GSC4: A CSOUND PROGRAM FOR GRANULAR SYNTHESIS

by Eugenio Giordani

1. Introduction

The notion of a Granular Synthesis Csound patch was initially stimulated by the work of Barry Truax. During the Italian seminar, Musica/Complessità, held in the summer 1988, he presented material concerning his research and musical compositions based on real-time granular synthesis. One of the most didactically inexpensive yet powerful synthesis software packages available at the time was Csound. So, I decided to challenge the Music-N dualism of orchestra and score that is the typical feature of this type of programming language, try to write a simple program for this complex sound synthesis.

Since Granular Synthesis creates many acoustic events (grains) per unit of time, it is not a practical to generate each grain using one note score statements. My goal was to realize an automated process for grain generation processed at the micro-level while preserving control over the synthesis process and overall parameters, as well.

The recent releases of Csound, now include the granulation process (see the opcodes *grain* and *granule*) but at the time this patch was written, no granulation unit was available in the Csound opcodes. The goal of this lecture is to serve as a practical exercise for implementation of a complete sound synthesis algorithm in this language - from the conception of an idea to a working program.

2. General structure of the algorithm and synthesis parameters description

Using GSC4 is possible to generate four independent stereo streams of sonic grains. The number four derives from the demand of minimum vertical grain density and real-time capabilities of personal computers.

According to this scheme, the *orchestra* set up includes four grain generation instruments plus one instrument for the control and one instrument for sound mix and scale:

instr 1, 2, 3, 4 : grain generators instruments
instr 11 : control instrument
instr 21 : sound mix, out and scale instrument

In order to generate a complex sound event, we need to switch on six instruments at the same start time (p2) with the same duration (p3).

The majority of the parameters (up to p13) are utilized by the control instrument, whereas the others instruments each contains only four parameters (p1 to p4).

Control of the granulation process is determined by control functions that describe the evolution in time of the synthesis parameters. Those parameters refer to instrument 11 and are:

1)	center grains duration in ms	p4
2)	random grains duration in ms	p5
3)	center inter-grain delay in ms	p6
4)	random inter-grain delay in ms	p7
5)	grain envelope ramp scale factor in non-dimensional units	p8
6)	center waveform-file frequency in Hz	p9
7)	random waveform-file frequency in Hz	p10
8)	center waveform-file phase or file pointer (normalized)	p11
9)	random waveform-file phase or file pointer (normalized)	p12
10)	overall amplitude (normalized)	p13

Each of these parameters states the function number assigned for the relative parameter. So, during the instrument activation, ten functions (created by some GENi method) must exist inside the score. Referring to the score file included in the Appendix, we can summarize the meaning of those synthesis parameters:

f11 :
the average (center) grains duration is defined by a linear function (GEN 7) with initial value of 10 ms that after 256/512 of p3 (the total event duration) reaches the value of 20 ms, keeps costant for 128/512 and moves to the final value of 16 ms after 128/512 of p3.

f12:
the peak random duration value of the grains is defined by a linear function (GEN 7) with initial value of 4 ms that after 256/512 of p3 reduces itself to 1 ms and after 256/512 goes to the final value of 0 ms (no random deviation).

f13:
the inter-grain delay is defined by a linear function (GEN 7) with initial value of 10 ms that after 256/512 of p3 raises to 20 ms and after 256/512 reaches the final value of 5 ms.

f14:
the peak random inter-grain delay value of the grains is defined by a linear function (GEN 7) with initial value of 0 ms (no random deviation) that after 128/512 of p3 stays constant to 0 ms; after 256/512 raises to the value of 2 ms and after 128/512 reaches the final value of 0 ms (no random deviation).

f15:

grain envelope ramp scale factor is defined by a linear function (GEN 7) with initial value of 2 that after 256/512 of p3 rises to 4 and after 256/512 reaches the final value of 2. Practically, the grain envelope shape change gradually from a triangle at the beginning towards a trapezium and back to triangle. When the envelope shape looks like a triangle, the duration of both the attack and decay ramp is equal to half duration of the whole envelope (no sustain). When the envelope shape looks like a trapezium, the duration of both the attack and decay ramp is equal to one quarter of the whole envelope, so the sustain duration is equal to two quarter of the whole envelope.

f16:

the frequency of the granulated waveform is defined by a linear function (GEN 7) with initial value of 220 Hz that stays constant during the whole event. Ins the score there is a comment line referring to the values of control frequency (from 1.345 to 3.345 Hz) in the case of a granulated sampled waveform (sample.wav) of 32768 samples size instead of a single cycle wave. In this case, the initial value of 1.345 derives from the ratio 44100/32768 and represents the original pitch of the waveform.

f17:

the peak random frequency of the granulated waveform is defined by a linear function (GEN 7) with initial value of 0 hz (no random deviation) that after p3 reaches the final value of 110 hz (50% of frequency modulation).

f18:

the phase (or file pointer) of the granulated waveform is defined by linear function (GEN 7) with initial value of 0 that stays constant over the whole event.

f19:

the peak random phase (or file pointer) of the granulated waveform is defined by linear function (GEN 7) with initial value of 0 that stays constant over the whole event.

f20:

the overall amplitude waveform is defined by a linear function (GEN 7) with initial value of 0 that after 128/512 of p3 rises to 1, stays constant for 256/512 and reaches 0 after 128/512 of p3.

f1:

the granulated waveform is defined by a simple Fourier additive function (GEN 10).

It is important to notice that, except in functions f1 and f20, the parameter p4 is always negative because the function breakpoints must represent their values in an absolute scale. For the four generation instruments (i1,i2,i3,i4), is sufficient to specify (besides the 3 obligatory parameters p1,p2 and p3) the function number (f1 in this case) and for instrument 21 the output scale factor.

For the audio waveform is possible to specify a single cycle wave or a sampled sound using a GEN1 function. In the first case, the value of the parameter p4 of the instrument 21 must contain the maximum output value (in the range 0 , 32767) whereas in the second one, the value is defined in the range 0, 1. The reason of this fact is that the sampled audio signal is not post-normalized during the table reading (GEN -1).

As stated before, is important to point out that the audio signal may be both a single cicle wave or a sampled sound.

