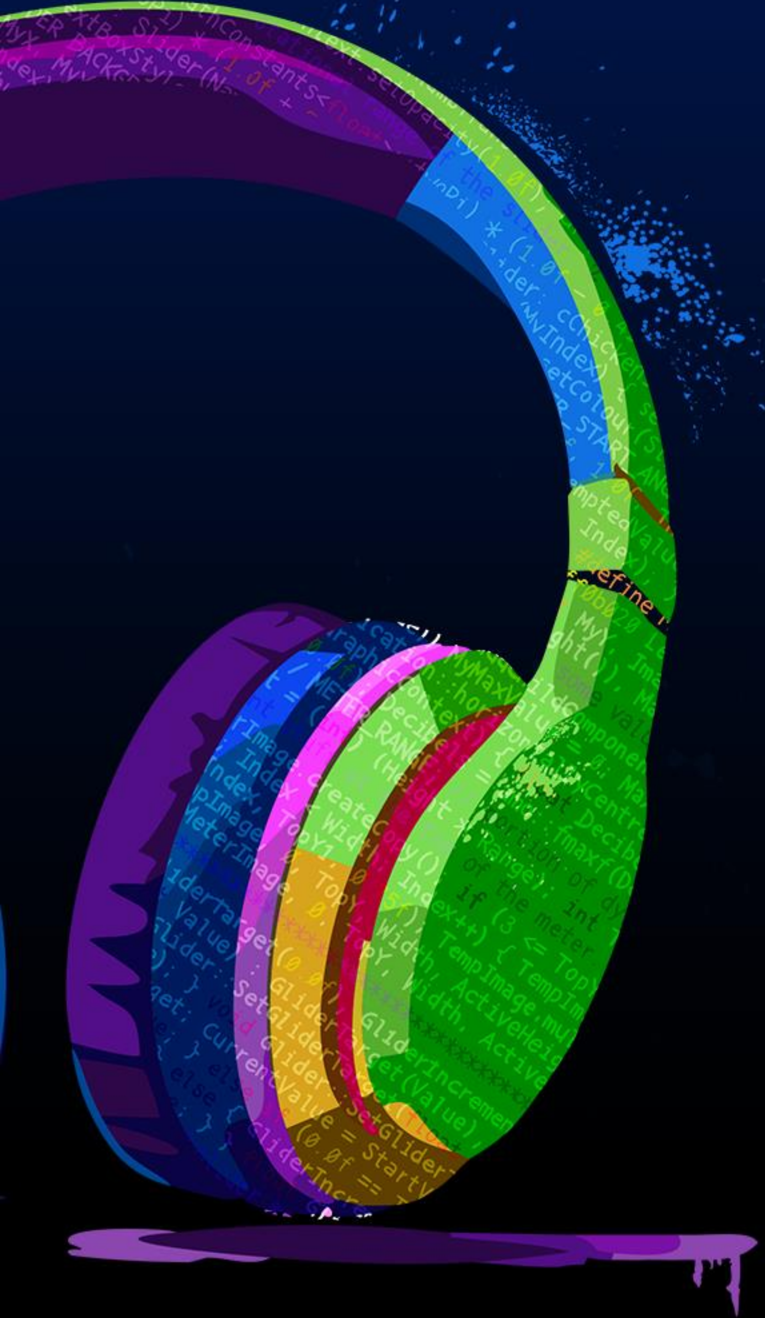


ADC24 Bristol

HOW TO READ SCIENTIFIC DSP PUBLICATIONS AND TURN THEM INTO CODE

MAKING SENSE OF ACADEMIC PAPERS AND PATENTS FOR PEOPLE WHO ARE NOT ACADEMICS OR PATENT LAWYERS

MATTHIJS HOLLEMANS

Where to find papers

AES – Journal of the Audio Engineering Society
Journal of the Acoustical Society of America
MIT Computer Music Journal
DAFx – International Conference on Digital Audio Effects

arXiv.org
scholar.google.com
www.jstor.org

https://doi.org/10.2307/3680280

Papers

Library

1 € filter

DISCOVER

Search

Recommendations

PERSONAL LIBRARY

My Papers

Recently Read

Flagged

Tags

Reinforcement Learning

Biology

Computer Science

Functional Programming

Health

Languages and Compilers

Machine Intelligence

Mathematics

Music & Audio

Object-Oriented Programming

Scientific Software

Writing & Research

SHARED LIBRARIES

Create Library

IMPORT STATUS 307

Add

Search within music & audio...

