



MINISTERO DELL'ISTRUZIONE, DELL'UNIVERSITA' E DELLA RICERCA
Alta Formazione Artistica e Musicale

**Conservatorio statale di musica
"G. Rossini"**

Triennio I livello

Musica Elettronica

Geometria frattale nella sintesi sonora e strutture autogenerative

Relatore:

Prof. Pasquale Mainolfi

Tesi di:

Claudio Marcozzi

SOMMARIO

Abstract	4
Introduzione	6
CAPITOLO 1: Musica generativa, alea e composizione algoritmica	10
1.1 Il fenomeno aleatorio	10
1.2 Composizione algoritmica	14
1.3 Musica generativa	17
CAPITOLO 2: La geometria frattale	19
2.1 Caso e Caos	19
2.2 Geometrie e strutture frattali	21
2.3 <i>Tipologie di frattali</i>	24
2.4 Esempi di frattali lineari	26
2.5 Il frattale di Mandelbrot	30
CAPITOLO 3: Geometria frattale applicata alla musica elettronica	33
3.1 I frattali nella musica	33
3.2 Algoritmi per il calcolo frattale aleatorio	38
3.3 Impiego della geometria frattale nella composizione musicale	46
3.4 Geometria e calcolo frattale all'interno di una struttura generativa	50

CAPITOLO 4: Fractal Autogenerative Network	57
4.1 Presentation Mode	57
4.2 Gestione degli eventi sonori	59
4.3 Motore di sintesi	79
4.4 Spazializzazione e codifica Ambisonics	84
CAPITOLO 5: A Fractal Piece	90
5.1 Struttura del brano	90
5.2 Prima sezione – Thin Line	92
5.3 <i>Seconda sezione – Circle</i>	95
5.4 Terza sezione – The Big Area	97
Conclusioni	99
Bibliografia	102
Sitografia	105

Abstract

Da un lato, la cosiddetta *musica aleatoria*, basata sulla decisione di lasciare al caso la determinazione di alcuni parametri. Di qualunque sistema aleatorio si parli, questo risulterebbe imprevedibile, non garantirebbe un comportamento ben distinto, ma, appunto, interamente casuale. Dall’altro lato, l’avvento del computer, dell’elaborazione del suono digitale; la velocità di esecuzione permessa da queste macchine fa sì che esse possano essere utilizzate per compiere operazioni numerose e molto complesse quasi istantaneamente. La *composizione algoritmica* nasce da tutto ciò, verte intorno all’elaboratore e all’insieme di procedimenti che la definisce.

Posta in mezzo a questi due mondi, la geometria frattale offre numerose possibilità come elemento aleatorio o algoritmico; il potenziale che può assumere è da ricercarsi in un’analisi delle proprietà che definiscono gli oggetti frattali, della loro relazione col caos e con la natura, e delle loro possibilità, a partire dal fascino che questi esercitano già soltanto tramite una loro rappresentazione grafica bidimensionale fino alle applicazioni che hanno assunto nei vari campi. Alcuni di questi frattali, fra tutti il più famoso frattale di Mandelbrot, sono fondamentali per lo studio dei sistemi dinamici e della teoria del caos. I risultati del calcolo frattale hanno spesso un comportamento apparentemente caotico, ma che presenta una caratteristica di auto-similarità interna, detta *omotetia*. In altre parole, l’oggetto frattale è simile a sé stesso a qualunque scala lo si osservi; la figura generale riappare continuamente man mano che ci si addentra all’interno della sua struttura. La generazione di un frattale, inoltre, è lei stessa un algoritmo, nello specifico un algoritmo ricorsivo. La *ricorsione*, appunto, è un’ulteriore caratteristica fondamentale degli oggetti frattali, ad ogni iterazione del procedimento il risultato corrente è legato al risultato precedente.

In virtù di tali considerazioni, un utilizzo appropriato del calcolo frattale come elemento aleatorio è capace di garantire un ottimo grado di imprevedibilità e

casualità in un contesto evolutivo, paradossalmente, non del tutto caotico, riconoscibile ed in parte prevedibile.

L'elevata potenza di calcolo dei sistemi digitali odierni permette di utilizzare la geometria frattale in maniera ottimale, trasformandola in un mezzo con il quale controllare dei parametri di una composizione aleatoria. Il lavoro che segue intende porre la geometria frattale al centro di una ricerca musicale, sia nell'articolazione degli eventi sia nella struttura timbrica, realizzando qualcosa che non provenga soltanto dai puri calcoli matematici, ma che prenda anche ispirazione dalla struttura stessa dei frattali.

INTRODUZIONE

I frattali hanno assunto, dal XX secolo al giorno d'oggi, una notevole rilevanza in una moltitudine di campi scientifici; dalla matematica che vi si cela dietro alla geometria che ne scaturisce, per arrivare all'economia e alla finanza, alla teoria del caos, l'anatomia, la computer grafica. Ciò si deve allo studio portato avanti da Benoit Mandelbrot, che ha fatto emergere le potenzialità che hanno questi oggetti in numerose discipline.

La scoperta della *matematica frattale* risale circa al XVII secolo; il filosofo e matematico Gottfried Leibniz fu il primo a parlare di auto-similarità e ricorsione, come fu anche il primo ad usare il termine *fractional* per descrivere questo tipo di comportamenti. Il suo pensiero sulla realtà anticipava la caratteristica di omotetia propria dei frattali.

« Ogni porzione di materia può essere concepita come un giardino pieno di piante e come uno stagno pieno di pesci; ma ogni ramo di pianta, ogni membro di animale, ogni goccia dei loro umori, è ancora un giardino simile, un simile stagno ». ¹

Per i due secoli successivi, l'intuizione di Leibniz non fu portata avanti; l'idea di questi frattali sembrava troppo eterea, non tangibile, non vi era familiarità con i concetti che si pongono alla base della geometria frattale. Soltanto nel XIX secolo la ricerca su questo mondo fu ripresa, facendo emergere dei primi esempi di figure frattali. Karl Weierstrass, padre della moderna analisi matematica, presentò una prima definizione di una funzione continua ma non differenziabile in nessun punto, che si rivelò essere negli anni più recenti un esempio di frattale. Georg Cantor, seguendo le letture di Weierstrass, pubblicò il suo *insieme di Cantor*, un sottoinsieme della linea e che è un frattale a tutti gli effetti. Felix Klein ed Henri Poincaré introdussero una nuova categoria di frattali, detti frattali auto-inversi. Giuseppe Peano, negli ultimi anni del XIX secolo, partendo dall'insieme di Cantor scoprì una curva in grado di riempire un piano senza lasciare punti vuoti, che prese il nome di *curva di Peano*, anch'essa catalogata come frattale.

¹ G.W. VON LEIBNIZ, *La Monadologia*, Milano: SE, 2018.

Nel secolo successivo lo sviluppo di questo mondo continuò; Helge Von Koch e Waclav Sierpinski pubblicarono altre figure frattali che prenderanno poi il loro nome - *curva di Koch* e *triangolo di Sierpinski* - con annessi i procedimenti ricorsivi che portano alla loro realizzazione.

A gettare le basi per il lavoro di Benoit Mandelbrot e portare notevolmente avanti la ricerca sui frattali furono Pierre Fatou² e Gaston Julia;³ entrambi arrivarono a descrivere il comportamento frattale avvalendosi dei numeri complessi e di funzioni iterative, ideando gli insiemi oggi conosciuti come *insieme di Fatou* e *insieme di Julia* - l'uno complementare dell'altro. Affascinato dal comportamento caotico di alcuni fenomeni, Benoit Mandelbrot partì dall'eredità da Fatou e Julia; durante il suo lavoro sui frattali, fu in grado di mostrare le possibilità di utilizzo in vari campi, fra cui l'economia, diede il suo nome ad una specifica famiglia di frattali, appunto frattale di Mandelbrot, il più conosciuto, e mostrando come la natura offra numerosissimi esempi di figure e forme che sono dei frattali a tutti gli effetti.⁴ Con le scoperte di Mandelbrot, il mondo frattale diviene un sistema capace di dare un'interpretazione forte a molti fenomeni con comportamenti caotici e altrimenti inspiegabili, sia che questi vengano osservati in maniera più generica, sia che vengano guardati nelle più piccole variazioni.

È proprio da tutto ciò che nasce questo mio interesse per il calcolo frattale. È affascinante osservare come il caos, in realtà, possa definirsi attraverso regole a volte molto semplici; i risultati caotici generati dal calcolo frattale sono strettamente connessi con la natura, ogni oggetto frattale è al suo interno simile a sé stesso. Queste particolarità rendono i frattali utilizzabili anche in ambito musicale; in un contesto di struttura auto generativa, la geometria frattale può garantire dei risultati caratteristici, seppur governati da un certo grado di apparente imprevedibilità.

Dal mio punto di vista, la musica generativa rappresenta un diverso modo di comporre incredibilmente affascinante; riuscire a creare un organismo musicale

2 cfr P. FATOU, *Teoria dell'iterazione*, 1917

3 cfr. G. JULIA, *Memoria sull'iterazione delle funzioni razionali*, 1918

4 cfr. B. MANDELBROT, *The Fractal Geometry of Nature*, New York: Times Books, 1982.

autonomo con il quale è possibile interfacciarsi, che sia in grado di generare, partendo dagli stessi presupposti, qualcosa di diverso continuamente, ma a sua volta simile, disegnare uno “stile”, un *modus operandi* proprio della struttura generativa sono stati obiettivi principali del mio percorso.

Il concetto di struttura generativa pone storicamente le basi sulla composizione aleatoria e sulla composizione algoritmica. L’aleatorietà musicale è la scelta del compositore di svincolarsi dal dover stabilire il valore di un parametro, che sia esso frequenza o intensità, timbro, durata, lasciando tale compito al caso, ed è il principio che sta dietro ad una struttura generativa. La composizione algoritmica ne è l’immediata conseguenza e al tempo stesso il suo ampliamento; tramite l’elaboratore, è possibile creare algoritmi complessi per una scelta casuale, seppur ben controllata, dei parametri musicali. Una struttura auto-generativa fa propri i concetti di alea e di algoritmo, per poter creare continuamente nuova musica.

Date le loro caratteristiche così particolari, i frattali si pongono immediatamente come perfetti strumenti casuali all’interno di un sistema aleatorio. Il calcolo frattale è un esempio di caos controllato, ha un suo comportamento ben definito; il suo utilizzo può dare risultati sempre diversi, ma questi seguiranno tutti delle articolate restrizioni che contribuiscono a definire uno stile ben riconoscibile.

Il seguente elaborato, dopo aver affrontato separatamente i concetti di geometria frattale, struttura generativa e composizione algoritmica, andrà ad illustrare come i due temi possano connettersi sotto numerosi aspetti. Saranno descritte le nozioni matematiche basilari per quanto riguarda il calcolo frattale e le nozioni teoriche per quanto riguarda l’ambito algoritmico ed aleatorio; tuttavia, ritengo ben più importante focalizzare il discorso più sugli aspetti che hanno avuto un impatto sul mio percorso e sul mio lavoro; in primo luogo, l’analisi storica e le motivazioni che hanno portato alla nascita della composizione aleatoria fino allo sviluppo della musica generativa; successivamente le possibilità della geometria frattale, alcune delle applicazioni finora sviluppate tramite il loro utilizzo, il legame col mondo naturale, le caratteristiche distintive di omotetia e ricorsione.

Partendo da tali presupposti, si andrà poi ad analizzare l'utilizzo dei frattali in ambito musicale; verranno messi in relazione i diversi concetti descritti precedentemente gettando alcuni spunti per l'utilizzo del calcolo frattale in un'ottica compositiva, per poi spiegare come questi possano essere utilizzati all'interno di una struttura generativa, come metodo di generazione degli eventi sonori, come base per la sintesi del suono, come metodo di determinazione della posizione nello spazio di una sorgente. Infine, viene spiegato il lavoro svolto nella realizzazione di una vera e propria struttura generativa basata sul mondo frattale utilizzata poi per una composizione aleatoria/algoritmica, che basa sulla geometria frattale sia la vera e propria generazione sonora sia la sua struttura.

CAP. 1 Musica generativa, alea e composizione algoritmica

1.1 Il fenomeno aleatorio

Il caso esercita una forte influenza su molti aspetti della nostra vita; comunemente esso assume una connotazione astratta, vista l'intrinseca impossibilità di definizione – se fosse definibile non si tratterebbe di casualità, ma di ovvia conseguenza - eppure i suoi effetti sono fortemente reali. Le interpretazioni e i tentativi di spiegare il termine “caso” sono innumerevoli, ma con un unico fattore comune: al netto di qualsivoglia interpretazione dovuta alle più disparate differenze culturali, il caso è visto come una forza superiore che controlla cose su cui l'uomo può avere un'influenza ridotta, se non addirittura nulla; spesso viene interpretato come una sorta di decisione inspiegabile della natura. Gli eventi casuali sono definiti fortuiti, accidentali o imprevisti che avvengono indipendentemente dalla nostra volontà o intenzione.

A livello compositivo, l'aspetto della casualità non è stato affrontato fino al XX secolo. Al di là di alcuni esempi sporadici nel corso della storia della musica (come il *Musikalischs Würfelspiel* di Mozart, un gioco musicale basato sul lancio di dadi), le prime tecniche compositive basate sulla casualità furono introdotte dagli americani Charles Ives e Henry Cowell, basate sul controllo del caso e sulle probabilità, ciò che viene conosciuto come processo aleatorio, da cui deriva la connotazione di *musica aleatoria*.⁵ Tuttavia, colui che incarna e più di tutti ha contribuito all'affermarsi di tale corrente musicale è indubbiamente John Cage.

A seguito del suo avvicinarsi alle culture orientali e al Buddhismo Zen, abbracciò l'aleatorietà come metodo compositivo allo scopo di spogliare il compositore da ogni relazione emotionale con la sua opera. Proprio perché è il caso a governare la natura e non l'uomo, la musica di Cage era una musica-processo, non un oggetto da modellare che subiva la volontà del compositore.⁶ John Cage applica i principi dello Zen nella sua idea artistica, rinunciando alle classificazioni,

⁵ Pierre Boulez rese popolare il termine *Aleatory Music*, utilizzando il termine *Alea* nelle sue *Note di un apprendistato* del 1957.

all’organizzazione del materiale; tale disciplina prevede l’accettazione della natura in quanto tale – casuale, per non dire *caotica*, senza alcuna organizzazione comprensibile – e la musica deve seguire lo stesso percorso, ovvero l’accettazione dei suoni per come sono. Si tratta di un tentativo di simulare e forse anche omaggiare il comportamento naturale che ci circonda.

La lettura del libro cinese dei mutamenti, l’I-Ching, un’ulteriore fonte d’ispirazione dal mondo orientale, porta Cage a derivarne un sistema casuale per la composizione in cui non vi è alcun intento né soggettività posta da parte dell’autore.⁷ In seguito, il caso verrà utilizzato come mezzo compositivo anche da parte dei compositori europei della scuola di Darmstadt, provenienti da una formazione completamente diversa e con un’estetica lontana da quella di John Cage – per questo l’idea compositiva che vi è dietro risulta antitetica all’idea del compositore americano - come ad esempio Stockhausen,⁸ Pierre Boulez⁹ e Bruno Maderna.¹⁰

Detto ciò, occorre sottolineare che il termine aleatorio non è sinonimo di casuale. Esso sì incapsula il concetto di casualità al suo interno, ma lo limita al tempo stesso; di per sé l’alea è una casualità controllata, non è possibile prevedere il risultato esatto per ogni processo aleatorio, ma il numero di possibilità è finito e calcolabile. L’etichetta di *musica aleatoria* si basa proprio su questo concetto, il caso influisce in maniera più o meno ampia sul risultato agendo all’interno di un insieme

6 cfr. *Funghi e Zen: casualità e complessità nella musica di John Cage*,
<https://www.lavoroculturale.org/musica-di-john-cage/>, consultato il 2 febbraio 2020).

7 Il metodo ricavato dall’I-Ching portò Cage a realizzare la sua celebre composizione per pianoforte *Music Of Changes* (1951). Il titolo prende ispirazione proprio dall’I-Ching, il libro dei cambiamenti – *Book of Changes*.

8 In Klavierstück XI il pianista può scegliere liberamente l’ordine dei 19 frammenti di cui è composto il brano. Sebbene sia a discrezione dell’esecutore, tale decisione può essere vista come una volontà di lasciare la struttura dell’opera alla casualità, dato che ogni esecutore sceglierà combinazioni diverse senza alcun meccanismo prevedibile.

9 Nella *Terza Sonata* per pianoforte Boulez chiama le cinque parti di tale opera *Formanti* e non *Movimenti*, a seguito della decisione di escludere una consequenzialità fissa; le parti di brano che andranno a formare l’opera avranno un ordine scelto a discrezione dell’esecutore.

10 Tra le composizioni aleatorie più importanti di Maderna troviamo *Serenata per un Satellite* (1969), *Grande Aulodia per flauto e oboe soli ed Orchestra* (1970), *Quadrivium* (1966), *III Concerto per Oboe ed Orchestra* (1973). Il compositore italiano utilizzò ampiamente le tecniche aleatorie nell’ultima parte della sua carriera.

di possibilità. Ogni processo aleatorio adoperato dai compositori, persino l’I-Ching di Cage, limita la casualità ad un certo insieme di valori.

Analizzando in modo generico un fenomeno qualsiasi, questo può definirsi aleatorio se produce sempre un risultato diverso a partire dalle stesse condizioni iniziali; esso non è *deterministico*, ma *non-lineare* e per questo non è possibile fare una previsione del risultato, ma è sempre possibile effettuare un calcolo statistico. Si prenda in considerazione un lancio di un dado da gioco; pur lanciando il dado più volte nella stessa condizione iniziale il risultato sarà ogni volta differente e imprevedibile, ma le possibilità ad ogni lancio che esca una delle 6 facce è pari a $1/6$, ovvero 16.67 %. Il risultato del lancio del dado corrisponde al concetto di *variabile aleatoria* nello studio delle probabilità, una variabile che assume connotazioni diverse a seconda di uno o più eventi aleatori. In tal senso, un processo aleatorio – o stocastico – è un insieme di variabili aleatorie tutte dipendenti da un parametro t , solitamente identificato con il tempo.¹¹

Partendo da un insieme T di valori assumibili da t , da una *funzione di probabilità* che assegna ad ogni variabile una certa probabilità e da uno spazio campione Ω – vale a dire un insieme dei possibili risultati – un processo stocastico è definibile formalmente come l’insieme delle variabili aleatorie dipendenti da t , il tempo, e definite all’interno dello spazio Ω . Si possono considerare tali variabili come delle funzioni che cambiano nel tempo, ognuna associata ad un elemento di Ω ; fissando un valore di partenza e ripetendo il processo più volte sempre dallo stesso valore iniziale otterremo ogni volta un risultato diverso, un valore diverso dello spazio campione. Configurando tutto ciò sull’esempio del lancio di un dado: le variabili aleatorie sono le sei facce del dado, la funzione di probabilità assegna ad ognuna delle sei facce il valore 0.1677 e gli istanti di tempo possibili sono essenzialmente due, lo stato iniziale e lo stato di arrivo.

11 cfr. *Introduzione ai processi stocastici*, Università di Pisa,
http://users.dma.unipi.it/~flandoli/dispense_Istituzioni.pdf, (consultato il 2 febbraio 2020).

- $T = \{1, 0\}$;

- $t \in T$;

- $\Omega = \{1, 2, 3, 4, 5, 6\}$;

- $x \in \Omega$;

- $P(x) = 0.1667, \forall x \in \Omega$

I processi stocastici si dividono in processi Markoviani o non Markoviani, in base alla legge probabilistica che li governa. Il risultato di un processo di Markov dipende unicamente dallo stato iniziale; qualora vi sia una dipendenza coi risultati precedenti, il processo è detto non Markoviano. Questo tipo di processi aleatori sono stati ampiamente usati dal compositore Iannis Xenakis verso la fine degli anni '50. Tramite l'utilizzo di sistemi probabilistici come le catene di Markov,¹² Xenakis introduce nella musica aleatoria un'idea compositiva differente sia da quella di Cage, fondata sull'abbandono dell'intento compositivo e l'accettazione del caso, sia da quella figlia della scuola di Darmstadt di Boulez, Stockhausen, Maderna ecc.; le probabilità nelle opere aleatorie di Xenakis sono calcolate rigorosamente.¹³ Egli si approccia alla composizione attraverso le sue competenze matematiche – ai tempi della seconda guerra mondiale frequentava gli studi di architettura ed ingegneria, interrotti a causa dell'invasione tedesca – avvalendosi in maniera importante dei procedimenti Markoviani per la determinazione dei vari parametri della composizione.¹⁴ La musica *stocastica* di Iannis Xenakis è frutto di una serie di calcoli probabilistici ben ponderati allo scopo di ottenere un comportamento analogo a quello dei fenomeni naturali.

12 I. XENAKIS, *Analogique A*, Parigi (1958).

13 cfr. I. XENAKIS, *Formalized Music. Thought and Mathematic in Composition*. Parigi, Pendragon Press (1963).

14 cfr. T. BOLOGNESI, *Composizione assistita dal calcolatore*. Pisa, CNUCE (1981), p. 83.

1.2 Composizione algoritmica

I procedimenti compositivi attuati da Xenakis hanno trovato il loro miglior utilizzo nel momento in cui sono apparsi i primi elaboratori. Il compositore greco è stato fra i primi ad avvalersi dei computer nella sua musica e la potenza di calcolo di queste nuove macchine ha permesso di affrontare la composizione mediante un processo matematico in maniera più vasta, rapida e complessa.

La *composizione algoritmica* è, semplicemente, una tecnica compositiva che fa uso di algoritmi. Il termine *algoritmo* è molto generico ed indica una serie di procedimenti da attuare o regole da seguire per ottenere un risultato; questo può essere estremamente semplice, come il calcolo di una media aritmetica, oppure molto complesso. In realtà, gli algoritmi sono stati sempre utilizzati nella composizione, basti pensare al classico contrappunto occidentale e alle sue regole, o alla dodecafonia e al serialismo, sebbene mediato dall'intervento umano. C'è sempre stata una tendenza a formalizzare il processo compositivo in una serie di istruzioni più o meno rigide. Il termine algoritmo prende piede a seguito della rivoluzione digitale, per questo un procedimento viene chiamato algoritmo nel momento in cui è eseguito da una macchina invece che dall'uomo.

In un contesto musicale, l'utilizzo di algoritmi deterministici per la composizione difficilmente è in grado di dare sviluppi particolarmente interessanti. Difatti, gli algoritmi utilizzati dai compositori sono sempre di natura non deterministica, garantendo un esito musicale sempre diverso; per tale ragione si può affermare che la composizione algoritmica discende dalla corrente aleatoria, ponendo le sue radici proprio nel concetto di alea ed arrivando anche ad espanderne gli orizzonti grazie alla rapidità di esecuzione di tutti questi procedimenti da parte dei computer.

A proposito delle possibilità offerte in ambito musicale dai processi automatizzati, Ada Lovelace - ricordata come la prima programmatrice al mondo - più di un secolo prima dell'avvento del computer si espresse in tal modo:

« Supponendo, ad esempio, che le relazioni fondamentali fra le note nell'ottica dell'armonia e della composizione musicale fossero suscettibili a qualche tipo di espressione o adattamenti, il motore di calcolo potrebbe comporre brani musicali elaborati e scientifici di qualsiasi grado di complessità o estensione. »¹⁵

I primissimi esempi di composizione algoritmica, ancor prima di Xenakis, sono da riscontrarsi nelle figure di Lejaren Hiller e Leonard Isaacson, entrambi professori all'università dell'Illinois. Il brano *Illiac Suite* per quartetto d'archi del 1957 è conosciuto come il primo brano della storia realizzato mediante l'utilizzo del computer.

La strategia compositiva adottata da Hiller e Isaacson per *Illiac Suite* è stata quella di generare dei materiali tramite l'elaboratore¹⁶ per poi trasformarli mediante diversi tipi di funzioni matematiche, attuare una selezione delle parti considerate più interessanti ed infine trascrivere il risultato in uno spartito, adattando la parte per un quartetto d'archi. Questi esperimenti portarono Hiller a realizzare, qualche anno dopo, il linguaggio di programmazione MUSICOMP, sviluppato per la composizione musicale dal quale scaturì il brano *Computer Cantata* (1963).

L'approccio stocastico adottato da Xenakis e Hiller porta avanti il concetto di aleatorietà mediante il controllo delle probabilità; l'utilizzo di processi di questo tipo e lo studio statistico che vi è dietro produce una complessità notevole che sfocia in un risultato sonoro molto interessante, riuscendo in questo modo a controllare il tipo di comportamento caotico che il processo andrà ad assumere. La nascita e lo sviluppo della composizione algoritmica va di pari passo con quello della *computer*

15 J. A. MAURER, *A Brief History of Algorithmic Composition*. Stanford, CCRMA (1999).

16 Venne utilizzato l'ILLIAC I (Illinois Automatic Computer), da cui deriva il titolo del brano. Si tratta di uno dei primi esemplari di computer, realizzato e posseduto dall'università dell'Illinois. Fra i vari sistemi per la generazione del materiale, fu utilizzato per la prima volta un esempio di Catene di Markov.

*music*¹⁷ e dei linguaggi di programmazione musicali. Il lavoro di Max Mathews ai Laboratori Bell risulta decisivo; durante la sua carriera ha sviluppato una serie di ambienti di sviluppo per applicazioni musicali, i cosiddetti MUSIC-N¹⁸. Sebbene il suo interesse fosse principalmente focalizzato sulla sintesi sonora piuttosto che sull'organizzazione del materiale e sugli algoritmi per la composizione, questa serie di software e i suoi più recenti discendenti offrono la possibilità di definire algoritmi per la realizzazione di processi stocastici.

Col tempo, le tipologie di algoritmi per la composizione sono aumentate di numero, così come le possibilità e la complessità come conseguenza dell'enorme sviluppo tecnologico degli ultimi decenni.

17 Termine generico per indicare la musica realizzata con l'ausilio del computer. La composizione algoritmica è un sottogenere della computer music.

18 I linguaggi MUSIC-N sono una famiglia di software e linguaggi di programmazione sviluppati da Max Mathews, tutti derivanti dal primo linguaggio MUSIC, scritto nel 1957.