Although there is no functional differences in the two cases, it is better to pay attention in the frequency specification.

In the first case, the frequency value and the relative random deviations are simply the desired values.

In the second case, the nominal value of the *natural frequency* (Fn) of the oscillator (by that we mean the frequency value to reproduce the audio signal at the original pitch) is derived from the ratio of the sampling rate (*sr*) and the length (in samples) of the table that contains the sampled waveform (Fn = sr / table length).

For example, if the length of an audio waveform sampled at 44.1 kHz is 64k samples (about 1.486 seconds) , the natural frequency will be 44100 / 65536 = 0.672 Hz.

In general, since the effective duration of the audio signal hardly ever equivalent to a powers of two, we have to provide a table to store the values in excess of the waveform size minus the nearest power of two.

If the total duration of the audio signal sampled at 44.1 kHz is 1.2 seconds, to be effective, the table should be 44.1 x 1.2 = 52920 samples. This implies one should choose a table size of 64 k-samples. However, the natural frequency will still be 0.672 Hz because the Csound oscillator modules work with tables whose length is equivalent to a power of two. The only difference, in this case, is that the phase parameter ranges from 0 and 0.807. The value 0.807 is obtained from the ratio 52920/65536.

It is also important that the random frequency fluctuation must be congruent with the corresponding deterministic value.

For example, with a natural frequency value of 0.672 Hz and a fluctuation of 10 percent the corresponding fluctualtion is about 0.06 Hz.

The concept of *natural frequency* is very important in this context because we can granulate a single cycle of a waveform or a whole sample of sound. From the point of view of the oscillator there is no difference. We can generalize this with the equation:

Fn = I x SR/ L

where :

Fn = oscillator natural frequency
SR = sampling rate
I = increment (table reading step)
L = table length (samples)

With respect to the previous example, if we want to granulate a 1.2 sec stored sound sampled at 44.1 kHz and reproduce it at the original pitch, we have to specify something like

```
f16   0   512  -7   0.672   512   0.672
```

When the granulation is applied only to sampled sounds, it is a good idea to multiply the variable *ifreq* with the expression *sr/ftlen(ifun)*, where *ftlen(x)* is a function that returns the table length in samples. Hence, the score line controlling the frequency will be:

```
f16   0   512  -7   1   512   1
```

In this way, the sound's pitch is handled as a ratio with respect to the natural frequency, and we avoid computing the real values of the frequency itself. For example, to create a continuos pitch glide of the granulated sound, starting from its original frequency and moving up to the natural fifth above (interval ratio of 3:2=1.5), the following score line is required

```
f16   0   512  -7   1   512   1.5
```

The same control is possible for random frequency deviation.

If the audio function is a single cycle of a periodic wave, the phase parameter of the oscillator has an influence on the acoustic result. But if the audio function is a sampled sound, its role is crucial. In fact, in this case, the phase parameter becomes the *table pointer*, allowing us to granulate different sections of the sampled sound.

When you want to granulate the whole sample, from the beginning to the end of sound, we can use a linear function (GEN 7) that moves from 0 to 0.999:

```
f18   0   512  -7   0   512   0.999
```

Here, the sampled sound is granulated without time warp. By exchanging two values, it is possible to reverse the sound. Following this approach, we can achieve different and interesting acoustic results.

For example, in the next score line, during the first half of the event (p3/2), the granulated sound is reversed starting from its middle point (0.5) to the origin, and in the second half, the sampled sound is forward time compressed:

```
f18  0   512  -7   0.5  256  0   256  0.999
```

In any case, the total event duration (p3) may be kept the same as the duration of the original sampled sound or it maybe increased or reduced.

To increase the time transformation of the original sound, one can use a more complex control function (i.e. non-linear) and add an additional random control with the function 19 (see code listing).

3. Origins of the synthesis algorithm

The basic algorithm is based on the model proposed by Barry Truax and implemented on the DMX-1000 real time processor controlled by a host microcomputer. Fig. 1 shows the basic process of the granulation technique in this early implementation. Here, the synthesis generation probably used two programming sections: a background section for the envelope generator bank running on the processor and managed by an interrupt logic (i.e. using a 1 ms timer), and a control section running on the host computer.

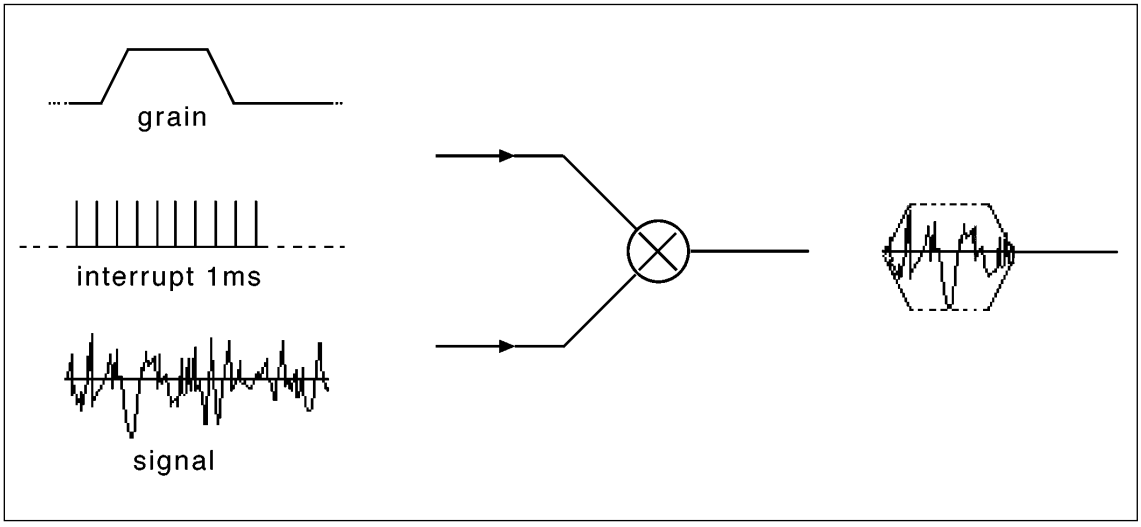

Fig. 1

Since this Csound instrument for the granulation process was implemented in 1989 (an era with few real-time applications for general purpose computers) the main goal of its development was to make a basic strategy for implementing this innovative synthesis technique in a non real-time environment.