Matched Second Order Digital Filters
Vicanek

Matched Two-Pole Digital Shelving Filters

Measurements of the vibrato rate of ten singers
Prame The Journal of the Acoustical Society of America 1994

Measures and parameter estimation of triodes, for the real-time simulation of a multi-stage guitar preamplifier

Mechanics of human voice production and control
Zhang The Journal of the Acoustical Society of America 2016

MelGAN: generative adversarial networks for conditional waveform synthesis

MelNet: A Generative Model for Audio in the Frequency Domain
Vasquez et al. arXiv 2019

Scanned synthesis can be looked upon as a descendent of wave table synthesis. In wave table synthesis, points in a function of one independent variable are computed and stored in successive memory locations in a computer. This chunk of memory (the wave table) is scanned or read by a periodic scanning function to produce the samples of the audio sound wave. The period of the scanning function is the period of the synthesized sound. The scanning process is computationally simple and efficient. The computation of the wave table need only be done once, and thus can be computationally intensive.

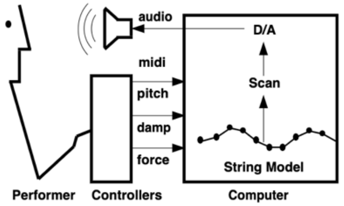


Figure 2 A performer uses a variety of controllers to send pitch to the scan rate and parameters (e.g. damping) and disturbances (e.g. force) to the model (generator).

A general block diagram of our model is shown in Fig 2. It is a real-time program which generates a sequence of samples that are read out of the computer at the audio sampling rate (typically 44100 samples per second) through a d-to-a converter and sent to a loud speaker. It also contains input channels through which the performer "plays" the model. The input channels (typically midi) are connected to physical controllers that the performer touches and manipulates. These can include midi keyboards, radio-batons, and Phantom sticks. We have also used a video camera as an input device.

The haptic generator in the model generates sampled spatial frequencies. The scanning path, becomes an array of numbers in the computer memory. This array can be looked upon as a dynamic wave table. These numbers are changed at haptic rates by the haptic generator. Thus the

In order to be useful, the numbers in the dynamic wave table must represent useful spatial frequencies. It is not sufficient that a number in given location in the array change in time at haptic frequency. An individual number must be related to its neighbors along the scanning path in a way which represents a desired spatial frequency. This spatial frequency is converted to a time frequency by the scanning function. This property is achieved by the choice of mathematical functions computed by the haptic generator.

4. ONE-DIMENSIONAL STRING MODEL

A useful model which generates spatial frequencies can be derived from the finite element approximation to a string. It can be thought of as a string of masses connected by springs. The equations of motions of the masses can be simply derived from Newton's equations. The resulting differential equations can be approximated by difference equations which can be solved by a computer as shown in Appendix A.

We have studied Scanned Synthesis chiefly with a finite element model of a generalized string. Our finite element models are a collection of masses connected by springs and dampers. We have generalized a traditional string by adding to each mass a damper and a spring connected to earth. All parameters -- mass, damping, earth spring strength and string tension -- can vary along the string. The performer manipulates the model by pushing or hitting different masses and by manipulating parameters.

Musical applications of finite element models were pioneered in the 1970's by Cadoz (1978, 1979, 1993) and his associates. Our work differs from that of Cadoz in that our models vibrate at low (haptic) frequencies and must be scanned to obtain audio frequencies. The models of Cadoz generally vibrated directly at audio frequencies. This difference is important. The performer can directly manipulate the motions of our haptic models. Also, slow models also require much computer power.

We have generalized the string model by:

1) allowing each element to have an arbitrary mass, M_i,

SCANNED SYNTHESIS

Write your note here...

Annotations

Type to filter annotations...

