C I & S S L a b

UNIVERSITÀ PARTHENOPE
DiST
DIPARTIMENTO DI SCIENZE E TECNOLOGIE

Intelligent Signal Processing

Effects and Sound Synthesis

Angelo Ciaramella

Introduction

- Effects
  - are correlated with the modifications of an acoustic signal
  - events in time and space domains
    - Wave reflection

- Some effects
  - Distortion
  - Vibrato
  - Flanger
  - Chorus
  - Doubling
  - Echo
  - Reverberation
  - …

Traditional effects

- Features
  - the output of a system depends on the music signal input and a certain number of its repetitions
    - e.g., echo, chorus, flanger
  - by using a modulated delay line
    - time invariant filter

Filters

- To elaborate the signals
  - comb filters
  - all-pass (universal comb filters)
  - delay lines (DL)

FIR Comb filters

- The input-output relation is expressed by the following difference equation

$$y[n] = x[n] + gx[n - D]$$

- The transfer function is

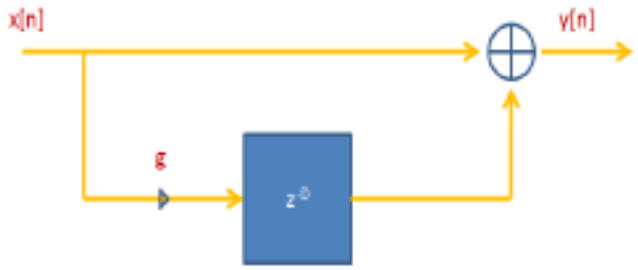
FIR Comb filters

- The input-output relation is expressed by the following difference equation

$$y[n] = x[n] + gx[n - D]$$

- The transfer function is

$$H(z) = 1 + gz^{-D}$$



IIR Comb filters

- The input-output relation is expressed by the following difference equations

$$y[n] = x[n-D] + gy[n-D] \qquad y[n] = x[n] + gy[n-D]$$

- The transfer function is

IIR Comb filters

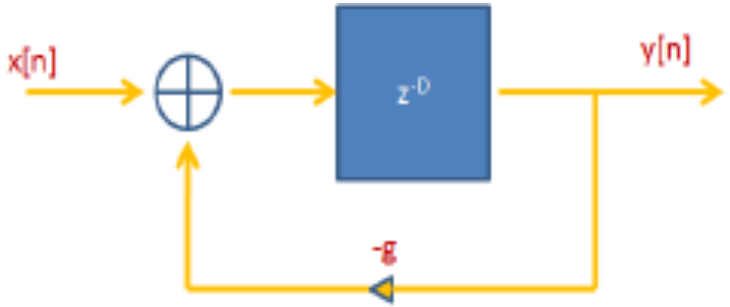
- The input-output relation is expressed by the following difference equations

$$y[n] = x[n-D] + gy[n-D] \qquad y[n] = x[n] + gy[n-D]$$

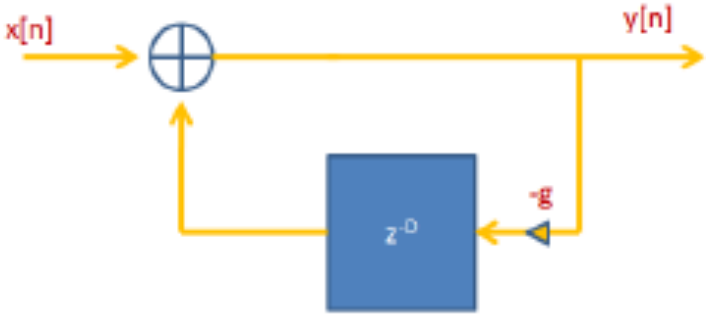
- The transfer function is

$$H(z) = \frac{z^{-D}}{1+gz^{-D}} \qquad H(z) = \frac{1}{1+gz^{-D}}$$

IIR Comb filters



$$H(z) = \frac{z^{-D}}{1 + gz^{-D}}$$



$$H(z) = \frac{1}{1 + gz^{-D}}$$

All-Pass filters

- The input-output relation is expressed by

$$y[n] = a_N x[n] + \ldots + x[n-N] - a_1 y[n-1] - \ldots - a_N y[n-N]$$

- The transfer function is

$$H(z) = \frac{a_N + a_{N-1} z^{-1} + \ldots + a_1 z^{-(N-1)} + z^{-N}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}}$$

