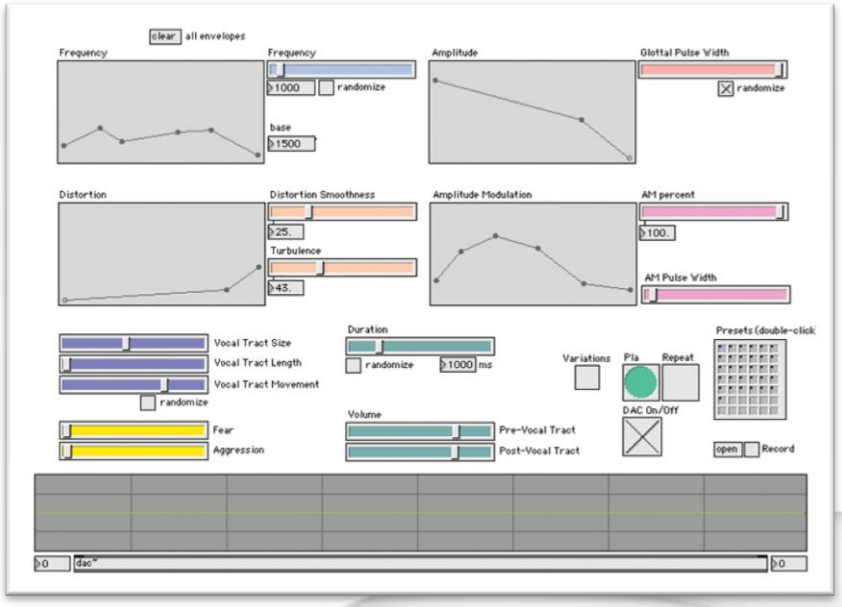
- Physical modelling of the vocal tract (Kelly-Lochbaum model using waveguides)
- Glottal oscillator



Procedural Model Example : Creature

Synthasaurus: an animal vocalization synthesizer from the 90s.





Procedural Model Example : Creature

Eye Pet vocalizations:

- Over a thousand recordings of animals
- 634 waveforms used
- In 95 sound scripts