Order by: Scroll Position

Direction: Descending

2/8/2022

Scanned synthesis involves a slow dynamic system whose frequencies of vibration are below about 15 hz. The system is directly manipulated by motions of the performer. The vibrations of the system are a function of the initial conditions, the forces applied by the performer, and the dynamics of the system.

2/8/2022

The pitch is determined by the speed of the scanning function.

2/8/2022

The essence of scanned synthesis is to use a slowly vibrating object whose resonant frequencies are low enough so the performer can directly manipulate the object's vibrations by motions of his body and to scan (measure) the shape of the object along a periodic path by a periodic scanning function whose period is the fundamental frequency of the sound we wish to create.

2/14/2022

In order to be useful, the numbers in the dynamic wave table must represent useful spatial frequencies.

2/8/2022

It is not sufficient that a number in given location in the array change in time at haptic frequency. An individual number must be related to its neighbors along the scanning path in a way which represents a desired spatial frequency. This spatial frequency is converted to a time frequency by the scanning function.

2/14/2022

Uncomfortable facts about papers

Uncomfortable facts about papers

Papers are not written for laypeople

Uncomfortable facts about papers

Papers are not written for laypeople

Not all academics are good writers

Uncomfortable facts about papers

Papers are not written for laypeople

Not all academics are good writers

There will be mistakes!

$$\begin{aligned}\sum_{k=0}^{N-1} a^k \sin(\theta + k\beta) &= \frac{1}{1 \Leftrightarrow 2a\cos(\beta) + a^2} \times \\ [\sin(\theta) \Leftrightarrow a\sin(\theta \Leftrightarrow \beta) &\Leftrightarrow a^N \sin(\theta + N\beta) \\ &+ \; a^{N+1} \sin[\theta + (n \Leftrightarrow 1)\beta]\end{aligned}$$

The above closed-form expression is derived in a straightforward manner using the identity $2j\sin(x) = e^{jx} \Leftrightarrow e^{-jx}$ on the left-hand side, and applying the closed-form expression for a geometric series:

$$\sum_{k=0}^{N-1} z^k = \frac{1 \Leftrightarrow z^N}{1 \Leftrightarrow z}$$

Alias-Free Digital Synthesis of Classic Analog Waveforms (Stilson & Smith, 1996)

All of the above relationships are expressed in the trigonometric expansion of Eq. [2]

$$
\begin{aligned}
e = A\{\; & J_0(I)\ \sin\alpha t \\
& + J_1(I)\ [\sin(\alpha + \beta)t - \sin(\alpha - \beta)] \\
& + J_2(I)\ [\sin(\alpha + 2\beta)t + \sin(\alpha - 2\beta)] \\
& + J_3(I)\ [\sin(\alpha + 3\beta)t - \sin(\alpha - 3\beta)] \\
& + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \}. \quad (2)
\end{aligned}
$$