Universal Comb filter

- The input-output relation is expressed by

$$y[n] = gx[n] + x[n-D] - gy[n-D]$$

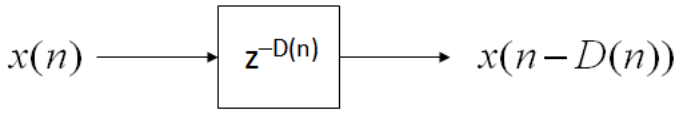
- The transfer function is

$$H(z) = \frac{g + z^{-D}}{1 + gz^{-D}}$$

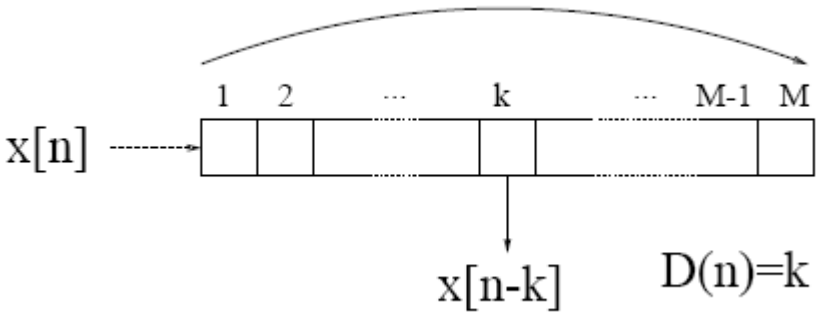
Delay line

- To obtain the delay, a parameter of constant modulation is adopted

$$y[n] = x[n - D]$$



- The simulation can be obtained by using a circular queue



Delay Line

```
// Delay line version I
double delay_line(vector<double> &w, int D, double x)
 {
  int i;
  for(i=D; i>=1; i--) w[i] = w[i-1];

  w[0] = x;

  return w[D];
 }



//Delay line version II
double delay_line_1(vector<double> &w, int D, int &p, double x)
{
 w[p] = x;

 p = (p + 1)%D;

 return w[p];
}
```


Delay Line

```
// Delay line version III
double delay_line_2(vector<double> &w, int D, int &p, double x)
{
 double y = w[p];
 w[p++] = x;

 if(p>=D) p-=D;

 return y;
}
```


Lines with fractional delay

- Generally a fractional delay is needed
  - Fractional Delay (FD)

- Solutions
  - Linear interpolation
  - All-pass Filter
  - Lagrange interpolation

Linear interpolation

- Linear interpolation between two samples of the signal

$$y[n] = x[n-1] + \alpha(x[n] - x[n-1])$$

- otherwise

$$y[n] = (1-\alpha)x[n] + \alpha x[n]$$

All-pass interpolation

- Interpolation with an all-pass filter

$$H(z) = \frac{a + z^{-1}}{1 + az^{-1}}$$

- Difference equation

$$y[n] = ax[n] + x[n-1] - ay[n-1]$$

Lagrange interpolation

- Interpolation on N points

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

C++ implementation

```
// Lagrange interpolation
double lagrange_delayline(vector<double> &w, int D, int &p,
double x)
{
 w[p] = x;

p = (p + 1)%D;

int k,j; double prod=1.0,sum=0.0;

 for(k=1;k<=4;k++)
  {
   prod = w[(p + (k-1))%D];

        for(j=1;j<=4;j++)
            {
                    if(j != k ) prod = prod*(double)(2-j)/(k-j);
            }

   sum = sum + prod;
  }

 return sum;
}
```


Time-varying delay lines

- Most audio effects are based on a DL with a time-varying size

$$y[n] = x[n - D[n]]$$

- A practical way is

$$D[n] = D_0 + D_1 f_D[n] = D_0 (1 + m_D f_D[n])$$

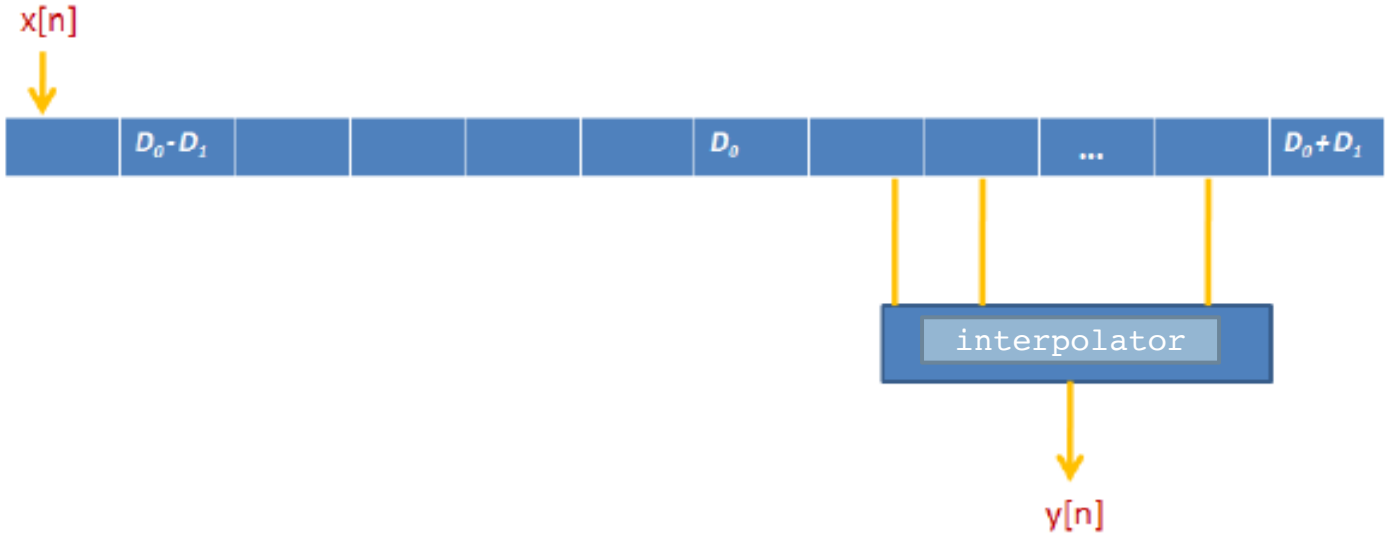
DL size

Variation function

Modulation index

Time-varying delay lines

- Generally `D[n]` is not integer and must be interpolated



Fractional time-varying delay lines

```
double linear_delayline_f_md(double x, int *wptr, vector<double>
&D, int N, double y_n, int D0, double md)
{

double i_frac = ((*wptr) + D0*(1 + md*y_n)) ;

D[(*wptr)++] = x;

long rpi = ((long) floor(i_frac))%N;
double a = i_frac  - (double) floor(i_frac);

double y = a * D[rpi]  + (1 - a) * D[rpi + 1];

if((*wptr) >= N) { *wptr -= N; }

return y;
}
```