Given the inherent necessity to generate a great amount of micro-events per units of time, it was not practical to pursue a note by note generation approach. An alternative solution would have been the utilization of a pre-processing front-end program for the generation of the innumerable lines of notes, each one corresponding to a single grain.

4. The Csound implementation of Granular Synthesis (GSC4)

GSC4 attempts to generate sequences of large number of grains (micro-events) in the high level macro-events that corresponds to single note statement in the Music-N languages. The major difficulty is how to integrate the low-level activity and the control process of the synthesis into a single level. In other words, we need an automated procedure to manage the two different levels. The most useful Csound opcode for this purpose is surely the *timout* statement. It was around this that the whole program algorithm was developed We remember that the syntax of timeout (included in the **program control statement** class) is:

timout istrt, idur, label

During P-time (performance time), a branch to the specified *label* will synthesize depending on the elapsed note time. The branch will be realized at time *istrt* and will remain so for just *idur* seconds. It is very important to note that *timout* can be re-initialized for multiple activations for a single note.

This is equivalent to a program time-counter that produces an interrupt at the end of the countdown. As a consequence, during the re-initialization process it is possible to update a predefined set of i-type variables. The core of the program is then based on this following control structure:

```
;- - - - - - - - - - Interrupt simulation section - - - - - - - - - -
;
;timeout work as an interrupt logic. It is loaded with the current grain duration and automatically decrement to zero

timout 0, igrain, cont          ;if the current value of the timer
                                ;is not zero,branch to the program section labelled with cont

reinit loop                     ;otherwise jump to re-init pass
```

The variable *igrain* contains, at each re-initialization, the current duration value of the next grain. Until this duration is not zero, the program continuously transfers the control to the *cont* label. When the timer reaches zero, the program flow is interrupted and forced by the statement *reinit* to enter a re-initialization section of the program. The label *loop* is a pointer to this section.

The basic structure of the *orchestra* consists of three fundamental blocks:

1) *Grains generator* : it corresponds to an individual instrument that contains, besides the timeout-reinit process, the re-initialization section, the updating of the synthesis parameters and the waveform reading.

2) *Grains control* : it corresponds to another instrument that provides to the control functions generation.

3) *Rescale, Mix&Out*: self determining instrument.

These three blocks are shown in figure 2. The user can easily duplicate those three fundamental blocks in order to expand the total number of voices.

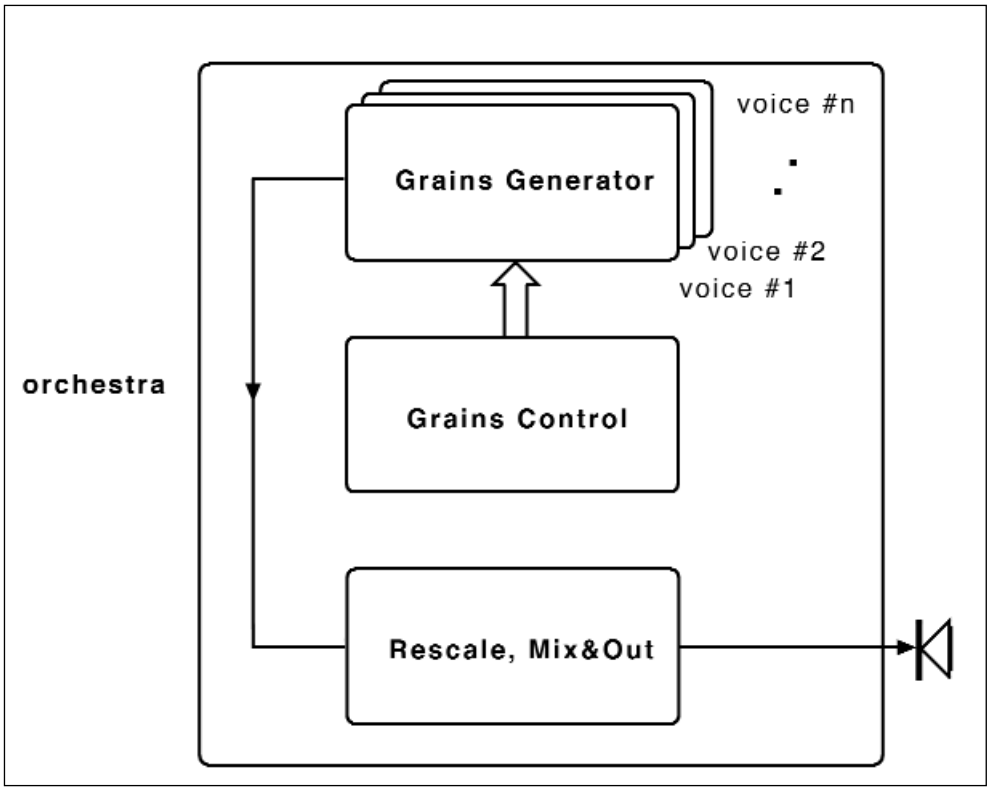

Fig. 2

As shown in figure 2, it is possible to notice a signal flow that connects the various blocks. This fact reveals the use of global variables that send and receive shared parameters.

For each grain, the synthesis parameters are updated on the timeout branch to the re-initialization section.

```
instr 1
;====================== GRAINS GENERATOR (VOICE #1)=========================

ifun     =  p4              ;audio function
;

;
;Grains parameters update (re-initialization)
;     ____________________________________________________________-
loop:
idu      = i(gkdur)        ;the current value of gkdur is sampled
                           ;from the corresponding generator in instr 11
                           ;and i-rated into a i-type variable

idurr    = i(gkrnd1)       ; the current value of gkrnd1 is ......

itrpz    = abs(0.001* (idur + idurr)) ;compute the trapezoid duration

iramp    = i(gkramp)+ 0.1            ; the current value of gkramp (plus a magic)is ..

idel     = i(gkdel)        ; the current value of gkdel is....
idelr    = i(gkrnd1y)      ; the current value of gkrnd1y is ....
idely    = abs(0.001 * (idel + idelr)) ;compute the total delay

ifreq    = i(gkfreq)       ; the current value of gkfreq is ....
ifreqr   = i(gkran)        ; the current value of gkran is ....
iphase   = i(gkphase)      ; the current value of gkphase is ....
iphaser = i(gkrnd1p)       ; the current value of gkrnd1p is ....
iamp     = i(gkamp)        ; the current value of gkamp is ....
```

Each grain consists of a trapezoid envelope plus a delay. The trapezoid in its turn includes an attack ramp (irise), a sustain (isus) and a decay ramp (isus) as showed in figure 3.

