Pt3: Sound as a Process

Suono come modello event based/data driven

Vincere la ripetitività è sinonimo di ricerca di realismo in ambito sonoro (ricordiamo che in natura non esistono suoni ripetitivi!). Nel mondo videoludico, costretto costantemente dalla ristrettezza di risorse, la soluzione adottata è quella di adottare il paradigma **data driven**.

Questo modello _guidato dai dati_ in sostanza - l'abbiamo visto ampiamente la scorsa lezione - è costituito da un enorme database che racchiude al suo interno una moltitudine di file audio preprodotti.

Questi audio files vengono messi in riproduzione all'occorrenza, in seguito al verificarsi di particolari eventi (**event based**).

Per garantire maggiore variabilità, i sample che vengono riprodotti si applicano delle **modificazioni in tempo reale** come abbiamo già detto (attenuazione dovuta alla distanza, combinazioni e layering, random e granularità).

La storia del vidogioco ci insegna che l'audio ha sempre rivestito un ruolo fondamentale e che per moltissimi anni è stato il **motore trainante** principale per lo **sviluppo tecnologico** del mezzo.

Ne è la prova il fatto che in molti advertise dell'epoca (di seguito una pubblicità del gioco _Jaws_ dell'Atari, siamo nel 1975) il suono venisse esplicitamente menzionato e se ne vantassero le caratteristiche di realismo!

Un altro esempio l'abbiamo visto esaminando il sintetizzatore audio a 5 voci del NES.
Questo ruolo di preminenza dell'audio è continuato a lungo culminando nel periodo d'oro delle schede audio come la fortunata _Sound Blaster_ e _Roland MT-32_ fino all'avvento del supporto CD, approssimativamente nella metà degli anni '90, periodi in cui il sample di alta qualità (44100@16bit) faceva il suo ingresso.

Grafica per la ricerca del realismo

D'altro canto, per quanto concerne l'aspetto grafico si può evidenziare un processo evolutivo analogo che ha portato il mercato a preferire via via sempre più la grafica tridimensionale a quella storica bidimensionale.

Possiamo dire che il termine "_realismo_", se nel mondo del suono si traduce in _sample_, nella sfera della grafica esso significa _tridimensionale_ (soprattutto in certi generi)!

> Premesso che il realismo non è una condizione indispensabile, anzi, ci sono videogiochi (basti pensare a PacMan, o a diverse produzioni indipendenti come Fez, Limbo, etc...) che tutt'altro ci propongono rispetto ad una esperienza "_realistica_", ce ne occupiamo perchè la ricerca del realismo in ambito grafico è stato anc'essa preponderantein abito videogames.

In un gioco che si basi pesantemente sull'aspetto grafico, l'immagine mostrata a schermo è fondamentale per restituire la senzazione di **realismo**.

Consideriamo un semplice modello 3D costituito da 4 facce triangolari: occorrono 3 (OpenGL ne usa 4 in realtà) valori numerici corrispondenti ai 3 assi cartesiani per identificare la posizione di ciascuno dei suoi vertici nello spazio tridimensionale.

Un modello estrapolato da un moderno videogioco tuttavia è composto da una moltitudine di poligoni, centinaia se non migliaia (_high poly_). Questi vertici e la loro configurazione nello spazio costituisce la **mesh** del modello ma da sola, non basta per creare l'illusione di realismo.

Serve una **texture** da poter mappare sulla mesh che riproduca fedelmente le caratteristiche visive come colori e dettagli dell'oggetto (UV mapping).

Perchè il modello possa essere visto nel mondo tridimensionale occorrono **luci**: ogni oggetto dunque, colpito dai raggi irradiati da tutte le fonti di luce presenti nel mondo virtuale, verrà descritto in modo ancora più realistico, in più se il modello riporta alcuni valori per le grandezze fisice associate ai materiali di cui è composto, come coefficienti di riflessione e assorbimento, e così via, l'effetto potrà essere ancora migliore.

Moltiplichiamo il tutto per la moltitudine di modelli simultaneamente presenti all'interno del mondo 3D virtuale e avremo quanto computato da una componente del game engine, il **rendering engine** (che traspone il tutto su una superficie flat 2D: il nostro schermo), coadiuvato da una speciale componente hardware, la **GPU**, che aiuta accelerando e parallelizzando il processing.