1.3 Musica generativa

La *musica generativa* rappresenta un’ulteriore estensione del concetto di composizione algoritmica. Il termine è stato coniato e reso famoso da Brian Eno e descrive una musica creata da un sistema a sé stante, autonomo, la quale ha una continua evoluzione ed è diversa ad ogni reiterazione del sistema. Se nella composizione algoritmica vengono realizzati algoritmi su misura per la realizzazione di un brano musicale, la musica generativa è invece concepita come un flusso sonoro in continuo mutamento.

Sebbene i due concetti siano molto vicini e a tratti possano confondersi l’uno l’altro, ci sono delle dovute differenze che è necessario sottolineare. Innanzitutto, l’origine della musica generativa ha un forte legame con la *musica ambient*, altro termine coniato da Brian Eno che descrive una musica d’atmosfera, che può essere ascoltata distrattamente e che crea a livello immaginario un ambiente che circonda l’ascoltatore senza rendersi protagonista.¹⁹ L’idea di un sistema capace di generare musica continuamente e sempre in maniera diversa è stata pensata da Eno come un qualcosa che rientrasse nella definizione di musica ambient; interessante notare, tuttavia, come egli stesso sia stato ispirato sì da Erik Satie con la sua *musique d’ameublement*²⁰ (musica d’arredamento) e dalla corrente del minimalismo degli anni ‘60, ma anche dalla musica del già citato John Cage ed il suo utilizzo dell’I-Ching come metodo compositivo. La musica d’ambiente trova esempi precursori di alcune delle sue caratteristiche anche nelle musiche aleatorie di Cage, la sua ricerca sullo spazio e sul silenzio,²¹ nei primi lavori elettronici di Karlheinz Stockhausen, nella *musique concrète* di Piérre Schaffer e nei primi lavori sul *paesaggio sonoro*²².

L’idea di sistema generativo o struttura generativa deriva da diverse influenze, oltre la già citata musica aleatoria. La prima influenza riguarda la concezione di

19 B. ENO, *Ambient I - Music for Airports*. Polydor Records (1978).

20 E. SATIE, *Relâche* (1924).

21 J. CAGE, 4'33", (1955).

22 Termine coniato da Raymond Murray Schafer nel suo libro, *Tuning of the World* (1977). Indica una musica fatta di suoni naturali non estrapolati dal loro contesto, come può essere la *musique concrète*. Lo scopo di queste composizioni è sia di natura artistica sia di natura ambientale.

musica come processo graduale²³ già sperimentata da Steve Reich a partire da *It's Gonna Rain* (1968) e che è alla base del suo percorso artistico successivo; Eno prende esempio dal processo di *phasing* attuato da Reich sui due registratori a nastro e osserva come questo sistema non produce lo stesso risultato se riprodotto più volte e possa avere lunghezza infinita senza ripetersi mai. La seconda importante influenza riguarda, sempre rimanendo nel mondo minimalista, il celebre brano *In C* (1964) di Terry Riley. Si tratta di una composizione aleatoria composta di 53 frasi musicali e una serie di indicazioni per l'esecuzione; ogni musicista può scegliere quale frase suonare e quante volte ripeterla. La peculiarità colta da Brian Eno in questa composizione sta nella sua potenziale esecuzione per una durata indefinita; l'unica nota di Riley in merito è un suggerimento che consiglia che il brano duri dai 45 minuti ad un'ora e mezza.

Nel 1966 Brian Eno collaborò con la compagnia SSEYO per la realizzazione di un sistema per la musica generativa chiamato *Koan*, usato da lui stesso per comporre il disco *Generative Music 1*; buona parte dei lavori successivi del compositore inglese derivano dallo sviluppo ed utilizzo di sistemi generativi.²⁴

« Uno dei miei interessi a lungo termine è stato l'invenzione di "macchine" e "sistemi" in grado di produrre esperienze musicali e visive. L'obiettivo era di fare musica con materiali e processi che ho specificato, ma in combinazioni e interazioni che non ho fatto. »

In sintesi, le strutture generative possono essere considerate dei sistemi che incapsulano il concetto di composizione algoritmica allo scopo di generare un *continuum* sonoro in evoluzione e che ha una durata non definita, decisa dall'utente in base alla sua volontà. Questi sistemi, vista anche la potenza di calcolo dei computer odierni, possono anche avere un design interattivo con un controllo ad alto livello, possibilità espresive molto elevate ed un'interfaccia utente relativamente semplice.

23 S. REICH, *Musica come processo graduale* (1968).

24 cfr. B. ENO, P. CHILVERS, <http://www.generativemusic.com>, consultato il 2 febbraio 2020.

CAP. 2 La geometria frattale

2.1 Caso e Caos

Il precedente capitolo si è aperto con una breve riflessione sul significato e sull'interpretazione del *caso*. Prima di proseguire il percorso verso l'universo che circonda i frattali, trovo utile esporre una mia personale considerazione sul confronto fra *caso* e *caos*.

Come già anticipato, per *caso* s'intende un evento imprevedibile; qual è invece il significato attribuito alla parola *caos*? Etimologicamente, la definizione più semplice è quella di disordine, confusione. Analizzando quindi i due termini nella maniera più generica possibile, le differenze risultano lampanti; ma se contestualizziamo il termine associandolo ad un comportamento o ad un evento, il suo reale significato assume ulteriori sfumature. Un evento governato dal caos è un evento imprevedibile, un comportamento caotico è anch'esso un comportamento imprevedibile; se analizzati in tal senso, *caos* e *caso* sembrano coincidere. Dunque, un evento fortuito accade per *caso* o accade per *caos*?

La generica concezione già esposta di *caso* come evento di cui non conosciamo le cause e che per tale ragione è opera di qualcosa più grande di noi, è in realtà molto più vicina a ciò che s'intende per *caos*. Ciò che separa i due concetti sta nel contesto in cui vengono usate le due parole; mentre il *caso* tende a riferirsi ad un singolo evento non prevedibile, il *caos* si riferisce ad un insieme di eventi non comprensibili ed imprevedibili, talvolta anche insignificanti se valutati singolarmente, che tendono a causare a loro volta eventi più rilevanti e anch'essi non prevedibili. Fondamentalmente, *caos* e *caso* si intrecciano, l'uno è conseguenza dell'altro. Un evento casuale è generato da una serie di eventi casuali, e tale serie di eventi è *caos*. Un avvenimento casuale è sia causa che conseguenza del *caos*.

Ponendo tutto ciò in un contesto di processo aleatorio, si può parlare di procedimento *caotico*? Prendendo in esame l'esempio posto in precedenza sul lancio del dado, in base alle considerazioni fatte poc'anzi ciò che ne determinerà il risultato

è una serie innumerevole di micro-eventi non deterministici, per cui il *caso* genera l'evento casuale; quindi un procedimento aleatorio come il lancio di dado può essere considerato come una conseguenza del *caos*. Proprio a causa di questo legame, lo studio del *caos* viene oggi affiancato alla matematica statistica nell'analisi di alcuni fenomeni reali; esso rappresenta un fattore troppo significativo per essere ignorato.²⁵

25 cfr. R. L. DEVANEY, *A First Course In Chaotic Dynamical Systems: Theory And Experiment (Studies in Nonlinearity)*, Westview Press (1992)

2.2 Geometrie e strutture frattali

I frattali costituiscono una categoria di oggetti geometrici con delle caratteristiche molto particolari e che hanno un'implicazione piuttosto rilevante all'interno dei *sistemi dinamici* e della *teoria del caos*. Un oggetto frattale è così definibile se possiede le seguenti proprietà:

- *Omotetia interna*: detta anche auto-similarità. L'omotetia è una trasformazione che dilata o contrae un oggetto geometrico, mantenendone invariata la forma. La caratteristica di omotetia interna indica che l'oggetto geometrico possiede una forma che si ripete a qualsiasi scala di grandezza. In altre parole, l'oggetto è formato da oggetti più piccoli simili o uguali all'oggetto stesso.
- *Struttura ricorsiva o irregolarità*: il frattale è definito tramite un algoritmo ricorsivo, un procedimento applicato più volte in cui il risultato ad ogni step dipende dal risultato dello step precedente. Più volte tale procedimento viene applicato, maggiore sarà la complessità dell'oggetto che ne scaturisce.
- *Perimetro illimitato o nullo*: man mano che il processo viene iterato, ogni segmento che compone il frattale viene riproposto e ridotto di dimensioni, perciò al tendere del numero di passi anche il perimetro tenderà ad infinito.
- *Area finita o nulla*: l'area del frattale è racchiusa dentro un confine ben stabilito
- *Dimensione anche non intera*: le figure geometriche classiche hanno una dimensione intera (monodimensionale, bidimensionale, tridimensionale).²⁶ Gli oggetti frattali possono avere una dimensione che non è intera, ovvero frazionaria; questa è stata generalizzata da Felix Hausdorff, che, nel 1918, introdusse il concetto di dimensione frattale oggi denominata *dimensione di Hausdorff*, ovvero il valore della dimensione di uno specifico frattale. La dimensione di Hausdorff si indica con $\dim_H(A)$.
- *Struttura fine*: un oggetto frattale rivela sempre più dettagli man mano che lo si ingrandisce.
- *Dinamica caotica*: piccole variazioni del sistema iniziale cambiano drasticamente il risultato.
- *Oggetto geometrico non euclideo*: la geometria frattale è una geometria non euclidea.²⁷

26 La dimensione di una figura si calcola raddoppiando le dimensioni della figura stessa. Nel caso di una linea avremo una linea lunga il doppio, quindi $2*l$; nel caso di un quadrato raddoppieremo sia la sua base che la sua altezza, ottenendo un quadrato quattro volte superiore, quindi $4*q = 2^2$. Nel caso di un cubo, avremo $8*c = 2^3$. La dimensione corrisponde all'esponente da dare a 2 per ottenere il fattore moltiplicativo.

27 cfr. M.J. GREENBERG, *Euclidean and Non-Euclidean Geometries: Development and History*. W.H. Freeman, (2008)

La geometria non euclidea nasce dalla negazione del quinto postulato di Euclide.²⁸ Gli studi di Benoit Mandelbrot sulla geometria frattale dimostrano come tali oggetti non siano definibili attenendosi alla geometria classica basata su di un'equazione tradizionale; la loro generazione si basa infatti, come già anticipato, su di un processo ricorsivo.

Il termine frattale deriva dal latino *fractus*, che significa spezzato, da cui deriva anche il termine *frazione*; fu coniato da Benoit Mandelbrot che lo scelse appunto per la caratteristica frazionaria della dimensione dei frattali.

« Le nuvole non sono sfere, le montagne non sono coni, le coste di un'isola non sono cerchi e le corteccie non sono lisce, e nemmeno i fulmini viaggiano secondo una linea dritta. » B. Mandelbrot

Gli esempi di forme frattali in natura sono molto frequenti, a partire dalla forma del cavolfiore o del broccolo romano, per passare ai fiocchi di neve, le foglie, gli alberi, le montagne. Persino la struttura a doppia elica del DNA, i vasi sanguigni del cuore, la forma del cervello umano e i neuroni costituiscono esempi di frattale, poiché ognuno di questi si struttura su copie di se stesso più piccole. La fascinazione che queste forme e questi oggetti riescono a suscitare è presto spiegata; essi richiamano fortemente la realtà, possiedono una complessità enorme, un comportamento caotico e allo stesso tempo una forte organizzazione e coerenza, il tutto partendo da un unico calcolo svolto ricorsivamente.

La geometria frattale nasce proprio dagli studi di Benoit Mandelbrot, il quale, partendo dall'insieme di Julia, ne approfondì le caratteristiche, applicando le sue scoperte anche a discipline non puramente matematiche. Ad oggi, i frattali sono impiegati all'interno di settori come l'economia, la finanza,²⁹ la computer graphic, la fisica, la biologia, l'anatomia, la medicina.

28 La geometria euclidea è la geometria basilare che ha permesso l'introduzione dei concetti di Punto, Retta, Area, Piano ecc., e si basa sui cinque postulati di Euclide: 1) *tra due punti qualsiasi è possibile tracciare una e una sola retta*; 2) *si può prolungare un segmento oltre i due punti indefinitamente*; 3) *dato un punto e una lunghezza, è possibile descrivere un cerchio*; 4) *tutti gli angoli retti sono uguali fra loro*; 5) *data una retta r e un punto P che non appartiene alla retta, esiste una e una sola retta parallela ad r passante per P*.

Parte dei suoi studi si sono concentrati anche sul calcolo della lunghezza della costa della Gran Bretagna, arrivando a descriverla come un insieme di figure casuali auto-similari con dimensione frattale.³⁰

I frattali rappresentano un potente mezzo matematico per descrivere i comportamenti e gli oggetti naturali e al tempo stesso un nuovo modo di guardare la realtà. Le caratteristiche che li definiscono, inoltre, conferiscono loro una rilevanza notevole nello studio del *caos*; come Mandelbrot dimostrò analizzando il modello di Jean Baptiste Perrin, persino il *moto browniano* ha un comportamento frattale. Il *pink noise*, un tipo di rumore chiamato anche $1/f$ noise a causa del modo in cui le ampiezze di ogni relativa frequenza si dispongono (rapporto di $1/f$, appunto, dove f sta per frequenza), sembra avere una sua autocorrelazione ed essere per questo legato al mondo dei frattali.³¹ ³² Alcuni di essi rientrano nella descrizione di *sistemi dinamici caotici*,³³ modelli matematici che mostrano una sensibilità spiccata al minimo cambiamento delle condizioni iniziali, tant'è che persino alcune delle strutture fondamentali per la teoria del caos, come ad esempio l'*attrattore di Lorenz*, mostrano una dimensione non intera, e vengono quindi catalogato come dei particolare esempio di frattale.

29 La *Finanza Frattale* deriva dagli studi di Mandelbrot. Questa applica i calcoli della geometria frattale all'analisi dell'andamento dei mercati finanziari.

30 cfr. B. MANDELBROT, *How Long Is the Coast of Britain? Statistical Self-Similarity and Fractal Dimension* (1967).

31 cfr. M. BULMER, *Music from Fractal Noise*, Melbourne, University of Queensland (2000).

32 cfr. B. MANDELBROT, *Multifractals and 1/f noise*, New Haven, Springer-Verlag (1999).

33 cfr. KENNETH FALCONER, *Fractal Geometry. Mathematical Fundations and Applications*, Chichester, John Wiley & Sons LTD, (2003).

2.3 Tipologie di frattali

La geometria frattale ci offre numerosissimi esempi di oggetti con struttura auto-similare poiché ognuno di essi può essere descritto da uno specifico algoritmo ricorsivo. In base alla natura dell'algoritmo possiamo catalogare in maniera molto generica i frattali in tre diverse categorie: *frattali lineari*, *frattali non lineari* e *frattali aleatori*.

I frattali lineari sono generati da un'equazione di prim'ordine, per questo definita lineare³⁴. I frattali di questo tipo sono fra i più semplici ed hanno un aspetto piuttosto netto che richiama gli oggetti geometrici più classici; possono essere definiti tramite un algoritmo basato su di una trasformazione di un oggetto geometrico applicata ricorsivamente. Ad esempio, la *curva di Von Koch* parte da un segmento; tale segmento viene diviso in tre segmenti più piccoli uguali fra loro, per poi eliminare il segmento centrale e sostituirlo con altri due segmenti di pari lunghezza posti come a formare un triangolo equilatero. Questo procedimento viene reiterato un numero finito di volte per ogni segmento presente nella figura. Altri frattali di questo tipo, che verranno esposti in maniera più dettagliata più avanti, sono la curva di Peano, l'insieme di Cantor, il frattale della serie di Fibonacci,³⁵ il triangolo di Sierpinski.

I frattali non lineari sono invece generati da un'equazione di ordine maggiore di uno. Di questo tipo di frattali si sono occupati Gaston Julia e Pierre Fatou, il cui lavoro nei primi anni del XX secolo è stato fondamentale per lo sviluppo della geometria frattale. Dai loro studi nascono due insiemi complementari che prendono i loro nomi, ovvero l'insieme di Julia e l'insieme di Fatou, entrambi basati sulla funzione quadratica $f(z) = z^2 + c$. Dall'insieme di Julia scaturiranno in seguito gli esperimenti di Mandelbrot, la nascita del frattale di Mandelbrot, della famiglia di frattali basati sulla funzione $f(z) = z^n + c$ conosciuta anche come *Multibrot*, e di

34 Le equazioni di prim'ordine sono definite lineari poiché definiscono una retta sul piano cartesiano.

35 La serie di Fibonacci è una serie, detta anche successione aurea, è una serie numerica in cui ogni numero è la somma dei due precedenti (0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ecc...). Si tratta di una serie molto famosa e importante, grazie anche al suo legame con la sezione aurea, e viene utilizzata in molti campi, anche in ambito compositivo da personaggi come Stockhausen, Stravinsky, Xenakis, John Chowning, Ligeti. Da tale serie è anche possibile derivarne un frattale lineare.

tutto lo studio frattale odierno. I frattali aleatori sono la classe di frattali forse più particolare e rilevante in ambito artistico. Finora non è stato specificato che i frattali lineari e non lineari sono a conti fatti *deterministici*; sebbene il loro comportamento sia apparentemente caotico e, specialmente nei frattali non lineari si può notare una spiccata sensibilità anche al più piccolo cambiamento dello stato iniziale, essi sono interamente calcolabili e prevedibili. Difatti non possiedono alcun elemento aleatorio all'interno del procedimento che li genera. I frattali aleatori possiedono invece una componente casuale all'interno del loro algoritmo che li rende *non deterministici*. Esistono molti algoritmi aleatori di diverso tipo, ma è possibile fare un semplice esempio partendo dalla costruzione di un frattale lineare. Se riprendiamo in esame la curva di Von Koch, questa risulta deterministica, in quanto da un segmento di lunghezza l ed un numero finito di iterazioni otterremo sempre lo stesso risultato, la sua forma non cambia; questo procedimento può divenire aleatorio inserendo un elemento di casualità. L'algoritmo base prevede che ad ogni re-iterazione dell'algoritmo il segmento centrale venga rimosso e sostituito con due segmenti posti a formare un triangolo equilatero, ma se la sostituzione avviene inserendo i segmenti in modo casuale, la forma del frattale cambierà e ripetendo il tutto più volte otterremo sempre risultati diversi; ciò che ne scaturirà sarà un'immagine sì caotica, ma allo stesso tempo omotetica, la sua struttura interna sarà sempre la stessa. Tale procedimento può essere applicato a qualsiasi frattale di tipo deterministico; la condizione essenziale è che l'elemento aleatorio sia presente ad ogni passaggio dell'algoritmo ricorsivo. In sostanza, l'equazione o il procedimento geometrico ricorsivo deve avere al suo interno una funzione caotica.

I frattali aleatori sono in grado di realizzare forme molto simili agli oggetti e fenomeni naturali, sono la categoria di frattali che più si avvicina alla realtà, tant'è che sono utilizzati, come vedremo, nell'ambito della *computer graphic* per la realizzazione di oggetti e paesaggi naturali con risultati piuttosto realistici. Uno degli esempi di frattale aleatorio naturale è il moto *browniano*,³⁶ già citato in precedenza.

36 cfr. TOM LINDSTROM, *Brownian Motion on Nested Fractals*, Providence, American Mathematical Society (1992).

2.4 Esempi di frattali lineari

Si è parlato delle varie tipologie di frattali. Di seguito verranno illustrati alcuni frattali lineari, il rispettivo algoritmo generativo e la relativa dimensione di Hausdorff.

Il primo esempio, già in parte illustrato in precedenza, riguarda la curva di Koch.³⁷ A partire da un segmento di lunghezza l , questo viene diviso in tre parti uguali $\frac{l}{3}$, il segmento centrale viene eliminato e sostituito con altri due segmenti $\frac{l}{3}$ posti in modo da formare un triangolo equilatero con il segmento eliminato, per un totale di quattro segmenti $\frac{l}{3}$. Allo step successivo, questo procedimento viene riapplicato ad ogni segmento; per ogni segmento $\frac{l}{3}$ avremo quattro nuovi segmenti $\frac{l}{9}$ disposti secondo lo stesso procedimento, ottenendo in totale $4 * 4 = 16$ segmenti di lunghezza $\frac{l}{9}$. Questo oggetto frattale è una curva continua³⁸ ed ha un perimetro infinito, poiché ad ogni iterazione i segmenti di partenza vengono sostituiti da quattro segmenti di lunghezza pari ad $\frac{1}{3}$ del segmento di partenza, per un totale di $\frac{4}{3}$; ad ogni iterazione, il perimetro aumenterà di $\frac{4}{3}$. La dimensione di Hausdorff della curva di Koch è di $\log_3 4$, poiché la curva ad ogni iterazione si divide in quattro linee di grandezza pari ad $\frac{1}{3}$ della linea originaria. Questo frattale ha numerose varianti, una di queste è il fiocco di neve di Koch (*Koch snowflake*). Il procedimento è simile a quello della curva. Si parte stavolta da un triangolo equilatero; ogni segmento del triangolo viene suddiviso in tre parti $\frac{l}{3}$, la parte centrale viene rimossa e sostituita con altri due segmenti di lunghezza $\frac{l}{3}$. Reiterando il procedimento otteniamo il fiocco di neve di Koch, un frattale a perimetro finito e con dimensione pari di Hausdorff pari a $\log_3 4$.

37 cfr. HELGE VON KOCH, *Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire* (1904).

38 Una curva viene considerata tale se è unidimensionale e continua. Esempi di curva sono la circonferenza e la retta.

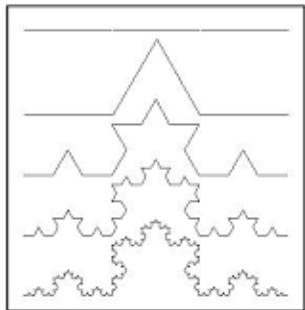


Fig. 2.1 Generazione curva di Koch

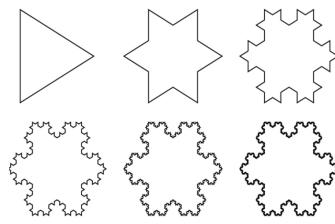


Fig 2.2 Generazione fiocco di neve di Koch

Altro frattale in parte simile alla curva di Koch è la curva di Peano;³⁹ L’idea della realizzazione di questo frattale nasce dall’intento di Giuseppe Peano di realizzare una curva che riempisse tutto lo spazio. La procedura per ottenere la curva di Peano parte da una figura base di tre segmenti di pari lunghezza l , posta come un quadrato senza il segmento superiore; a questo punto, il segmento inferiore viene sostituito da due figure tali e quali all’elemento base ma di grandezza quattro volte inferiore, per cui ogni segmento sarà lungo $\frac{l}{4}$, i segmenti laterali vengono sostituiti da una figura ciascuno, ma orientata una verso sinistra una verso destra. Dopodiché, queste figure vengono connesse fra loro, fatta eccezione per le figure superiori. Al passo successivo, la figura ottenuta viene replicata su ogni segmento presente con grandezza quattro volte inferiore, quindi otto volte inferiore alla figura di partenza, attuando le operazioni di rotazione e connessione eseguite alla prima iterazione. Replicando tale processo all’infinito, si ottiene la cosiddetta *curva limite*, che andrà a coprire tutto lo spazio in cui è definita.

Anche questa è una curva continua con perimetro illimitato; seguendo un ragionamento analogo a quello descritto per la curva di Koch, ad ogni iterazione i segmenti lunghi l vengono sostituiti con altri tre segmenti lunghi $\frac{l}{3}$, più altri tre di connessione, uno per ogni figura, ottenendo dunque quindici nuovi segmenti lunghi $\frac{l}{3}$. Sommando questi segmenti, otteniamo una lunghezza pari a 5 volte l , mentre la figura di partenza è pari a 3 volte l , per cui l’aumento è di $\frac{5}{3}$. Ad ogni passo del processo questo valore non cambia, per cui la curva aumenta di $\frac{5}{3}$ ad ogni iterazione.

39 cfr. GIUSEPPE PEANO, *Sur une courbe qui remplit toute une aire plane*, Mathematische Annalen, vol. 36, pp. 157-160, Torino (1890).

Tuttavia, per quanto riguarda la dimensione Hausdorff, questo frattale ha la particolarità di avere una dimensione intera pari a 2. Esistono anche altre curve in grado di riempire tutto lo spazio messo a disposizione, come la curva di Hilbert, tutte di dimensione di Hausdorff pari a due, poiché vanno a coprire l'intero piano bidimensionale.

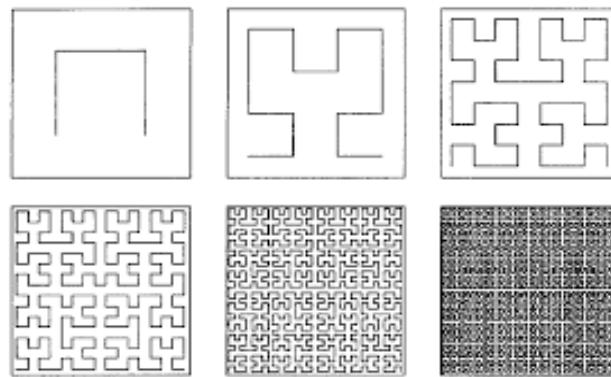


Fig. 2.3 Generazione della curva di Peano

Uno dei più importanti frattali è l'insieme di Cantor, definito da Georg Cantor.⁴⁰ Si tratta di un insieme molto importante e molto particolare, date le sue caratteristiche; esso è un sottoinsieme dei numeri reali, definito nell'intervallo $\{0,1\}$. Inoltre, come vedremo, è un frattale con perimetro nullo e contiene tutti i punti dell'intervallo in cui è definito. La sua costruzione è tra le più semplici; partendo da un segmento, questo viene diviso in tre parti uguali, per poi eliminare la parte centrale. Tale processo viene fatto su ogni segmento che ne scaturisce, fino ad ottenere dei singoli punti quando il numero di passi tende ad infinito. Per questa ragione viene anche chiamato col nome di *polvere di Cantor* ed ha un perimetro pari a 0, poiché all'aumentare delle iterazioni, la lunghezza di ciascun segmento diminuisce di $2/3$. La sua dimensione di Hausdorff è, appunto, $\log_2 3$.