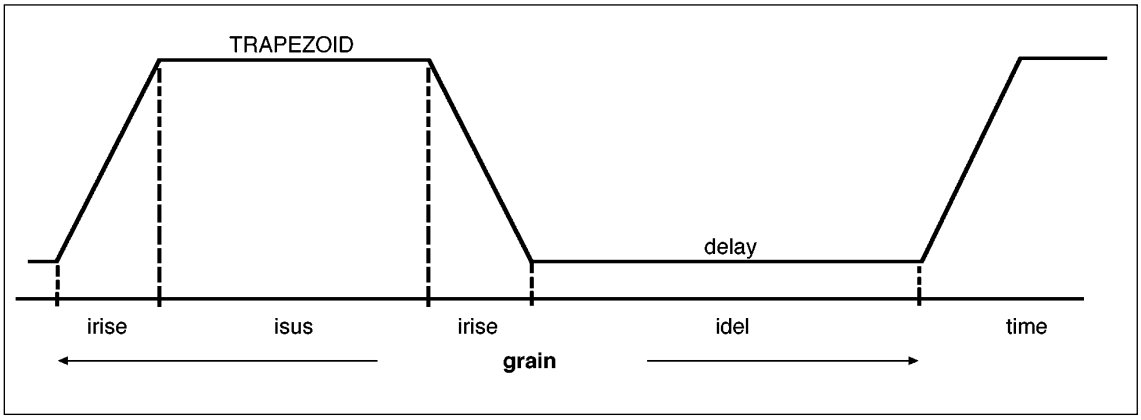

Fig. 3

The sum *irise+isus+irise+idel* represents the total grain duration (variable *igrain*). The attack and decay time of the two ramps, are computed as a fraction of the trapezoid duration (variable iramp).

For example, if *iramp* = 5, the attack (decay) time is equal to 1/5 of the trapezoid duration. The lower limit of this variable is 2. Infact, when *iramp* = 2, the trapezoid degenerates in the triangle.

This fact implies that the duration of stationary segment (*isus*) is equal to the difference between the current trapezoid duration minus the duration of the two ramps. The effective duration of each grain is calculated taking in account the random variations produced by a random generator within the Grain Control block (variable *idurr*).

```
irise   = itrpz/iramp          ;calculate the attack time of the trapezoid
isus    = itrpz - (2 * irise)      ;calculate the sustain time of the trapezoid
igrain  = itrpz + idely            ;calculate the trapezoid+delay duration
iph     = abs(iphase + iphaser)    ;calculate the phase (deterministic + randomic)
ifq     = ifreq + ifreqr           ;calculate the freq. (deterministic + randomic)
```

In substance, duration, delay, phase and frequency are always the sum result of a deterministic and aleatoric component.

The connection between the Grain Control Block and the Grain Generation Block is realized by the use of the *i(x)* function. This statement returns an Init-type equivalent of the argument, thus permitting a K-time value to be accessed (“frozen”) in at init-time o reinit-time, whenever valid.

The deterministic and aleatory components of the grain parameters, are continuously updated (K-time) within the Grain Control Block. These values are then sampled by the Grain Generation Block at the end of the *timeout* cycle. Both the generation and control

processes are performed in asynchronous way so that the user can specify the control variables of the synthesis at hi-level.