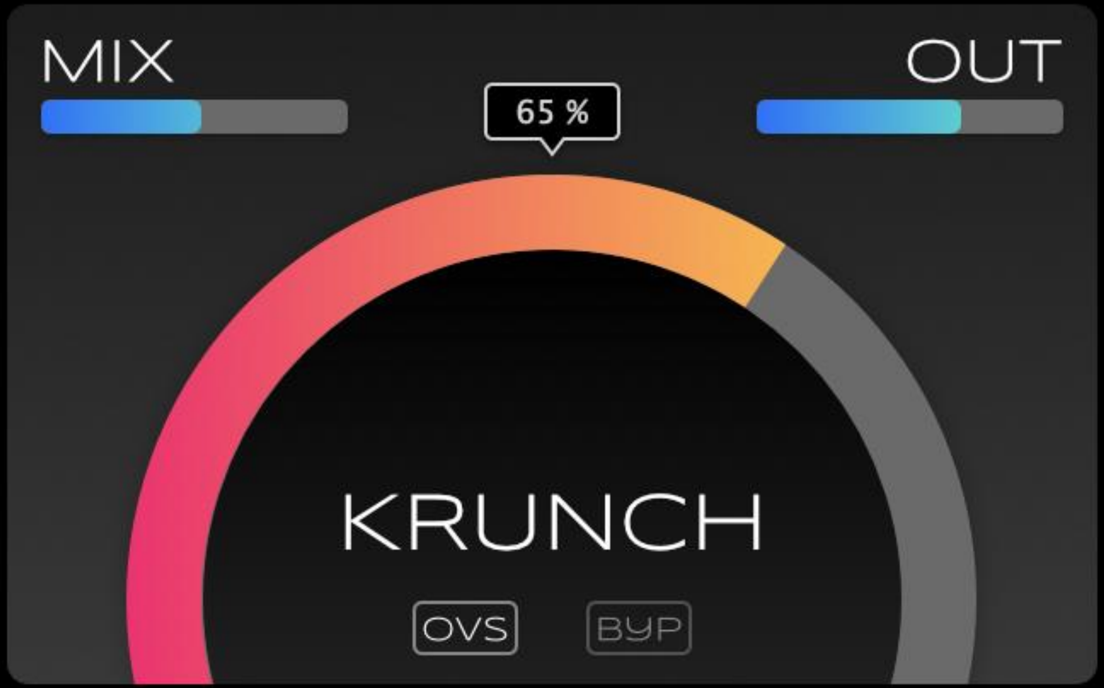
It can be seen in Eq. 2 that the odd-order lower-side frequencies, $\sin(\alpha\text{-}\beta)$, $\sin(\alpha\text{-}3\beta)$, etc., are preceded by a

All of the above relationships are expressed in Eq. 2 of the trigonometric expansion of Eq. 1.[2]

$$
\begin{aligned}
e = A\Big\{ & J_0(I)\sin\alpha t \\
& + J_1(I)\left[\sin(\alpha + \beta)t - \sin(\alpha - \beta)t\right] \\
& + J_2(I)\left[\sin(\alpha + 2\beta)t + \sin(\alpha - 2\beta)t\right] \\
& + J_3(I)\left[\sin(\alpha + 3\beta)t - \sin(\alpha - 3\beta)t\right] \\
& \ldots \\
& + J_n(I)\left[\sin(\alpha + n\beta)t \pm \sin(\alpha - n\beta t)\right]\Big\}
\end{aligned}
\qquad (2)
$$

It can be seen in Eq. 2 that the odd-order lower-side frequencies, sin(a-,S), sin(a-3~), etc., are preceded by a negative sign, and that

The Synthesis of Complex Audio Spectra by Means of Frequency Modulation (Chowning, 1973)



github.com/hollance/krunch

1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems

Géry Casiez[1,2,3], Nicolas Roussel[3] & Daniel Vogel[4]
[1]LIFL, [2]University of Lille & [3]Inria Lille, France
[4]Cheriton School of Computer Science, University of Waterloo, Canada
gery.casiez@lifl.fr, nicolas.roussel@inria.fr, dvogel@uwaterloo.ca

ABSTRACT
The 1€ filter ("one Euro filter") is a simple algorithm to filter noisy signals for high precision and responsiveness. It uses a first order low-pass filter with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to reduce lag. The algorithm is easy to implement, uses very few resources, and with two easily understood parameters, it is easy to tune. In a comparison with other filters, the 1€ filter has less lag using a reference amount of jitter reduction.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User interfaces - Input devices and strategies.

General Terms
Algorithms, Performance, Human Factors

Author Keywords
Signal; noise; jitter; precision; lag; responsiveness; filtering

INTRODUCTION
Noisy signals occur when an original time varying value undergoes undesirable and unpredictable perturbations. These may be caused by things like heat and magnetic fields affecting hardware circuitry, the limits of sensor resolution, or even unstable numerical computation. Noisy signals are a common problem when tracking human motion, particularly with custom sensing hardware and inexpensive input devices like the Kinect or Wiimote. In addition, even signals from established high-end sensing systems can become noisy when interaction techniques use large scaling effects. A common example is using a Vicon tracking system to implement ray casting with a wall display [6]: calibration problems and hand tremor add further perturbations to the ones amplified by the pointing technique.