L'interità dei modelli e delle loro mesh non sono fondamentali soltanto per ottenere una immagine 2D in uscita ma anche per creare la cosìdetta **__word geometry__** per il calcolo delle **__collisioni__**, indispensabili per prevedere e computare i comportamenti fisici.

Tutto questo è appannaggio del **__physics engine__** che non si occupa solo di collisioni ma valuta l'interà fisicità del mondo virtuale in cui siamo immersi: masse, densità, velocità e accelerazioni, forze, torsioni.

Ebbene tutto questo viene calcolato di continuo, sempre in funzione dei dati di movimento ottenuti dalle azioni del giocatore, 60 se non più volte al secondo.

Quando la grafica 3D cominciava ad affermarsi nel mondo del videogioco (siamo agli inizi degli anni '90) i modelli erano composti da un numero limitato di poligoni (_low poly_) per una questione di limitazioni della capacità hardware delle macchine del tempo, PC o PlayStation 1 ad esempio.

Eppure negli stessi anni al cinema si poteva assistere alla proiezione di **__Toy Story__**, il primo film di animazione interamente realizzato in computer grafica. Nel film i modelli sono molto più definiti dei loro corrispettivi in ambito videogames, ma questo solo perchè i numerosi calcoli richiesti per renderizzare ciascuno dei frame erano svolti da grandi "_render farm_", niente di comparabile ad una contemporanea console per uso domestico. Inoltre il risultato di un rendering spesso era pronto dopo diverso tempo che i dati erano stati introdotti.

Sotto questa luce, il _low poly_ dei primi videogiochi 3D diventa una prova evidente che la grafica dovesse (e deve) necessariamente essere calcolata in tempo reale!

Interessante notare dunque che l'immagine che vediamo quando giochiamo **__non esiste__** prima del run-time! Quanto si vede è frutto di un calcolo basato su parametri ricavati dalle azione del giocatore.

Suono come processo

Ci potremmo allora chiedere quindi che significato ha la parola **__realismo__** in ambito audio?

Il sample audio invece è una registrazione, e come tale si tratta di un qualche cosa fissato nel tempo e immutabile: una registrazione cattura la perturbazione della densità dell'aria, l'effetto di un movimento nello spazio in un particolare istante ma non ci dice nulla in merito al comportamento.

In questo senso l'audio fruito attraverso i campioni resta un procedimento **__statico__**, si riduce alla pressione del tasto play, una **__fotografia__** fissa ed immutabile anzichè vivida e dinamica.

Vero che alla registrazione possono essere applicate variazioni in tempo reale ma questo non ha nulla a che spartire con quanto avviene in ambito grafico il che fa capire quanto il termine "_realismo_ sonoro" suoni alquanto **__riduttivo__**, soprattutto contando che anche il suono ha la sua importanza:

* il suono riveste un ruolo importante di guida emotiva all'interno del gioco;
* l'audio è una parte importante anche per la narrazione;
* contribuisce alla percezione di _verosimiglianza_ (vedi più avanti);

E' possibile descrivere adeguatamente una scultura con una fotografia?

Niente trucchi da quattro soldi!

In generale, anche per una questione di risorse, l'audio del videogioco è stato per lungo tempo un audio sintetizzato in tempo reale. Un audio che rispecchiava linearmente le azioni del giocatore e le reazioni dell'engine.

Sotto questa luce, i "_trucchi_" descritti poco fa, usati normalmente per ridurre o mascherare la ripetitività, sono in realtà dei rimedi temporanei.

> Attenzione, questo **__non significa che in passato i sample non venissero utilizzati__**; Nintendo, SEGA e ancora prima nelle coin-op, dove possibile, si inserivano sistemi DAC in grado di riprodurre, seppure non con la stessa qualità CD, campioni e sample pre-registrati, soprattutto per la voce.

Un modo alternativo, mutuato dalla storia del videogioco, è quello di pensare il sonoro come **__sintetico__**: in questo modo due suoni associati allo stesso evento non suonerebbero mai esattamente allo stesso modo.