Eye Pet waveforms

Synthasaurus

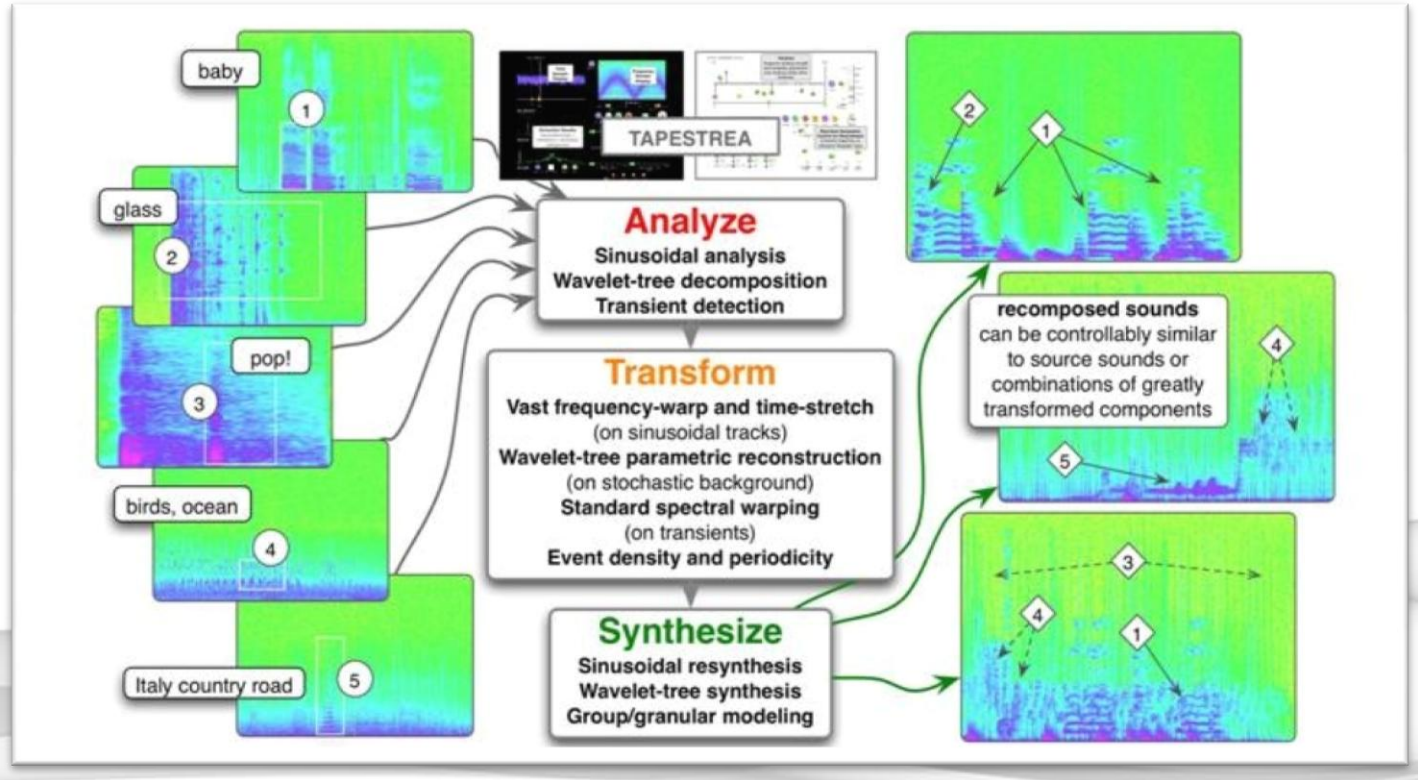
Sound texture synthesis / modelling

A sound texture is usually decomposed into:

- deterministic events
  - composed of highly sinusoidal components
  - often exhibit a pitch
- transient events
  - brief non-sinusoidal sounds
  - e.g. footsteps, glass breaking…
- stochastic background
  - everything else !
  - resynthesis using wavelet-tree learning algorithm

Sound texture synthesis / modelling

Example: Tapestrea from Perry R Cook and co.