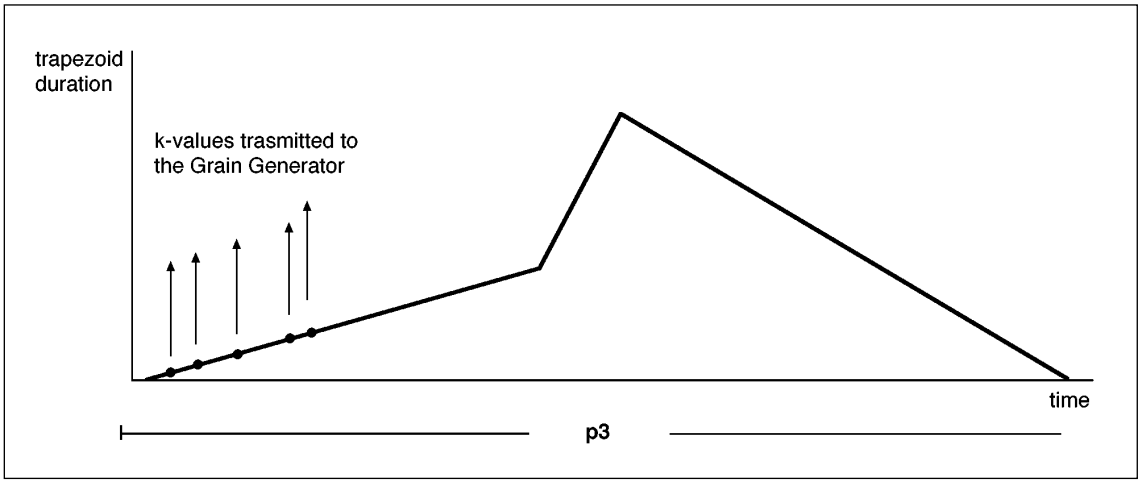

Fig. 4

In fig. 4 is shown an example of the diagram of a control function for the trapezoid duration over the note duration (p3). Inside the instrument 11 (Grain Control) are concentrated the major parts of modules for the deterministic and aleatory control parameters (see next code listing).

```
;============================== GRAIN CONTROL BLOCK ========================
instr 11
;NOTE: all the global variables are transmitted to the  instr 1, 2, 3, 4
;
gkdur       oscil1      0, 1, p3, p4     ;control generator for    idur
gkdurr      oscil1      0, 1, p3, p5     ;                         idurr
gkdel       oscil1      0, 1, p3, p6     ;                         idel
gkdelr      oscil1      0, 1, p3, p7     ;                         idelr
gkramp      oscil1      0, 1, p3, p8     ;                         iramp
gkfreq      oscil1      0, 1, p3, p9     ;                         ifreq
gkfreqr     oscil1      0, 1, p3, p10    ;                         ifreqr
gkphase     oscil1      0, 1, p3, p11    ;                         iphase
gkphaser    oscil1      0, 1, p3, p12    ;                         iphaser
gkamp       oscil1      0, 1, p3, p13    ;                         iamp

krnd1   rand 1 , 0.1                     ;random generator  (VOICE #1)
```

```
krnd2   rand    1 , 0.9             ;           (VOICE #2)
krnd3   rand    1 , 0.5             ;           (VOICE #3)
krnd4   rand    1 , 0.3             ;           (VOICE #4)

;The instantaneous values of the random generators are re-scaled to obtain the ;appropriate values of frequency, duration, delay and phase.

gkran   = krnd1 * gkfreqr/2     ;random frequency re-scale (VOICE #1,2,3,4)
gkrnd1  = krnd1 * gkdurr/2          ;random duration re-scale   (VOICE #1)
gkrnd2  = krnd2 * gkdurr/2      ;                           (VOICE #2)
gkrnd3  = krnd3 * gkdurr/2      ;                               (VOICE #3)
gkrnd4  = krnd4 * gkdurr/2      ;                           (VOICE #4)

gkrnd1y = krnd1 * (0.05 + gkdelr /2) ;random delay re-scale(VOICE #1)
gkrnd2y = krnd2 * (0.05 + gkdelr /2) ;              (VOICE #2)
gkrnd3y = krnd3 * (0.05 + gkdelr /2) ;              (VOICE #3)
gkrnd4y = krnd4 * (0.05 + gkdelr /2) ;              (VOICE #4)

gkrnd1p = krnd1 * gkphaser/2  ;random phase re-scale (VOICE #1)
gkrnd2p = krnd2 * gkphaser/2  ;                      (VOICE #2)
gkrnd3p = krnd3 * gkphaser/2  ;                          (VOICE #3)
gkrnd4p = krnd4 * gkphaser/2      ;                          (VOICE #4)
endin
```

Referring to the previous listing code, the global variable *gkdur* is the output of the ìone-shotî oscillator *oscil1* that read a control function (parameter *p4*) defined within the score. Four independent random generators were used to generate all the aleatory, control parameters.

All variables that include the aleatory component, produce complex control function that generate ideal *tendency masks* as represented in fig. 5b. Referring to the figure 4 and 5, it is possible to visualize the combination of the deterministic and random component of the trapezoid duration as a function of time: the effective value of the trapezoid duration is represented by the shaded area of figure 5b. The variables with deterministic and aleatory components are:

a) grain duration (independent aleatory component for each voice)
b) grain delay (independent aleatory component for each voice)
c) phase (independent aleatory component for each voice)
d) frequency (same aleatory component for all voices)

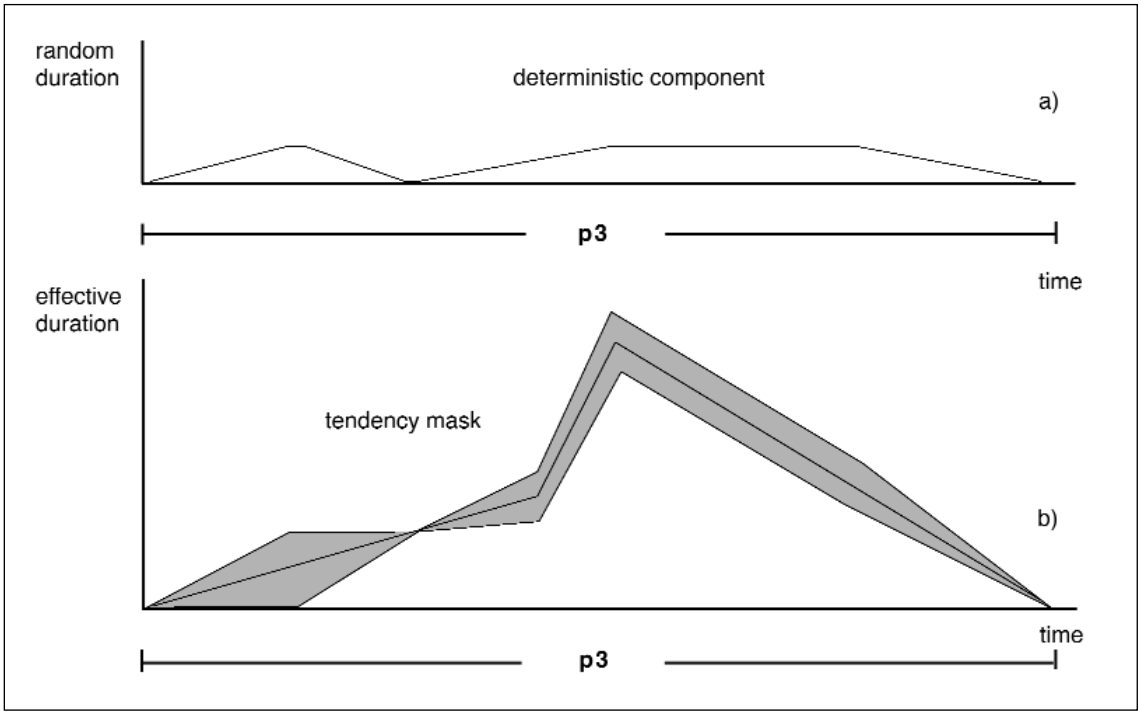

Fig.5

After the grains parameters are updated (at re-initialization time), the grain envelope is generated usig the simple *linseg* opcode, whose parameters are *irise*, *isus*, *idel*, *iamp* (see fig. 3):

```
cont:
k1 linseg 0, irise, iamp, isus, iamp, irise, 0, idel, 0   ;generate grain envelope

ga1 oscili k1, ifq, ifun, iph                             ;generate voice 1

endin
```

The same envelope function is used to control the amplitude (*kl*) of an interpolated oscillator (*oscili*).