Si tratta in fondo di una caratteristica peculiare della sintesi sottrattica, la quale fa uso del rumore come forma d'onda base. Anche se le differenze potrebbero essere estremamente sottili, il nostro **__cervello__** ha la capacità particolare di riconoscere queste sorgenti come "_vive_".

Inoltre, l'approccio _data driven_ ha i suoi svantaggi vediamo un paio di esempi:

La porta

Supponiamo di avvicinarci ad una abitazione sconosciuta. L'intento è quello di entravi senza destare l'attenzione di chi sta all'interno. Nel gioco guideremo il nostro avatar fin sulla soglia e, a questo punto, a seconda dell'ispirazione del momento, potremo aprire la porta di colpo e preciparci all'interno oppure socchiuderla lentamente e magari interromperci al primo sospetto di non essere soli.

Secondo l'approccio tradizionale _sample based_ un campione preregistrato di scricchiolio viene riprodoto non appena si verifica l'evento "_collisione_" tra il nostro avatar e l'oggetto _porta_. Specie se l'audio file è lungo si nota che la riproduzione dell'audio file prosegue, anche se l'azione sulla porta si interrompe dopo breve.

Anche nel caso il game sound engine utilizzasse meccanismi di dissolvenza in uscita per temporizzare correttamente il livello, comunque risulterebbe chiara la disomogeineità tra i domini grafico e uditivo.

Immaginiamo cosa accadrebbe se poi scostassimo la porta ripetutamente, avanti ed indietro...

Lastra percossa

Alcuni oggetti come le campane o i tubi creano differenti suoni in base al proprio stato. un tubo che è già in vibrazione e che venga colpito in un punto diverso, incorporerà le nuove eccitazioni nel pattern di vibrazione a creare un suono diverso rispetto a quello che avrebbe fatto se colpito in stato di quiete. Niente di simile è ottenibile usando i campioni.

Immaginiamoci con un arma di fronte alla fortificazione di un piccolo bunker dal quale dobbiamo stanare i nostri nemici. Cominciamo a sparare e colpiamo ripetutamente una delle lastre di metallo che rivestono l'esterno della costruzione.

Ad ogni impatto (di un proiettile sulla lastra) la lastra viene eccitata e di tutti i possibili modi strutturali saranno quelli primari ad assorbire il maggior quantitativo di energia e, come risultato, percepiremo sempre più chiaramente le frequenze di risonanza della lastra mano a mano che i proiettili continuano ad incidere su essa.

Lo stesso effetto non è ottenibile riproducendo lo stesso audiofile ripetuttamente per ogni nuovo impatto.

La sintesi in musica

Se, i videogiochi moderni hanno via via preferito i campioni, i paradigmi di audio sintetizzato, più tradizionali se vogliamo, sono sopravissuti in una parte del mercato legato all'audio: la **musica** e gli **strumenti musicali**. In questi campi ci si accorge subito delle limitazion intrinseche dell'audio per campioni (il sample è un _tradimento_ della realtà), per questo la ricerca e lo sviluppo di nuovi sistemi di sintesi continua a progredire.

Nascono i primi sistemi di sintesi basati su **modelli**: i fenomeni della fisica del suono vengono studiati e sintetizzati in modelli matematici e poi innestati all'interno dei circuiti integrati o dei software per computer.

Quando si vuole simulare i suoni dei vecchi sintetizzatori analogici, si sviluppano tecnologie volte a riprodurre in digitale tutte le idiosincrasie dei componenti elettrici discreti che li costituivano e nascono i modelli **virtual analog**.

ad esempio: la tecnologia **True Analog Emulation (TAE)** di Arturia, usata in plugin come il _mini V_ o l'_arp-2600_. Il **Pod** di Line6 implementa un suond engine in grado di simulare molti preamplificatori per chitarra e basso, cabinets e acustia degli ambienti. Anche l'italiana **Acustica-Audio** usa molta tecnologia interessante all'interno dei loro plugin come il _Nebula3_ ad esempio, che "_is a multi-effect plug-in that is able to emulate and replicate several types of audio equipment and uses libraries which are created using a sophisticated “sampling approach” making it possible to “record” aspects of the sound of audio devices and play them back_". Interessantissimi anche i lavori dell'italiana **AudioThing**; I virtual instruments della svizzera **Togu Audio Line (Tal)**; **Hexter**, sintetizzatore che emula gli strumenti Yamaha serie X;

Quando invece si vuole simulare strumenti musicali acustici o elettroacustici nascono modelli volti a riprodurre i fenomeni fisici delle sollecitazioni, vibrazioni, attenuazioni dei materiali: nasce il **physical modelling**.