Implementation

Implementation Requirements

- Adapted tools
  - higher-level tools to develop procedural audio models
  - adapted pipeline
- Experienced sound designers
  - sound synthesis
  - sound production mechanisms
- Experienced programmers
  - sound synthesis
  - DSP knowledge

Implementation with Scripting

Current scripting solutions:

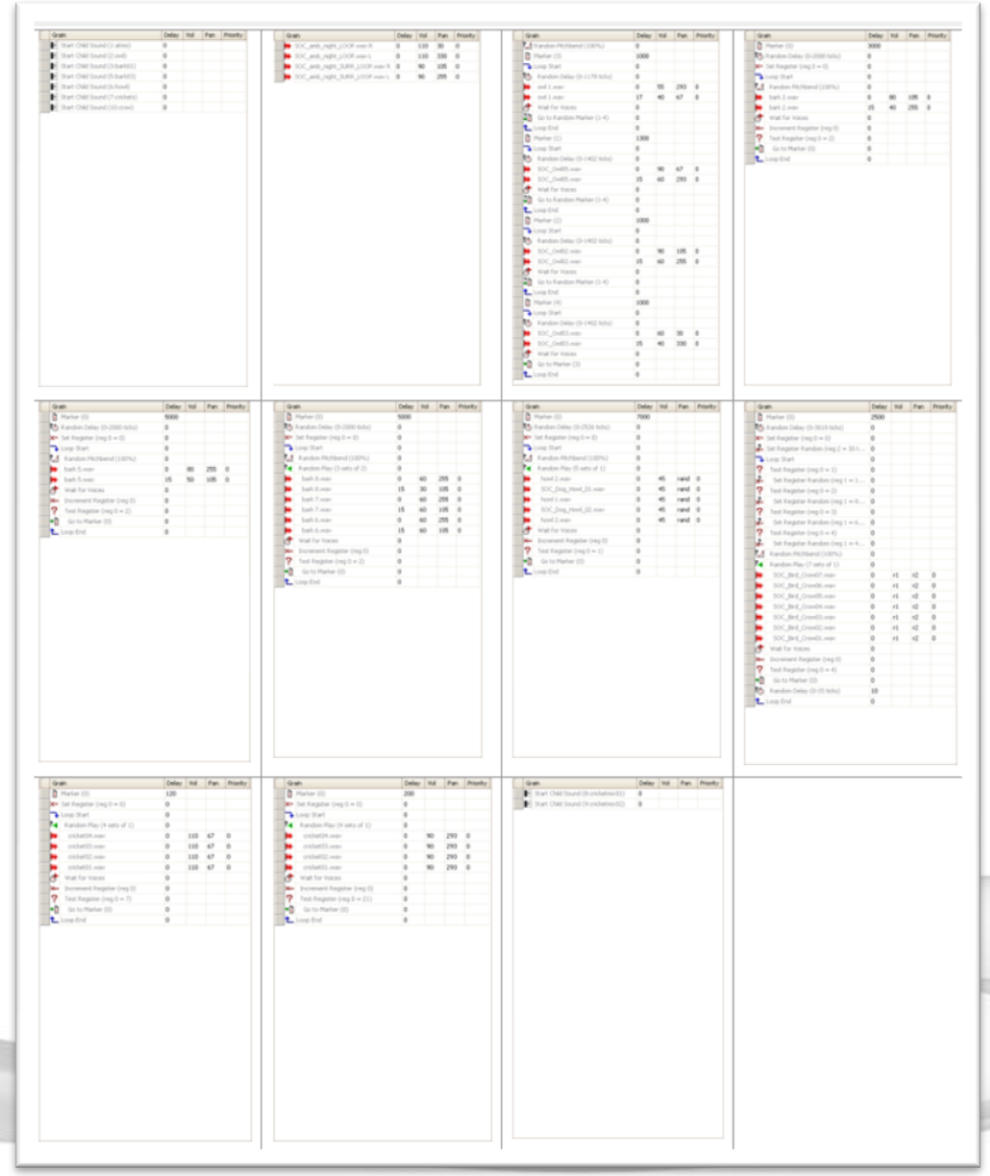
- randomization of assets
- volume / pan / pitch variations
- streaming for big assets