40 cfr. GEORG CANTOR, *Sulla potenza degli insiemi perfetti di punti*, Acta Mathematica Vol. 2 (1884)

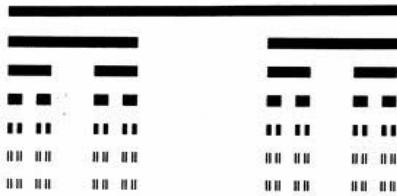


Fig. 2.5 Generazione dell'insieme di Cantor

L'ultimo frattale lineare che *verrà esposto* è il triangolo di Sierpiński. La generazione di questo frattale parte da un triangolo equilatero, ogni lato di lunghezza l . Per ogni lato, si individua il punto medio, ovvero il rispettivo punto pari a $\frac{l}{2}$, e si uniscono i punti, ottenendo un altro triangolo equilatero con lati pari a $\frac{l}{2}$ all'interno del triangolo originario. La figura che ne scaturisce è formata da 4 triangoli, di cui uno, il centrale, capovolto. Al passo successivo verranno trovati i punti medi dei lati di ogni triangolo non capovolto, per poi unirli in modo da formare altri triangoli con lati pari a $\frac{l}{4}$. Man mano che il processo viene re-iterato, maggiore sarà il numero di triangoli presenti nella figura e, di conseguenza, la complessità. Il triangolo di Sierpiński ha un perimetro che tende a zero, poiché questo diviene man mano pari a $\frac{3}{2}$ del valore precedente. La sua dimensione di Hausdorff, infatti è di $\log_3 2$.

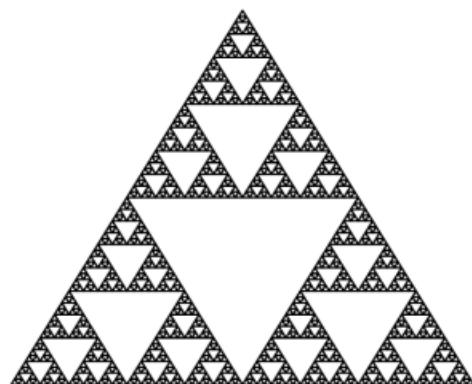


Fig. 2.6 Triangolo di Sierpinski

2.5 Il frattale di Mandelbrot

Esposti alcuni dei frattali lineari più famosi, verrà di seguito trattato il frattale più conosciuto ed importante, il frattale di Mandelbrot.⁴¹ Esso è in realtà un insieme a cui Mandelbrot, che lo scoprì, diede il suo nome; è un sottoinsieme del piano dei numeri complessi⁴² la cui successione $f(z)$, definita in tal modo

$$z_0 = 0 ; z_{n+1} = z_n^2 + c ; \quad \forall n \in \mathbb{R}; \quad c \in \mathbb{C};$$

risulta essere limitata e non caotica. La prima immagine di tale frattale risale al 1978 ad opera di Robert Brooks e Peter Matelski, fu poi ripresa e studiata da Benoit Mandelbrot nel 1980. Questo frattale deriva direttamente dall'insieme di Julia, il quale è basato sulla stessa funzione $f(z_{n+1}) = z_n^2 + c$, con $c \in \mathbb{C}$ ed è definito come l'insieme di tutti i punti c del piano complesso per i quali, dato un numero complesso z_0 di partenza, $f(z)$ tende ad infinito. Scegliendo un qualsiasi punto z_0 del piano complesso, il suo relativo insieme di Julia comprenderà tutti i punti c_1 che, sommati a z ad ogni iterazione del processo, fanno tendere all'infinito la funzione. L'insieme di Fatou, essendo complementare, comprende tutti i punti che convergono ad un valore.

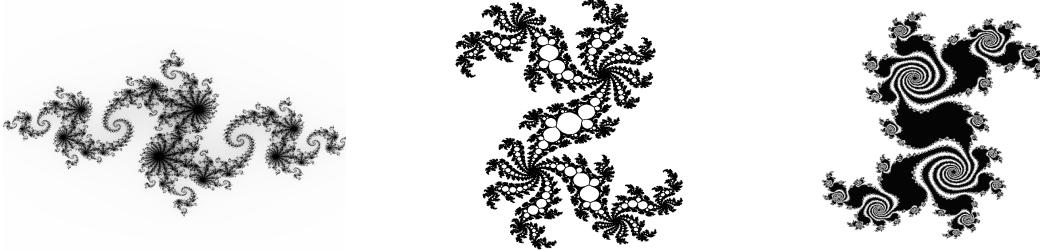


Fig. 2.6 Alcuni esempi dell'insieme di Julia.

41 BENOIT MANDELBROT, *Fractals and chaos: the Mandelbrot set and beyond*, Springer Verlag, New York (2004).

42 Il piano dei numeri complessi fu introdotto da Carl Friedrich Gauss. Alla base del piano complesso c'è il numero complesso e la scoperta del numero immaginario i , definito come un numero che, se elevato alla seconda, restituisce il valore -1 . Un numero complesso non è altro che una somma fra un numero reale ed un numero immaginario (multiplo di i). Il piano complesso pone è un piano cartesiano che presenta, sull'asse delle ascisse la parte reale, sull'asse delle ordinate la parte immaginaria.

L’insieme di Mandelbrot scaturisce anch’esso da $f(z_{n+1}) = z_n^2 + c$; la differenza è che questo frattale comprende tutti i punti c per i quali la ricorsione converge ad un certo valore, partendo da uno stato iniziale $z_0 = 0$. La colorazione delle figure di Mandelbrot realizzate al computer si basano su un algoritmo denominato *escape time*, in cui ogni punto viene colorato in base al numero di iterazioni che impiega per divergere, per cui maggiore sarà il numero di iterazioni che il sistema impiega a divergere, più forte sarà la colorazione. I punti all’interno dell’insieme rimangono neri, non assumono colore poiché convergono. Sia l’insieme di Julia che l’insieme di Mandelbrot hanno una dimensione di Hausdorff pari a 2.

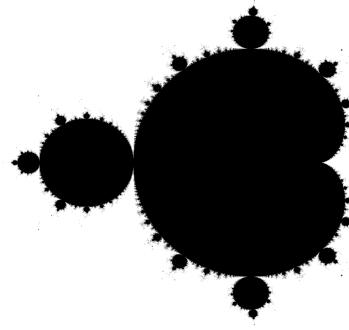


Fig. 2.7 Insieme di Mandelbrot

Ciò che accomuna l’insieme di Mandelbrot e l’insieme di Julia non è soltanto la funzione di partenza e la loro dimensione; tra loro c’è una corrispondenza riassumibile nella proprietà di indicizzazione degli insiemi di Julia insita nel frattale di Mandelbrot; in sostanza, i punti che sono nell’insieme di Mandelbrot possiedono tutti un insieme di Julia. Si dice che l’insieme di Julia è *connesso*⁴³ se questo è relativo ad un punto c del piano complesso che appartiene all’insieme di Mandelbrot. Altra particolarità a sostegno della forte connessione fra gli insiemi di Mandelbrot e Julia, è la presenza di immagini di vari insiemi di Julia all’interno del frattale di Mandelbrot, visibili effettuando uno *zoom* molto ampio su quest’ultimo.

43 Un insieme si definisce *insieme connesso* quando questo non può essere diviso in due o più insiemi aperti, non vuoti e disgiunti. Ad esempio, l’insieme costituito in \mathbf{R} dai punti $\{0,1\}$ e $\{1,2\}$ non è connesso, mentre l’intero insieme dei numeri reali \mathbf{R} è connesso. In parole semplici, un insieme connesso è formato “*tutto d’un pezzo*”.

Oltre alla funzione $f(z) = z^2 + c$, sono stati definiti altri frattali, sempre considerati frattali di Mandelbrot, basati su funzioni analoghe ma di ordine superiore a due. Questa generalizzazione viene chiamata *Multibrot* e rappresenta una famiglia di frattali per cui $f(z) = z^n + c$, con $n > 2$ ed $n \in \mathbb{I}$.

CAP. 3 Geometria frattale applicata alla musica elettronica

3.1 I frattali nella musica

L'utilizzo dei frattali nei vari campi scientifici e anche artistici, compresa la musica, ha subito una notevole impennata negli anni successivi alle scoperte di Benoit Mandelbrot. Connottendo i vari concetti finora esposti, è facile intuire come il calcolo frattale sia effettivamente un perfetto mezzo aleatorio, capace di offrire un comportamento caotico e allo stesso tempo strutturato. Il determinismo dei frattali lineari e non lineari, d'altro canto, può risultare uno spunto da cui trarre un procedimento, una scala o serie, se si preferisce, un insieme di valori da utilizzare nel processo compositivo; un esempio lampante è il compositore ungherese Gyorgy Ligeti, il quale ha mostrato un certo interesse per la geometria frattale, utilizzandone delle caratteristiche per realizzare alcune sue composizioni.

Ci sono, però, anche delle interessanti teorie riguardo al legame fra la musica di alcuni autori classici e la struttura frattale ben prima della nascita di tale termine. Il primo esempio riguarda la *Suite no. 3 per Violoncello solo* di J. S. Bach;⁴⁴ il musicista e matematico Harlan J. Brothers notò come la struttura di alcune frasi utilizzate da Bach in questa composizione richiami in qualche modo la figura dell'insieme di Cantor, con schemi ripetuti a diverse grandezze e annidati fra loro.⁴⁵ Ponendo l'esempio di queste 16 misure estratte dalla *Suite no. 3*, si può notare come le frasi *s1* ed *s2* siano costruite metricamente secondo lo schema *breve(m1) – breve(m2) – lungo(m3)* ed ognuna occupi 8 quarti. La frase *s3* dura il doppio, 16 quarti, ma segue la stessa struttura *breve – breve – lungo*, ma con valori raddoppiati.

44 J. S. BACH, *Suite n. 3 in do maggiore per violoncello solo*, BWV 1009, H. A. Probst, Vienna (1825).

45 H. J. BROTHERS, *Structural scaling in Bach's Cello Suite No. 3*, Fractals Vol. 15, World Scientific Publishing Company, Madison (2007).

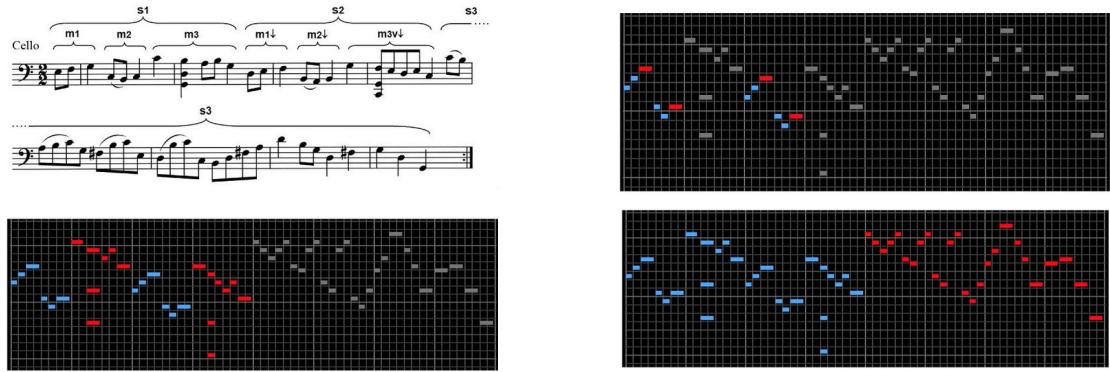


Fig. 3.3 Partitura e illustrazione dello schema breve-breve-lungo sui vari livelli

Di conseguenza, la struttura di s_1 ed s_2 può essere raffigurata in un modo che ricorda molto l'insieme di Cantor:

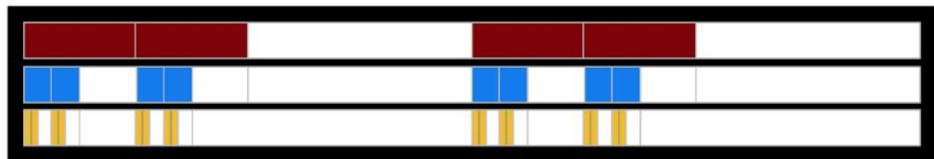


Fig. 3.4 Raffigurazione della struttura delle prime 16 misure della Suite no. 3

la parte rossa rappresenta lo spazio di 4 battiti, ovvero la lunghezza di un'intera frase, la parte blu indica le note da $\frac{1}{4}$ e la parte gialla indica le note da $\frac{1}{8}$.

Harlan J. Brothers, nella sua carriera di matematico e musicista, ha sempre nutrito particolare interesse nella relazione tra frattali e musica, cercando spesso di individuare una struttura frattale all'interno del suono o della composizione; Benoit Mandelbrot stesso espresse l'idea che fra i due mondi ci possa essere una forte connessione. Nel libro *Benoit Mandelbrot, a life in many dimension*,⁴⁶ Brothers fa notare come il rumore rosa, chiamato $1/f$ noise, rappresenti una sorta di legge intrinseca di distribuzione delle frequenze nella musica e che questa appaia in numerose opere degli artisti classici, fatto analizzato per la prima volta da Richard Voss e John Clarke, al punto che compositori come Charles Wuorinen e Charles Dodge usarono questa legge come punto di riferimento nella loro esplorazione

⁴⁶ MICHAEL FRAME, NATHAN COHEN, *Benoit Mandelbrot: a life in many dimension* vol. 1, Singapore, World Scientific Publishing Co. (2015).

nell'ambito della composizione algoritmica. In particolare, la formula $1/f^\beta$ permette di generalizzare questo fatto semplicemente definendo il valore di β ; un valore pari a 0 equivale ad un *white noise* (rapporto $1/1 = 1$), quindi una distribuzione omogenea con un risultato molto caotico, mentre un valore pari a 2 equivale al moto *browniano*, per cui la distribuzione risulterà fortemente dipendente dalla frequenza.

Questa distribuzione sembra essere valida anche per quanto riguarda durate e ritmiche; uno degli esempi che vengono posti è la *Sonata no. 6 in re maggiore* di Mozart,⁴⁷ chiamata anche *Sonata "Dürnitz"*; analizzandone le durate nel primo movimento, risulta che il numero di volte che ogni valore metrico appare è inversamente proporzionale alla sua stessa durata in un modo molto vicino alla distribuzione $1/f$, visibile rendendo logaritmiche le due grandezze.

Size (quarter notes)	Count
1/4	3322
3/4	323
5/4	104
7/4	21
9/4	9
11/4	10
13/4	3
15/4	0
17/4	0
19/4	0
21/4	0
23/4	2
25/4	1

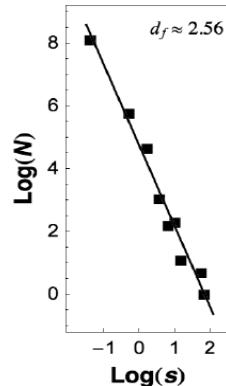


Fig. 3.4 Tabella con il conteggio dei risultati relativi alla Sonata no. 6 di Mozart e relativo grafico

47 WOLFGANG AMADEUS MOZART, *Sonata per pianoforte no. 6 in re maggiore K284*, Vienna, 1784.

Discorso analogo per le durate de *Le Tombeau de Couperin* di Maurice Ravel;⁴⁸ guardando i risultati con un adattamento logaritmico dei dati, la distribuzione si avvicina molto alla legge $1/f$.

Size (dotted quarters)	Count Version 1
1/6	1738
3/6	694
5/6	239
7/6	65
9/6	24
11/6	25
13/6	17
15/6	5
17/6	2
19/6	0
21/6	0
23/6	0
25/6	4

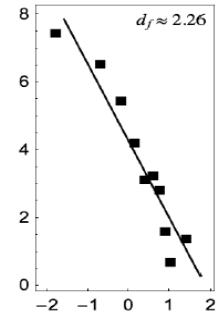


Fig. 3.5 Tabella con il conteggio dei risultati relativi a *Le Tombeau de Couperin* di Ravel e relativo grafico

Tale analisi è stata fatta anche per uno standard di musica jazz, *Don't Get Around Much Anymore* di Duke Ellington.⁴⁹ I risultati mostrano che anche in questo caso l'andamento si avvicina alla legge $1/f$.

Size (quarter notes)	Count
1/4	423
3/4	225
5/4	24
7/4	38
9/4	15
11/4	2
13/4	0
15/4	1

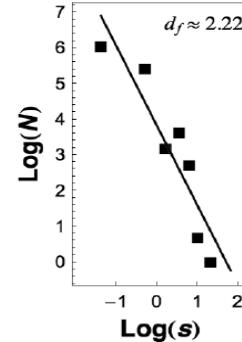


Fig. 3.6 Tabella con il conteggio dei risultati relativi all'esempio di Duke Ellington e relativo grafico

48 MAURICE RAVEL, *Le Tombeau de Couperin*, Parigi, Durand, 1919.

49 DUKE ELLINGTON, *Don't Get Around Much Anymore*, Ascap, 1942.

Differente risulta la presenza dei frattali nella musica di Gyorgy Ligeti; in tal caso, non si parla di distribuzione di frequenze, di ampiezze o di durate. Ligeti era notoriamente interessato alla geometria frattale e ne fa un uso fortemente reinterpretato e consapevole. In uno degli esempi più noti, *L'escalier du diable*,⁵⁰ Ligeti prende ispirazione dalla *funzione di Cantor*, osservando il modo in cui questa cresca verso l'alto, ogni volta di una piccola quantità; egli riporta tale andamento sul pianoforte, utilizzando la scala cromatica e compiendo un salto notevole a metà scala, imitandone l'andamento.

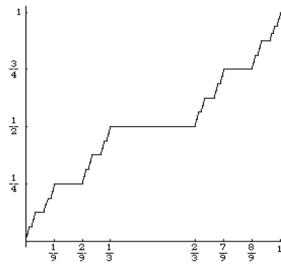


Fig. 3.7 Rappresentazione grafica della funzione di Cantor

dédicée à Volker Banfield
Étude 13: L'escalier du diable
 Auftragswerk des Süddeutschen Rundfunks Stuttgart für die Schwetzingen Konzerte

Presto legato, ma leggero, $\text{c} = 30$

*) 12 pp

una corda
quasi senza ped.

cresc. poco a poco - - - - -

(2)

9 7 9 11

Fig. 3.8 Estratto della partitura di *L'escalier du diable*.

50 GYORGY LIGETI, *Études for piano (book 2)* no. 13: *L'escalier du Diable*

3.2 Algoritmi per il calcolo frattale aleatorio

Il percorso finora descritto si è focalizzato nel mettere in evidenza tre aspetti che saranno fondamentali per il proseguo di questo elaborato: l'alea, intesa come processo in ambito musicale compositivo, la composizione algoritmica e la musica generativa, realizzata da processi aleatori tramite l'ausilio del computer, e la geometria frattale. Nell'intendo di realizzare una composizione aleatoria oppure una struttura generativa che si basi sul calcolo frattale, è necessario capire come sia possibile realizzare un procedimento frattale aleatorio. Come già detto in precedenza, un frattale aleatorio è realizzato da un algoritmo ricorsivo che include una componente caotica all'interno del suo processo; si possono prendere gli algoritmi descritti per alcuni frattali lineari e renderli aleatori semplicemente aggiungendo uno scostamento casuale ad ogni reiterazione.

L'algoritmo generale per la realizzazione di qualsiasi frattale, anche non aleatorio, è l'*Iterated Function System* (sistema di funzioni iterate),⁵¹ abbreviato *IFS*. Questo algoritmo è nato all'interno dello studio della teoria degli insiemi, per poi essere risultato un perfetto procedimento per la realizzazione dei frattali. Un *IFS* è definibile formalmente come un insieme di trasformazioni geometriche applicate all'interno di un piano; tutte le trasformazioni per la realizzazione di frattali come il triangolo di Sierpinski, l'insieme di Cantor, la curva di Peano, la curva di Koch non sono altro che trasformazioni geometriche reiterate, ovvero degli *IFS*. La descrizione generica consiste nel prendere un insieme di punti, traducibile in una qualsiasi figura geometrica o immagine, ed applicarvi le trasformazioni definite all'interno dell'*IFS* in maniera ricorsiva, ripetendo il processo ogni volta sopra al risultato precedente. In base al sistema definito, per ogni figura geometrica o immagine che verrà trasformata dall'*IFS*, il risultato finale dopo un numero sufficientemente elevato di volte sarà pressoché lo stesso; si dice che ogni *IFS* ha un suo attrattore caratteristico, una figura che viene sempre generata dopo un certo numero di iterazioni, indipendentemente

51 M. F. BARNSLEY, S. DEMKO, *Iterated function systems and the global construction of fractals*, London, Royal Society, 1985.

dalla figura di partenza. Un esempio di ciò può essere realizzato con l'*IFS* del triangolo di Sierpinski.



Fig. 3.9 Triangolo di Sierpinski realizzato a partire da un quadrato tramite IFS

L'algoritmo chiamato *Barnsley's Fern* per la realizzazione grafica di una foglia di felce è anch'esso un *IFS* basato su quattro trasformazioni geometriche: f_1 genera punti relativi alle foglie più piccole vicino al punto, f_2 genera i punti delle foglie più larghe a destra, f_3 genera i punti delle foglie più larghe a sinistra e f_4 genera i punti del gambo.

$$f_1(x, y) = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$$

$$f_2(x, y) = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 0.44 \end{bmatrix}$$

$$f_3(x, y) = \begin{bmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$$

$$f_4(x, y) = \begin{bmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Fig. 3.10 Generazione della foglia di felce tramite Barnsley's Fern

Esso è un algoritmo per la realizzazione dei frattali aleatori; viene scelto un punto del piano in maniera casuale, per poi scegliere casualmente una delle quattro trasformazioni. Le probabilità attribuite ad ogni trasformazione sono: $f_1 = 0.85$; $f_2 = 0.07$; $f_3 = 0.07$; $f_4 = 0.01$.

Generalizzando, un *IFS* non deterministico parte da un punto casuale del piano; dopodiché sceglie in maniera casuale una delle trasformazioni definite all'interno del sistema, ad ognuna delle quali è stato attribuito un valore di probabilità. A tale risultato riapplica una delle trasformazioni, scelta nuovamente in maniera casuale.

Reiterando il processo un numero piuttosto elevato di volte, si andrà a formare un frattale aleatorio. Nel caso del Barnsley's Fern, occorrono circa 1.000.000 punti.

La struttura degli *IFS* è praticamente la base di partenza per qualsiasi altro algoritmo per la generazione dei frattali; a partire da un dato input, che sia esso un punto o un insieme di punti, o che sia definito in uno spazio monodimensionale, bidimensionale o tridimensionale, ogni algoritmo applica delle trasformazioni in maniera ricorsiva. Persino algoritmi aleatori con regole stocastiche possono essere realizzati in maniera tale che questi producano figure frattali mediante la ricorsione. L'algoritmo più diffuso è il *Lindenmayer system*, abbreviato *L-system*;⁵² questo procedimento è di tipo grammaticale, ossia viene concepito come un alfabeto di simboli messo a disposizione per creare delle stringhe, ad ogni simbolo corrisponde una regola da applicare, ovvero una trasformazione che andrà ad espandere la stringa. Affiancando ad ogni lettera dell'alfabeto definito nel *L-system* una determinata figura basica e sostituendo poi la stringa con le rispettive figure, il frattale in questione verrà generato. Tramite un *L-system* è possibile definire anche i vari frattali di Cantor, Sierpinski, Peano e Koch. Questo algoritmo è utilizzato specialmente per realizzare modelli di piante sorprendentemente realistici. La definizione formale di un *L-system* è:

$$\mathbf{G} = (V, \omega, P);$$

in cui V è l'insieme dei simboli, ω è l'assioma, cioè la stringa iniziale, P è l'insieme delle regole assegnate ad ogni simbolo. In alcuni casi, specialmente nella realizzazione di modelli di piante, viene definito anche l'angolo di rotazione per alcuni tipi di trasformazioni. Le regole dietro ad ogni carattere possono assumere diverse sfumature, ad esempio possono essere *context sensitive*, vale a dire che la regola da applicare varia in base al carattere precedente e/o al carattere successivo. La variante aleatoria di questo algoritmo può essere realizzata secondo un modello stocastico, assegnando più regole ad un carattere, ognuna con un suo valore di probabilità, per poi sceglierle casualmente.

52 P. PRUSINKIEWICZ, J. HANAN, *Lindenmayer Systems, Fractals and Plants*, Regina, Springer-Verlag, 1992.

Axiom = F;
 $F \rightarrow FF+[+F-F-F]-[-F+F+F]$;
angolo = 22.5;

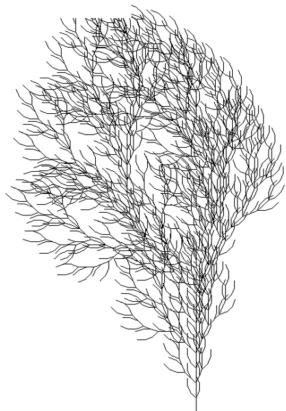


Fig. 3.11 Primo esempio di pianta realizzata con L-System

Axiom = X;
 $F \rightarrow FF$;
 $X \rightarrow F-[X]+X]+F[+FX]-X$
angolo = 22.5



Fig. 3.12 Secondo esempio di pianta realizzata con L-System

Due degli algoritmi più usati nel campo della *computer graphic* sono il *Diffusion-Limited Aggregation Aggregation*⁵³ (algoritmo di aggregazione a diffusione limitata), abbreviato *DLA* ed il *Diamond-Square Algorithm*,⁵⁴ abbreviato *DSA*, entrambi capaci di realizzare delle riproduzioni di paesaggi ed elementi naturali anche molto complesse e dettagliate.



Fig. 3.13 Il film “Avatar” di James Cameron è un esempio di computer graphic realizzata con algoritmi frattali.



Fig. 3.14 Un’immagine di paesaggio realizzata col software *Terragen* basato sull’algoritmo *Diamond-Square*.

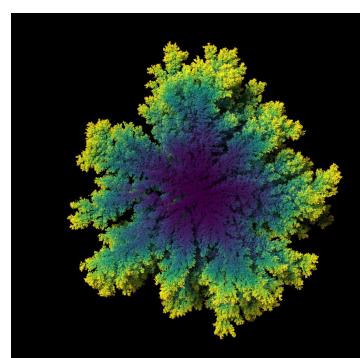


Fig. 3.15 Immagine realizzata con un algoritmo *Diffusion-Limited Aggregation*.