Noise affects the quality of a signal in two primary ways [9]. It can reduce *accuracy*, by adding an *offset* between the observed values and the true ones. More often, it reduces *precision*, where repeated observations of values exhibit *jitter* – many different values are observed for a single true one. Jitter has a large effect on the way people perceive and act. For example noisy values are harder to read and unstable cursors hinder target acquisition [3, 7, 5]. One usually wants to filter noisy signals to reduce, and possibly remove, the unwanted parts of the signal. However, filtering inherently introduces time latency – commonly called *lag* – which reduces system *responsiveness*. Lag may not be an issue in domains like artificial perception and decision making, but with interactive feedback, it is very important. In fact, it is the combination of precision and responsiveness that are crucial: people can point accurately in spite of an offset, but only with minimal lag and jitter. The difficulty is that implementing and tuning a filter to minimize both jitter and lag is challenging, especially with little or no background in signal processing.

In this paper we describe the 1€ filter ("one Euro filter"), a tool to improve noisy signal quality with a tunable jitter and lag balance. It uses a low-pass filter, but the cutoff frequency changes according to speed: at low speeds, a low cutoff reduces jitter at the expense of lag, but at high speeds, the cutoff is increased to reduce lag rather than jitter. The intuition is that people are very sensitive to jitter and not latency when moving slowly, but as movement speed increases, people become very sensitive to latency and not jitter. We compare the 1€ filter to alternative techniques and show how it can reduce that same amount of jitter with less lag. It is also efficient and easy to understand, implement, and tune: the algorithm can be expressed in a few lines; it uses only basic arithmetic; and it has only two independent parameters that relate directly to jitter and lag. Other researchers and ourselves have already used variations of it in many projects. In fact, the "dynamic recursive low-pass filter" used by the third author in [6] established the basic principle, but it required four parameters and a fixed sample rate. The '1€' name is an homage to the $1 recognizer [10]: we believe that the 1€ filter can make filtering input signals simpler and better, much like the $1 recognizer did for gestures.

After a review of the jitter, lag, and alternative filtering techniques, we describe the 1€ filter in detail with an implementation, discuss tuning with different applications, and conclude with an illustrative comparison.

JITTER, LAG, AND FILTERING
Several studies show jitter and lag have a negative impact on performance. MacKenzie et al. found mouse movement times increased 16% with 75 ms lag, and up to 64% with 225 ms lag [3]. With 3D hand tracking, Ware and Balakrishnan found that only 50 ms lag reduced performance by more than 8% [7]. Pavlovych and Stuerzlinger found no performance degradation below 58 ms lag using a mouse or Wiimote, but increasing jitter from 4 to 8 pixels doubled error rates for small targets [5]. Assuming a 100 PPI screen, 4 pixels corresponds to 1mm of jitter mean-to-peak: close to the 0.4 mm of jitter they found with the established Optitrack system.

Although the precision of an input device may be very good, it does not take into account scaling effects introduced by interaction techniques. Device input is often scaled up, so people can cover more display distance with less device movement. For example, default operating system mouse transfer functions can be scaled up 12× [1] and factors as high as 90× have been used when ray casting on wall sized displays [6]. Regardless of native device precision, scaling amplifies even small sensing perturbations, increasing jitter.

These results highlight the importance of balancing jitter and lag. Jitter should be less than 1mm mean-to-peak, but lag should be below 60 ms. As we shall see, any filter introduces some lag and considering 40-50 ms of inherent system lag [5], that leaves less than 10-20 ms for the filter.

Moving average
By the Central Limit Theorem and reasonable assumptions, averaging enough successive values of a noisy signal should produce a better estimate of the true one [9]. As a result, a *moving average* of the last n data values is commonly used by computer scientists as a kind of filter. For example, Myers et al. [4] used one for laser pointers and reduced hand tremor jitter from ±8 pixels to between ±2 and ±4 pixels using a 0.5s window (n = 10). Since all n values are weighted equally, this creates a lag up to n times the sampling period.