* **Arturia Stage-73 V**; Antares **Auto-tune** oppure **Throath**; Celemony **Melodyne** e **Capstan**, **Izotope RX**. Altri esempi potrebbero essere l'**Aerophone AE-10** di Roland, che fa

uso dell'engine di sintesi per modelli _SuperNATURAL_ in parallelo con la tradizionale sintesi per campioni; **Native Instruments B4**, etc… i pianoforti virtuali di **PianoTeq**;

> Solo un appunto interessante a proposito di **Pianoteq**: il peso del software è di soli 40MB. In confronto al **Ravenscroft Virtual Piano 275** - che fa uso di 17.000 samples - che richiede 6GB per funzionare (perchè i dati sono compressi) e un HD a stato solido per garantire una performance ottimale!

Esistono da diversi anni scuole di pensiero volte a traslare questi metodi di sintesi, da tempo diffusisi sul mercato e molto apprezzati per le loro qualità, nel mondo videoludico e dell'interazione più in generale.

Pionieri di questa filosofia di pensiero sono persone come **Perry Cook** e **Andy Farnell**, solo per citare i più noti, i quali ritengono possibile derivare dagli studi fatti fino ad ora, modelli finalizzati non tanto a simulare suoni appartenenti al dominio musicale ma piuttosto volti a sintetizzare una moltitudine di classi sonore associate ad oggetti e fenomeni più generali: così da rendere possibile la sintesi, potenziale, di ogni tipo di suono possibile.

L'audio procedurale

Da questo punto di vista il suono nel videogioco diventa un modello inteso come processo (**sound as a process**), in contrasto con il paradigma precedente di _event driven/data driven model_.

Il termine spesso associato a questo paradigma è **audio procedurale**, eccone una definizione:

> *“Procedural audio is non-linear, often synthetic sound, created in real time according to a set of programmatic rules and live input.” — "An introduction to procedural audio and its application in computer games.” by Andy Farnell*

Processo guidato da uno stream continuo di dati provenienti dall'interazione dell'utente, manipolati e usati come parametri per controllare in tempo reale una serie di algoritmi che sintetizzano audio.

A ben vedere il concetto di audio procedurale non ci è del tutto estraneo; un esempio a cui siamo abituati è il _riverbero digitale_ che usiamo tutti i giorni sottoforma di plug-in o di outboard fisico in studio.

Esempi in PureData tratti principalmente dal lavoro di Andy Farnell...

Behaviour, Model, implementation

TODO: argomenta immagini raster vs. vettoriali.

TODO: altro esmpio della torta e della ricetta.

Vantaggi e svantaggi del paradigma procedurale

Tra i vantaggi possiamo considerare:

Posticipazione

Mentre l'atto di registrare un suono è un azione che fissa nel tempo senza lasciare alcune possibilità di intervento successivo, l'audio procedurale è dinamico e lascia che molte delle decisioni, anche strutturali, vengano rimandate al real-time.

Talvolta è semplicemente impossibile conoscere a priori quale sarà il suono che il gioco dovrà riprodurre e, di conseguenza, impossibile sapere come registrarlo o realizzarlo in fase di produzione. In un caso come questo, il moderno sound designer potrebbe avvantaggiarsi dell'audio procedurale e predisponendo un modello per il suono e lasciare che questo venga confezionato "_durante_" il gioco.