53 ANDRE OFFRINGA, *Diffusion Limited Aggregation*,

54 HEINZ-OTTO PEITGEN, DIETMAR SAUPE, *The Science of Fractal Images*, New York, Springer-Verlag, 1988.

L'algoritmo *DLA* deriva dal moto browniano, generato dal movimento delle particelle che collidono fra loro; per questo motivo le forme che ne derivano, spesso simili ad alberi o piante, vengono chiamate *Brownian Tree*. Il procedimento parte come un qualunque *IFS*, scegliendo un punto casuale nello spazio; in questo punto viene posizionata una particella. Successivamente, viene posizionata una seconda particella nello spazio disponibile, piuttosto lontana dalla prima, e viene fatta muovere in maniera casuale finché non collide con l'altra particella. Lo stesso viene fatto con la terza, che verrà fermata nel momento in cui entrerà in contatto con una delle altre due. Il processo viene quindi reiterato un numero sufficientemente elevato di volte.

L'algoritmo *Diamond Square* è invece quello più utilizzato in ambito di grafica tridimensionale, nei videogame e nel cinema per la realizzazione dei terreni. È conosciuto anche con il nome *random midpoint displacement fractal*. L'algoritmo basilare è definito a partire da una matrice $N*N$ con $N = (2^n + 1)$, di cui vengono selezionati gli angoli estremi come punti iniziali, considerando la matrice di valori come fosse un quadrato. Il processo di trasformazione che ne segue si divide in due fasi, una definita *diamond step* ed una *square step*; alla prima iterazione, nella fase *diamond step* andiamo a selezionare il punto medio del quadrato, che a questo punto dell'algoritmo sarà esattamente al centro della figura, per poi spostarlo di un valore casuale; in questo modo, la figura si può dividere in quattro triangoli. Nella fase successiva, lo *square step*, si vanno a trovare i punti medi per ogni triangolo scaturito dalla fase *diamond step*, spostando ogni punto di un certo valore casuale, dividendo quindi la figura in quattro figure quasi quadratiche. In breve, i due processi servono per dividere la figura man mano in quadrati irregolari sempre più piccoli e sempre diversi, dato lo scostamento aleatorio dei punti medi; ad ogni iterazione, il *diamond step* divide ogni figura in quattro triangoli, lo *square step* trasforma questi triangoli in quadrati irregolari. L'algoritmo termina nel momento in cui sono stati selezionati $N*N$ punti.

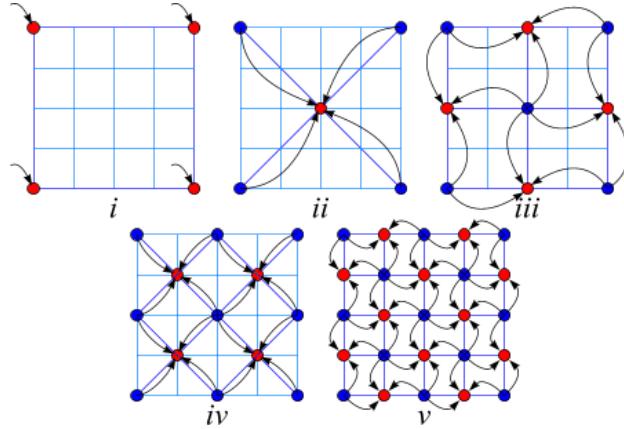


Fig. 3.16 Rappresentazione grafica dell’algoritmo Diamond-Square

La quantità di scostamento casuale dei punti medi risulta un parametro fondamentale che cambia le caratteristiche della figura risultante; viene chiamato *roughness* ed è compreso fra 0 e 1; per ogni punto medio, lo scostamento sarà compreso fra $-roughness$ e $+roughness$. Più questo valore è piccolo più la superficie risultante sarà liscia e pressoché regolare; al contrario, più sarà alto più la superficie sarà irregolare. Una volta ottenuta la cosiddetta *terrain map*, questa viene trasformata in un’immagine grafica tramite appositi algoritmi.⁵⁵

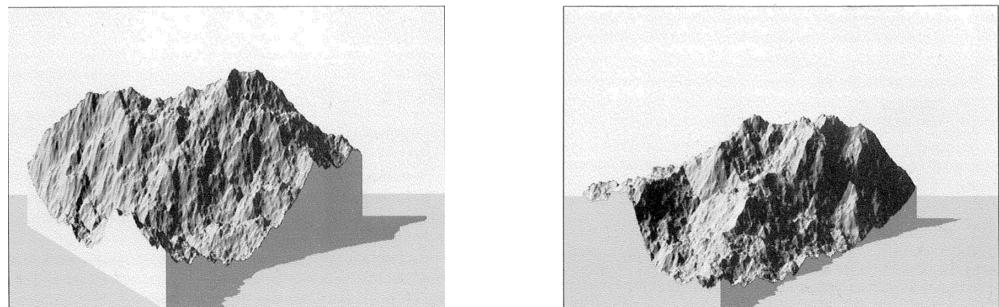


Fig. 3.17 Esempi di terreno realizzati con il Diamond-Square Algorithm

⁵⁵ Il *Tessellating* va a sostituire tutte le varie porzioni della *terrain map* con delle figure triangolari. Dopodiché vengono effettuate le varie operazioni grafiche per la definizione dei dettagli, come la colorazione, le ombre e le luci ecc.

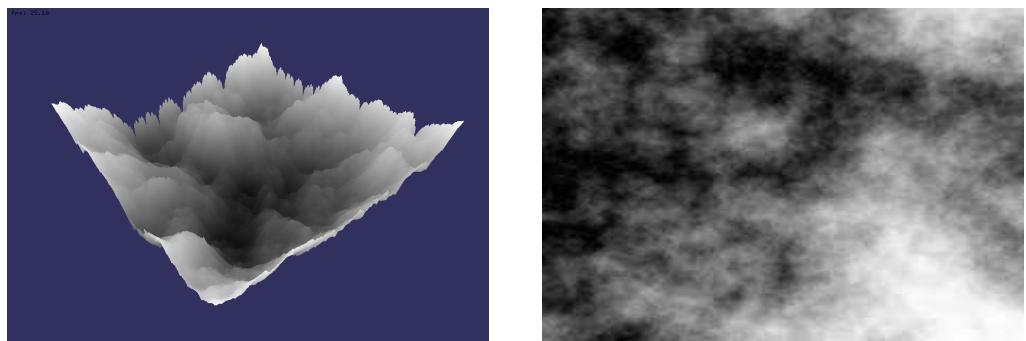


Fig. 3.18 Il *Diamond Square Algorithm* è capace anche di generare delle texture di nuvole

3.3 Impiego della geometria frattale nella composizione musicale

L’obiettivo del presente elaborato consiste nell’analizzare le possibilità dei frattali in ambito musicale in virtù delle loro caratteristiche più importanti, per poi concretizzare tutto ciò all’interno di un sistema per la musica generativa e di una composizione musicale. Per quanto concerne quest’ultimo punto, la geometria e gli algoritmi frattali possono essere indirizzati in diverse maniere verso un utilizzo compositivo. Oltre agli esempi che Ligeti e Bach hanno posto, seppure quest’ultimo non intenzionalmente, ed oltre alla legge distributiva $1/f$, intendo offrire alcuni spunti personali per l’utilizzo dei frattali nella musica che sono anche stati adottati, come vedremo, nella realizzazione di un brano di musica aleatoria/algoritmica.

Parte degli stratagemmi che intendo presentare si basano su un concetto molto semplice e diffuso soprattutto in ambito informatico, ovvero il processo di *mapping*,⁵⁶ che consiste nel trasferire una serie di dati analizzati o generati in un certo dominio ben definito di grandezze su di un altro dominio; in altre parole, si attua una sostituzione delle grandezze ed un eventuale adattamento dei valori. Prendendo come esempio un piano complesso raffigurante il frattale di Mandelbrot, l’operazione di *mapping* va a sostituire le grandezze sui due assi per poi, eventualmente riadattare la figura nella maniera più opportuna.

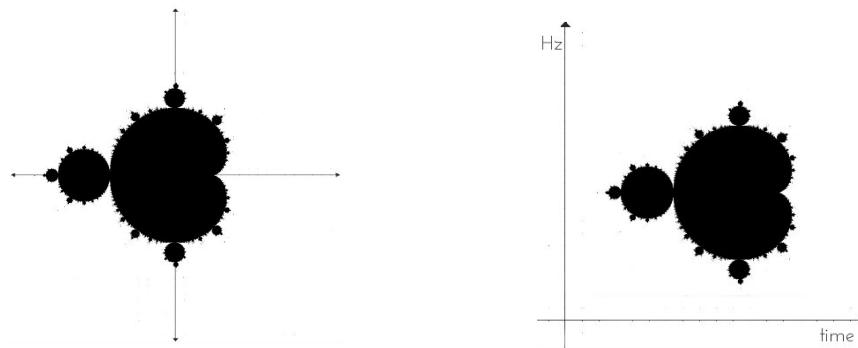


Fig 3.19 L’insieme di Mandelbrot sul piano cartesiano e l’insieme di Mandelbrot traslato nel quadrante positivo

⁵⁶ Il *mapping* si utilizza in vari campi scientifici. Un esempio già proposto è quello dell’algoritmo *DSA*; una volta generata la *terrain map* viene fatto un algoritmo di *mapping* per trasformare la matrice in un’immagine.

Una delle possibilità è utilizzare l'immagine frattale come spunto per realizzare una partitura; si tratterebbe più che altro di un'indicazione che lascia una certa libertà d'azione nella stesura della composizione. Se nell'asse delle ascisse consideriamo il tempo, l'asse delle ordinate può ad esempio indicare la presenza di suono in una determinata fascia spettrale; oppure, in maniera più astratta, si può considerare la superficie nera del frattale per la sua estensione verticale ad ogni istante di tempo come indicatore di dinamica generale della composizione, di sovrapposizioni, di complessità. In tal senso, si può considerare l'immagine frattale, in questo caso il frattale di Mandelbrot, come un indicatore della macrostruttura del brano; sarà necessario, tuttavia, scegliere opportunamente l'oggetto frattale di riferimento. Nel caso del frattale di Mandelbrot, ci troviamo di fronte ad una forma piuttosto marcata e compatta; se si prendesse l'insieme di Cantor, ad esempio, effettuare un *mapping* del tipo *tempo-ascisse densità-ordinate* risulta poco sensato, poiché tale frattale ha una forma fatta di più linee ben distinte e sovrapposte; in un certo senso, ha una “polifonia” intrinseca nella sua dimensione verticale.

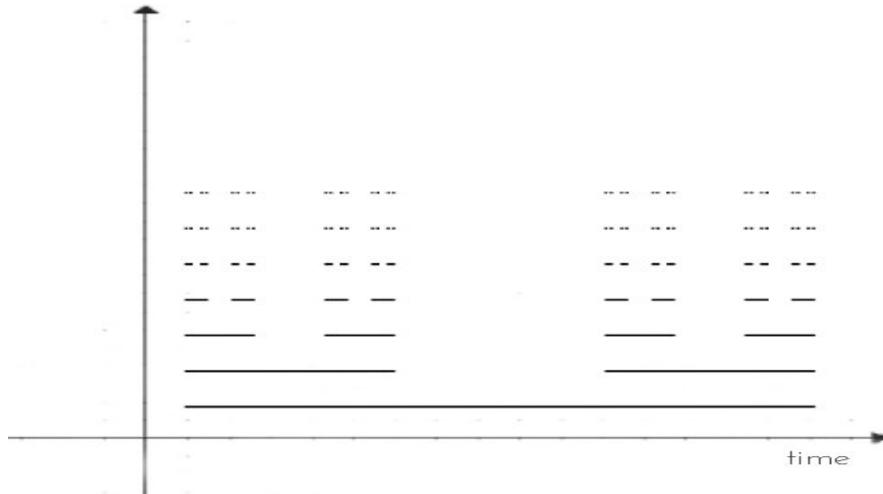


Fig. 3.20 L'insieme di Cantor posizionato nel quadrante positivo

Allo stesso tempo, l'insieme di Cantor utilizzato con un *mapping tempo-ascisse frequenze-ordinate* può risultare monotono se il range temporale è troppo dilatato. Una forma simile può essere fruttuosa se il range temporale definito nel *mapping* è relativamente breve oppure come una struttura generale, in cui ogni linea corrisponde

ad un evento sonoro o alla linea di uno strumento, considerando allo stesso tempo l'asse verticale come indicatore di un intervallo di frequenze in cui tale evento è posizionato.

Un ulteriore spunto scaturisce dalla caratteristica omotetica dei frattali; come abbiamo visto nell'esempio di Bach, è possibile riproporre la stessa struttura su più livelli, riproponendo la caratteristica omotetica nell'organizzazione delle varie sezioni del brano. Portando la cosa all'estremo, la composizione può essere organizzata in modo interamente omotetico, la struttura frattale viene ripetuta, alterata o meno, in ogni singola suddivisione. Tramite i moderni mezzi elettronici, la caratteristica omotetica è riproponibile persino nella struttura timbrica; si potrebbe infatti generare una linea melodica qualsiasi in qualsiasi stile musicale, per poi utilizzare tale linea melodica come sorgente sonora, riproducendola ad una velocità tale che generi un suono periodico (si parla di una velocità pari ad almeno qualche decina di volte superiore all'originale).

Per stabilire i valori di un parametro musicale qualsiasi, l'utilizzo della ricorsione è un'altra possibilità vicina ai procedimenti seriali; a partire da un valore arbitrario possiamo applicare un procedimento matematico in maniera ricorsiva e utilizzare il risultato che scaturisce ad ogni passo per controllare un aspetto della composizione, come ad esempio la frequenza di una linea musicale. La via più semplice è quella di scegliere il procedimento matematico da applicare in base a ciò che si desidera ottenere; i procedimenti matematici dietro a frattali come l'insieme di Cantor, il triangolo di Sierpinski, o anche il frattale di Mandelbrot, sono procedimenti che lavorano su di un piano a due dimensioni, generando un insieme di punti con coordinate x ed y , mentre la definizione di un parametro musicale come la frequenza è praticamente monodimensionale; di conseguenza non è possibile riproporre questi processi direttamente, hanno bisogno di essere riadattati. Una strategia può essere quella di effettuare un *mapping* su due grandezze musicali, come ad esempio frequenza ed intensità, anziché tenere il tempo sull'asse delle ascisse, ottenendo dei risultati decisamente particolari; oppure fare una media dei valori che caratterizzano ciascun punto; o, semplicemente, utilizzare soltanto uno dei due assi.

Tutti i procedimenti descritti finora non escludono affatto una componente aleatoria al loro interno. Avvalendosi di un'organizzazione di tipo frattale e/o omotetica, si ha certamente un grado di libertà tale da lasciare il contenuto dettagliato di ogni parte ad un processo aleatorio. Riprendendo l'insieme di Cantor come esempio, le linee sovrapposte possono indicare una struttura generale di voci o strumenti o eventi sonori che possono essere gestite nella loro articolazione da procedimenti casuali. I procedimenti ricorsivi per la determinazione dei parametri possono includere anch'essi uno scostamento casuale ad ogni iterazione, in modo da risultare non deterministiche allo stesso tempo modellarli secondo una certa forma e gli algoritmi per la realizzazione dei frattali aleatori risultano molto potenti in un'ottica di questo tipo.

3.4 Geometria e calcolo frattale all'interno di una struttura generativa

Nel basare un sistema digitale per la musica generativa sull'universo frattale, ritornano molte delle ispirazioni già espresse riguardo alla composizione; trattandosi però di una musica generativa, l'alea risulta una caratteristica fondamentale all'interno del calcolo frattale. Una struttura generativa è caratterizzata da tre diversi aspetti: la gestione degli eventi sonori, che si occupa di calcolare metriche, ampiezze, frequenze ecc., la sintesi sonora, che si occupa della generazione e successivo *processing* del suono, e la *spazializzazione*, ovvero il posizionamento dei suoni nello spazio; la struttura generativa realizzata verrà descritta nel dettaglio più avanti, ma è organizzata esattamente in questo modo, con un'attenzione particolare posta nell'aspetto dello spazio.

La gestione degli eventi sonori passa quindi per una serie di algoritmi ricorsivi aleatori. Le problematiche da affrontare nella realizzazione di tali algoritmi riguardano innanzitutto la già citata bidimensionalità a fronte di una monodimensionalità costituita dal parametro che verrà controllato, in secondo luogo la caratterizzazione dell'algoritmo in modo tale che questo possa offrire un risultato sonoro più gradevole possibile. La struttura generativa realizzata produce degli eventi sonori che si susseguono e/o si accavallano, a seconda delle impostazioni scelte; gli aspetti di questi eventi sonori che sono stati controllati mediante algoritmi frattali aleatori sono metrica, frequenza ed intensità. Le metriche vengono gestite da un algoritmo *L-system* realizzato appositamente per gestire la ripartizione nel tempo di ogni evento; si tratta quindi di un sistema di grammatica formale, nel quale ogni simbolo dell'alfabeto ha una corrispondenza, nel caso della grafica nello spazio bidimensionale o tridimensionale, nel caso corrente nell'insieme delle durate definito e basato sulla notazione musicale comune (semibreve, minima, semiminima, croma ecc.). Questa caratteristica del *L-system* va ad ovviare al problema della bidimensionalità precedentemente posto.

L'algoritmo si dirama in due parti; la prima si occupa della creazione della cellula iniziale ed è basata su un *L-system* di tipo aleatorio, per cui: si sceglie il primo simbolo casualmente, viene poi generata tutta la stringa iniziale scegliendo casualmente fra una delle regole descritte, quattro per ogni carattere. La seconda parte è anch'essa un *L-system* che agisce ricorsivamente partendo dalla stringa base secondo un comportamento *context sensitive*. Viene quindi scelta una delle quattro sostituzioni previste per ogni carattere in base al carattere che lo precede, fino a sostituire ogni carattere sostituibile della stringa di partenza. In sostanza, ci sono due algoritmi frattali all'interno di uno. Le regole di sostituzione dei vari simboli sono state definite in modo tale da avvicinarsi più possibile alla distribuzione $1/f$, prendendo spunto dagli esempi posti da Harlan Brothers riguardo Mozart, Ravel e Duke Ellington. Perciò più il valore metrico è breve più volte comparirà nella stringa.

Per quanto riguarda il controllo della frequenza, la scelta è ricaduta su un *Iterated Function System* di tipo stocastico; anche in questo caso, il procedimento è diviso in due parti, una per la creazione della cellula iniziale, una per la sua trasformazione, con scelta casuale fra diverse tipologie: manipolazione dell'ordine dell'insieme dei valori generato, inserimento o scarto di un valore e calcoli matematici. Anche in questo caso, le regole stocastiche sono state definite al fine di ottenere una distribuzione di tipo $1/f$.

Per il controllo dell'intensità è stato scelto un algoritmo differente, basato sull'attrattore di Ikeda o *Ikeda map*, ovvero una particolare categoria di sistema dinamico. Un attrattore⁵⁷ è un insieme di valori che un sistema dinamico tende ad assumere dopo un certo numero di iterazioni detto *bacino di attrazione*; nel caso in cui un attrattore risulti avere una dimensione di Hausdorff non intera, questo viene chiamato *attrattore strano*. Edward Lorenz scoprì il primo attrattore strano, poi

57 cfr. JULIEN CLINTON SPROTT, *Strange attractors: creating patterns in chaos*, Madison, M & T Books, 1993.

chiamato *attrattore di Lorenz*, basato su di un sistema di tre equazioni differenziali⁵⁸ di ordine pari ad uno derivante dalle equazioni del moto.⁵⁹ ⁶⁰

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = \rho x - xz - y \\ \dot{z} = xy - \beta z \end{cases}$$

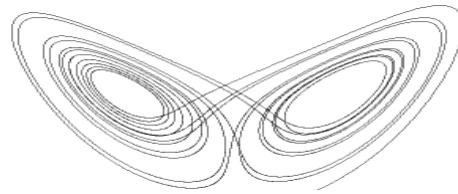


Fig. 3.21 L'attrattore di Lorenz

Nel caso del *Ikeda map*, non ci troviamo di fronte ad equazioni differenziali, ma ad una funzione complessa definita come

$$z_{n+1} = A + Bz_n e^{i(|z_n|^2 + C)}$$

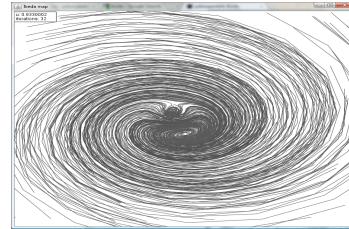


Fig. 3.22 L'attrattore di Ikeda

Il comportamento dell'attrattore di Ikeda presenta una dinamica più caotica; fu in principio proposto da Ikeda come modello comportamentale della luce che attraversa un risonatore ottico non lineare. La formula del *Ikeda map* riportata in un contesto bidimensionale è così formulata:

$$x_{n+1} = I + u (x_n \cos t_n - y_n \sin t_n);$$

$$y_{n+1} = u (x_n \sin t_n + y_n \cos t_n);$$

$$t_n = 0.4 - 6 / (1 + x_n^2 + y_n^2)$$

58 Un'equazione differenziale lega l'incognita ad una funzione i cui termini sono le derivate della funzione stessa.

59 Le equazioni del moto descrivono il moto di un sistema fisico in funzione dello spazio e del tempo.

60 cfr. JUN NI, *Principles of Physics: from Quantum Field Theory to Classical Mechanics*

Il comportamento di questo attrattore varia in maniera drastica in funzione del parametro u ; nel caso questo sia impostato come maggiore di 0.6 il comportamento sarà quello di un attrattore strano. Questi esempi mostrano i diversi comportamenti in relazione al parametro u dopo un numero di iterazioni di circa 1.000.000.

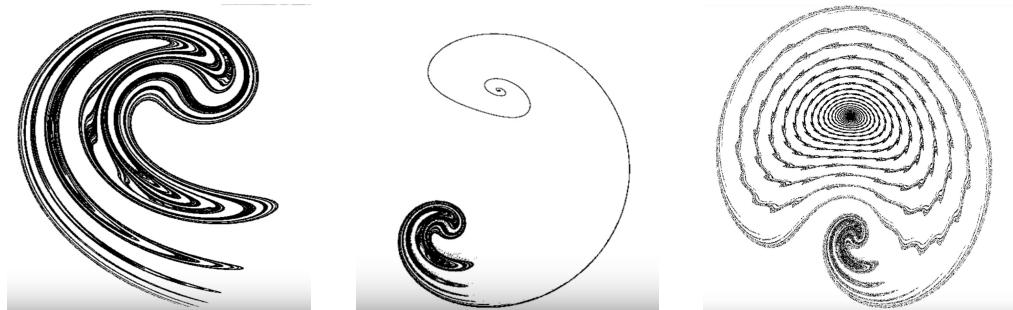


Fig. 3.23 Ikeda map con $u = 0.9$, con $u = 0.92$ e con $u = 0.99$

Partendo dall'algoritmo fondamentale del *Ikeda map*, ritroviamo la problematica della bidimensionalità e, in aggiunta, una problematica relativa alla quantità di valori; un risultato soddisfacente a livello grafico richiede circa 1.000.000 iterazioni, ma un numero così elevato di dati risulta ridondante e non si riesce a ricavare dei valori utilizzabili in un contesto di continua evoluzione come la musica, specialmente quella generativa. Per ovviare a tutto ciò, l'algoritmo realizzato per le dinamiche compie innanzitutto una media fra i valori dei due assi per ogni punto, per poi compiere una media fra tutti i valori calcolati dall'attrattore; posto che l'algoritmo compia 1.000.000 di passi, tutti i punti calcolati verranno raccolti in 64 gruppi e da ognuno di questi gruppi viene ricavato il valore medio, ottenendo un'unica serie di 64 valori che andranno ogni volta a controllare le dinamiche degli eventi sonori.

Gli algoritmi descritti necessitano dunque di una fase di adattamento per poter essere applicati in un contesto musicale, specialmente all'interno di una struttura generativa. Ciò che bisogna considerare in tale processo è la differente percezione che l'uomo ha di immagini e di suono; riportando un'immagine in una dimensione sonora l'effetto prodotto è drasticamente diverso, ed il processo di adattamento va

ponderato in maniera tale da estrarre dall’immagine le caratteristiche determinanti o, in un obiettivo di ricerca artistica, quelle che servono allo scopo designato.

Questo è proprio ciò che accade nella generazione sonora che è all’interno della struttura realizzata. Il motore di sintesi attua un *mapping* ben definito del tipo *ascissa-tempo ordinata-frequenza*, allo scopo di utilizzare un’immagine frattale come sonogramma. Al netto delle trasformazioni e adattamenti che il sistema può effettuare e che vedremo più avanti, agendo in questo modo la traslitterazione dell’immagine in suono non subisce adattamenti, per cui il risultato è sicuramente distante dalla percezione che si ha a livello visivo dell’immagine, salvo alcuni casi. Ciò che si mantiene, tuttavia, è il livello di complessità dell’immagine e la sua evoluzione, trasportata in un contesto temporale/spettrale, ed è ciò che si intende estrapolare tramite questo procedimento.

Per quanto riguarda la *spazializzazione*, viene implementato il sistema *ambisonics*,⁶¹ ⁶² un sistema per la simulazione di uno spazio reale tridimensionale piuttosto recente. L’idea originale risale agli anni ‘70 da parte di Michael Gerzon, ma fu allora abbandonata a causa dei mezzi ancora non sufficientemente sviluppati. È stata poi ripresa a partire dal nuovo millennio ed ha visto incrementare la sua diffusione negli ultimi dieci anni. Il formato *ambisonics* è sia una tecnica di registrazione sia una codifica multicanale del segnale audio in grado di replicare tutte le informazioni riguardo alla posizione nello spazio tridimensionale di una sorgente, ampliando quindi di molto le possibilità offerte dalla comune stereofonia, quadrifonia o ottofonia, non limitandosi a cambiare l’intensità di ogni altoparlante in relazione alla posizione desiderata, ma sfruttando ogni diffusore in modo che questo contribuisca alla resa sonora della posizione di un suono nello spazio. Offre, inoltre, la possibilità di una decodifica successiva ad un qualsiasi formato di riproduzione, che sia esso binaurale, multicanale planare,⁶³ o in formato 5.1.

61 cfr. FRANZ ZOTTER, MATTHIAS FRANK, *Ambisonics: a practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement and Virtual Reality*, Graz, Springer Nature, 2019.