Fractional time-varying delay lines

```
double all_delayline_f(double x, int *wptr, vector<double> &D,
int N, double y_n, int D0, int D1, double y_p)
{
double i_frac = (*wptr) + D0 + D1 * y_n;
D[(*wptr)++] = x;

long rpi = ((long) floor(i_frac))%N;
double a = i_frac  - (double) floor(i_frac);


double y = (1-a) * D[rpi]  + D[rpi + 1] - (1 - a) * y_p;

if((*wptr) >= N) { *wptr -= N; }

return y;
}
```

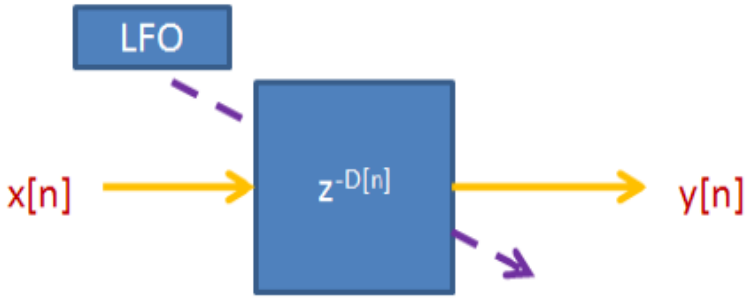

Fractional time-varying delay lines

```
double lagrange_delayline_f(double x, int *wptr, vector<double> &D, int N,
double y_n, int D0, int D1)
{

double i_frac = (*wptr) + D0 + D1 * y_n;
D[(*wptr)++] = x;

long rpi = ((long) floor(i_frac))%N;

int k,j; double prod=1.0,sum=0.0;

 for(k=1;k<=4;k++)
  {
   prod = D[(rpi + (k-1))%N];

        for(j=1;j<=4;j++)
             {
                      if(j != k ) prod = prod*(double)(2-j)/(k-j);
             }

   sum = sum + prod;
  }

if((*wptr) >= N) { *wptr -= N; }

return sum;
}
```


Time-varying delay lines

- Generally `D[n]` is continuous
  - Integer and fractional part

- A Low Frequency Oscillator (LFO) can be used for modulation



Pitch shift

$$D[n] = (1 - p)n$$

pitch change ratio

$$x[n - D[n]] = x[pn]$$

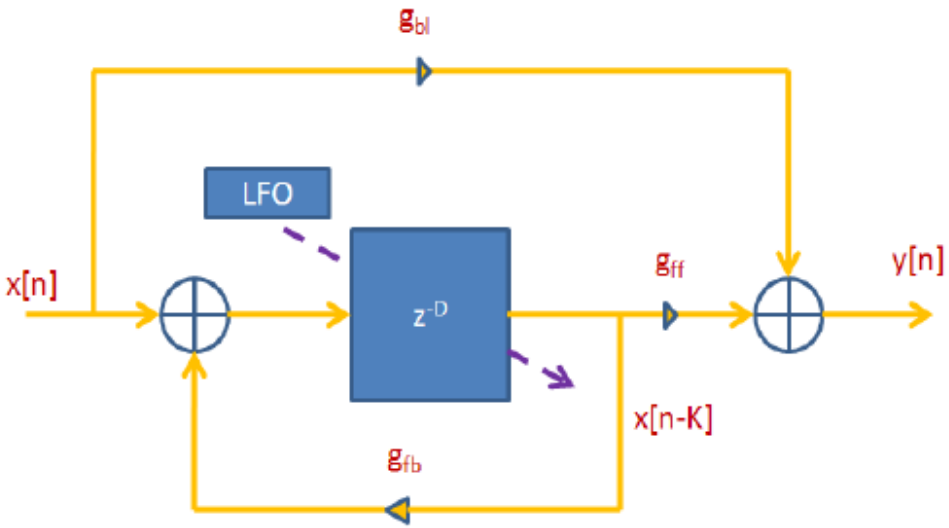
for p = 2

Pitch shift

```
double pitch_modulation(double x, int *wptr, vector<double> &D, int N, int
i, int D0, int p)
{

long rpi = (D0 + (1 - p)*i)%N;

D[(*wptr)++] = x;

if((*wptr) >= N) { *wptr -= N; }

return D[rpi];
}
```


Effects

- General model for effects



$$H(z) = \frac{g_{bl} + g_{ff} z^{-D[n]}}{1 + g_{fb} z^{-K}}$$

Vibrato

- From the general model eliminating feedback and blending
- Delay less than 5 ms
- Size of delay line

$$D_1 = \frac{f_c}{2 f_0} m$$

Effects

```
void vibrato_lagrange(int N, int D0, int D1, int fc)
{
          int i, wptr = 0;

          vector<double> A(N,0.0);

          for(i = 0; i < sequenze<double>::x.size() ; i++)
      {
           y.push_back(lagrange_delayline_f(sequenze<double>::x[i], &wptr,
A,
                  N, sin(2*PI*i/fc), D0, D1));

           }
}
```