Low-pass filters and exponential smoothing
With human movements, noise typically forms high frequencies in the signal while actual limb movements have lower frequencies. A low-pass filter is designed to let these desired low frequency portions pass through, while attenuating high frequency signals above a fixed cutoff frequency. The *order* of a low-pass filter relates to how aggressively it attenuates each frequency: first order filters reduce the signal amplitude by half every time the frequency doubles, while higher order variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i = \alpha X_i + (1-\alpha)\hat{X}_{i-1} \quad (1)$$

The first term of the equation is the contribution of new input data value, and the second term adds inertia from previous values. As α decreases, jitter is reduced, but lag increases since the output responds more slowly to changes in input. Since the contribution of older values exponentially decreases, a low-pass filter will have less lag than a high n moving average filter.

Smoothing techniques used in business and economic forecasts are similar in approach to a low-pass filter. The equation for *single exponential smoothing* is very similar to Equation 1. As the name suggests, *double exponential smoothing* uses two of these equations to handle trends in the signal. Although not formally documented, the Microsoft Kinect skeleton filters appear to be a variant of this type of smoothing[1]. LaViola extended double exponential smoothing for predictive tracking [2], building on Equations 1 and 2 to predict positions τ time steps in the future (Equation 3):

$$\hat{X}_i^{[2]} = \alpha\hat{X}_i + (1-\alpha)\hat{X}_{i-1}^{[2]} \quad (2)$$

$$P_{i+\tau} = \left(2 + \frac{\alpha\tau}{1-\alpha}\right)\hat{X}_i - \left(1 + \frac{\alpha\tau}{1-\alpha}\right)\hat{X}_i^{[2]} \quad (3)$$

Kalman filters
Unlike the techniques above, Kalman filters make assumptions about the system generating the signal. Typically used for navigation and tracking, they work well when combining data from different sensors (e.g. a GPS and a speedometer) or when the system can be modeled by equations (e.g. determining vehicle acceleration from accelerator pedal position). Kalman filters rely on a *process model* and a *measurement model*. The standard Kalman filter uses a discrete-time linear stochastic difference equation for the process model and assumes that process and measurement noise are independent of each other, white, and are normally distributed [8]. When estimating the true position of a moving object, the process model is typically a linear function of the speed and the previous estimated position. With additional complexity, Extended and Unscented variants of Kalman filters can also model non-linear processes and observations [8].

In the frequent case where the process and measurement noise covariances are not known, one must determine them empirically. This task can be challenging, and an improperly tuned filter can increase and even degrade the signal [9], by creating artificial "overshooting" movements for example. Moreover, understanding Kalman filters requires mathematical knowledge beyond basic linear algebra such as statistics, random signals, and stochastic methods. Implementing them requires a language or library with matrix operations. And, as demonstrated by LaViola for predictive tracking, they can be considerably slower to compute than double exponential smoothing predictors (approximately 135×) with similar jitter and lag performance [2].

THE 1€ FILTER
The 1€ filter is an adaptive first-order low-pass filter: it adapts the cutoff frequency of a low-pass filter for each new sample according to an estimate of the signal's speed, or more generally, its derivative value. Even though noisy signals are often sampled at a fixed frequency, filtering can not always follow the same pace, especially in event-driven systems. To accommodate possible fluctuations, we rewrite equation 1 to take into account the actual time interval between samples. Using a direct analogy with an electrical circuit, where a resistor in series with a capacitor defines a first order low-pass filter, α can be computed as a function of the sampling period T_e and a time constant τ, both expressed in seconds (Equation 4). The resistor and capacitor values define the time constant ($\tau = RC$) and the corresponding cutoff frequency f_c, in Hertz, of the circuit (Equation 5).

$$\alpha = \frac{1}{1+\frac{\tau}{T_e}} \quad (4)$$

$$\tau = \frac{1}{2\pi f_c} \quad (5)$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1+\frac{\tau}{T_e}} \quad (6)$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \quad (7)$$