62 ANGELO FARINA, *Ambisonics Pages*, <http://pcfarina.eng.unipr.it/Ambisonics.htm>, consultato il 10 febbraio 2020.

Il controllo della posizione spaziale nel formato *ambisonics* passa per due variabili, l'*azimuth*, ovvero la posizione orizzontale, e l'elevazione; la determinazione di questi due parametri tramite un algoritmo frattale propone un legame sicuramente più forte con quello delle immagini, poiché la percezione di posizione a livello sonoro è connessa alla percezione visiva. Per tale ragione, un algoritmo frattale non sembra necessitare di un adattamento troppo drastico; tuttavia, la musica non è un'immagine, ma un'arte evolutiva e in continuo cambiamento, indi per cui l'algoritmo non dovrà generare immagini, ma dovrà generare posizioni specifiche o traiettorie. Il processo realizzato si basa su una versione aleatoria del frattale del *Multibrot*, ovvero la famiglia di frattali che deriva dalla formula $f(z_{i+1}) = z_i^n + c$, dove n è un numero intero e c un numero complesso, formato quindi da una parte reale e una parte immaginaria. L'algoritmo è in grado di calcolare questo frattale in maniera ricorsiva, indipendentemente dal valore dell'esponente n .

Il procedimento, al passo iniziale, estrae i coefficienti per la parte reale e la parte immaginaria del numero complesso c e l'esponente n in maniera casuale; successivamente, il sistema esegue il calcolo $z_i^n + c$ in maniera ricorsiva, applicando al tempo stesso un leggero scostamento casuale dei due coefficienti prima di effettuare il calcolo. Il processo viene quindi ripetuto in maniera ricorsiva; ogni volta che il totale delle iterazioni o la *norma*⁶⁴ dei due valori superano un certo limite, l'algoritmo riparte dal suo stato iniziale, estraendo un nuovo esponente e due nuovi coefficienti. Ad ogni iterazione, sincronizzata ad una certa metrica di estrazione, il risultato corrente viene utilizzato come posizione nello spazio; la parte reale controlla l'*azimuth* e la parte immaginaria controlla l'elevazione, a seguito di un'opportuno adattamento dei valori dove è possibile notare come la distribuzione sia sì caotica, ma in parte prevedibile, dato che assume forme sempre simili. In quanto alla distribuzione, il sistema tende naturalmente ad un comportamento del tipo $1/f$, in cui i valori attorno al centro compaiono più spesso e i valori più lontani compaiono meno frequentemente.

63 Si intende una pluralità di diffusori acustici disposti attorno all'ascoltatore. Quadrifonia o ottofonìa sono tecniche di diffusione multicanale planari.

64 Si tratta di una sorta di media applicata ad un numero complesso. È calcolata come $0.5 * (a+b)^2$, dove a è il coefficiente della parte reale, b il coefficiente della parte immaginaria.

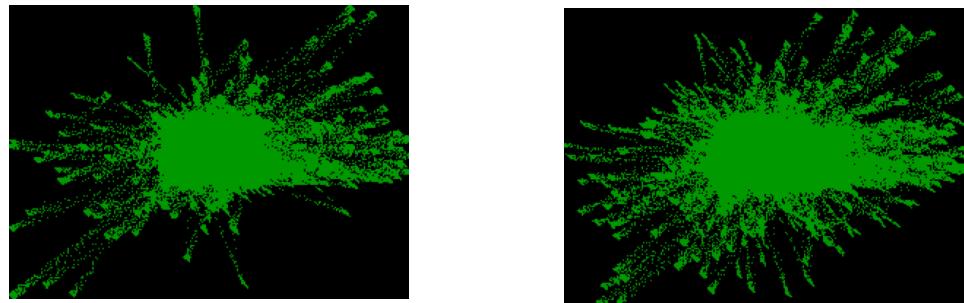


Fig. 3.24 Risultati del calcolo Multibrot dopo numerose iterazioni

Analizzando le figure, tuttavia, si evince un problema: un rapporto lineare nella fase di adattamento, a partire dai valori risultanti dall'algoritmo per ottenere dei valori utili al processo di *spazializzazione*, porterà il sistema a muovere la sorgente sempre nelle stesse aree, non occupando alcune porzioni di piano come la zona in alto a sinistra e la zona in basso a destra. Per tale ragione, l'adattamento dovrà seguire una curva logaritmica oppure esponenziale, piuttosto che un andamento lineare; così facendo, l'area in cui la sorgente si muove più frequentemente può essere ampliata, con una distribuzione logaritmica, o diminuita, con una distribuzione esponenziale, e le zone più estreme vengono talvolta coperte.

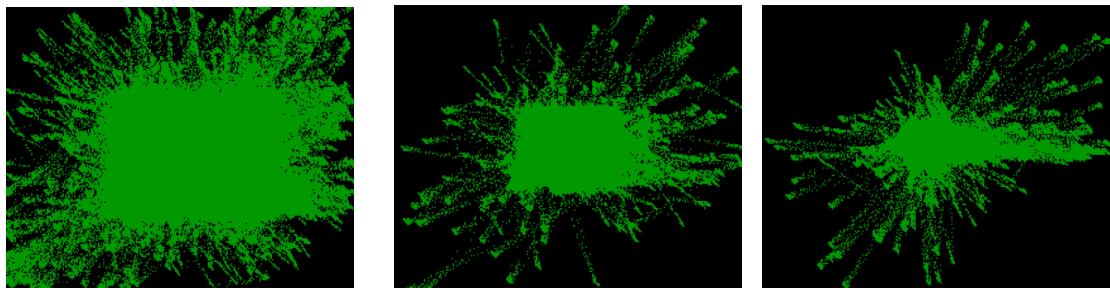


Fig. 3.25 Risultati del Multibrot con diversi tipi di distribuzione (da logaritmica a esponenziale)

CAP. 4 Fractal Autogenerative Network

4.1 Presentation Mode

Il sistema generativo è stato realizzato in Max/MSP,⁶⁵ un ambiente di sviluppo per applicazioni audio strutturato ad oggetti,⁶⁶ ⁶⁷ che offre la possibilità di realizzare applicazioni anche in ambito grafico tramite *Jitter*, una parte del linguaggio Max/MSP. Questa è stata la motivazione principale che ha portato alla scelta di questo software; dal momento che il sistema prevede di estrapolare delle informazioni da un'immagine, *Jitter* è in grado di favorire notevolmente questo processo.

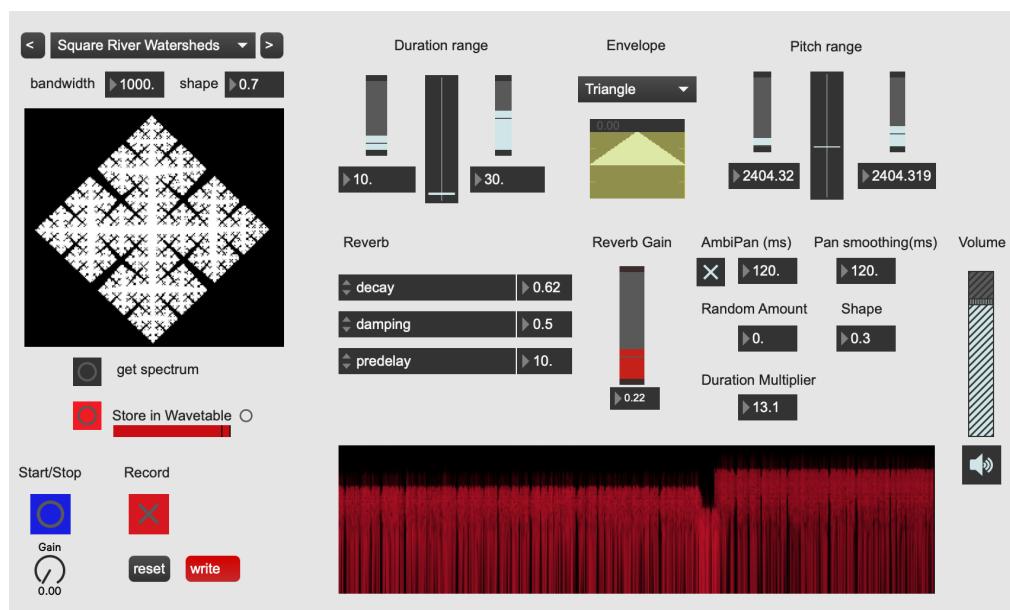


Fig. 4.1 Modalità di presentazione del Fractal Autogenerative Network

La *patch* si presenta divisa in sezioni. Per cominciare, una sezione riguarda l'estrapolazione dei dati per la sintesi sonora; possiamo scegliere fra diverse immagini di frattali, impostare la larghezza di banda ed il parametro di curvatura

65 V.J. MANZO, *Max/MSP/Jitter for music: a practical guide to developing interactive music system for education and more*, Oxford, Oxford University Press, 2016.

66 La programmazione orientata agli oggetti è un tipo di programmazione in cui ogni entità o struttura o funzione viene visualizzata come un oggetto.

67 ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.

della trasposizione in frequenza dell'asse verticale; in altre parole, permette di passare da una distribuzione logaritmica ad una esponenziale, passando per una distribuzione lineare se il parametro è impostato ad 1. Sotto al riquadro di visualizzazione dell'immagine, abbiamo i due controlli per estrarre i dati dall'immagine e per memorizzare il tutto in un *buffer* audio.

Troviamo poi i controlli per la sintesi sonora: il *Duration range*, che controlla l'intervallo di valori relativo alle durate, un selettore per l'inviluppo, il *Pitch range*, che controlla l'intervallo di valori di frequenza, i parametri di *decay*, *damping* e *pre-delay* del riverbero (in formato *ambisonics*, come vedremo), il *Reverb Gain*, ovvero l'ammontare di riverberazione, i controlli relativi alla spazializzazione, quindi l'intervallo di tempo tra un'estrazione e l'altra dei valori, lo *smoothing*, il *Random amount* per impostare l'ampiezza della componente aleatoria nel calcolo del *Multibrot*, lo *Shape* per la distribuzione nello spazio dei valori del *Multibrot*, ed il *Duration Multiplier*, un coefficiente che va ad aumentare o diminuire le durate attribuite ad ogni evento sonoro, in modo da garantire sia dei suoni ben separati temporalmente, sia dei suoni che si accavallano; in tal senso, in combinazione con il *Duration range*, la *patch* è in grado di cambiare il suo comportamento drasticamente, passando da flussi sonori ben distinti fino ad una pura sintesi granulare, impostando il *Duration range* fra i 2 e i 50 millisecondi e il *Duration Multiplier* ad un valore almeno pari ad 8. Infine, troviamo il controllo di *Start/Stop* per azionare e fermare la generazione, il *Gain* che controlla l'ammontare di distorsione, in modo da passare da suoni lisci e cristallini a suoni anche molto ruvidi, i controlli relativi alla registrazione del flusso audio, ed un sonogramma.

4.2 Gestione degli eventi sonori

Come anticipato, sono stati utilizzati tre algoritmi per frattali aleatori, ognuno atto a gestire frequenza, metrica e intensità.

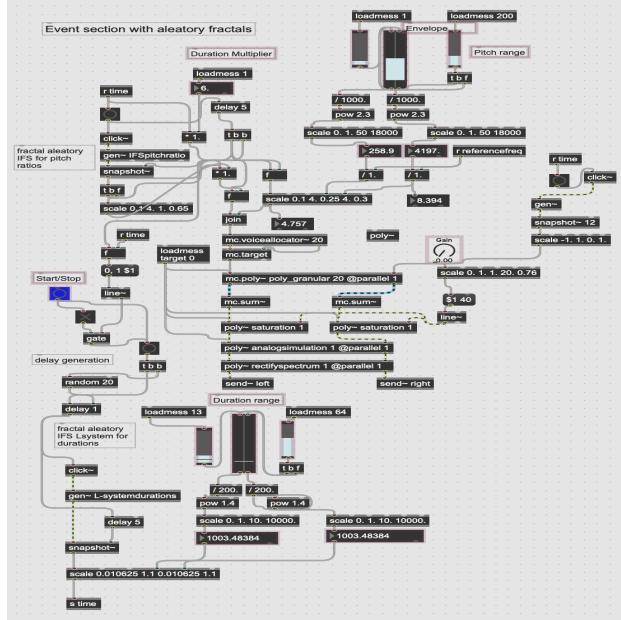


Fig. 4.2 La parte del software relativa alla gestione degli eventi sonori

Il tutto parte dall'algoritmo *L-system* per la gestione della metrica, che fa da *scheduler* agli eventi sonori e all'estrazione di frequenza ed intensità negli altri due algoritmi; il *L-system* restituisce il valore, dopodiché viene inviato un *bang*, che nel linguaggio Max/MSP indica un *trigger*, un comando che aziona una determinata funzione, agli altri algoritmi aleatori, in modo da azionare il loro calcolo; il bang viene inviato anche ad un timer che, nel momento in cui termina il periodo di tempo definito dal *L-system*, restituisce un bang che fa ripartire tutto il procedimento; il tutto viene adattato ai valori definiti negli oggetti grafici presenti nella modalità presentazione. Inoltre, è stata creata una relazione fra durata e frequenza, in modo tale che gli eventi sonori disposti in alta frequenza abbiano mediamente una durata inferiore rispetto a quelli in bassa frequenza.

I tre algoritmi aleatori sono stati realizzati mediante l'oggetto *gen~*, ovvero un oggetto di Max/MSP che permette di lavorare in un codice derivante dal linguaggio *C++*,⁶⁸ con una precisione più elevata e richiedendo meno risorse di calcolo.

Il codice per l'algoritmo *L-system* delle durate è il seguente:

```

Buffer metrics ("metrics");
Buffer initialmetrics("initialmetrics");
Buffer values ("values");
Buffer infometrics("infometrics");

trig = in1;

// set values
poke (values, 1/8, 0, 0);
poke (values, 1/4, 1, 0);
poke (values, 2/4, 2, 0);
poke (values, 1/16, 3, 0);
poke (values, 4/4, 4, 0);
poke (values, 1/32, 5, 0);
poke (values, 1/64, 6, 0);
poke (values, 3/4, 7, 0);
poke (values, 3/8, 8, 0);
poke (values, 3/16, 9, 0);
poke (values, 3/32, 10, 0);
poke (values, 3/64, 11, 0);
poke (values, 3/128, 12, 0);

if (trig == 1) { // set initial sequence
initialized = peek (infometrics, 0, 0);

if (initialized == 0) {
    rand_value = int (scale (noise(), -1., 1., 0, 6));
    val0 = peek (values, rand_value, 0, 0);
    poke (metrics, val0, 0, 0);
    //----set other values
    if (rand_value == 0) { // A case (1/8)
        rand_algo = int (scale (noise(), -1., 1., 1., 4.));

```

68 Il linguaggio *C++*, discendente dal linguaggio *C*, è uno dei linguaggi di programmazione più utilizzati al giorno d'oggi. Molte delle applicazioni software in campo audio sono scritte in questo linguaggio.

```

if (rand_algo == 1) { // AAA
    poke (initialmetrics, 1/8, 1, 0); // A
    poke (initialmetrics, 1/7, 2, 0); // A
    poke (infometrics, 3, 1, 0); // update number of values
} // end 1st rule
else if (rand_algo == 2) { // ABA
    poke (initialmetrics, 1/4, 1, 0);
    poke (initialmetrics, 1/8, 2, 0);
    poke (infometrics, 3, 1, 0);
} // end 2nd rule
else if (rand_algo == 3) { // AAC
    poke (initialmetrics, 1/8, 1, 0);
    poke (initialmetrics, 2/4, 2, 0);
    poke (infometrics, 3, 1, 0); // update number of values
} // end 3d rule
else if (rand_algo == 4) { // A --
    poke (initialmetrics, 3/16, 1, 0); //
    poke (initialmetrics, 3/16, 2, 0); //
    poke (infometrics, 3, 1, 0); // update number of values

} // end 4th rule
}// end A case
else if (rand_value == 1) { // B
    rand_algo = int ( scale ( noise(), -1., 1., 1., 4. ) );
    if (rand_algo == 1) { // BAA
        poke (initialmetrics, 1/8, 1, 0); //set A value (4/4)
        poke (initialmetrics, 1/8, 2, 0); //set A value (4/4)
        poke (infometrics, 2, 1, 0); // update number of values
    } // end 1st rule

    else if (rand_algo == 2) { // BADD
        poke (initialmetrics, 1/8, 1, 0); //A
        poke (initialmetrics, 1/16, 2, 0); //D
        poke (initialmetrics, 1/16, 3, 0); //D
        poke (infometrics, 4, 1, 0);
    } // end 2nd rule
    else if (rand_algo == 3) {
        poke (initialmetrics, 1/4, 1, 0); //B
        poke (initialmetrics, 1/8, 2, 0); //A
        poke (infometrics, 3, 1, 0); // update number of values
    } // end 3d rule
    else if (rand_algo == 4) { // BCB
        poke (initialmetrics, 2/4, 1, 0); //C
        poke (initialmetrics, 1/4, 2, 0); //B
        poke (infometrics, 3, 1, 0); // update number of values

    } // end 4th rule
} // end B case
else if (rand_value == 2) { // C
    rand_algo = int ( scale ( noise(), -1., 1., 1., 4. ) );
    if (rand_algo == 1) {
        poke (initialmetrics, 2/4, 1, 0); //C
        poke (infometrics, 2, 1, 0); // update number of values
    } // end 1st rule
}

```

```

else if(rand_algo == 2) {
    poke (initialmetrics, 1/8, 1, 0); //A
    poke (initialmetrics, 1/4, 2, 0); //B
    poke (infometrics, 3, 1, 0);
} //end 2nd rule
else if(rand_algo == 3) {
    poke (initialmetrics, 1, 1, 0); //E
    poke (initialmetrics, 1/16, 2, 0); //D
    poke (initialmetrics, 1/16, 3, 0); //D
    poke (infometrics, 4, 1, 0); // update number of values
} //end 3d rule
else if(rand_algo == 4) {
    poke (initialmetrics, 2/4, 1, 0); //C
    poke (initialmetrics, 4/4, 2, 0); //E
    poke (infometrics, 3, 1, 0); // update number of values

} //end 4th rule
} //end C case
else if(rand_value == 3) { //D
    rand_algo = int ( scale ( noise(), -1., 1., 1., 4. ) );
    if(rand_algo == 1) {
        poke (initialmetrics, 1/8, 1, 0); //A
        poke (initialmetrics, 3/32, 2, 0); //.
        poke (infometrics, 3, 1, 0); // update number of values
    } //end 1st rule
    else if(rand_algo == 2) {
        poke (initialmetrics, 1/16, 1, 0); //D
        poke (initialmetrics, 1/16, 2, 0); //D
        poke (infometrics, 3, 1, 0);
    } //end 2nd rule
    else if(rand_algo == 3) {
        poke (initialmetrics, 1/32, 1, 0); //F
        poke (initialmetrics, 1/32, 2, 0); //F
        poke (initialmetrics, 1/8, 3, 0); //A
        poke (infometrics, 4, 1, 0); // update number of values
    } //end 3d rule
    else if(rand_algo == 4) {
        poke (initialmetrics, 1/4, 1, 0); //B
        poke (initialmetrics, 3/16, 2, 0); //-
        poke (infometrics, 3, 1, 0); // update number of values

    } //end 4th rule
} //end D case
else if(rand_value == 4) { //E
    rand_algo = int ( scale ( noise(), -1., 1., 1., 4. ) );
    if(rand_algo == 1) {
        poke (initialmetrics, 1/4, 1, 0); //B
        poke (infometrics, 2, 1, 0); //update number of values
    } //end 1st rule
    else if(rand_algo == 2) {
        poke (initialmetrics, 1/8, 1, 0); //A
        poke (initialmetrics, 1/8, 2, 0); //A
        poke (infometrics, 3, 1, 0);
    } //end 2nd rule
}

```

```

else if(rand_algo == 3) {
    poke (initialmetrics, 1/16, 1, 0); // D
    poke (initialmetrics, 1/16, 2, 0); // D
    poke (initialmetrics, 1/8, 3, 0); // A
    poke (infometrics, 4, 1, 0); // update number of values
} // end 3d rule
else if(rand_algo == 4) {
    poke (initialmetrics, 1, 1, 0); // E
    poke (initialmetrics, 1/16, 2, 0); // D
    poke (initialmetrics, 1/32, 3, 0); // F
    poke (initialmetrics, 1/32, 4, 0); // F
    poke (infometrics, 5, 1, 0); // update number of values

} // end 4th rule
} // end E case
else if(rand_value == 5) { // F
    rand_algo = int ( scale ( noise(), -1., 1., 1., 4. ) );
    if(rand_algo == 1) {
        poke (initialmetrics, 1/32, 1, 0); // F
        poke (initialmetrics, 1/32, 2, 0); // F
        poke (initialmetrics, 1/32, 3, 0); // F
        poke (initialmetrics, 1/32, 4, 0); // F
        poke (infometrics, 5, 1, 0); // update number of values
    } // end 1st rule
    else if(rand_algo == 2) {
        poke (initialmetrics, 1/64, 1, 0); // G
        poke (initialmetrics, 1/64, 2, 0); // G
        poke (initialmetrics, 1/16, 1, 0); // D
        poke (infometrics, 4, 1, 0);
    } // end 2nd rule
    else if(rand_algo == 3) {
        poke (initialmetrics, 1/8, 1, 0); // A
        poke (initialmetrics, 3/32, 2, 0); // .
        poke (infometrics, 3, 1, 0); // update number of values

    } // end 3d rule
    else if(rand_algo == 4) {
        poke (initialmetrics, 3/64, 1, 0); //
        poke (initialmetrics, 1/16, 2, 0); // D
        poke (initialmetrics, 3/128, 3, 0); //
        poke (infometrics, 4, 1, 0); // update number of values

    } // end 4th rule
} //end F case
else if(rand_value == 6) { // G
    rand_algo = int ( scale ( noise(), -1., 1., 1., 4. ) );
    if(rand_algo == 1) {
        poke (initialmetrics, 1/64, 1, 0); // G
        poke (initialmetrics, 1/64, 2, 0); // G
        poke (initialmetrics, 1/64, 3, 0); // G
        poke (initialmetrics, 1/64, 4, 0); // G
        poke (infometrics, 5, 1, 0); // update number of values
    } // end 1st rule

    else if(rand_algo == 2) {

```

```

        poke (initialmetrics, 1/64, 1, 0); // G
        poke (initialmetrics, 1/32, 2, 0); // F
        poke (initialmetrics, 3/64, 3, 0); // {
        poke (initialmetrics, 1/64, 4, 0); // G
        poke (initialmetrics, 3/128, 5, 0); // }
        poke (infometrics, 6, 1, 0);
    } // end 2nd rule
else if (rand_algo == 3) {
    poke (initialmetrics, 3/64, 1, 0); // {
    poke (initialmetrics, 1/8, 2, 0); // A
    poke (initialmetrics, 3/128, 2, 0); // }
    poke (infometrics, 4, 1, 0); // update number of values
} // end 3d rule
else if (rand_algo == 4) {
    poke (initialmetrics, 1/16, 1, 0); // D
    poke (initialmetrics, 3/32, 2, 0); // .
    poke (initialmetrics, 1/64, 3, 0); // G
    poke (infometrics, 4, 1, 0); // update number of values
} // end 4th rule
} // end G case
} // end create initial sequence

// output
n_values = peek (infometrics, 1, 0);
currentcount = peek (infometrics, 2, 0);
val_x = peek(metrics, currentcount, 0);
poke(infometrics, val_x, 4, 0);
currentcount += 1;
currentcount = wrap (currentcount, 0, n_values -1);

//----- create new values
if (currentcount == 0) {
    poke(infometrics, 0, 3, 0);
    n_transformations = peek(infometrics, 4, 0);
    n_transformations += 1;
    poke(infometrics, n_transformations, 4, 0);
    prev_value = 0;
    next_value = 0;
    for (i=0; i<n_values; i+=1){
        writing_count = peek(infometrics, 3, 0);
        if (i==0) {
            prev_value = 0;
        }
        else {
            prev_value = peek (initialmetrics, i-1, 0);
        }
        if (i == n_values -1) {
            next_value = 0;
        }
        else {
            next_value = peek (initialmetrics, i+1, 0);
        }
        currentvalue = peek (initialmetrics, i, 0);
        if (currentvalue == 1/8) {
            if (prev_value == 1/8) { //first rule: BA ----> B AAB

```

```

    poke(metrics, 1/8, writing_count, 0); //A
    poke(metrics, 1/8, writing_count+1, 0); //A
    poke(metrics, 1/4, writing_count+1, 0); //B
    writing_count += 2;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);

} //end first rule
else if (prev_value == 1/4) {
    poke(metrics, 1/16, writing_count, 0); //D
    poke(metrics, 1/16, writing_count + 1, 0); //D
    writing_count += 2;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} // end second rule
else if (prev_value == 1/4 && next_value == 1/4) {
    poke(metrics, 1/8, writing_count, 0); //A
    poke(metrics, 1/8, writing_count+1, 0); //A
    poke(metrics, 1/4, writing_count + 2, 0); //B
    poke(metrics, 1/64, writing_count + 3, 0); //G
    poke(metrics, 1/64, writing_count + 4, 0); //G
    poke(metrics, 1/16, writing_count + 5, 0); //D
    poke(metrics, 1/16, writing_count + 6, 0); //D
    poke(metrics, 1/8, writing_count + 7, 0); //A
    writing_count += 8;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} //end 3rd rule
else {
    poke(metrics, 1/4, writing_count, 0); //B
    poke(metrics, 1/8, writing_count+1, 0); //A
    poke(metrics, 1/16, writing_count + 2, 0); //D
    poke(metrics, 1/16, writing_count + 3, 0); //D
    poke(metrics, 1/64, writing_count + 4, 0); //G
    poke(metrics, 1/64, writing_count + 5, 0); //G
    writing_count += 6;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
}
} // A context sensitive rules application
else if (currentvalue == 1/4) { // B rules
    if (prev_value == 1/8) {
        poke(metrics, 1/4, writing_count, 0); //B
        poke(metrics, 2/4, writing_count+1, 0); //C
        poke(metrics, 2/4, writing_count+2, 0); //C
        poke(metrics, 1/4, writing_count+3, 0); //B
        poke(metrics, 1/8, writing_count+4, 0); //A
        writing_count += 5;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end first rule
    else if (prev_value == 1/4) {
        poke(metrics, 1/8, writing_count, 0); //A
        poke(metrics, 1/16, writing_count + 1, 0); //D
        poke(metrics, 1, writing_count + 2, 0); //E
    }
}