Flanging

- From the general model eliminating feedback and blending
- Delay in the range 1 ms – 10 ms
- Equations

$$y[n] = x[n] + g_{ff} x[n - D[n]]$$

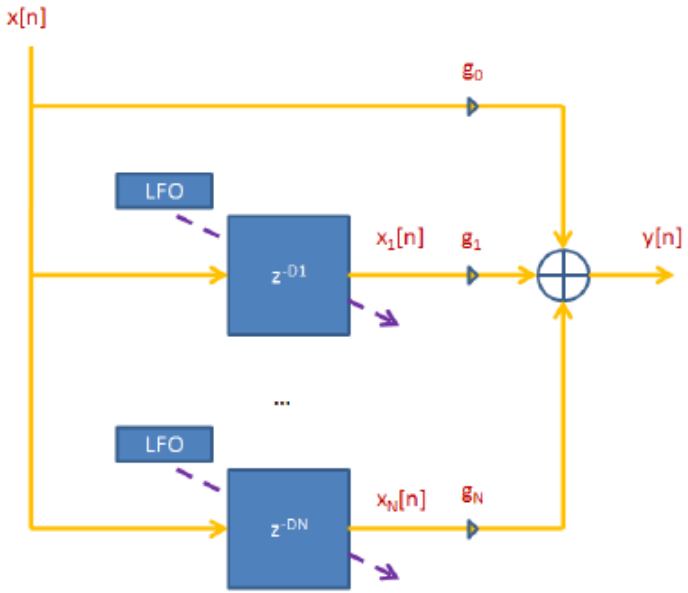
$$D[n] = D_0 + D_1 \sin(2\pi f_{FL} n)$$

Flanging

```
void flanging_all(int N, int D0, int D1, int fc, double g)
{
        int i, wptr = 0;

        vector<double> A(N,0.0);

        double app = 0;

        for(i = 0; i < sequenze<double>::x.size() ; i++)  {

           y.push_back(all_delayline_f(sequenze<double>::x[i], &wptr, A,
N, sin(2*PI*i/fc), D0, D1, app));

           app = y[i];
        }


    for(i = 0; i < y.size() ; i++) y[i] = (x[i] + g * y[i])/2;


}
```


Chorus

- At least two voices



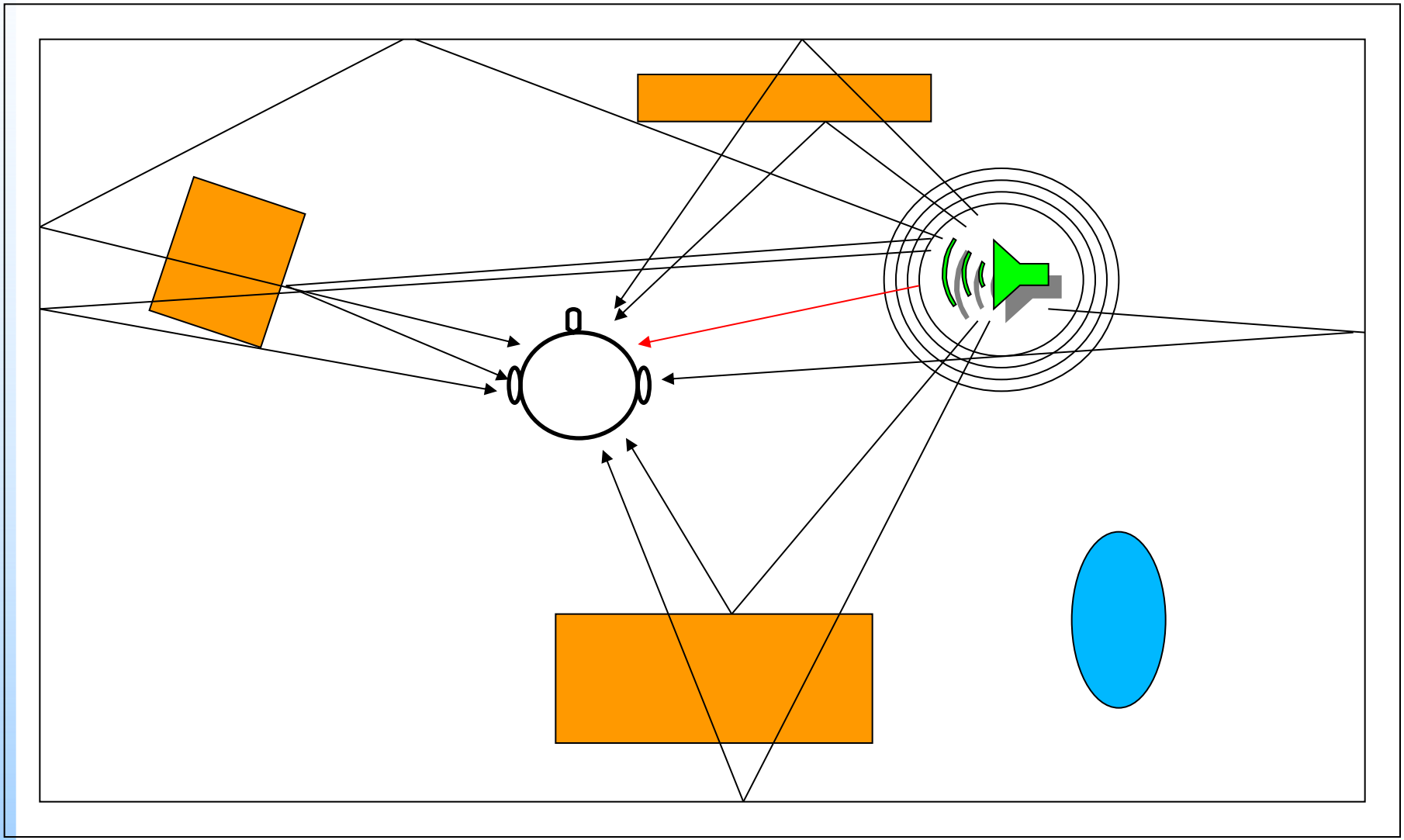
Chorus

```
void chorus_2_voci(int N1, int D01, int D11, int N2, int D02, int D12, int
fc, double g1, double g2)
{
        int i, wptr = 0;

        vector<double> A1(N1,0.0), A2(N2,0.0);

        int wptr1 =0, wptr2 = 0;

        for(i = 0; i < sequenze<double>::x.size() ; i++)  {

          y.push_back(lagrange_delayline_f(sequenze<double>::x[i],
&wptr1, A1, N1, sin(2*PI*i/fc), D01, D11));

      y[i] = (g1*y[i] + g2*lagrange_delayline_f(sequenze<double>::x[i],
&wptr2, A2, N2, sin(2*PI*i/fc), D02, D12))/2;

        }


}
```