Un esempio pratico di quanto detto lo si trova nel videogame "_No Man Sky_". Il gioco si basa interamente su contenuti generati proceduralmente (non escluivamente sonori). Proprio per la natura del gioco, Paul Weir il suond designer di _Hello Games_, intento a risolvere il problema del suono delle creature, diversissime tra loro e inconoscibili prima della partita, ha realizzato il tool _VocAlien_: una sorta di plugin che integrato nel game audio engine si occupasse di sintetizzare i suoni necessari secondo dei parametri di volta in volta diversi, passatigli dall'engine di gioco.

Differimento e Variabilità

Caratteristica fondamentale del suono procedurale che garantisce la possibilità pressochè completa di modificazione del suono in tempo reale e di produrre in questo modo risultati sonori anche molto diversi tra loro pur facendo capo ad uno stesso modello.

Costo dinamico (Dynamic Level of Details) e LOAD

Il suono sintetico si dimostra vincente rispetto all'approccio _data driven_ quando si debbano descrive ampie scene con innumerevoli sorgenti sonore (tra 100 e 1000 sorgenti concorrenti).

I dati sottoforma di campione hanno un costo fisso (lo streaming dati da disco non cambia se il suono deve essere riprodotto completamente dry oppure deve subire real-time processings).
Inoltre, la densità di suoni che possono essere riprodotti in una scena ha un limite:
* sia in termini di accuratezza del mix dei vari suoni (più suoni = più difficile sommarli tra loro, problemi di dinamica e saturazione);
* sia dal punto di vista psicoacustico (alcuni dei contibuti sonori sono inutili all'atto pratico perchè non posso essere uditi per via dei diversi mascheramenti, in tempo o in frequenza).

Già quando si arriva a sommare 100 suoni simultanei, si può vedere che il contributo di ogni nuovo suono aggiuntivo, in rapporto al costo computazionale associato, diminuisce.

D'altro canto, una approccio sintetico può offrire un costo variabile ad ogni sorgente: un esempio potrebbe essere il suono di un bicchiere che si infrange cadendo a terra.

Un metodo sintetico potrebbe produrre un suono molto "_realistico_" e mantenere ancora un alto grado di correlazione audio-video rimpiazzando i frammenti perimetrali con singoli segnali sinusoidali o granuli di rumore, fornendo un alto grado di dettaglio per quei frammenti che cadono vicino al player.

Il paradigma procedurale permette il LOAD - Level Of Audio Details - (che sfrutta le conoscenze in ambito di acustica e psico-acustica) caricamento dinamico sulla CPU esattamente come già fa la parte grafica cone il **mipmapping**.

Consideriamo una esempio quale potrebbe essere un suono in grande lontananza:
nel caso di un motore audio tradizionale, al suono verrebbe applicato notevole processing DSP in tempo reale da far sì che il sample - attenuato e filtrato - dia l'impressione di provenire da lontano. Tuttavia, nonostante il suono ne risulti pesantemente modificato rispetto all'originale, comunque il costo in termini di risorse sarebbe invariato rispetto alla normale riproduzione di un audio file.

Nel caso il problema venga affrontato con il paradigma procedurale invece, soprattutto facendosi forti del fatto che il modello è stratificato e costituito da più parti indipendenti tra loro, il carico di lavoro potrebbe essere ridotto: certe componenti del suono, proprio per via del fatto che la sorgente sonora è ascoltata da grande distanza, potrebbero essere del tutto tralasciate rispetto ad altre. Il lavoro del processore sarebbe avvantaggato.

Lo stesso dicasi se il suono riprodotto da un modello si trova riprodotto in associazione con altri suoni che causerebbero il _mascheramento_ nel tempo o in frequenza di alcune sue componenti.

Default forms

Dal momento che la **crescita** dei sounds assets è **combinatoria**, diventa difficile provvedere alla creazione di tutti i suond assets necessari mano a mano che il mondo virtuale cresce. Il vantaggio di un modello procedurale è che il suono può essere generato in modo automatico derivando le proprietà dagli oggetti presenti nel gioco.

Questo **non elimina** la figura del **sound designer**, il quale interviene laddove alcuni suoni necessitino di particolari caratteristiche perchè più importanti per la narrazione, ma **garantisce** che ogni oggetto abbia sempre un **suono di default** associato, senza incorrere così nel rischio che qualche evento sonoro non possa essere triggerato.