The frequency input of this oscillator is controlled by the i-variable *ifq*. This includes the respective deterministic and aleatory component. The same for the phase (*iph*) whereas the variable *ifun* points to the audio function to be granulated.

The complete algorithm includes three further instruments (instr 2,3,4) that provide to the generation of additional de-correlated sequences of grains, in order to achieve a

minimum sound texture. The de-correlation of the four instruments is audible only when the respective aleatory components are not zero.

```
krnd1  rand 1 , 0.1      ;random generator  (VOICE #1)
krnd2  rand 1 , 0.9      ;                  (VOICE #2)
krnd3  rand 1 , 0.5      ;                  (VOICE #3)
krnd4  rand 1 , 0.3      ;                  (VOICE #4)
```

The only difference in the previous code lines consists of different seed number in the random generation process (in the above example 0.1, 0.9, 0.5, 0.3), so the resulting sound is much more rich and ìchorusedî. When the amount of all the aleatory components is zero, the four voices are synchronized, thie ìchoralî effect is canceled and the sound is not so interesting as before. For that reason, an offset value (0.05) is permanetely added to the random generator.

5. Conclusions and future expansions

A Csound implementation of Granular Synthesis was implemented starting from the model proposed by Truax. The program structure consists of a basic group of six instruments controlled by a score, in which are specified all the synthesis parameters. It is possible to granulate any periodic waveform or a sampled sound. Starting from this basic version, it is easy to expand the synthesis process to a greater number of voices. We suggest, also, that you experiment with the granulation of different sampled sound and with different grain envelope shapes and realizing a real-time control of all the synthesis parameters. We have developed an experimental real-time C version of this program that allows up to sixteen full stereo voices. This version is integrated in a custom version of Csound.