Effects

	g_{bl}	g_{ff}	g_{fb}	Onset	Depth	Modulazione
Vibrato	0.0	1.0	0.0	0 ms	0-5ms	0.1-5 Hz sinusoidale
Flanger	0.707	0.707	-0.707	0ms	1-10 ms	0.1- 1 Hz sinusoidale
Chorus	1.0	0.707	0.0	1-30 ms	5-30 ms	Lowpass noise
White chorus	0.707	1.0	0.707	1-30 ms	5-30 ms	Lowpass noise
Doubling	0.707	0.707	0.0	10-100 ms	1-100 ms	Lowpass noise
Eco	1.0	≤ 1.0	< 0	50-∞	80-∞	-

Reverberation



Reverberation

- A simple approach is based on the convolution of the room impulse response

$$y(n) = h_i \otimes x(n)$$

- A more sophisticated methodology is based on a perspective approach

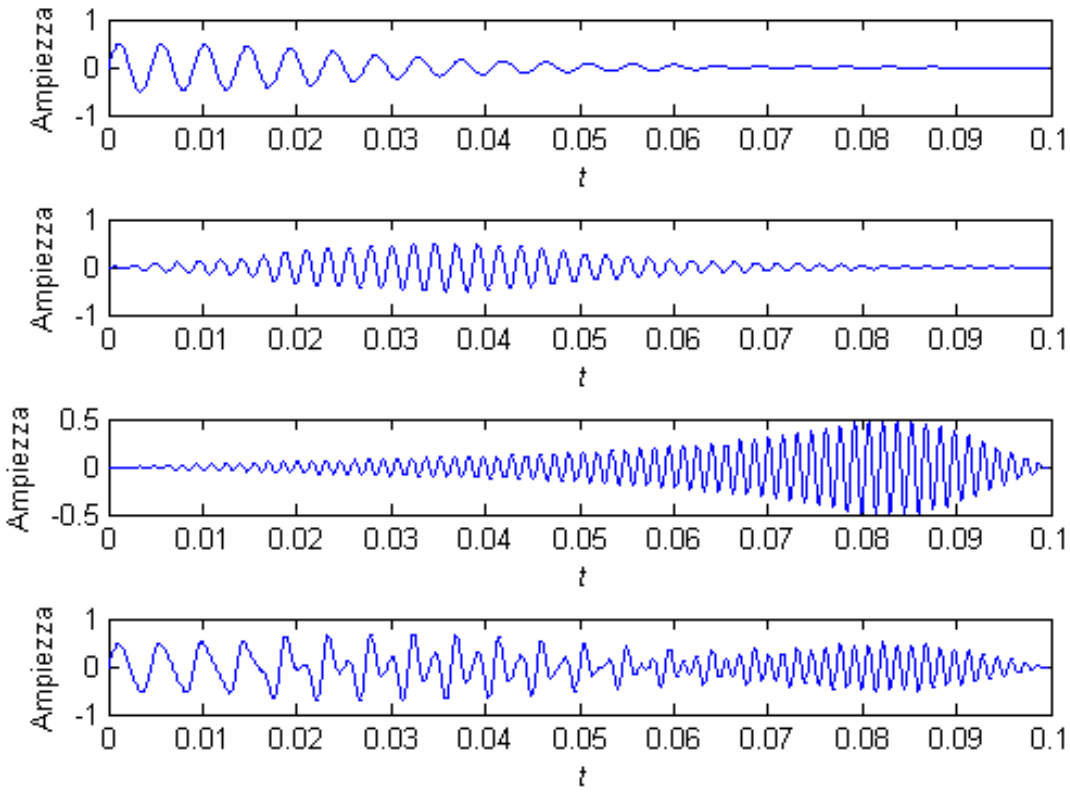
Sound Synthesis

- A waveform of the sound to be generated is computed by using models

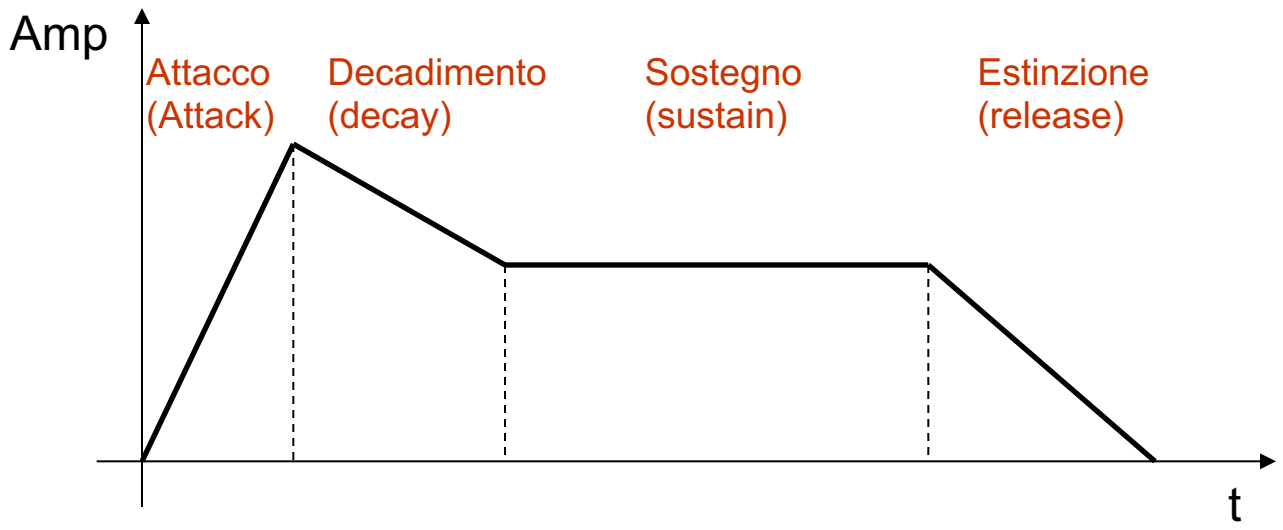
- Some approaches
  - Additive synthesis
  - Physical modelling synthesis

Additive syntheis

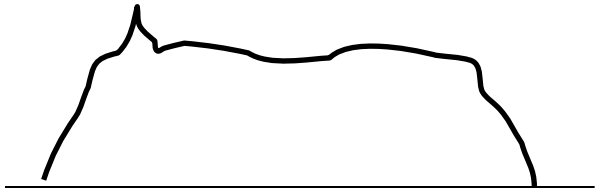
- Directly from the Fourier Theorem



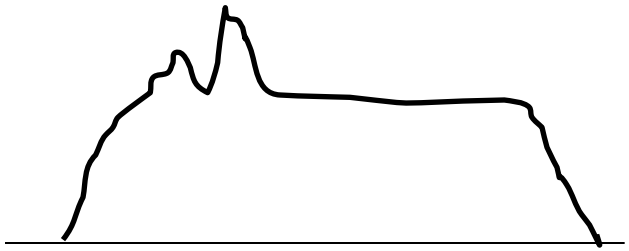
Envelope


Amp
Attacco
(Attack)
Decadimento
(decay)
Sostegno
(sustain)
Estinzione
(release)
t