L'audio procedurale è da intendersi come uno dei tanti strumenti nella toolbox del sound designer. L'abilità propria del sound designer tradizionale è ancora fondamentale laddove determinati suoni abbiano bisogno di essere particolarmente caratterizzati, magari per ottenere il così detto **hyper-realism** ("_more than reality_").

Sì perchè in fondo il **realismo** non serve!
Lo sanno bene i sound designer e tutti coloro che, in generale, hanno già qualche esperienza nel mondo dell'intrattenimento, il realismo spesso delude.

Non c'è bisogno che il suono sia perfettamente fedele al fenomeno percepito nella realtà, quello che è veramente importante è il **verosimile**, è la **resa** (come dice molto bene **Chion**).

> "*Il tutto è più grande della somma delle sue parti*." *(Aristotele, Metafisica)*

Questo perchè suono ed immagine combinati assieme generano sempre un risultato inaspettato, assolutamente assente se lo si ricercasse in una sola delle sue componenti. E' questo il concetto del **valore aggiunto** di Chion.

Si tratta di una cosa che si può sperimentare anche quando si fa un **mix**: supponiamo di averne ottenuto uno che suoni davvero bene.

Tutto è al suo posto e si è ricavato la propria posizione sia nello spazio sterofonico che nel range dinamico e di frequenza. Non ci sono sovrapposizioni, conflitti di alcun genere e il tutto risulta comprensibile e omogeneo.

Ora, se tornassimo ad ascoltare una delle singole tracce che compongono il mix, magari la chitarra elettrica, il basso o il pianoforte, ci accorgeremmo molto probabilmente di quanto il suono processato, se preso singolarmente, risulti in realtà molto diverso da quello emesso dallo strumento reale.

Il suono è stato probabilmente filtrato, equalizzato, compresso, ripulito di eventuali fastidiose risonanze, emagari suona anche un po' più "_piccolo_" che nella realtà eppure, se reinserito nel mix, ecco che lo strumento riacquista di colpo tutte le sue componenti e trasmette un senso tutto diverso.

Questo è possibili grazie al fatto che lo strumento è affiancato ad altri, si fa forza dell'essere parte di un insieme, arricchisce, ed è arricchito al contempo di dignificati nuovi che, non avrebbe se preso singolarmente!

Svantaggi

Tra gli svantaggi del paradigma _audio procedurale_:

Industrial inertia: you gotta ship titles

Al momento attuale non sembra ci sia interesse nell'implementare quanto necessario per inserire questo paradigma nel workflow per la produzione di giochi. Spesso in questi casi ci si scontra con metodi di lavoro consolidati che sono difficili da modificare, soprattutto per via dei costi e dei tempi necessari per la transizione.

New workflows, new skills

Il paradigma procedurale richiede personale che sappia operare in campo audio in nuovi modi. Studiare modelli derivati o ispirati dal mondo reale e implementarli poi in una forma hardware o software richiede abilità e competenze che normalmente non fanno parte del bagaglio culturale di un sound designer tradizionale.

Synthesis = fake (what?)

Permane la falsa concezione che la sintesi audio sia, in qualche modo, sinonimo di finzione (sintesi = suono "_di plastica_") e, come tale, sia qualcosa di insoddisfacende, di deludente. In realtà non è così lo abbiamo visto poco fa parlando anche di _hyper-realism_ e _valore aggiunto_.

The Future

Cosa ci riserva il futuro dell'audio nei videogames?

Se avessimo fatto questa domanda a metà degli anni '90 declinandola al mondo della grafica di sicuro nessuno avrebbe potuto immaginare che, nell'arco di pochi anni a venire si sarebbe profilata tutta una serie di nuovissime figure professionali che prima non esistevano: professionisti che si occupano esclusivamente di _rigging_, _textures_, _animazione_, _modellazione_, _light_, _visual fxs_, _compositing_ etc...

In ambito audio, in un futuro presumibilmente non troppo lontano, il paradigma procedurale avrà preso ancora più piede e il mondo del lavoro nel settore dell'audio per videogiochi si arricchirà di tutta una serie di nuove figure professionali.