Remaining issues:

- no timbral modifications
- still uses a lot of resources (memory or disk)
- not really dynamic

A “simple” patch in Sony Scream Tool:

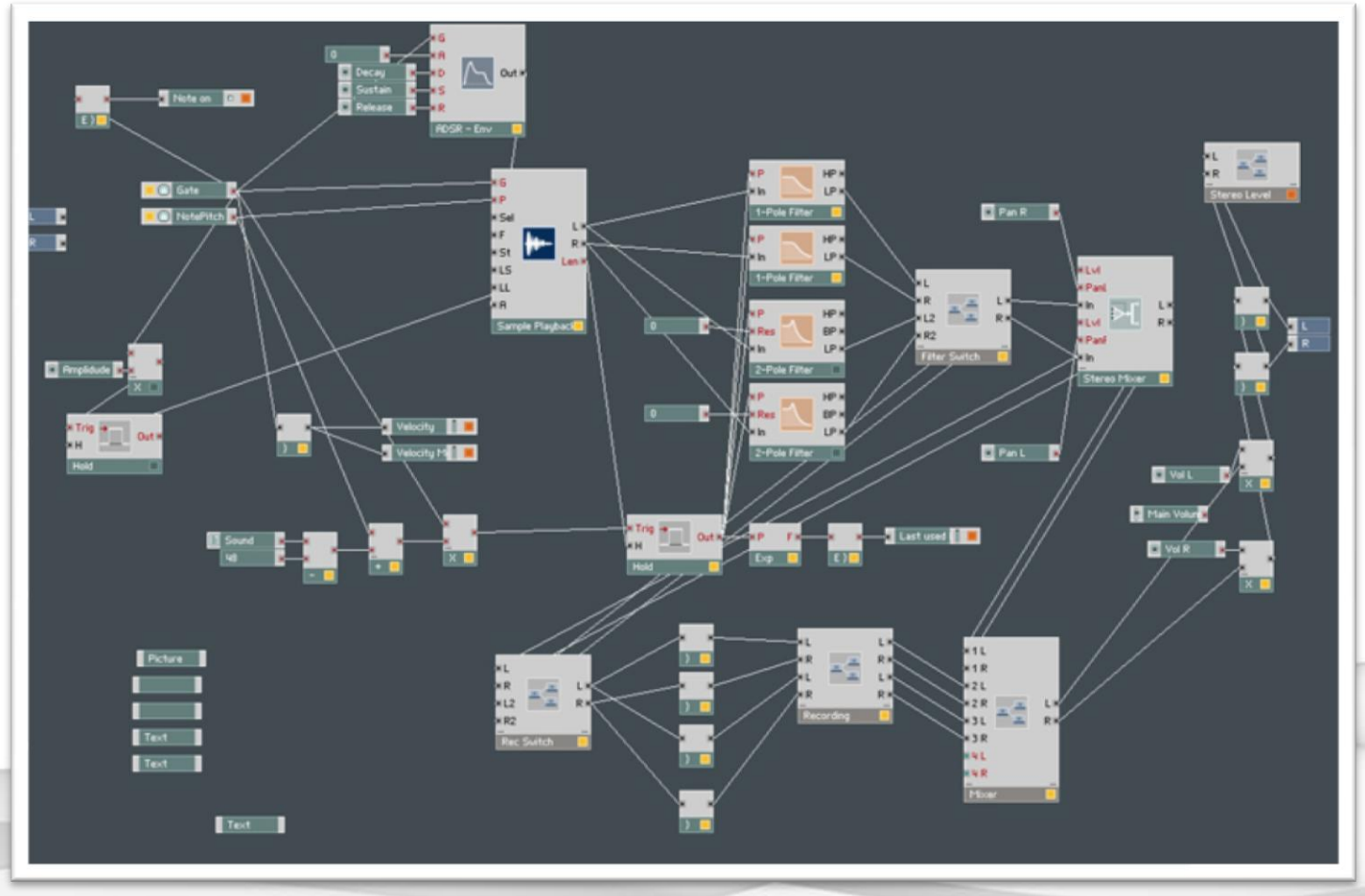
- 11 concurrent scripts
- each “grain” has its own set of parameters