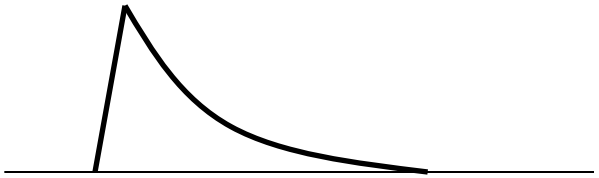
Envelope



Flute

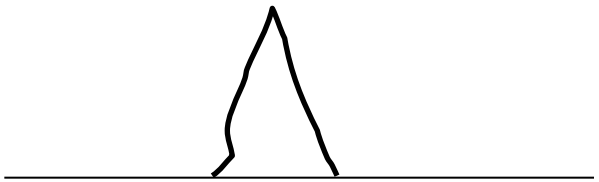


Trumpet



Piano

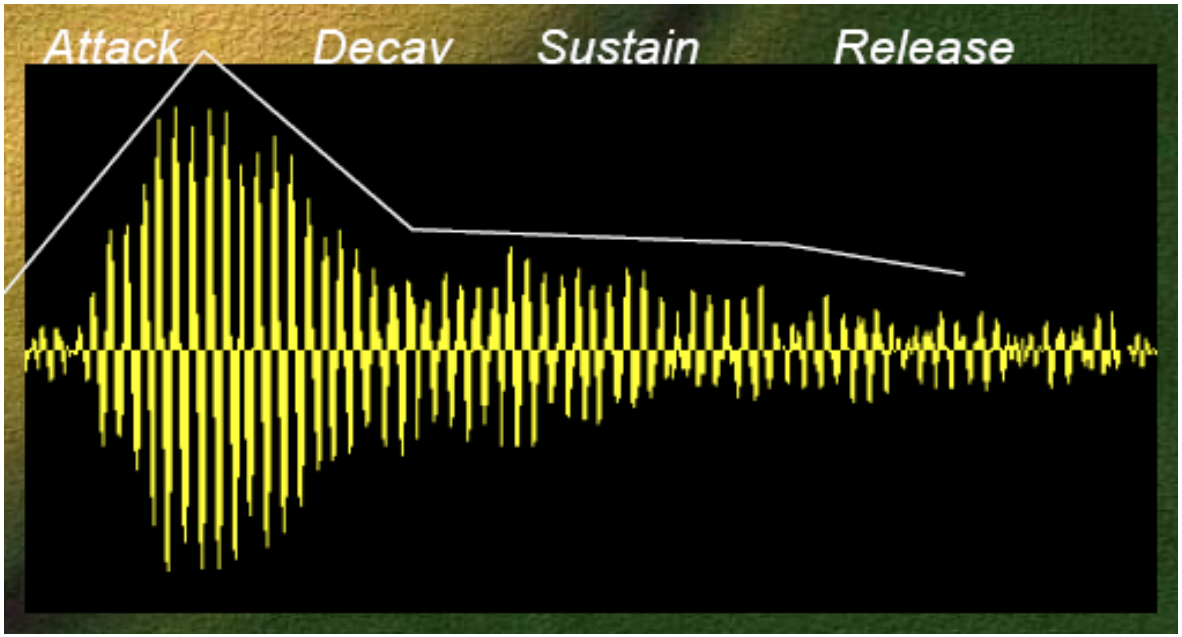
Violin



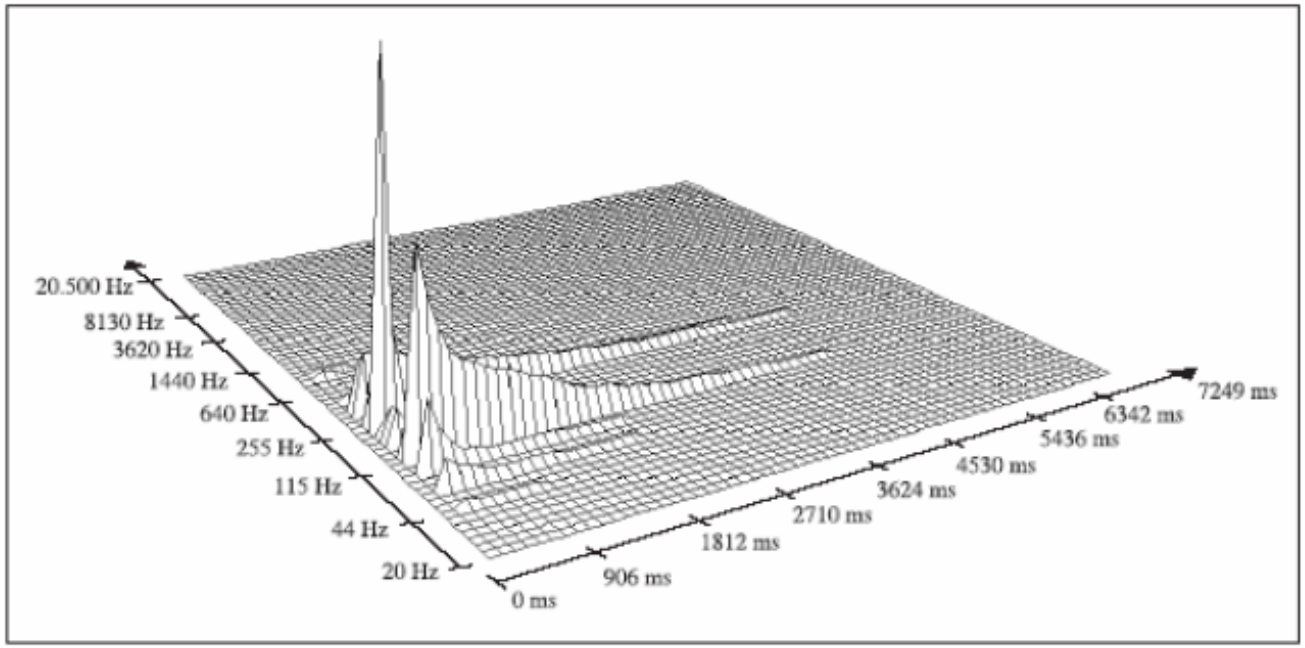
Wood blocks

Contrabass

Envelope



Spectogram


20.500 Hz
8130 Hz
3620 Hz
1440 Hz
640 Hz
255 Hz
115 Hz
44 Hz
20 Hz
0 ms
906 ms
1812 ms
2710 ms
3624 ms
4530 ms
5436 ms
6342 ms
7249 ms

Physical modelling synthesis

- A mathematical model is used
  - equations or algorithms to simulate a physical source of sound
    - usually a musical instrument
- Methodology
  - Karplus-Strong algorithm

Karplus-Strong algorithm

- Karplus-Strong method
  - use the delay line

$$y[n] = x[n] + R^L y[n-L]$$

Difference equation

Comb filter

$$H(z) = \frac{1}{1 - R^L z^{-L}} = \frac{1}{z^L - R^L}$$

Transfer fuction

Chord - Karplus-Strong

```
class corda_lagrange
{
public:
corda_lagrange(int M)
{
    m = M;
    in = vector<double>(m+2,0.0);
}


double var_lagrange(double x, double d) { ... }

private:
int m;
vector<double> in;

};
```