```

```

        poke(metrics, 2/4, writing_count + 3, 0); // C
        writing_count += 4;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } // end second rule
else if (prev_value == 1/8 && next_value == 1/8) {
    poke(metrics, 1/8, writing_count, 0); // A
    poke(metrics, 1/64, writing_count + 1, 0); // G
    poke(metrics, 1/64, writing_count + 2, 0); // G
    poke(metrics, 1/16, writing_count + 3, 0); // D
    poke(metrics, 1/8, writing_count+4, 0); // B
    writing_count += 5;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} //end 3rd rule
else {
    poke(metrics, 1/4, writing_count, 0); // B
    poke(metrics, 1/8, writing_count + 1, 0); // A
    poke(metrics, 1/8, writing_count + 2, 0); // A
    poke(metrics, 1/64, writing_count + 3, 0); // G
    poke(metrics, 1/64, writing_count + 4, 0); // G
    writing_count += 5;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
}
} // B context sensitive rules application
else if (currentvalue == 2/4) {

    if (prev_value == 1/8) {
        poke(metrics, 1/4, writing_count, 0); // B
        poke(metrics, 1/4, writing_count+1, 0); // B
        writing_count += 2;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end first rule

    else if (next_value == 2/4) {
        poke(metrics, 1, writing_count, 0); // E
        poke(metrics, 3/4, writing_count + 1, 0); // F
        poke(metrics, 1/32, writing_count + 2, 0); // F
        poke(metrics, 1/32, writing_count + 3, 0); // F
        writing_count += 4;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } // end second rule
    else if (next_value == 1/8 || next_value == 1/4) {
        poke(metrics, 2/4, writing_count, 0); // C
        poke(metrics, 1/16, writing_count + 1, 0); // D
        poke(metrics, 1/4, writing_count + 2, 0); // A
        poke(metrics, 1/16, writing_count + 3, 0); // D
        writing_count += 4;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end 3rd rule
}

```

```

else {
    poke(metrics, 1/4, writing_count, 0); // B
    poke(metrics, 1/4, writing_count + 1, 0); // B
    poke(metrics, 3/16, writing_count + 2, 0); // -
    poke(metrics, 3/16, writing_count + 3, 0); // -
    writing_count += 4;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
}
} // C context sensitive rules application
else if (currentvalue == 1/16) { // D
    if (prev_value == 1/8) {
        poke(metrics, 3/32, writing_count, 0); // .
        poke(metrics, 3/64, writing_count+1, 0); // {
        poke(metrics, 1/32, writing_count+2, 0); // F
        poke(metrics, 1/32, writing_count+3, 0); // F
        poke(metrics, 1/64, writing_count + 4, 0); // G
        poke(metrics, 1/16, writing_count + 5, 0); // D
        poke(metrics, 1/64, writing_count + 6, 0); // G
        writing_count += 7;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end first rule
    else if (prev_value == 2/4) { // -C D ---> -E FF.A.D
        poke(metrics, 1/32, writing_count, 0); // F
        poke(metrics, 1/32, writing_count + 1, 0); // F
        poke(metrics, 3/32, writing_count + 2, 0); // .
        poke(metrics, 1/8, writing_count+3, 0); // A
        poke(metrics, 3/32, writing_count + 4, 0); // .
        poke(metrics, 1/16, writing_count + 5, 0); // D
        writing_count += 6;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } // end second rule
    else if (prev_value == 3/64 || next_value == 3/128) {
        poke(metrics, 3/32, writing_count, 0); // .
        poke(metrics, 1/64, writing_count + 1, 0); // G
        poke(metrics, 1/64, writing_count + 2, 0); // G
        poke(metrics, 1/8 , writing_count+3, 0); // A
        poke(metrics, 3/32, writing_count + 4, 0); // .
        writing_count += 5;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end 3rd rule
    else {
        poke(metrics, 1/16, writing_count, 0); // D
        poke(metrics, 1/16, writing_count + 1, 0); // D
        writing_count += 2;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } // end 4th rule
} // D context sensitive rules application
else if (currentvalue == 1) { // E

```

```

if(next_value == 1/4) {
    poke(metrics, 2/4, writing_count, 0); // C
    poke(metrics, 2/4, writing_count+1, 0); // C
    poke(metrics, 1/4, writing_count+2, 0); // B
    writing_count += 3;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);

} //end first rule

else if (prev_value == 2/4) {
    poke(metrics, 3/4, writing_count, 0); // [
    poke(metrics, 1/16, writing_count + 1, 0); // D
    poke(metrics, 1/16, writing_count + 2, 0); // D
    poke(metrics, 1/4, writing_count + 3, 0); // B
    poke(metrics, 3/8, writing_count + 4, 0); // ]
    writing_count += 5;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} // end second rule

else if (next_value == 1/32) {
    poke(metrics, 1, writing_count, 0); // E
    poke(metrics, 3/4, writing_count + 1, 0); // [
    poke(metrics, 1/16, writing_count + 2, 0); // D
    poke(metrics, 1/32, writing_count + 3, 0); // F
    poke(metrics, 1/32, writing_count + 4, 0); // F
    poke(metrics, 3/8, writing_count + 5, 0); // ]
    poke(metrics, 1/64, writing_count + 6, 0); // G
    writing_count += 7;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} //end 3rd rule

else {
    poke(metrics, 1, writing_count, 0); // E
    poke(metrics, 1/8, writing_count + 1, 0); // A
    writing_count += 1;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} // end 4th rule

} // E context sensitive rules application

else if (currentvalue == 1/32) { // F rules

if(prev_value == 1/4) {
    poke(metrics, 1/64, writing_count, 0); // G
    poke(metrics, 1/64, writing_count+1, 0); // G
    poke(metrics, 1/32, writing_count+2, 0); // F
    poke(metrics, 1/32, writing_count+3, 0); // F
    poke(metrics, 3/128, writing_count+4, 0); // }
    writing_count += 5;
    poke(infometrics, writing_count, 3, 0);
}

```

```

    poke(infometrics, writing_count+1, 1, 0);

} //end first rule

else if (prev_value == 2/4) {
    poke(metrics, 1/64, writing_count, 0); //G
    poke(metrics, 3/64, writing_count + 1, 0); //{
    poke(metrics, 1/32, writing_count + 2, 0); //F
    poke(metrics, 3/128, writing_count + 3, 0); //}
    writing_count += 4;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} // end second rule

else if (prev_value == 1/32) {
    poke(metrics, 3/32, writing_count, 0); //.
    poke(metrics, 1/32, writing_count + 1, 0); //F
    poke(metrics, 1/32, writing_count + 2, 0); //F
    poke(metrics, 3/16, writing_count + 3, 0); //-
    poke(metrics, 3/64, writing_count + 4, 0); //{
    poke(metrics, 1/64, writing_count + 5, 0); //G
    poke(metrics, 3/128, writing_count + 6, 0); //}
    writing_count += 7;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} //end 3rd rule

else {
    poke(metrics, 3/16, writing_count, 0); //-
    poke(metrics, 3/64, writing_count + 1, 0); //{
    poke(metrics, 1/64, writing_count + 1, 0); //G
    poke(metrics, 1/32, writing_count + 1, 0); //F
    poke(metrics, 3/128, writing_count + 1, 0); //}
    writing_count += 5;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} // end 4th rule
} // F context sensitive rules application
else if (currentvalue == 1/64) { // G
    if (prev_value == 1/64) {
        poke(metrics, 1/32, writing_count, 0); //F
        poke(metrics, 1/64, writing_count+1, 0); //G
        poke(metrics, 1/64, writing_count+2, 0); //G
        poke(metrics, 3/16, writing_count+3, 0); //-
        writing_count += 4;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end first rule
    else if (next_value == 1/32) {
        poke(metrics, 1/64, writing_count, 0); //G
        poke(metrics, 1/64, writing_count + 1, 0); //G
        poke(metrics, 1/64, writing_count + 2, 0); //G
        poke(metrics, 1/32, writing_count + 3, 0); //F
        writing_count += 3;
        poke(infometrics, writing_count, 3, 0);
    }
}

```

```

        poke(infometrics, writing_count+1, 1, 0);
    } // end second rule
    else if (next_value == 1/16) {
        poke(metrics, 1/64, writing_count, 0); // G
        poke(metrics, 1/16, writing_count + 1, 0); // D
        poke(metrics, 1/64, writing_count + 2, 0); // G
        poke(metrics, 1/32, writing_count + 3, 0); // F
        poke(metrics, 1/64, writing_count + 4, 0); // G
        writing_count += 5;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } //end 3rd rule
    else {
        poke(metrics, 1/32, writing_count, 0); // F
        poke(metrics, 1/64, writing_count + 1, 0); // G
        poke(metrics, 1/32, writing_count + 1, 0); // F
        writing_count += 3;
        poke(infometrics, writing_count, 3, 0);
        poke(infometrics, writing_count+1, 1, 0);
    } // end 4th rule
} // G context sensitive rules application
else { // costant case
    poke(metrics, currentvalue, writing_count, 0);
    writing_count += 1;
    poke(infometrics, writing_count, 3, 0);
    poke(infometrics, writing_count+1, 1, 0);
} // end costant case
} // end for. It reads the entire sequence
n_values = peek(infometrics, 1, 0); //read again n_values
for (i=0; i<n_values; i+=1) { //transfer data into initialmetrics
    val_0 = peek(metrics, i, 0);
    poke(initialmetrics, val_0, i, 0);
} // end transfer data
} // end currentcount = 0 ---- create newvalues
n_transformations = peek(infometrics, 5, 0);
if (n_transformations >= 4) {
    rand_reinit = noise();
    if (rand_reinit <= -0.7){ //reinit
        poke(infometrics, 0, 0, 0);
    } // end if rand_reinit < 0
} //end if n-transformation >= 4
//----- output values
} // end trig
out1 = peek(infometrics, 4, 0);

```

L'alfabeto di questo *L-system* possiede 7 caratteri con relative regole più 6 caratteri per le costanti, ovvero caratteri che non hanno regole di trasformazione, vengono lasciati inalterati. I 7 caratteri sono *A* per la croma ($1/8$), *B* per la semiminima ($1/4$), *C* per la minima ($2/4$), *D* per la semicroma ($1/16$), *E* per la semibreve ($4/4$), *F* per la biscroma ($1/32$) e *G* per la semibiscroma ($1/64$), mentre i 6 caratteri sono i relativi valori puntati, eccezion fatta per la semibreve, [per la minima

$(\frac{3}{4})$,] per la semiminima ($\frac{3}{8}$), - per la croma ($\frac{3}{16}$), . per la semicroma ($\frac{3}{32}$), { per la biscroma ($\frac{3}{64}$), e } per la semibiscroma ($\frac{3}{128}$). L'algoritmo è diviso in due *L-system* separati, uno stocastico per la stringa iniziale ed uno *context-sensitive* per le sostituzioni.

L'algoritmo iniziale estrae un carattere casuale fra i 7 non puntati dell'alfabeto, quindi ABCDEFG; dopodiché applica una scelta stocastica fra le 4 regole da applicare relative al carattere scelto, che sono:

$$\begin{aligned} A &\rightarrow 1. AAA; 2. ABA; 3. AAC; 4. A---; \\ B &\rightarrow 1. BAA; 2. BADD; 3. BBA; 4. BCB; \\ C &\rightarrow 1. CC; 2. CAB; 3. CEDD; 4. CBE; \\ D &\rightarrow 1. DA.; 2. DDD; 3. DFFA; 4. DB-; \\ E &\rightarrow 1. EB; 2. EAA; 3. EDDA; 4. EFDFF; \\ F &\rightarrow 1. EFFF; 2. FGGD; 3. FA.; 4. F\{D\}; \\ G &\rightarrow 1. GGGG; 2. GF\{G\}; 3. G\{A\}; 4. GD.G; \end{aligned}$$

Scelta una di queste trasformazioni iniziali, l'algoritmo inizia ad effettuare le sostituzioni in maniera ricorsiva. Le regole *context sensitive* sono:

$$\begin{aligned} A &\rightarrow 1. \text{ se preceduta da } B, \text{ allora la } A \text{ diventa } AAB; \\ &2. \text{ se preceduta da una } A, \text{ allora la } A \text{ corrente diventa } DD; \\ &3. \text{ se si trova fra due } A, \text{ allora la } A \text{ diventa } ABGGDDA; \\ &4. \text{ in tutti gli altri casi, la } A \text{ diventa } BADDGG; \\ B &\rightarrow 1. \text{ se preceduta da una } A, \text{ la } B \text{ diventa } BCCBA; \\ &2. \text{ se preceduta da una } B, \text{ la } B \text{ corrente diventa } ADEC \\ &3. \text{ se fra due } A, \text{ la } B \text{ diventa } AGGDB \\ &4. \text{ in tutti gli altri casi, la } B \text{ diventa } BAAGG \\ C &\rightarrow 1. \text{ se preceduta da una } A, \text{ la } C \text{ diventa } BB; \\ &2. \text{ se precede una } C, \text{ la } C \text{ corrente diventa } E[FF; \\ &3. \text{ se precede una } B \text{ o una } A, \text{ la } C \text{ diventa } CDAD; \\ &4. \text{ in tutti gli altri casi la } C \text{ diventa } BB-; \end{aligned}$$

$D \rightarrow$ 1. se preceduta da una A, la D diventa .
 $\{FFGDG;$
 2. se preceduta da una E, la D diventa FF.A.D;
 3. se posizionata fra una { e una }, la D diventa .GGA.;
 4. in tutti gli altri casi, la D diventa DD;
 $E \rightarrow$ 1. se precede una B, la E diventa CCB;
 2. se preceduta da una C, la E diventa [DDB];
 3. se precede una F, la E diventa E[DFF]G;
 4. in tutti gli altri casi, la E diventa EF;
 $F \rightarrow$ 1. se preceduta da una F, la F corrente diventa GGFF};
 2. se preceduta da una D, la F diventa G{F};
 3. se preceduta da una G, la F diventa .F-{G};
 4. in tutti gli altri casi, la F diventa -{GF};
 $G \rightarrow$ 1. se preceduta da una G, la G corrente diventa FGG-;
 2. se precede una F, la G corrente diventa GGGF;
 3. se precede una D, la G diventa GDGFG;
 4. in tutti gli altri casi, la G diventa FGF;

Ad ogni iterazione c'è un 50% di possibilità che l'algoritmo ricominci da zero, se sono state compiute almeno quattro trasformazioni. Come si può notare dalle regole di sostituzione, la seconda parte dell'algoritmo è deterministica; l'alea è concentrata soltanto nella prima parte dell'algoritmo. Facendo ripartire da zero l'esecuzione, in gergo informatico un *reinit*, si garantisce un continuo cambiamento e una copertura di tutti i simboli e tutte le possibilità cercando di seguire la legge distributiva $1/f$. La realizzazione di tutte queste casistiche passa per una serie di costrutti condizionali⁶⁹ che verificano il contesto grammaticale in cui il simbolo corrente si trova.

69 Un costrutto condizionale è detto *if* oppure un *if then else*. Un costrutto *if*, seguito da una condizione, agisce quando la condizione è vera. Un costrutto *if then else* consente di far eseguire una diversa serie di operazioni nel caso in cui la condizione non sia vera.

Il codice dell'algoritmo *IFS* per determinare la frequenza di ogni evento è il seguente:

```

Data factors(100, 1);
Data dump(100,1); //for random re-order
Data informations (10, 1);

Trig = in1;

//----set initial values everytime set_start = 0;
set_start = peek (informations, 0, 0);

if (set_start == 0) { //set initial cell
    n_elements = int( scale( noise(), -1., 1., 5, 10 ) );
    poke ( informations, n_elements, 1, 0);
    value_algo_1 = 1/n_elements;
    val_x = abs( noise() ); //getting absolute value, unipolar random
    val_x *= 9; // go from 0 to 9
    val_x += 1; // range between 1 and 10
    val_x = log10(val_x); // apply logarithmic ( x = 1 --> give 0
    //x = 10 ---> give 1
    val_x = scale ( noise(), 0., 1., 2., 0.1); //by this way we have more probability
    //that x will follow the distribution law 1/f
    poke(factors, val_x, 0, 0);
    for (i=1; i < n_elements; i+=1) {
        rand_value = noise();
        if (rand_value <= 0.) {
            val_x = (val_x * value_algo_1) + val_x;
            val_x = wrap(val_x, 0.1, 2.);
            poke (factors, val_x, i, 0);
        } //close if randvalue <= 0

        else {
            if (val_x < 1.) {
                val_x *= 8;
                val_x += 0.10*noise();
                val_x = wrap(val_x, 0.1, 4.);
                poke (factors, val_x, i, 0);
            }
            else if (val_x == 1.) {
                val_x = val_x*2;
                val_x = wrap(val_x, 0.1, 4.);
                poke (factors, val_x, i, 0);
            }
            else if (val_x > 1.) {
                val_x *= 0.15;
                val_x = wrap(val_x, 0.1, 4.);
                poke (factors, val_x, i, 0);
            }
        } //close else randvalue > 0
        poke (informations, 1, 0, 0);
    } //close for
} // close if set_start = 0
if (Trig == 1) {

```

```

//algorithms of manipulation

n_elements = peek (informations, 1, 0);
n_count = peek (informations, 2, 0);
if(n_count == 0) { // compute another algorithm
    rand_algo_type = int ( scale ( noise(), -1., 1., 0, 5 ) );
    poke (informations, rand_algo_type, 9, 0);

    if(rand_algo_type == 0) {
        //rand_algo = int ( scale ( noise(), -1., 1., 0, 2 ) );
        rand_algo = abs (noise());
        poke (informations, rand_algo, 8, 0);

        if (rand_algo < 0.25) {
            for (i=0; i<n_elements; i+=1) {
                lastvalue = peek(factors, i, 0);
                newvalue = lastvalue + 0.1 * noise();
                if(newvalue > 4.) {
                    newvalue = peek(factors, wrap(i-
1, 0, n_elements-1), 0);
                }
                newvalue = wrap(newvalue, 0.1, 4);
                poke (factors, newvalue, i, 0);
                poke (factors, lastvalue, n_elements-i + 1,
0);
            }
        } //end rand_algo < 0.25
        else if (rand_algo >= 0.25 || rand_algo <= 0.75) { //invert
sequence
            for (i=0; i<n_elements/2; i+=1){
                lastvalue = peek(factors, i, 0);
                newvalue = peek(factors, n_elements-
(i+1), 0) + 0.15 * noise();
                newvalue = wrap(newvalue, 0.1, 4);
                poke (factors, newvalue, i, 0);
                poke (factors, lastvalue, n_elements-
(i+1), 0);
            }
        }//end else if rand_algo = 1

        else if (rand_algo > 0.75) { //random reorder
            for (i=0; i<n_elements; i+=1){
                currentvalue = peek(factors, i, 0) + 0.15
* noise();
                currentvalue = wrap(currentvalue, 0.1,
4);
                poke (dump, currentvalue, i, 0);
            }
        }
        for (i=0; i<n_elements; i+=1) { //random
selection
            rand_order = int ( scale ( noise(), -1., 1.,
0, n_elements-1));
            check_value = peek(dump, rand_order,
0);
        }
    }
}

```

```

        while(check_value == 0) {
            rand_order = int ( scale
                ( noise(), -1., 1., 0, n_elements-1 ) );
            check_value = peek(dump,
                rand_order, 0);
        }
        if(check_value != 0) {
            poke (factors, check_value, i, 0);
            poke (dump, 0, rand_order, 0);
        }
    } //end for i=0; i<n_elements; i+=1
} //end else if
} //end if rand_algo_type = 0
else if (rand_algo_type == 1 || rand_algo_type == 2) { //insert/cut
one value
    if(n_elements <= 3) {
        newvalue = abs(noise()); //getting absolute value,
unipolar random
        newvalue *= 9; //go from 0 to 9
        newvalue += 1; //range between 1 and 10
        newvalue = log10(newvalue);
        newvalue = scale ( newvalue, 0., 1., 4, 0.25 );
        newvalue = wrap(newvalue, 0.1, 4);
        n_elements += 1;
        poke (informations, n_elements, 1, 0);
        poke (factors, newvalue, n_elements-1, 0);
    }
    else {
        rand_value = noise();
        poke(informations, rand_value, 8, 0);
        if (rand_value <= 0.5) { //add one value
            newvalue = abs(noise());
            newvalue *= 9;
            newvalue += 1;
            newvalue = log10(newvalue);
            newvalue = scale ( newvalue, 0., 1., 4,
0.25 );
            newvalue = wrap(newvalue, 0.1, 4);
            n_elements += 1;
            poke (informations, n_elements, 1, 0);
            poke (factors, newvalue, n_elements-1,
0);
        }
        else { //cut one value
            poke(informations, n_elements-1, 1, 0);
            n_elements -= 1;
            poke(factors, 0, n_elements, 0);
        }
    }
} //end else n_elements > 3
} //end algotype = 1-2
else if (rand_algo_type == 3) {

rand_algo = int ( scale ( noise(), -1., 1., 0, 2 ) );
poke(informations, rand_algo, 8, 0);
}

```

```

        if (rand_algo == 0) { //log
            for (i=0; i<n_elements; i+=1){
                lastvalue = peek(factors, i, 0);
                newvalue = pow(e, lastvalue);
                newvalue = wrap(newvalue, 2., 4.);
                poke (factors, newvalue, i, 0);
            } // end for log
        } //end algo1
        else if (rand_algo == 1) { //square root
            for (i=0; i<n_elements; i+=1) {
                lastvalue = peek(factors, i, 0);
                newvalue = sqrt(lastvalue);
                newvalue = wrap(newvalue, 0.1, 4.);
                poke (factors, newvalue, i, 0);
            } // end for
        } // end algo2
        else if (rand_algo == 2) { //sin
            for (i=0; i<n_elements; i+=1){
                lastvalue = peek(factors, i, 0);
                lastvalue = scale ( lastvalue, 0.25, 4., -
twopi, twopi);
                newvalue = sin(lastvalue);

                newvalue = scale ( newvalue, -1., 1., 0.1,
4.);

                newvalue = wrap(newvalue, 0.1, 4);
                poke (factors, newvalue, i, 0);
            } //end for
        } // end algo3
    } // end algotype == 3
    else if ( rand_algo_type == 3) {
        randomshift = scale ( noise(), -1., 1., -3, 3);
        for (i=0; i<n_elements; i+=1) {
            lastvalue = peek(factors, i, 0);
            newvalue = randomshift + lastvalue;
            newvalue = wrap(newvalue, 0.1, 4);
            poke (factors, newvalue, i, 0);
        } // end for
    } // end algotype = 4
    else if ( rand_algo_type == 4) {
        n_algo_computes = peek (informations, 3, 0);
        if (n_algo_computes > 4) {
            poke(informations, 0, 0, 0);
            break;
        }
        else {
            continue;
        }
    } // end algo4
    currentvalue = peek(factors, n_count, 0);
    currentvalue = wrap(currentvalue, 0.1, 4);
    n_transformations = peek(informations, 3, 0);
    poke (informations, currentvalue, 4, 0); //store current value
    poke (informations, 1, 2, 0); //update counter
    poke (informations, n_transformations+1, 3, 0);

```

```

        } // end update factors
    else { // output factors
        currentvalue = peek(factors, n_count, 0);
        currentvalue = wrap(currentvalue, 0.1, 4);
        poke (informations, currentvalue, 4, 0); //store current value
        n_count += 1;
        if(n_count == n_elements-1) {
            n_count = 0;
        }
        poke (informations, n_count, 2, 0);
    } // end out factors
} //close trig = 1

out1 = peek (informations, 4, 0);
out2 = peek (informations, 9, 0);
out3 = peek (informations, 8, 0);

```

Questo algoritmo *IFS* estrae casualmente, alla sua prima iterazione, sia il numero di elementi sia il primo elemento, per poi scegliere fra 2 operazioni differenti: la prima prende il valore precedente e lo moltiplica per $1/n$ dove n è il numero di elementi estratto casualmente, per poi sommarlo al valore precedente ($x_1 = x_0 * 1/n + x_0$); la seconda moltiplica il precedente valore per 8 se questo è minore di 1, lo moltiplica per 0.15 se questo è maggiore di 1 oppure lo moltiplica per 2 se questo è pari ad 1. Creata e memorizzata la cellula iniziale, questa viene poi manipolata; ci sono 4 tipi di trasformazioni e per ognuna ci sono diverse possibilità, scelte casualmente. Il primo tipo di trasformazione è una rielaborazione della cellula; nel primo caso viene lasciata invariata, nel secondo caso, che fra i 3 ha il 50% di possibilità di essere scelto, la cellula viene invertita, nel terzo viene riordinata in maniera casuale. Il secondo tipo di trasformazione elimina l'ultimo valore o aggiunge un nuovo valore in coda, con un 50% di possibilità ognuno; nel caso in cui gli elementi siano soltanto 3, questo potrà soltanto aggiungere un altro valore. Il secondo caso è quello a cui è stata attribuita più probabilità di essere estratto, in modo da arrivare più facilmente a generare serie di valori non brevi. Il terzo tipo sceglie fra 3 diverse operazioni matematiche; la prima di queste è basata su una modifica dei di tipo logaritmico, la seconda calcola una radice quadrata e la terza calcola il seno dei valori memorizzati. La quarta tipologia trasposta tutti i valori della cellula di un certo valore scelto casualmente.

Anche in questo caso, ad ogni trasformazione applicata c'è la possibilità che l'algoritmo venga reinizializzato e riparta dallo stato iniziale a calcolare i valori.

L'ultimo algoritmo è relativo alle intensità ed è l'attrattore di Ikeda, già descritto nel capitolo precedente.

```

Buffer infointensities("infointensities");
Buffer ikedamap("ikedamap");
Buffer ikedamapinterp("ikedamapinterp");
trig = in1;
maxsteps = 1000000;
initialset = peek (infointensities, 0, 0);

if (initialset == 0) {
    // generating map
    rand_u = scale(noise(), -1., 1., 0.6, 1);
    poke(infointensities,rand_u, 1, 0);

    for (i=0; i<maxsteps; i+=1) {
        x = abs( noise()) * 2. ;
        y = abs( noise()) * 2. ;
        u = peek(infointensities, 1, 0);
        t = 0.4 - ( 6 / ( 1 + (x*x)+(y*y) ) );
        tmp = 1 + u * ((x * cos(t)) - (y * sin(t)) );
        midpoint = (t + tmp) / 2;
        poke (ikedamap, midpoint, i, 0);
    }
    //an average of values
    for (i=0; i<maxsteps; i+=15625) { // we read 15625 values at time
        //and make the average. We find 64 midpoints of the ikeda map
        point = 0;
        for (j=0; j<15625; j+=1) {
            point += peek(ikedamap, i, 0); // read ikedamap
            //accumvalue += point; //sum of values
        }
        point /= 15625; // average
        poke(ikedamapinterp, point, i/15625, 0);
    }
    poke(infointensities, 1, 0, 0);
} // initial set
if (trig == 1){
    count = peek(infointensities, 4, 0);
    x = peek(ikedamapinterp, count, 0);
    poke(infointensities, x, 5, 0);
    count += 1;
    if (count > 63){
        count = 0;
        poke (infointensities, 0, 0, 0);
    }
    poke (infointensities, count, 4, 0);
}
out1 = abs( peek(infointensities, 5, 0));

```

4.3 Motore di sintesi

La sintesi sonora di questa *patch* si suddivide in *due* fasi; la prima riguarda l'assunzione dei dati dall'immagine per poi trasformarli in suono, la seconda è la vera e propria generazione del suono.