APPENDIX (GSC4 - ORCHESTRA)

```
;gsc4.orc
;
;
;                       GRANULAR SYNTHESIS
;                  - - - - - - - - - - - - - - - -
;                              Ver 2.1
;                           Eugenio Giordani
;
;
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;This orchestra implements Granular Synthesis based on the model proposed by B. Truax.
;
;- - - - - - - - - - - - - - - - -  ORCHESTRA HEADER - - - - - - - - - - - - - - - - -;
;
            sr      =   44100
            kr      =   22050
            ksmps   =   2
            nchnls  =   2
;- - - - - - - - - - - - - - - - - Global control variables initialization - - - - - - - - - - - - - - - - -
;
gkdur       init 0          ;average grains duration
gkdurr      init 0          ;random grains duration
gkdel       init 0          ;average grains delay
gkdelr      init 0          ;random grains delay
gkramp      init 0          ;ramp ratio
gkfreq      init 0          ;average audio frequency
gkfreqr     init 0          ;random audio frequency
gkphase     init 0          ;average phase
gkphaser    init 0          ;random phase
gkamp       init 0          ;overall amplitude

gkran       init 0      ;instantaneous random frequency

gkrnd1      init 0      ;instantaneous random grains duration   (VOICE #1)
gkrnd2      init 0      ;                                       (VOICE #2)
gkrnd3      init 0      ;                                       (VOICE #3)
```

```
gkrnd4      init 0        ;                                        (VOICE #4)

gkrnd1y     init 0            ;instantaneous random grains delay      (VOICE #1)
gkrnd2y     init 0            ;                                       (VOICE #2)
gkrnd3y     init 0            ;                                       (VOICE #3)
gkrnd4y     init 0            ;                                       (VOICE #4)

gkrnd1p     init 0            ;instantaneous random grains phase(VOICE #1)
gkrnd2p     init 0            ;                                       (VOICE #2)
gkrnd3p     init 0            ;                                       (VOICE #3)
gkrnd4p     init 0            ;                                       (VOICE #4)
;
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
instr 1
;====================== GRAINS GENERATOR (VOICE #1)=========================
ifun     =  p4                ;audio function
;

;Grains parameters update (re-initialization)

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
loop:
idur     = i(gkdur)               ;current value of gkdur is sampled from
                            ;the corresponding generator in instr 11
                            ;and converted in a i-type variable

idurr    = i(gkrnd1)              ; current value of gkrnd1 is ......

itrpz    = abs(0.001* (idur + idurr))  ;calculates the trapezoid duration

iramp    = i(gkramp)+ 0.1     ;current value of gkramp (plus a magic) is ..

idel     = i(gkdel)               ; current value of gkdel is....
idelr    = i(gkrnd1y)             ; current value of gkrnd1y is....
idely    = abs(0.001 * (idel + idelr)) ;calculates the total delay

ifreq    = i(gkfreq)              ; current value of gkfreq is ....
ifreqr   = i(gkran)               ; current value of gkranis ....
iphase   = i(gkphase)     ; current value of gkphase is ....
```

```
iphaser = i(gkrnd1p)     ; current value of gkrn1p is ....
iamp    = i(gkamp)       ; current value of gkamp is ....

irise    = itrpz/iramp          ;calculates the trapezoid attack time
isus     = itrpz - (2 * irise)  ;calculates the trapezoid sustain time
igrain   = itrpz + idely        ;calculates the sum trapezoid+delay duration
iph      = abs(iphase + iphaser);calculates the phase(deterministic + random)
ifq      = ifreq + ifreqr       ;calculates the freq.(deterministic + random)

;- - - - - - - - - - - - - - - - - Interrupt simulation section  - - - - - - - - - - - - - - - - -
;
;timeout work as an interrupt logic. It is loaded with the current grain ;duration and automatically decrements to zero

timout  0, igrain, cont         ;if the current value of the timer
                                ;is not zero,branch to the program section labelled with cont

reinit loop                     ;otherwise jump to re-init pass

cont:
k1 linseg 0, irise, iamp, isus, iamp, irise, 0, idel, 0   ;generate grain envelope

ga1 oscili k1, ifq, ifun, iph                       ;generate voice 1


endin

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
instr 2
;====================== GRAINS GENERATOR (VOICE #2)=========================
ifun   =  p4                ;audio function
;

;Grains parameters update (re-initialization)

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
loop:
idur     = i(gkdur)         ;current value of gkdur is sampled from
                                ;the corresponding generator in instr 11
                            ;and converted in a i-type variable
```

```
idurr    = i(gkrnd2)                    ;current value of gkrnd2 is ......

itrpz    = abs(0.001* (idur + idurr))   ;calculates the trapezoid duration

iramp    = i(gkramp)+ 0.1               ;current value of gkramp (plus a magic) is ..

idel     = i(gkdel)                     ; current value of gkdel is....
idelr    = i(gkrnd2y)                   ; current value of gkrnd2y is....
idely    = abs(0.001 * (idel + idelr)) ;calculates the total delay

ifreq    = i(gkfreq)                    ; current value of gkfreq is ....
ifreqr   = i(gkran)                     ; current value of gkranis ....
iphase   = i(gkphase)                ; current value of gkphase is ....
iphaser = i(gkrnd2p)                 ; current value of gkrn2p is ....
iamp     = i(gkamp)                  ; current value of gkamp is ....

irise    = itrpz/iramp               ;calculates the trapezoid attack time
isus     = itrpz - (2 * irise)          ;calculates the trapezoid sustain time
igrain   = itrpz + idely                ;calculates the sum trapezoid+delay duration
iph      = abs(iphase + iphaser)        ;calculates the phase(deterministic + random)
ifq      = ifreq + ifreqr               ;calculates the freq.(deterministic + random)

;- - - - - - - - - - - - - - - - - Interrupt simulation section - - - - - - - - - - - - - - - - -
;
;timeout work as an interrupt logic. It is loaded with the current grain ;duration and automatically decrements to zero

timout 0, igrain, cont                  ;if the current value of the timer
                                        ;is not zero,branch to the program
                                        ;section labelled with cont

reinit loop                             ;otherwise jump to re-init pass


cont:
k1 linseg 0, irise, iamp, isus, iamp, irise, 0, idel, 0   ;generate grain envelope

ga2 oscili k1, ifq, ifun, iph           ;generate voice 2

endin
```

```
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
instr 3
;====================== GRAINS GENERATOR (VOICE #3)=========================
ifun     =  p4                              ;audio function
;

;Grains parameters update (re-initialization)

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
loop:
idur     = i(gkdur)                  ;current value of gkdur is sampled from
                                     ;the corresponding generator in instr 11
                                     ;and converted in a i-type variable

idurr    = i(gkrnd3)                     ;current value of gkrnd3 is ......

itrpz    = abs(0.001* (idur + idurr))    ;calculates the trapezoid duration

iramp    = i(gkramp)+ 0.1                ;current value of gkramp (plus a magic) is ..

idel     = i(gkdel)                      ; current value of gkdel is....
idelr    = i(gkrnd3y)                    ; current value of gkrnd3y is....
idely    = abs(0.001 * (idel + idelr)) ;calculates the total delay

ifreq    = i(gkfreq)                     ; current value of gkfreq is ....
ifreqr   = i(gkran)                      ; current value of gkranis ....
iphase  = i(gkphase)                ; current value of gkphase is ....
iphaser = i(gkrnd3p)                ; current value of gkrn3p is ....
iamp    = i(gkamp)                  ; current value of gkamp is ....

irise    = itrpz/iramp               ;calculates the trapezoid attack time
isus     = itrpz - (2 * irise)           ;calculates the trapezoid sustain time
igrain   = itrpz + idely                 ;calculates the sum trapezoid+delay duration
iph      = abs(iphase + iphaser)         ;calculates the phase(deterministic + random)
ifq      = ifreq + ifreqr                ;calculates the freq.(deterministic + random)

;- - - - - - - - - - - - - - - - - Interrupt simulation section - - - - - - - - - - - - - - - - -
;
;timeout work as an interrupt logic. It is loaded with the current grain ;duration and automatically decrements to zero
```

```
timout 0, igrain, cont                  ;if the current value of the timer
                                        ;is not zero,branch to the program
                                        ;section labelled with cont

reinit loop                             ;otherwise jump to re-init pass


cont:
k1 linseg 0, irise, iamp, isus, iamp, irise, 0, idel, 0   ;generate grain envelope

ga3 oscili k1, ifq, ifun, iph           ;generate voice 3

endin
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
instr 4
;====================== GRAINS GENERATOR (VOICE #4)=========================
ifun    =  p4                           ;audio function
;

;Grains parameters update (re-initialization)

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
loop:
idur    = i(gkdur)              ;current value of gkdur is sampled from
                                ;the corresponding generator in instr 11
                                ;and converted in a i-type variable

idurr   = i(gkrnd4)                 ;current value of gkrnd4 is ......

itrpz   = abs(0.001* (idur + idurr))    ;calculates the trapezoid duration

iramp   = i(gkramp)+ 0.1                ;current value of gkramp (plus a magic) is ..

idel    = i(gkdel)                  ; current value of gkdel is....
idelr   = i(gkrnd4y)            ; current value of gkrnd4y is....
idely   = abs(0.001 * (idel + idelr)) ;calculates the total delay

ifreq   = i(gkfreq)                 ; current value of gkfreq is ....
```

```
ifreqr    = i(gkran)                    ; current value of gkranis ....
iphase  = i(gkphase)                ; current value of gkphase is ....
iphaser = i(gkrnd4p)                ; current value of gkrn4p is ....
iamp    = i(gkamp)                  ; current value of gkamp is ....

irise     = itrpz/iramp             ;calculates the trapezoid attack time
isus      = itrpz - (2 * irise)          ;calculates the trapezoid sustain time
igrain    = itrpz + idely                ;calculates the sum trapezoid+delay duration
iph       = abs(iphase + iphaser)        ;calculates the phase(deterministic + random)
ifq       = ifreq + ifreqr               ;calculates the freq.(deterministic + random)

;- - - - - - - - - - - - - - - - - Interrupt simulation section - - - - - - - - - - - - - - - - -
;
;timeout work as an interrupt logic. It is loaded with the current grain ;duration and automatically decrements to zero

timout 0, igrain, cont                ;if the current value of the timer
                                      ;is not zero,branch to the program
                                    ;section labelled with cont

reinit loop                           ;otherwise jump to re-init pass


cont:
k1 linseg 0, irise, iamp, isus, iamp, irise, 0, idel, 0   ;generate grain envelope

ga4 oscili k1, ifq, ifun, iph         ;generate voice 4

endin

;============================== GRAIN CONTROL BLOCK ========================
instr 11
;NOTE: all the global variables are transmitted to the  instr 1,2,3,4
;
gkdur       oscil1  0, 1, p3, p4      ;control generator for    idur
gkdurr      oscil1  0, 1, p3, p5      ;                         idurr
gkdel       oscil1  0, 1, p3, p6      ;                         idel
gkdelr      oscil1  0, 1, p3, p7      ;                         idelr
gkramp      oscil1  0, 1, p3, p8      ;                         iramp
gkfreq      oscil1  0, 1, p3, p9      ;                         ifreq
```

```
gkfreqr     oscil1  0, 1, p3, p10       ;                       ifreqr
gkphase     oscil1  0, 1, p3, p11       ;                       iphase
gkphaser    oscil1  0, 1, p3, p12       ;                       iphaser
gkamp       oscil1  0, 1, p3, p13   ;                       iamp

krnd1       rand 1 , 0.1                ;random generator   (VOICE #1)
krnd2       rand 1 , 0.9                ;                   (VOICE #2)
krnd3       rand 1 , 0.5                ;                   (VOICE #3)
krnd4       rand 1 , 0.3                ;                   (VOICE #4)

;The instantaneous values of the random generators are re-scaled to obtain the ;appropriate values of frequency, duration,
delay and phase.

gkran       = krnd1 * gkfreqr/2     ;random frequency re-scale  (VOICE #1,2,3,4)
gkrnd1      = krnd1 * gkdurr/2      ;random duration re-scale   (VOICE #1)
gkrnd2      = krnd2 * gkdurr/2      ;                           (VOICE #2)
gkrnd3      = krnd3 * gkdurr/2      ;                           (VOICE #3)
gkrnd4      = krnd4 * gkdurr/2      ;                           (VOICE #4)

gkrnd1y= krnd1 * (0.05 + gkdelr /2) ;random delay re-scale      (VOICE #1)
gkrnd2y     = krnd2 * (0.05 + gkdelr /2) ;                      (VOICE #2)
gkrnd3y     = krnd3 * (0.05 + gkdelr /2) ;                      (VOICE #3)
gkrnd4y     = krnd4 * (0.05 + gkdelr /2) ;                      (VOICE #4)

gkrnd1p     = krnd1 * gkphaser/2        ;random phase re-scale      (VOICE #1)
gkrnd2p     = krnd2 * gkphaser/2        ;                           (VOICE #2)
gkrnd3p     = krnd3 * gkphaser/2        ;                           (VOICE #3)
gkrnd4p     = krnd4 * gkphaser/2        ;                           (VOICE #4)
endin

;======================== RESCALE, MIX & OUT ======================

instr 21
iscale      = p4                    ;read scale factor
outs1 (ga1/2 + ga2/2 ) * iscale         ;outputs voices 1 - 2
outs2 (ga3/2 + ga4/2 ) * iscale         ;outputs voices 3 - 4


endin
```