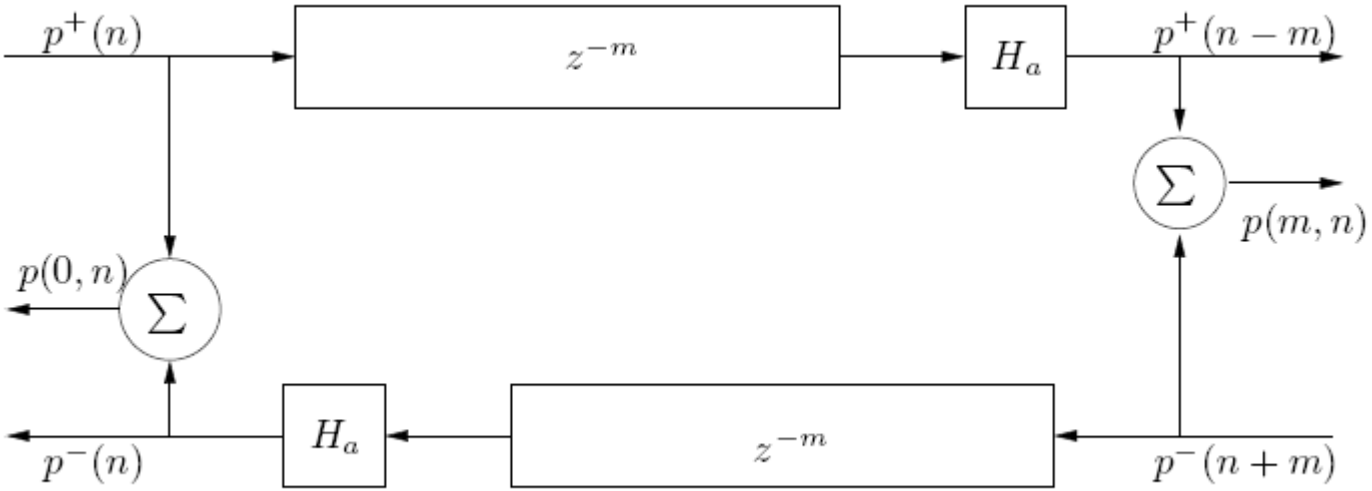

Chord - Karplus-Strong

```
double var_lagrange(double x, double d)
    {
     double sum = 0, prod = 1;
     int k,j, a = floor(m - d);
     for(k=1;k<3;k++)
     {
         prod = in[a + k];
          for(j=1;j<3;j++)
           {
             if( j != k)
               prod = prod * (2-j)/(k-j);
           }
       sum = sum + prod;
     }
    in.push_back(x);
    in.erase(in.begin());
    return sum;
}
```


Chord of an acoustic guitar (A - 440Hz)

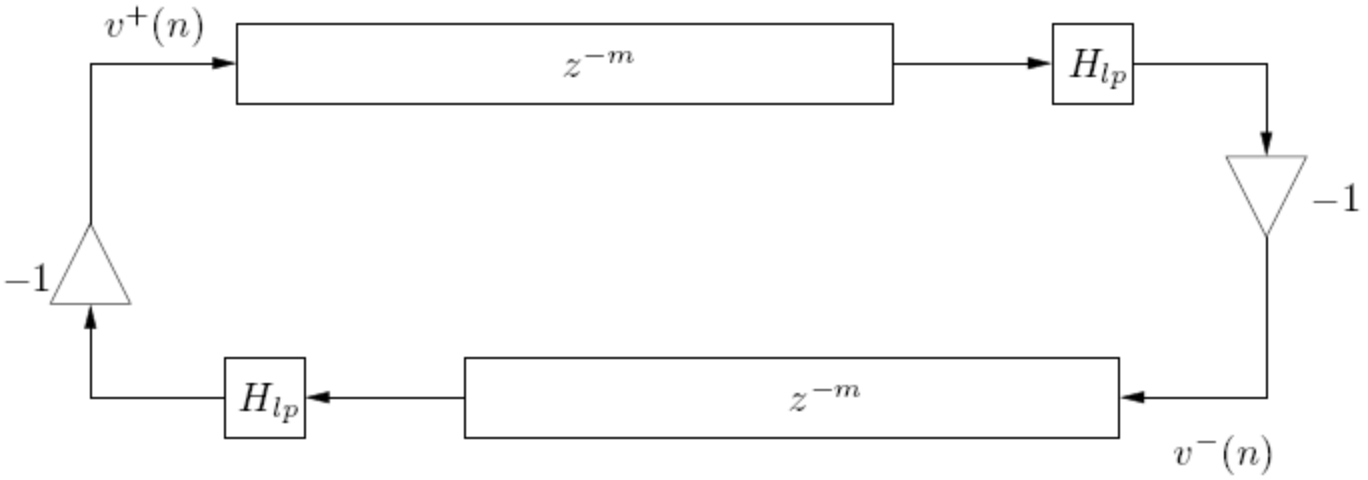
```
for(i = 0; i < (int)ceil(L); i++)
                x[i] = (2*(double)rand()/RAND_MAX - 1)/2;

        // Delay Line with Lagrange Interpolation

    EFFETTI E(x);
    double d = L;
    vector<double> w((int)d,0.0);
    double res =0;
    int p = 0;

    for(n=0;n< x.size(); n++)
        {
      y[n] = x[n] +  res*0.995;
      res = E.lagrange_delayline(w, d, p, y[n]);
        }
```


Chord of an acoustic guitar (A - 440Hz)



Using waveguides

Chord of an acoustic guitar (A - 440Hz)



Ideal chord with dissipation

$$L = \frac{lF_s}{\sqrt{\frac{T}{m/l}}}$$

Lenght of the delay line considering physical parameters

T – Tension of the chord

M – mass

l – length

F_s – sampling frequency

Chord of an acoustic guitar (A - 440Hz)

```
for(n=0;n< x.size(); n++)
       {

      a[n] = x[n] +  res*0.995;

      if(n==1)
         y[n] = a[n]/2;
         else y[n] = 0.5 * (a[n] + a[n-1]);


      res = E.var_lagrange(y[n], d);

      d = L * ( 1 + mod * sin(n/(mods*L)));
        }
```