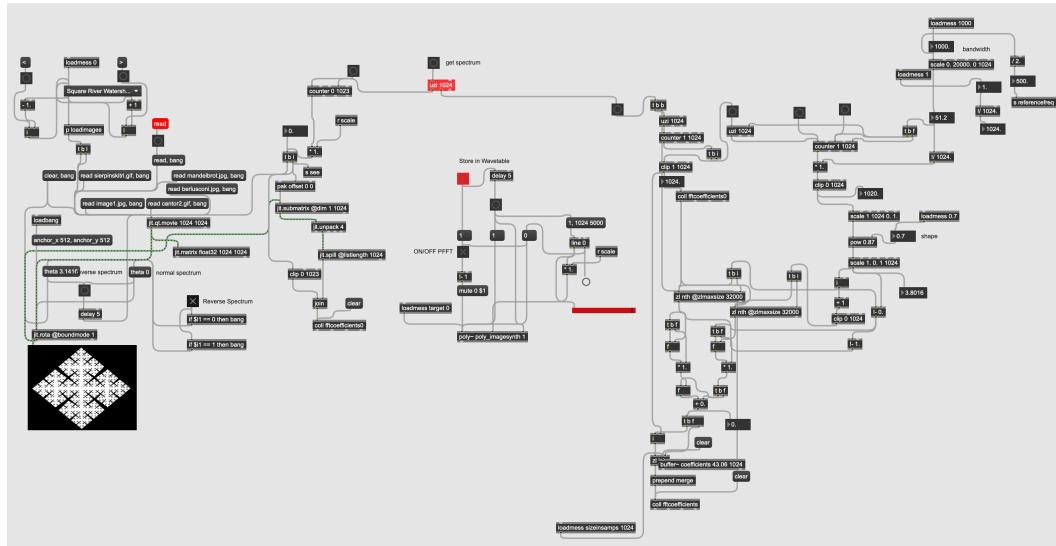


Fig. 4.3 Assunzione delle informazioni spettrali tramite Jitter

Una volta selezionata l'immagine, questa viene mostrata all'interno della matrice di visualizzazione – l'oggetto *jit.pwindow*. Tramite il *get spectrum*, il sistema riporterà i valori dei pixel all'interno di una *coll*, un file di testo all'interno di Max/MSP, per poi riadattare questi valori in base ai parametri di *bandwidth* e *shape*, in modo da avere un suono con una larghezza di banda limitata a piacere ed una distribuzione delle frequenze variabile, da logaritmico ad esponenziale. Azionando lo *Store in Wavetable*, questi dati verranno tramutati in suono tramite una sintesi spettrale e registrati all'interno di un *buffer* audio. L'oggetto *frame~* facilita questa operazione, prendendo i dati da ogni riga della *coll* e trasformandoli in coefficienti per la *pfft~* di Max/MSP.

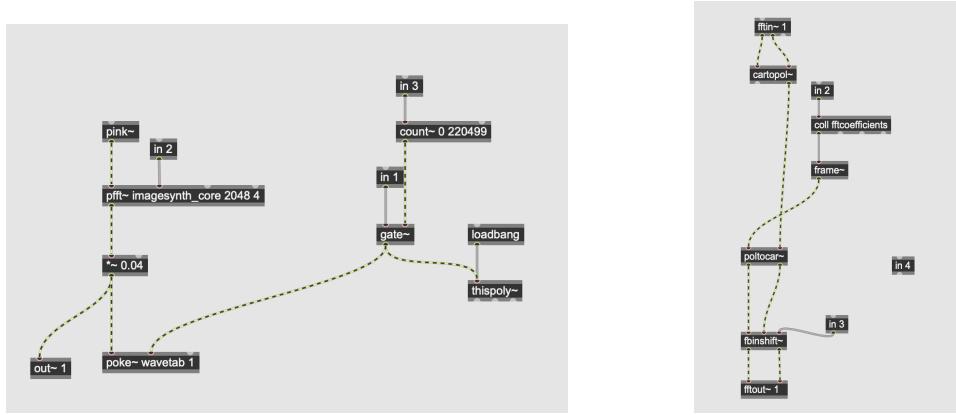


Fig. 4.4 Parti relative alla scrittura in tabella e alla generazione del suono

La seconda fase lavora tramite una sintesi granulare realizzata in *gen~* al fine di sintetizzare l'intera immagine; il *buffer* audio in cui è stato memorizzato il suono viene riprodotto a qualsiasi durata ed a qualsiasi frequenza grazie alla scomposizione granulare, ed a questo viene applicato un certo inviluppo selezionabile nei controlli principali.

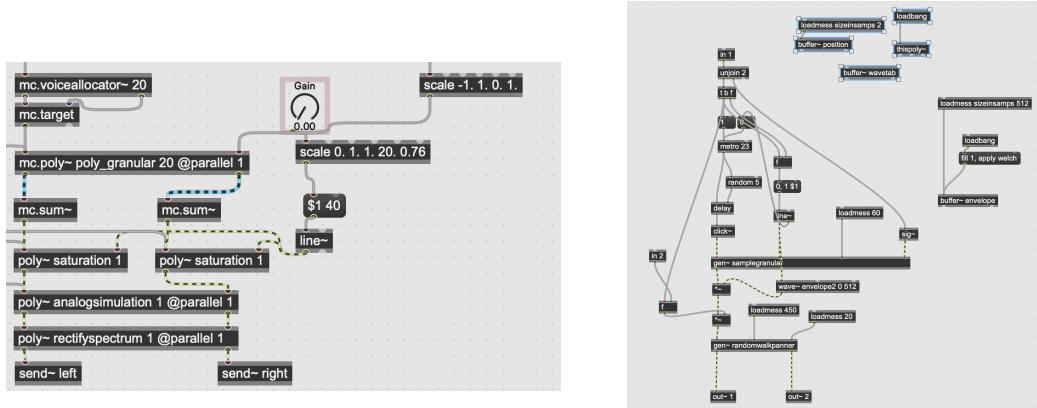


Fig. 4.5 Parti relative alla generazione degli eventi e alla sintesi sonora pura

Tramite il prefisso *mc*, gli oggetti di MaxMSP che operano su segnali audio entrano in modalità *multichannel*. In questa modalità, si è in grado di gestire a qualsiasi durata ed a qualsiasi frequenza tutti quei processi che richiedono diverse istanze della stessa porzione di programma; è infatti possibile realizzare una polifonia o una semplice granulare con pochissimi oggetti tramite l'utilizzo di *mc.voiceallocator* o *mc.noteallocator* a seconda dei casi per la gestione delle varie

istanze. Il suono che ne scaturisce viene poi processato tramite una saturazione, regolabile dal parametro *Gain*, un'ulteriore leggera saturazione stereofonica *mid-side*, in modo da ammorbidire e rendere più ampio il fronte sonoro (una simulazione del comportamento di una *console* analogica degli studi di registrazione degli anni '70) e un filtraggio elaborato del segnale, costituito da una serie di filtri ad un polo nel tentativo di avvicinarsi alla legge $1/f$ nell'ampiezza di ogni singola frequenza.

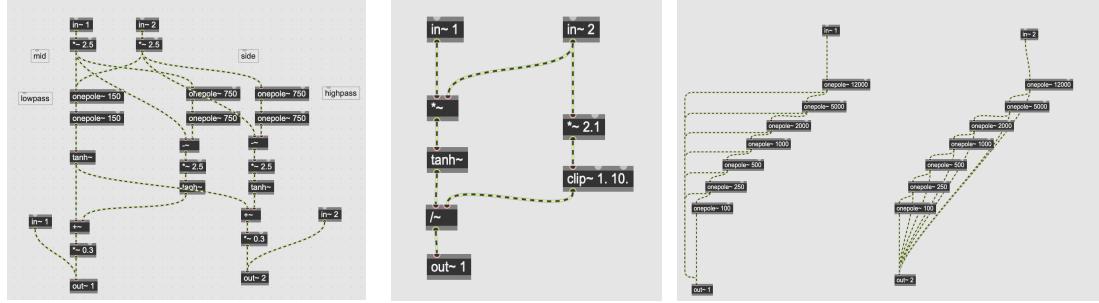


Fig. 4.6 Le parti relative al processing del suono

Il codice *gen~* relativo alla granulazione è il seguente:

```

Buffer wavetab1 ("wavetab");
Buffer envelope ("envelope");
Data util(50, 6);

maxgrains = 50;
table_length = dim(wavetab1);
envelope_table_length = dim(envelope);
soundout = 0;
Trig = in1;

position = in2 * table_length;
grainsize = mstosamps(in3);
grainpitch = in4;
graingain = in5;

if (Trig == 1) {
    for (i=0; i<maxgrains; i+=1) {
        busymap = peek(util, i, 5);
        if (busymap == 0) {
            dur = grainsize;
            grainphase = position;
            endpoint = position + (dur * float(grainpitch));
        }
    }
}

```

```

        endpoint = int(endpoint);
        phase_incr = grainpitch;
        envelope_phase = 0;
        envelope_phase_incr = (envelope_table_length-1) / dur;

        poke(util, grainphase, i, 0);
        poke(util, phase_incr, i, 1);
        poke(util, endpoint, i, 2);
        poke(util, envelope_phase, i, 3);
        poke(util, envelope_phase_incr, i, 4);
        poke(util, 1, i, 5);
        break;
    }
}
for ( i=0; i<maxgrains; i+=1 ) {
    busymap = peek(util, i, 5);

    if (busymap != 0) {

        phase = peek (util, i, 0);
        phaseincr = peek (util, i, 1);
        endpoint = peek (util, i, 2);
        envelope_phase = peek (util, i, 3);
        envelope_phase_incr = peek (util, i, 4);
        soundout = soundout + ( peek ( wavetab1, phase, 0, interp = "linear",
        boundmode = "clip") * peek ( envelope, envelope_phase, 0, interp = "linear", boundmode =
        "wrap") );

        nextphase = phase + phaseincr;
        nextphase_env = envelope_phase + envelope_phase_incr;
        if (nextphase < (endpoint) ) {
            phase = phase + phaseincr;
            poke (util, phase, i, 0);
        }
        else {
            poke(util, 0, i, 5);
        }

        if (nextphase_env < envelope_table_length) {
            envelope_phase = envelope_phase + envelope_phase_incr;
            poke (util, envelope_phase, i, 3);
        }
        else {
            poke(util, 0, i, 5);
        }
    }
}
outI = soundout;

```

Viene utilizzato un *buffer* come elemento di gestione polifonica dei vari grani; ogni volta che arriva il *trigger* esterno, un grano viene allocato all'interno di *util*, un *buffer* locale multicanale che può essere considerato come una tabella di dati; qui vengono scritti i valori necessari alla lettura della tabella *wavetab1* e della tabella d'inviluppo, come il punto di lettura, l'incremento di fase, il punto di fine lettura e lo status di *occupato* – *libero* per ogni grano. Il codice si divide in due cicli *for*,⁷⁰ uno per l'allocazione dei grani ed uno per la lettura. Il procedimento di allocazione è confinato all'interno di un controllo condizionale; ogni volta che la variabile *trig* risulta pari ad *1*, vale a dire ogni volta che si riceve il *trigger* esterno, il sistema inizializza il ciclo *for* in modo da controllare lo status di ogni singolo grano. Nel momento in cui uno status risulta libero, cioè pari a *0*, vengono memorizzate le informazioni relative alla produzione del grano, lo *status* viene settato ad *1*, cioè occupato ed il ciclo *for* viene interrotto tramite il comando *break*.

Il secondo ciclo *for* si occupa della lettura in tabella per ogni grano allocato; al contrario dell'inizializzazione, questo ciclo lavora continuamente. Vengono inizialmente controllati tutti gli status dei grani; ogni volta che questo risulta pari ad *1* vengono prese le informazioni necessarie per la riproduzione del grano, i *buffer* relativi al campione audio e all'inviluppo vengono letti e sommati alla variabile *soundout*, che fa da accumulatore per tutti i grani che vengono prodotti. Nel momento in cui la posizione di fase raggiunge il punto di fine lettura la posizione viene liberata settandone lo status a *0*. Infine, la variabile *soundout* viene mandata in *output*.

70 Il ciclo *for* è un *costrutto* dei linguaggi di programmazione. Ci sono diversi *costrutti*, che servono per effettuare determinate operazioni, spesso in maniera ciclica. Il ciclo *for* in particolare compie ciclicamente le operazioni definite al suo interno, finché una variabile utilizzata come contatore, solitamente chiamata *i*, non raggiunge il limite prestabilito.

4.4 Spazializzazione e codifica Ambisonics

L'intero flusso sonoro generato dal motore di sintesi subisce un processo di *encoding ambisonico* di ordine 3.

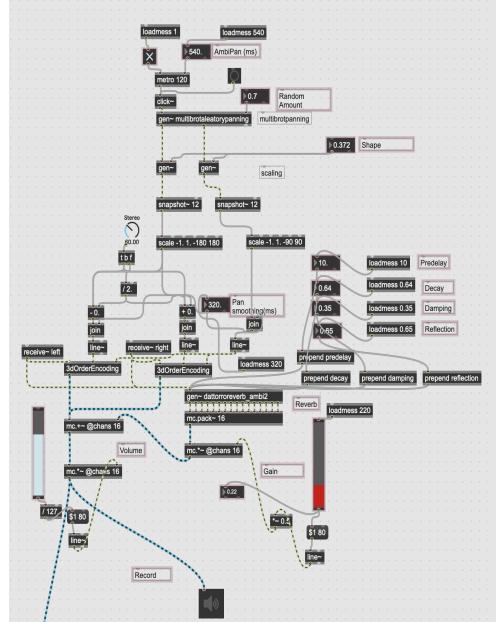


Fig. 4.7 Parte del codice relativa alla spazializzazione

L'ordine, nel formato *ambisonics*, indica la quantità di canali che vengono utilizzati per la riproduzione; un ordine maggiore favorisce una maggiore definizione nella risoluzione spaziale ed uno *sweet spot*, l'area ottimale d'ascolto di tale formato, più ampio. Nel dettaglio, ogni ordine o ha un numero di canali definito come $n = (o+1)^2$; quindi, nel primo ordine abbiamo 4 canali, nel secondo ordine abbiamo 9 canali, nel terzo 16 e così via.

La codifica *ambisonics* di ordine 3 è realizzata anch'essa in *gen~*

```

sig = in1; // input signal
azim = in2;
elev = in3;

ipi_div180 = pi / 180; // pi greek / 180
azimuth = azim * ipi_div180; // degrees to radians conversion
elevation = elev * ipi_div180;
  
```

```

afX = cos(azimuth)*cos(elevation);
afY = sin(azimuth)*cos(elevation);
afZ = sin(elevation);

afZ2 = afZ * afZ;
out1 = 0.2820947917738781 * sig;

afT1 = 0.48860251190292 * sig; // precalculate for optimization

out2 = afT1 * afY;
out3 = 0.4886025119029199 * sig * afZ;
out4 = afT1 * afX;

afC1 = afX*afX - afY*afY;
afS1 = afX*afY + afY*afX;

afT2 = 1.092548430592079 * sig * afZ;
afT3 = 0.5462742152960395 * sig;

out5 = afT3 * afS1;
out6 = afT2 * afY;
out7 = (0.9461746957575601 * sig * afZ2) - (0.31539156525252 * sig);
out8 = afT2 * afX;
out9 = afT3 * afC1;

afC2 = afX * afC1 - afY*afS1;
afS2 = afX * afS1 + afY*afC1;

afT4 = (2.285228997322329 * sig * afZ2) - (0.4570457994644658 * sig);
afT5 = 1.445305721320277 * sig * afZ;
afT6 = 0.5900435899266435 * sig;

out10 = afT6 * afS2;
out11 = afT5 * afS1;
out12 = afT4 * afY;
out13 = afZ*(1.865881662950577 * sig * afZ2 - 1.119528997770346 * sig);
out14 = afT4 * afX;
out15 = afT5 * afC1;
out16 = afT6 * afC2;

//SN3D normalization
out2 = out2*0.57735026918962584;
out3 = out3*0.57735026918962584;
out4 = out4*0.577350269189625844;
out5 = out5*0.44721359549995793;
out6 = out6*0.44721359549995793;
out7 = out7*0.44721359549995793;
out8 = out8*0.44721359549995793;
out9 = out9*0.44721359549995793;
out10 = out10 * 0.37796447300922720;
out11 = out11 * 0.37796447300922720;
out12 = out12 * 0.37796447300922720;

```

```

out13 = out13 * 0.37796447300922720;
out14 = out14 * 0.37796447300922720;
out15 = out15 * 0.37796447300922720;
out16 = out16 * 0.37796447300922720;

```

Il processo di *encoding* viene applicato su di una sorgente stereofonica, quindi ci sono due *encoder* uguali che lavorano con una differenza di *azimuth* di 60° , replicando il fronte stereofonico all'interno dello spazio *ambisonics*. La stereofonia è realizzata all'interno della granulazione, quindi ogni grano avrà un suo movimento all'interno di questi 60° , e viene controllata tramite un semplice algoritmo di *random walk*.

```

Buffer position ("position");
trig = in1;

if (trig == 1) {
    rand_value = noise();

    if (rand_value <= 0.) {
        x = peek(position, 0, 0);
        x -= 1;
        poke(position, x, 0, 0);
    }
    else {
        x = peek(position, 0, 0);
        x += 1;
        poke(position, x, 0, 0);
    }
}
pan = peek(position, 0, 0) / 100.:
pan = fold (pan, 0, 1);
out1 = pan;

```

Semplicemente, l'algoritmo parte da 0; ad ogni *trigger* esterno viene estratto un valore casuale e, se questo valore è minore di 0 il valore della posizione diminuisce di $1/100$, se è maggiore di 0 il valore aumenta di $1/100$. Il valore subisce poi un *folding*, un processo di limitazione simile al *wrapping*, con la differenza il valore che eccede i limiti non verrà specchiato sotto il limite opposto, ma sotto quello superato. Ad esempio, in un range tra 0 ed 1, il valore 1.1 diventerà 0.9; verrà calcolato quindi $1 - (1.1 - 1)$, con un valore di -0.2, il risultato sarà 0.2 calcolando $1 - (1 - 0.2)$. In

questo modo i valori sotto lo 0 torneranno non ripartiranno da 1 , quindi da destra nel caso del *panning*, ma da sinistra, per cui da 0 .

Il processo di controllo delle variabili *azimuth* ed *elevazione* della codifica effettuato dall'algoritmo derivante dal *Multibrot* è stato anch'esso realizzato in *gen~*, implementando una serie di operazioni in grado di calcolare $z_i^n + c$ indipendentemente dal valore di n .

```

Buffer infomandel("infoMandelIFS");
Buffer mandelbuf("mandelbrotcalc");

random_shift = in2;
i = 0;
trig = in1;

if (trig == 1) {
    init_extraction = peek(infomandel, 0, 0);
    // extraction initial values
    if (init_extraction == 0) {
        a_init = noise();
        b_init = noise();
        exp_init = int ( scale ( noise(), -1., 1., -100, 100 ) );
        if (b_init == 0) {
            i_b += 0.1; // avoid case a = 0 and b = 0 ---> not defined
        }
        poke(mandelbuf, a_init, 0, 0);
        poke(mandelbuf, b_init, 0, 1);
        poke(infomandel, exp_init, 3, 0);
        for (i=0; i < 20; i+=1) {
            a_sign = noise(); //change sign a possibility
            b_sign = noise(); // change sign b possibility

            if (a_sign >= 0){
                a_sign = 1; // no change sign
            }
            else {
                a_sign = -1; // change sign
            }
            if (b_sign >= 0){
                b_sign = 1; // no change sign
            }
            else {
                b_sign = -1; // change sign
            }

            a = peek(mandelbuf, 0, 0) + random_shift*noise();
            b = peek(mandelbuf, 0, 1) + random_shift*noise();
            a *= a_sign;
            b *= b_sign;
            arg = 0;
        }
    }
}

```

```

mod = sqrt ( a*a + b*b );

if( a == 0 && b > 0) {
    arg = pi/2;
    } //first case
else if(a == 0 && b == 0) {
    arg = 0; //just protection
    } // 2 case
else if(a == 0 && b < 0) {
    arg = pi * 3/2;
    } // 3 case
else if(a > 0 && b < 0) {
    arg = atan ( b/a) + twopi;
    } // 4 case
else if(a_init < 0) {
    arg = atan ( b/a) + pi;
    } // 5 case

mod = pow(mod, exp_init);
aa = mod * cos(exp_init * arg);
bb = mod * sin(exp_init * arg);
a = aa + a;
b = bb + b;

norm = ( (a+b) * (a+b) ) / 2.;

if(norm > 16) {
    poke(infomandel, i+1, 4, 0);
    break;
}
poke (mandelbuf, a, i+1, 0);
poke (mandelbuf, b, i+1, 1);
} //end for
} //end initialize set of values
// read values
n_values = peek(infomandel, 4, 0);
count = peek(infomandel, 1, 0);
current_a = peek(mandelbuf, count, 0);
current_b = peek(mandelbuf, count, 1);
poke(infomandel, current_a, 2, 0);
poke(infomandel, current_b, 3, 0);

count+=1;
if(count > n_values) {
    count = 0;
    poke(infomandel, 0, 0, 0);
}
poke(infomandel, count, 1, 0);
//poke(infomandel, 0, 0, 0);
}

out1 = peek(infomandel, 2, 0);
out2 = peek(infomandel, 3, 0);
out3 = peek(infomandel, 1, 0);

```

Infine, il flusso multicanale passa per un riverbero realizzato appositamente per il formato *ambisonics*. Il riverbero in questione è di tipologia *Feedback Delay Network* (abbreviato *FDN*), costituito quindi da un modulo di *early reflections* e da una più complessa rete di ritardi in cui il suono si propaga. Questo riverbero è anch'esso realizzato in *gen~*, sfruttandone stavolta la struttura ad oggetti così da avere un riscontro grafico della diramazione di questa *FDN*.

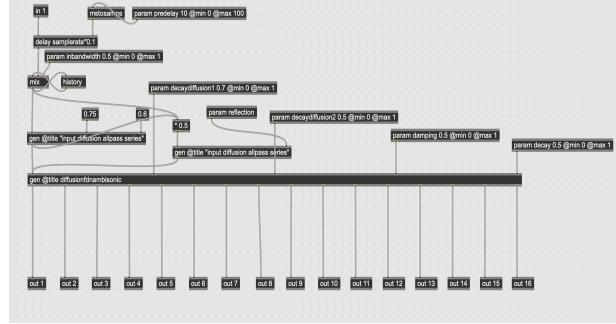


Fig. 4.8 Riverbero ambisonics

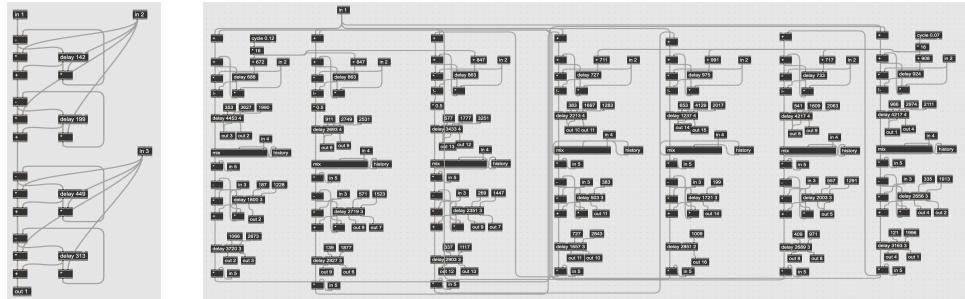


Fig. 4.9 Early reflections e Feedback Delay Network

CAP. 5 A Fractal Piece

5.1 Struttura del brano

Il brano realizzato s'intitola *A Fractal Piece*; si tratta di una composizione con una struttura ben definita, ma che lascia un certo grado di casualità all'interno di ogni cellula sonora che lo compone. Il brano è stato interamente generato tramite il *Fractal Autogenerative Network* ed è a tutti gli effetti una composizione aleatoria/algoritmica, ispirata alla geometria frattale.

La struttura si divide in 3 sezioni, due transizioni e una chiusura , la cui morfologia deriva da un'interpretazione dell'insieme di Mandelbrot.