Implementation with Patching

- Tools such as Pure Data / MAX MSP / Reaktor
- Better visualisation of flow and parallel processes
- Better visualisation of where the control parameters arrive in the model
- Sometimes hard to understand due to the granularity of operators

A “simple” patch in Reaktor…



Another solution

Vendors of ready-to-use Procedural Audio models:

- easy to use but…
- limited to available models
- limited to what parameters they allow
- limited to the idea the vendor has of the sound

Examples:

- Staccato Systems already in 2000…
- WWISE SoundSeed series
- AudioGaming

Going further…

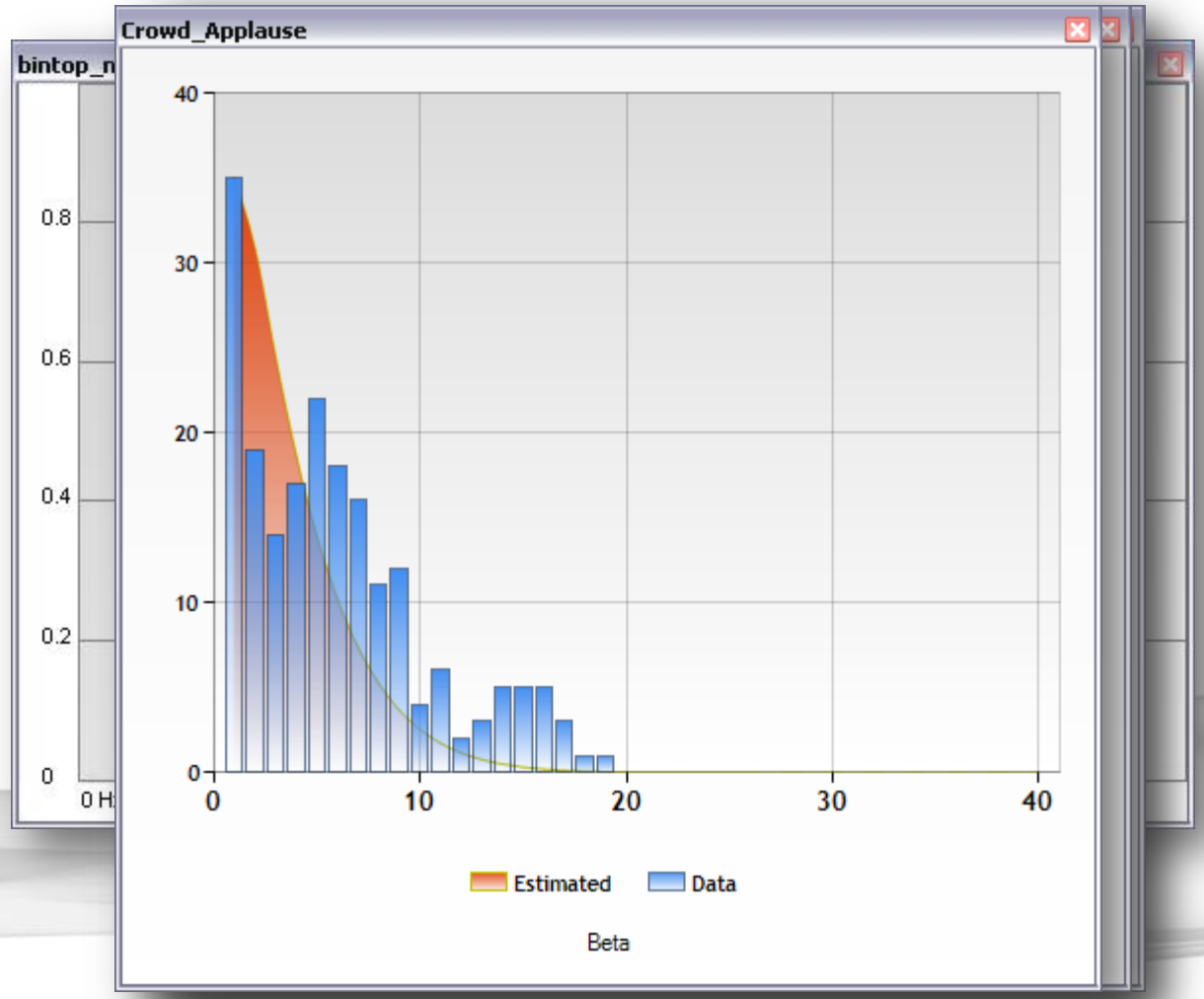
Need for higher-level tools that let the designer:

- create its own model
- specify its own control parameters
- without having an extensive knowledge of synthesis / sound production mechanisms
- without having to rely on third party models

Importance of audio features extraction

- To create models by detecting common features in sounds
- To provide automatic event modelling based on sound analysis
- To put the sound designer back in control

Think asset models, not assets



Implementation: Typical modules

Lots of different ways to organize modules, different levels of granularity

3 main types of modules:

- Event generation: probability distributions
- Audio synthesis: subtractive, modal, granular, F.M, waveguides…
- Parameter Control : envelope generators, Perlin noise, excitation modelling (friction, sliding etc…)

Implementation : Interface

Requires an even greater interaction between sound designer, game designer and programmer

Control parameters can come from a lot of subsystems:

- Animation
- Physics
- AI
- Gameplay

Requires a uniform interface with all game subsystems

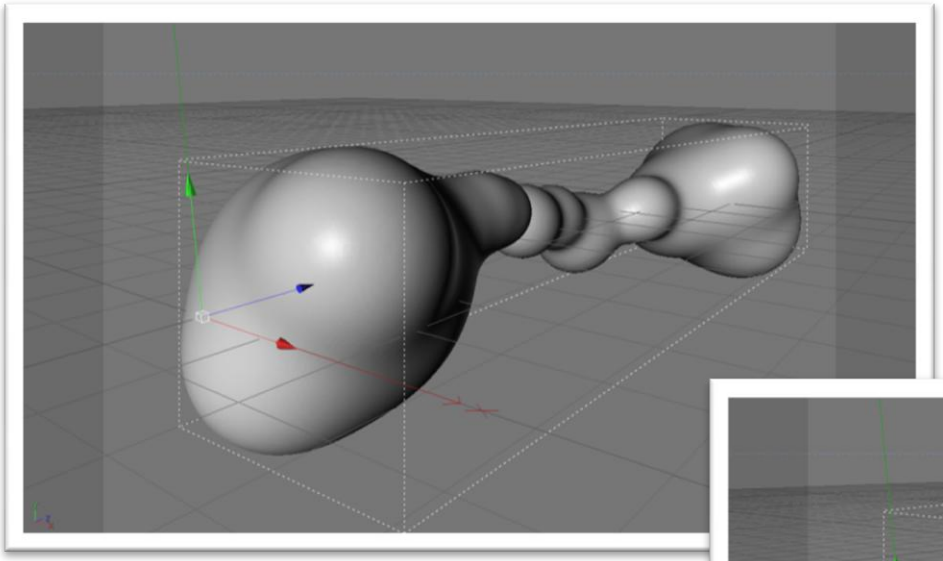
Implementation : Parameters

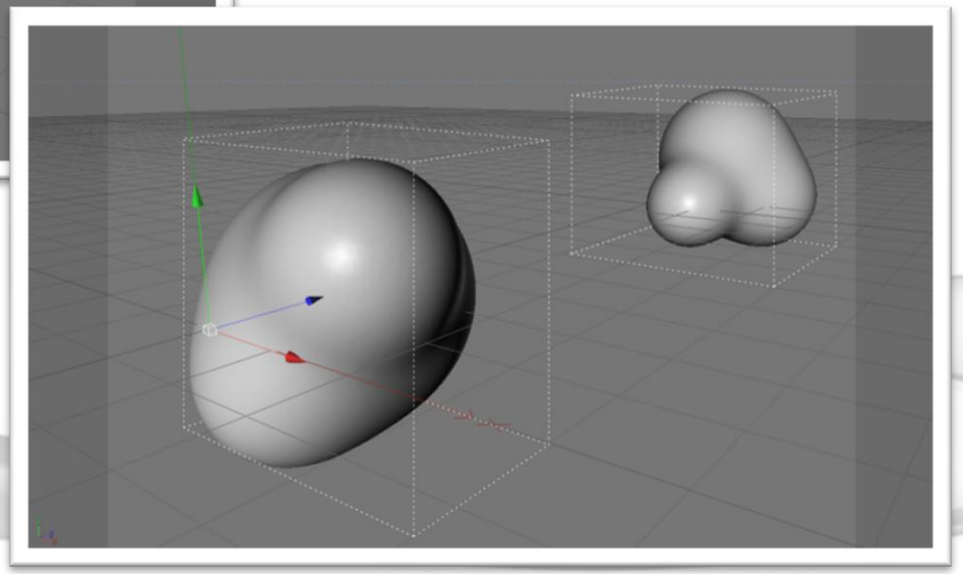
You can add all the parameters you want

It’s a trap !

- Limit the number of parameters
- Limit their range
- Test the stability of the model early

Implementation : Parameter space





Divide parameter space

to create stable models

Implementation: CPU Usage

The bad news

- Highly dependent on model
- Even dependent on parameters ! (e.g. number of grains, main pitch)
- Non linear models (FOF)

It’s not so bad…

- Typical sample playback uses resources also (resampling, filter…)
- Some algorithms are not more CPU hungry than a simple EQ

Implementation: CPU Usage

Mitigating factors:

- Depends if modular / fixed architecture for a few chosen models ("interpreted" a la PD, or "compiled")
- LOD: for different sounds and inside the same sound
- Dependent on update rate (control signal)
- Important to have tools display some metrics about CPU usage in the tools
- Granularity of modules

Quality Assurance

QA: typical sound bugs

- The sound effect is not playing
  - is it loaded ?
  - is it triggered ?
  - is it a voice management issue? Not enough free voices?
  - priority is too low?
- The sound effect is not looping
  - wrong looping points
  - bad settings (must be flagged as looping ?)
  - voice cut off by voice manager

QA: more typical sound bugs

- Wrong volume / panning:
  - wrong 3D settings
  - errors in 3D positioning code ?
- The sound is stuck in looping mode:
  - sfx not stopped
  - hardware voice not released
- Garbage data is played
  - sample data not correctly loaded / encoded / decoded
  - something is writing over our data etc…
  - stuttering → streaming issue

What kind of bugs are they ?

- Easily detectable
- Mostly quantitative bugs
- Do not require specific audio knowledge
- Any tester can be assigned
- There is a known list of possible causes

QA: PA sound bugs

- Synthesis vs. playback: qualitative aspect (sounds like this or that)
- P.A. model more complex and controlled by more subsystems than sample playback
  - harder to describe the exact conditions under which a bug occurs
  - harder to reproduce it
- CPU cost not linear: harder to deal with something not playing…

QA: PA sound bugs

- Fixing the issue is harder
- Modifying the model may be required
- Different structure will not have the same CPU cost or control parameters
- Might bring up new audio glitches

QA: solutions

- Education of testers (ideally a specific audio tester)
- Testers should know about the audio models or be able to refer to them
- The stability of the model must be tested in the tools as much as possible

Are we there yet ?

The good news

- Some models can be implemented very easily
  - Impacts / contacts
  - Footsteps
  - Air / Water
  - …
- They offer a lot of advantages compared to static sounds
- Procedural audio is not necessarily CPU expensive

The bad news

- Not a solution for everything
- It is still harder to implement
- Mostly due to lack of:
  - trained sound designers / programmers / testers
  - adapted tools / run-time
  - ready-to-use models

Solutions

- Get better tools (higher-level, importance of audio features extraction)
- Educate teams across disciplines
- This will help the creation of procedural models database
- Share models across the industry

Thank you !

Any questions ?