(GSC4 - score)

```
;gsc4.sco
;
;- - - - - - - - - - - - - - - - - Grains duration control function - - - - - - - - - - - - - - - - -
;
f11 0  512 -7 10 256 20  128 20 128 16

;- - - - - - - - - - - - - - - - - Random grains duration control function - - - - - - - - - - - - - - - - -
;
f12 0 512 -7 4 256 1 256 0

;- - - - - - - - - - - - - - - - - Grains delay control function - - - - - - - - - - - - - - - - -
;
f13 0 512 -7 10 256 20 256 5

;- - - - - - - - - - - - - - - - -  Random grains delay control function - - - - - - - - - - - - - - - - -
;
f14 0 512  -7 0 128  0 256 2 128 0

;- - - - - - - - - - - - - - - - - Ramp proportion control - - - - - - - - - - - - - - - - -
;
f15 0 512 -7 2 256 4 256 2

;- - - - - - - - - - - - - - - - - Frequency control function - - - - - - - - - - - - - - - - -
;
;f16 0 512 -7 1.345 512 3.345
f16  0 512 -7 220 512 220

;- - - - - - - - - - - - - - - - - Random frequency control function - - - - - - - - - - - - - - - - -
;
f17 0 512 -7  0  512  110

;- - - - - - - - - - - - - - - - - Phase control function  (Audio file pointer) - - - - - - - - - - - - - - - - -
;
f18 0 512 -7 0 512 0

;- - - - - - - - - - - - - - - - -  Random phase control - - - - - - - - - - - - - - - - -
f19 0 512 -7 0 128 0 256 0 128  0
```

```
;- - - - - - - - - - - - - - - - - Overall amplitude control function - - - - - - - - - - - - - - - - -
;
f20 0 512  7 0 128 1 256 1 128 0

;======================    Audio functions    ==============================;
;f1 0 32768 -1 "sample.wav" 0 0 0
f1 0 1024 10 0.6 0.8  1   0.5  0.3  0.5  0.7
;=============================================================================
;p1  p2  p3   p4   p5    p6   p7    p8    p9    p10    p11    p12
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ;
;              ifun
i1   0    40  1
i2   0    40  1
i3   0    40  1
i4   0    40  1
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;                        dur     durr    del     delr    ramp    freq    freqr   phase       phaser    amp
i11      0        40      11      12      13      14      15      16      17      18          19        20
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;                 scale
i21      0    40  20000
e
```

References

Gabor,D. 'Acoustical Quanta and the Theory of Hearing.' *Nature* May 3,1947

Truax,B. 'Real-Time Granular Synthesis with a Digital Signal Processor.' *Computer Music Journal* 12(2)

Truax,B. 'Real-Time Granular Synthesis with the DMX-1000' *ICMC 86 Proceedings*

Roads,C. 'Granular Synthesis' from *The Computer Music Tutorial* (MIT Press)

Roads,C. 'Granular Synthesis of Sound.' *Computer Music Journal* 2(2)

Roads,C. 'The Realization of nscor' from *'The Computer Music and Digital Audio Series'* C.Roads Editor

Truax,B. 'Handbook for Acoustic Ecology' B.Truax Editor

Vercoe,B. 'Csound User Manual' Media Lab MIT

(Translated from the Italian by the author)