Proprio come negli ultimi 20 anni sono nate specializzazioni di ogni tipo nel mondo della computer grafica, così anche nel mondo del sound design nasceranno nuove figure speciallizzate nella modellazione di suoni e fenomeni fisici differenti (acqua e **bolle**), fuoco, **fracture sound**, impatti, **frizioni e sfregamenti**, **accartocciamenti**, **acustica delle stanze**, etc...).

Animation driven by audio

C'è addirittura chi si specializza nel processo inverso, ovvero nel ricreare animazioni basandosi sul suono in una sorta di **inverse fooley**, il che può portare a risultati davvero sorprendenti:

Se, come spesso accade, l'audio è considerato come accessorio e secondario rispetto alla parte grafica/visiva, ci sono tuttavia alcuni casi in cui questo andamento è ribaltato: `event --> sound` diviene qui `sound --> event`.

Automatic lip sync: un sistema studiato da moltissimo tempo per animare grafiche e modelli 3D in modo automatico basandosi sul segnale audio (vedi **patent Sierra**).

Procedural animation: **Tom Clancy's Ghost Recon Advanced Warfighter 2** (Ubisoft 2007) case study: un aeroplano precipitato nel deserto esplode e continua a bruciare a terra. Le fiamme sono sferzate a destra e a sinistra da un vento irregolare. Polvere e fumo sono generati proceduralmetne in base al livello dell'audio pre-prodotto (vedi questo talk di Scott Selfon al minuto 23:14)!

Procedural Audio practice

Implementazione usando la sintesi
TODO

tipologie di suono: classi sonore relativamente piccole, canne, lastre di metallo o corde possono essere programmate con relativa facilità usando sorgenti di tipo generico.

Physical modelling

physical modelling / physical informed modelling (differenze)
Waveguide (difetto: uso della memoria per implementare delay per la propagazione e i risuonatori)
MSD (Mass, Spring, damper)

tipologie di suono: questo approccio lavora bene per "_dropped sound_", collisions, tubi o lastre percosse, eccitazione di strutture fisse.

Granular approximation

Utile l'analisi statistica della densità

tipologie di suono: suoni con struttura eterogenea, onde che si infrangono sulla spiaggia, pioggia, grandi gruppi di persone (ae esempio il suono degli applausi).

PureData: esempi di audio procedurale e musica generativa

Abbiamo detto che il paradigma del suono procedurale prevede una stratificazione delle diverse fasi. Questo significa che ciascuna di esse può essere svolta con un particolare strumento hardware o software piuttosto che un altro, a seconda delle esigenze del progetto o delle particolari propensioni del suond designer.

Tipico è il caso del layer di _implementazione_: in questa fase infatti qualsiasi strumento software può essere usato per implementare il modello. Esistono infatti svariati linguaggi che possono essere utilizzati per la fase di implementazione (Supercollider, Csound, Faust, C++ via STK, Chuck, etc...). Ogni linguaggio possiede determinate caratteristiche che lo rendono più indicato per un task specifico.

Parleremo di Pure data, che sembra essere un valido compromesso in termini di efficienza, facilità di apprendimento, di integrazione in altri tipi di sistema come può essere un game engine esistente (grazie in particolar modo a progetti software di integrazione come libpd, vedi sotto).

Pure Data, come Godot, è _free software_ quindi aperto per essere modificato ed esteso liberamente.

Introduzione al linguaggio di programmazione PureData

Che cosa è PureData? è un linguaggio di programmazione a nodi nato a metà degli anni '90 ad opera di Miller Puckette che all'epoca lavorava all'IRCAM di Parigi.

Pure Data è un linguaggio di programmazioni a paradigma _dataflow_ e, sebbene manchi di ricorsione e di una effettiva accuratezza "_al sample_" nell'implementazione di filtri FIR, IIR, è uno strumento molto produttivo e permette di risolvere il 90% dei problemi di sound design sintetico.

> Uno degli obiettivi cui si è interessati è quello di fare il minor uso possibile della memoria (evitando quindi, ove possibile, lookup table, ring buffers per dly, etc...). Si usano dunque i troncamenti delle approssimazioni in serie di Taylor delle varie funzioni anzichè ricorrere a lookup tables, il rumore è generato come numeri pseudo casuali, etc...
> Perchè evitare l'uso della memoria? Uno dei motivi è quello di facilitare l'esecuzione del codice in un ambiente multithread, in cui i diversi thread sono distribuiti su più processori.