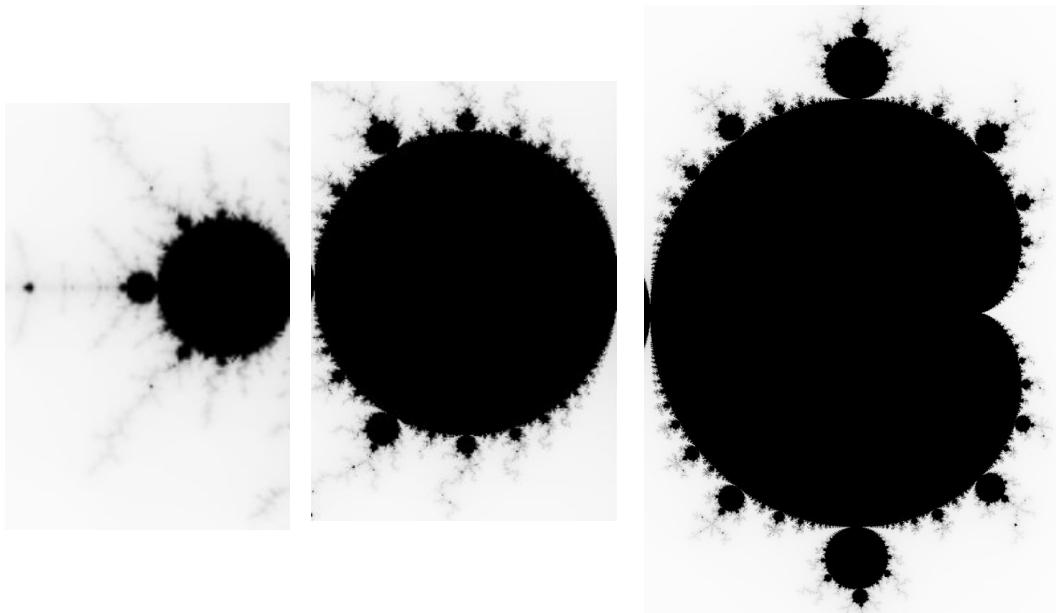


Fig. 5.1 Prima parte con transizione, seconda parte con transizione, terza parte con chiusura

Ognuna di queste parti si dirama in modi diversi; la prima parte va dal minuto 0.00 al minuto 2.10 ed è basata su dei suoni statici, con un'evoluzione non marcata. A questi si aggiungono sporadicamente altri suoni più articolati, cercando di imitare in qualche modo l'immagine. Il primo “cerchio” del frattale di Mandelbrot rispecchia il momento di transizione tra il primo movimento ed il secondo, rappresentato

attraverso una densità di eventi più ampia, una maggiore presenza nello spettro, in cui ogni suono presenta una complessa articolazione.

La seconda parte va dal minuto 2.15 al minuto 5.20 e vede predominare dei materiali apparsi solo marginalmente nella prima, suoni molto brevi che racchiudono la complessità delle immagini frattali al loro interno; questi, tramite la loro sovrapposizione crescente, conducono il brano verso eventi sonori ad ampia banda e molto intensi, per poi diradarsi nuovamente e ripetere questo ciclo. Il momento di transizione viene interpretato tramite una serie di suoni in glissando accompagnati da un tipico flusso granulare che rimanda al materiale sonoro della seconda parte, ma stavolta appena percepibile in lontananza.

La terza parte è invece quella con la maggiore densità; per tutta la sua durata, presenta una sovrapposizione di granulazioni di glissandi crescenti, i quali vanno via via aumentando la frequenza massima, per poi scendere, compiendo un *crescendo decrescendo* anche nella struttura che li racchiude; a posarsi su questa struttura sono dei nuovi materiali, costituiti da delle melodie generate dalla struttura generativa che si incrociano, assieme a dei brevi flussi granulari. La chiusura si sviluppa attorno alla coda derivante dalla terza parte e su dei glissandi articolati che si susseguono lentamente, in un lento discendendo delle dinamiche.

5.2 Prima sezione – Thin Line

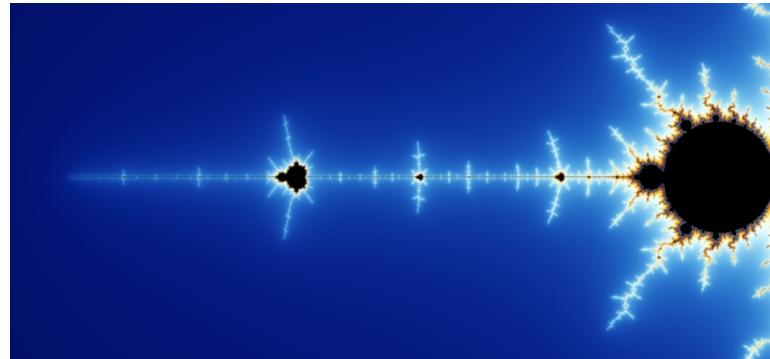


Fig. 5.2 Prima parte del frattale di Mandelbrot

Questa sezione si basa sulla porzione d'immagine del frattale di Mandelbrot, formata da una linea continua con delle irregolarità, ovvero altri insiemi di Mandelbrot; i suoni su cui si basa la maggior parte della sezione sono piuttosto leggeri e statici; questi non presentano una complessa evoluzione nell'intenzione di richiamare l'idea di una linea dritta; le immagini frattali su cui si basano i suoni portanti di questa prima parte sono l'insieme di Cantor ed una particolare variazione dell'insieme di Julia, in un range di frequenze limitato.

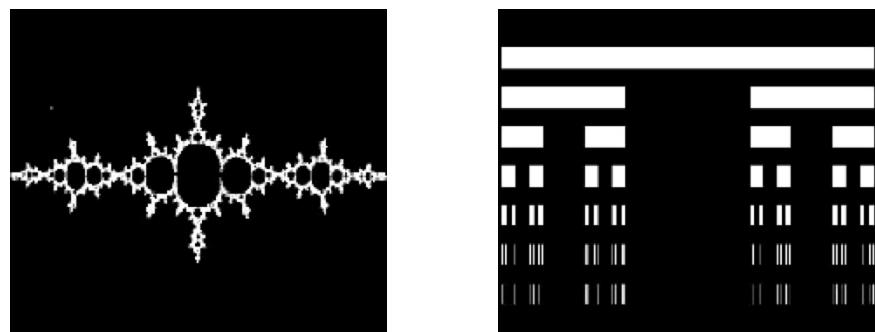


Fig. 5.3 L'insieme di Julia utilizzato e l'insieme di Cantor

Le “irregolarità” presenti sulla linea vengono realizzate tramite dei suoni molto brevi o delle granulazioni piuttosto dense della durata di qualche secondo, il cui timbro deriva da alcune variazioni del frattale di Mandelbrot.

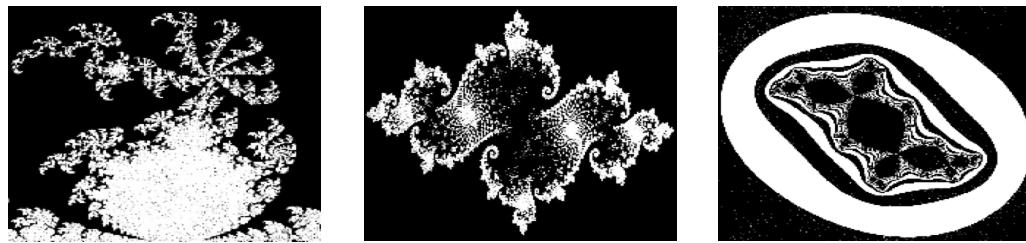


Fig. 5.4 Variazioni del frattale di Mandelbrot e dell’insieme di Julia utilizzati per creare le irregolarità

La prima sezione si divide in tre parti, separate da due brevi momenti di connessione sonorizzati tramite dei suoni *verticali* a più larga banda che, come per la macrostruttura, fanno da ponte fra le differenti sottosezioni.

La prima divisione, della durata di 45 secondi, presenta una serie di piccoli eventi in medio-alta frequenza, basati sull’insieme di Cantor, ed è la base per tutta questa sottosezione; insieme ad essi, dei suoni in bassa frequenza, generati dall’insieme di Mandelbrot, che appaiono in modo nascosto e meno frequentemente. Ai secondi 12.5 e 30 appaiono le “irregolarità”, rappresentate con delle brevi evoluzioni, la prima a banda ridotta, dato che l’irregolarità è piuttosto sottile, la seconda a più lunga e più ampia. A connettere la prima divisione con la seconda troviamo un breve flusso sonoro che si dirama dal basso verso l’alto, esprimendo una certa verticalità.

La seconda sottosezione dura invece soltanto 30 secondi; l’idea di base è simile alla prima divisione, ma con alcune differenze timbriche sostanziali. Il suono portante è basato su di una variazione dell’insieme di Julia, in un range più orientato verso le medie frequenze; le irregolarità sono riprodotte seguendo lo stesso principio della prima divisione, stavolta a 0.52 e ad 1.05. Troviamo però una novità sonora, dei *microsounds* con un’articolazione tipica della sintesi granulare, il cui timbro deriva da un’ulteriore variazione dell’insieme di Julia. Il secondo “ponte” segue lo stesso

principio del primo, ma, sempre seguendo l'immagine, risulta più morbido e meno esteso.

La terza sottosezione ripropone l'insieme di Cantor nel timbro, stavolta diviso in due serie di eventi ben distinti, una a stampo *percussivo* e l'altra con inviluppo breve e crescente. Anche qui compaiono le irregolarità al minuto *1.30*, *1.48* e al minuto *2.05*, ma con un'evoluzione marcata e una maggiore presenza rispetto alle precedenti, sfociando in un *climax* verso la fine della sezione. La durata della terza divisione è di 55 secondi.

La parte di connessione fra la sezione *1* e la sezione *2* è costituita da una sovrapposizione di flussi granulari molto densi e con diverse caratteristiche, da un flusso quasi *liquido* a dei suoni ruvidi, disposti lungo buona parte dello spettro. Corrisponde al primo cerchio della figura di Mandelbrot; essendo una prima espansione dell'insieme, viene rappresentata con un suono più ampio.

5.3 Seconda sezione – Circle

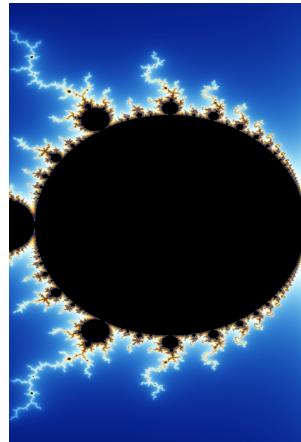


Fig. 5.5 Seconda parte del frattale di Mandelbrot

La seconda sezione presenta un cambio di sonorità. Dai suoni diradati e lisci della prima parte, si passa ad una maggiore presenza di suoni brevi che si accavallano, i *microsounds* sono centrali in questa parte. La struttura è basata su uno schema ciclico che si ripresenta: partendo con un flusso isolato, ad esso si accavallano man mano altri flussi simili, espandendone la dimensione spettrale e portando all'ingresso di un nuovo evento molto complesso, deciso ed espanso; segue infine un rapido diradamento degli eventi, tornando ad una situazione simile al punto di partenza.

Il primo ciclo ha una durata di 2 minuti circa e risulta molto asciutto rispetto a tutta la prima parte; i suoni sono molto ben distinguibili e asciutti, contribuendo sia alla sensazione di “novità sonora” sia al contrasto con l’evento di apertura spettrale che conclude il ciclo, il quale possiede una densa riverberazione atta a dare tridimensionalità e grandezza al suono. Il secondo ciclo dura invece 1 minuto; la sovrapposizione non è molto pronunciata, lasciando quindi spazio per una riverberazione di parte dei flussi *percussivi* che anticipano il *climax* che conclude la sezione. Quest’ultimo si compone di un bordone derivante da una granulazione in bassa frequenza che fa da piedistallo ad una serie di glissandi in medio alta frequenza che anticipano una delle caratteristiche fondamentali della terza sezione e della chiusura.

La connessione fra la seconda e la terza sezione inizia dalla coda della seconda; dopo una breve pausa, alcuni suoni molto fievoli appaiono in rapidi glissandi, mentre in lontananza ritroviamo i piccoli suoni *percussivi*, dipingendo una dimensione spaziale nuova e offrendo allo stesso tempo spazio per ciò che arriverà nella terza sezione.

Le immagini utilizzate in questa sezione sono numerose, a causa delle varie sovrapposizioni che ne seguono; essendo stati poi sviluppati in granulazioni molto dense o in suoni molto brevi, l'evoluzione timbrica che può avere l'immagine non viene percepita; ciò che rimane è la complessità e la struttura omotetica dei frattali, che, in ogni caso, produce un risultato diverso rispetto all'utilizzo di un oscillatore o di un campione audio qualsiasi. Ecco alcune delle immagini utilizzate in questa sezione

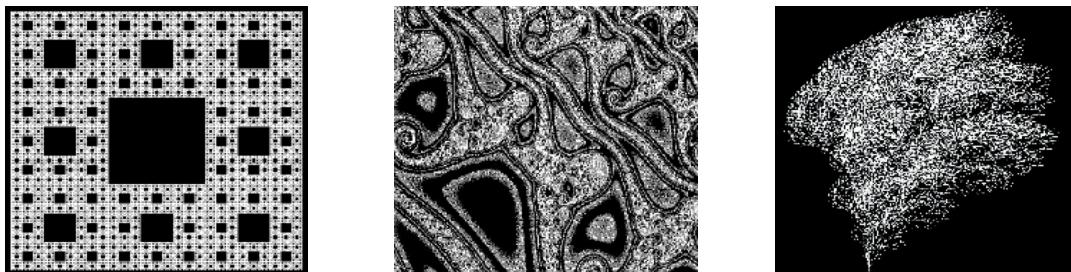


Fig. 5.6 Spugna di Menger, uno zoom del frattale di Mandelbrot e un albero risultato da un L-System

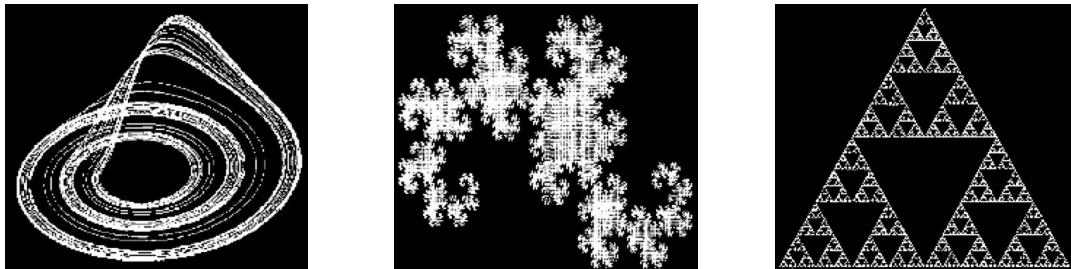


Fig. 5.7 Attrattore di Rossler, curva del drago di Koch e triangolo di Sierpinski



Fig. 5.8 Cantor target ed Isola quadratica di Koch

5.4 Terza sezione – the Big Area

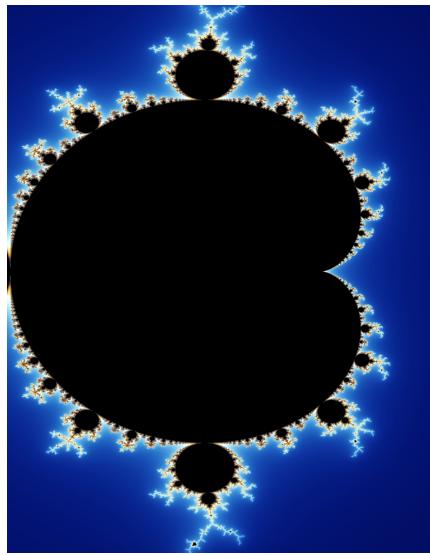


Fig. 5.9 Terza parte del frattale di Mandelbrot

Questa sezione va dai 6.00 ai 7.50 ed ha una struttura unica, senza alcuna divisione, ed è anche quella che lascia più spazio alla casualità e libertà nella sua realizzazione. Il materiale sonoro centrale è fondato su di una grande sovrapposizione di glissandi derivanti dalle immagini di alcuni frattali in *ascendendo* e *descendendo*, con numerose sovrapposizioni a diversi intervalli spettrali che vanno man mano creando un'espansione verticale.

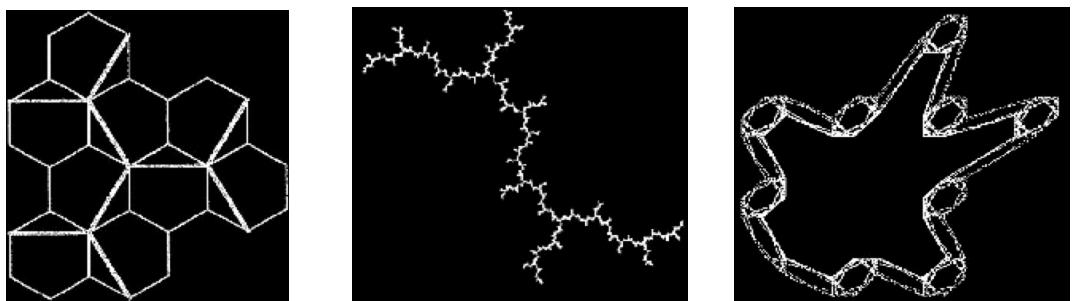


Fig. 5.10 Altre immagini frattali utilizzate per la realizzazione dei glissandi

Tale struttura viene riproposta anche ad un livello più elevato, poiché ognuno di questi *overlap* vede aumentare la sua frequenza centrale; man mano, i suoni

presentano un intervallo spettrale sempre più spostato verso le alte frequenze, per poi, a metà sezione, ritornare man mano al punto di partenza.

A tutto ciò si sommano altri materiali del tutto nuovi; viene usato il sistema generativo frattale per generare delle sequenze molto frenetiche di suoni brevi e di cui è riconoscibile un'intonazione, i cosiddetti *pitched sounds*. Questi vengono utilizzati come sfondo sonoro alla complessa architettura generata dai glissandi.

Questa sezione tenta di ricreare la grande area del frattale di Mandelbrot a livello sonoro, tramite una densità di eventi molto elevata, suoni in continua evoluzione, sovrapposizioni ed una spiccata eterogeneità dei materiali. La conclusione, da 7.50 a 8.56 porta avanti un diradamento continuo degli eventi già iniziato nella terza sezione, in un discendendo generale di frequenza e dinamica e densità, protraendo il flusso di *pitched sound* ed i glissandi, stavolta rilassati e ben separati.

In tutto lo svolgimento di *A Fractal Piece* ritroviamo parte degli spunti esposti nel capitolo 3. La caratteristica omotetica viene riproposta ad ogni livello strutturale, a partire dai singoli momenti sonori generati dal *Fractal Autogenerative Network* e che quindi vedono l'utilizzo dei frattali sotto ogni aspetto timbrico, fino alla struttura generale, divisa in parti che ripresentano una certa struttura anche nelle loro articolazioni interne.

CONCLUSIONI

Il percorso svolto in questo elaborato è frutto di numerose influenze, di cui lo studio svolto nei vari ambiti durante questo Triennio è sicuramente la più importante. Questo lavoro non è soltanto un punto di arrivo, ma anche una sintesi di tutto ciò che ho realizzato ed appreso in questo percorso di studi. Gli aspetti che ne fanno parte non riguardano soltanto la natura tecnica o matematica nell'applicare la geometria frattale ad un contesto musicale, ma anche i riferimenti storici e gli esempi che i grandi compositori hanno lasciato, l'idea artistica dietro alla scelta di un tipo di composizione che abbraccia l'aleatorietà; tutto ciò è stato parte integrante di questo lavoro e della mia crescita. La conoscenza teorica e pratica degli aspetti tecnici risulta un mezzo fondamentale per approcciarsi ad un progetto simile, di cui lo scopo non è però meramente tecnico, quanto piuttosto quello di comunicare la fascinazione che io personalmente ho subito nei confronti della geometria frattale nel momento in cui ho iniziato a comprenderne la natura e le possibilità, di mostrarne una mia interpretazione personale, tramite il *Fractal Autogenerative Network* ed il brano *A Fractal Piece* che ne consegue, concentrandovi tutto ciò che mi è rimasto del percorso di studi intrapreso.

La geometria frattale, dunque, presenta delle possibilità molto ampie anche in ambito artistico; la difficoltà principale sta nel riadattamento, come descritto nei vari capitoli, nel portare la funzionalità che il calcolo frattale ha in un'applicazione grafica e geometrica in un campo completamente diverso, musicale in questo caso. Essa trova la sua implicazione naturale all'interno del concetto di musica aleatoria e tutto ciò che ne deriva, visti i legami già descritti con il caos e con la natura e, a quanto pare, anche con la musica stessa; gli esempi posti riguardanti alcuni grandi compositori del passato, le quali composizioni hanno a che fare coi frattali o con la distribuzione *browniana*, sono testimoni di quanto possa offrire l'utilizzo di un approccio frattale nella composizione musicale. Inoltre, la complessità raggiungibile tramite una strutturazione omotetica è enorme, a fronte di uno sforzo nella sua creazione relativamente ridotto; l'organizzazione secondo questo principio costruisce

una struttura ben delineata semplicemente definendo alcune caratteristiche basilari, poiché queste verranno riproposte nei sottolivelli definiti. Il brano *A Fractal Piece* presenta un'organizzazione di questo tipo formata da 3 livelli: macro-struttura formata sulle sezioni, divisione di ogni sezione, eventi sonori singoli; se, però, si estende la stratificazione tramite questo stratagemma, si è capaci di stabilire un'architettura decisamente ricca in maniera semplice, dentro la quale è possibile muoversi in modo libero e coerente, indipendentemente dagli strumenti utilizzati, che sia un organico strumentale o dei mezzi elettronici. L'impiego di algoritmi frattali aleatori è incentrato, in questo caso, sul controllo di una struttura di stampo granulare e mono-timbrica, la realizzazione di tutto il *Fractal Autogenerative Network* è focalizzata sull'incorporare la geometria frattale sotto ogni aspetto possibile; l'utilizzo di questo approccio all'interno di una rete multi-timbrica, con più voci e diverse tipologie di suono, ognuna governata in modo diverso in base alla propria funzione e che si interfaccia con le altre secondo determinate regole è in grado di creare organismi musicali dalle mille sfaccettature, con risultati del tutto naturali, in continua evoluzione, proprio secondo l'idea originale di musica generativa di Brian Eno.

Infine, una menzione particolare va al suo impiego all'interno della spazializzazione. Le tecnologie *ambisonics* offerte dal progetto SPACE⁷¹ all'interno del Conservatorio Rossini sono state importantissime per la mia formazione e nella realizzazione di questo progetto. Proprio dall'ibridazione *calcolo frattale – controllo ambisonics* nasce questo lavoro; nel corso dell'ultimo anno, sono stati realizzati, assieme al maestro relatore Pasquale Mainolfi, dei primi modelli di controllo per il *panning ambisonico* tramite il frattale di Mandelbrot. Come descritto nel terzo capitolo, il controllo dello spazio con il sistema *ambisonics* è almeno bidimensionale ed i movimenti che una sorgente assume riescono a connettere suono ed immagine visiva; lo spazio è un parametro tridimensionale e l'udito tanto quanto la vista è in grado di darci l'informazione relativa alla posizione di un oggetto, se questo emana

71 E. GIORDANI, D. MONACCHI, <<http://www.rossinispace.org>>, consultato il 15 febbraio 2020.

suono e, per tali ragioni, risulta è uno degli aspetti del suono che più si prestano ad essere manovrati tramite i calcoli frattali.

BIBLIOGRAFIA

- G.W. VON LEIBNIZ, *La Monadologia*, Milano: SE, 2018
- P. FATOU, *Teoria dell'iterazione*, 1917
- B. MANDELBROT, *The Fractal Geometry of Nature*, New York: Times Books, 1982.
- I. XENAKIS, *Formalized Music. Though and Mathematic in Composition*. Parigi, Pendragon Press (1963).
- T. BOLOGNESI, *Composizione assistita dal calcolatore*. Pisa, CNUCE (1981)
- J. A. MAURER, *A Brief History of Algorithmic Composition*. Stanford, CCRMA (1999).
- S. REICH, *Musica come processo graduale* (1968)
- R. L. DEVANEY, *A First Course In Chaotic Dynamical Systems: Theory And Experiment (Studies in Nonlinearity)*. Westview Press (1992)
- M.J. GREENBERG, *Euclidean and Non-Euclidean Geometries: Development and History*. W.H. Freeman, (2008)
- B. MANDELBROT, *How Long Is the Coast of Britain? Statistical Self-Similarity and Fractal Dimension* (1967)
- M. BULMER, *Music from Fractal Noise*, Melbourne, University of Queensland (2000)
- B. MANDELBROT, *Multifractals and 1/f noise*, New Haven, Springer-Verlag (1999)

- *KENNETH FALCONER, Fractal Geometry. Mathematical Fundations and Applications*, Chichester, John Wiley & Sons LTD, (2003).
- *TOM LINDSTROM, Brownian Motion on Nested Fractals*, Providence, American Mathematical Society (1992)
- *HELGE VON KOCH, Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire* (1904)
- *GIUSEPPE PEANO, Sur une curve qui remplit toute une aire plane*, Mathematische Annalen, vol. 36, pp. 157-160, Torino (1890)
- *GEORG CANTOR, Sulla potenza degli insiemi perfetti di punti*, Acta Mathematica Vol. 2 (1884)
- *BENOIT MANDELBROT, Fractals and chaos: the Mandelbrot set and beyond*, Springer Verlag, New York (2004)
- *H. J. BROTHERS, Structural scaling in Bach's Cello Suite No. 3*, Fractals Vol. 15, World Scientific Publishing Company, Madison (2007)
- *MICHAEL FRAME, NATHAN COHEN, Benoit Mandelbrot: a life in many dimensions* vol. 1, Singapore, World Scientific Publishing Co. (2015)
- *M. F. BARNSLEY, S. DEMKO, Iterated function systems and the global construction of fractals*, London, Royal Society, 1985
- *P. PRUSINKIEWICZ, J. HANAN, Lindenmayer Systems, Fractals and Plants*, Regina, Springer-Verlag, 1992
- *HEINZ-OTTO PEITGEN, DIETMAR SAUPE, The Science of Fractal Images*, New York, Springer-Verlag, 1988

- JULIEN CLINTON SPROTT, *Strange attractors: creating patterns in chaos*, Madison, M & T Books, 1993
- JUN NI, *Principles of Physics: from Quantum Field Theory to Classical Mechanics*
- RANZ ZOTTER, MATTHIAS FRANK, *Ambisonics: a practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement and Virtual Reality*, Graz, Springer Nature, 2019
- V.J. MANZO, *Max/MSP/Jitter for music: a practical guide to developing interactive music system for education and more*, Oxford, Oxford University Press, 2016
- ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

SITOGRAFIA

- *Funghi e Zen: casualità e complessità nella musica di John Cage,*
<https://www.lavoroculturale.org/musica-di-john-cage/>
- *Introduzione ai processi stocastici*, Università di Pisa,
http://users.dma.unipi.it/~flandoli/dispense_Istituzioni.pdf
- B. ENO, P. CHILVERS, <http://www.generativemusic.com>
- ANDRE OFFRINGA, *Diffusion Limited Aggregation*,
<https://www.astro.rug.nl/~offringa/Diffusion%20Limited%20Aggregation.pdf>
- ANGELO FARINA, *Ambisonics Pages*,
<http://pcfarina.eng.unipr.it/Ambisonics.htm>
- E. GIORDANI, D. MONACCHI, <http://www.rossinispaces.org>