Vedremo ora alcuni esempi tratti dal lavoro di _Andy Farnell_, il quale usa PureData come linguaggio di programmazione (thanks to _Andy Farnell_, _Alexey Reshetnikov_ and _Rod Selfridge_).

Material and structure
* **telephone**: bakelite resonators

Wind
Il vento in sè non produce suono (molto interessante a tale proposito il seminario "_Squeezing out noise_" di Chris Woolf, technical consultant per _Rycote_)

Water
TODO

Machines
Vale la pena di sottolineare che tutti i suoni prodotti dalla macchine sono il risultato di un qualche tipo di inefficienza e di frizione: in teoria, la macchina perfetta non emette alcun rumore.

* **electric motors**:

* **fans**: il suono macchine come le ventole degli impianti di areazione, le quali funzionno di continuo, è più soggetto a problematiche di _ripetizione_. Se il suono viene generato per mezzo della sintesi (usando come base il rumore in questo caso) ecco che il problema è risolto: il suono acquista una caratteristica _viva_ e il cervello non è più in grado di identificare alcun _pattern_.

* **elicopter**:

* **clocks**: Il suono di un orologio che ticchetta può essere realizzato sovrapponendo diverse componenti più semplici per ottenere un risultato finale organico. Interessante a tale proposito l'aneddoto raccontato dallo stesso Farnell in un seminario sull'audio procedurale all'AES nel 2013 in cui ricorda d'avere analizzato il suono di una sveglia dopo averlo registrato ad alta risoluzione e riprodotto a velocità ridotta.

* **creaking**: movimento _slip-stick_. Un segnale di controllo (che simboleggi la forza impiegata nel movimento) che varia da 0 e 1 produce una serie di impulsi in uscita. Questi passano attraverso una serie di filtri passabanda per riprodurre le formanti di una struttura _quadrata_ in _legno_.

Di seguito alcune belle animazioni che mostrano i _modi_ principali per una membrana rettangolare, tratte dalla pagina del professor Daniel Russel della _Pennsylvania State University_.

Godot: audio architecture

Godot è un game engine libero! Interrogato dal **driver**, l'**audio server** deve fornire lui tutti i _samples_ (campioni audio 32 bit) di cui ha bisogno.

Per farlo deve copiare i dati immagazzinati nel proprio _internal buffer_ sul _p_buffer_ che il driver gli ha fornito.

Ad ogni richiestadel driver, l'audio server passa in rassegna tutte le **voci** (esaminandone con cura la _coda dei comandi_ rispettiva in modo tale da sapere se la voce deve essere riprodotta, messa in pausa, eliminata dalla codam etc...) e chiede al **mixer** di mixarle. In seguito si passa ad analizzare gli **stream** e a scrivere infine sul _p_buffer_.

Il **mixer** elabora l'audio che passa attraverso i suoi **canali** provvedendo eventualmente a calcolare eventuali **effetti** da applicare sull'audio passante. Una volta fatto questo, copia i dati contenuti sul proprio _internal_buffer_ a quello dell'_audio_server_.

L'**audio player**, lo **stream player**:'**event player**:

Godot & libpd integration

Il software libero è vincente in quanto consente l'accesso diretto al codice sorgente rendendo più veloci e agili integrazioni, modifiche e migliorie altrimenti impossibili in progetti software proprietari.

In questo modo ci è stato possibile avviare un progetto di integrazione in Godot dell'engine audio _PureData_ grazie al wrapper **libpd** ideato da Peter Brinkmann.

Call for partecipants

Il progetto è ancora in fase embrionale ma contiamo di refinirlo sempre più per poi rilasciarlo ufficialemtne al pubblico in modo che tutti possano avvantaggiarsi anche di un audio _veramente_ procedurale all'interno di Godot.

Qui il link al repository sul quale _limulo.net_ sta facendo i primi test: ogni contributo è benvenuto!