The sampling period T_e (or its inverse, the sampling rate) can be automatically computed from timestamps, so the cutoff frequency f_c is the only configurable parameter in equation 6. As with any low-pass filter, decreasing f_c reduces jitter, but increases lag. Finding a good trade-off between the two is difficult since people are more sensitive to jitter at low speeds, and more sensitive to lag at high speeds. This is why an adaptive cutoff frequency works well. To reduce jitter, a low f_c is used at low signal speeds, and to reduce lag, f_c is increased as speed increases. We found that a straightforward linear relationship between cutoff frequency f_c and the absolute speed works well (Equation 7). The speed (i.e the derivative $\dot{\hat{X}}_i$) is computed from raw signal values using the sampling rate and then low-pass filtered with a cutoff frequency chosen to avoid high derivative bursts caused by jitter. Our implementation uses a fixed value of 1 Hz, leaving only two configurable parameters: the intercept $f_{c_{min}}$ and the slope β shown in Equation 7. Details of the algorithm are provided in the Appendix.

Tuning and Applications
To minimize jitter and lag when tracking human motion, the two parameters can be set using a simple two-step procedure. First β is set to 0 and $f_{c_{min}}$ to a reasonable middle-ground value such as 1 Hz. Then the body part is held steady or moved at a very low speed while $f_{c_{min}}$ is adjusted to remove jitter and preserve an acceptable lag during these slow movements. Next, the body part is moved quickly in different directions while β is increased with a focus on minimizing lag. Note that parameters $f_{c_{min}}$ and β have clear conceptual relationships: if high speed lag is a problem, increase β; if slow speed jitter is a problem, decrease $f_{c_{min}}$. Rotational input uses a similar tuning process, but rotation axis and angle are filtered separately.

Another application of the 1€ filter is displaying noisy numerical values, such as an unsteady frame rate used to monitor graphical application performance. The goal is to reduce jitter to make the numerical output legible while minimizing lag so the value remains timely. Tuning is similar to above: adjust $f_{c_{min}}$ until the text becomes stable, then increase β until just before the text become unstable.

COMPARISON WITH OTHER FILTERS
To compare the 1€ filter with other techniques, we created a Python application that periodically samples the XY position of the system cursor, adds noise, and displays filtered cursor positions. Each filter can be tuned interactively and all filters can be shown simultaneously making it possible to visually compare jitter reduction and lag across parameter settings and filters. Once tuned, timestamped positions can be logged for the system cursor (with and without noise) and filtered positions of all filters. We used a MacBook Pro with a 1440 × 900 pixel display (109 PPI).

In our comparison, we used independent Gaussian white noises for X and Y with a 50 dB SNR[2], a public implementation of the Kalman filter[3], and custom implementations of a moving average, single exponential, and LaViola's double exponential smoothing. We tuned moving average first and used its performance as a baseline. We found that averaging more than 14 data values did not reduce jitter further and only increased lag, so we used n=14. Then we interactively tuned the other filters to primarily match the jitter reduction of moving average, and secondarily attempting to reduce lag.

Tuning single exponential smoothing to match the reference jitter requires a low alpha value (α=0.11) which introduces lag. This highlights the difficulty of tuning with only a single parameter. For LaViola's double exponential smoothing filter, the reference jitter is obtained with a lower alpha value (α=0.06) and with lower lag. However, this causes overshooting when the pointer abruptly decelerates. For the Kalman filter, we set the measurement noise covariance to the variance of the introduced noise (18.06) as in [2], and adjusted the process noise covariance until we obtained the reference jitter reduction (at a value of 0.3). The amount of lag for this setting was comparable to the moving average and single-exponential. For the 1€ filter, we matched the reference jitter and optimized lag using the tuning procedure described above. In the first tuning step, setting $f_{c_{min}}$ = 1 Hz and β = 0 matched the reference jitter and lag was similar to single exponential smoothing. In the second tuning step, increasing β to 0.007 made the lag almost imperceivable yet maintained the reference jitter when stationary or moving slowly. A supplementary video demonstrates this tuning process and visualizes filter performance.

For a quantitative comparison, we logged the system cursor at 60 Hz for about 1 hour during regular desktop use, then added white noise and applied the filters using the settings above. Figure 1 shows the distance from each filtered cursor position to the true one, binned into four speed intervals. Note that since we tuned the filters to match a reference jitter when not moving, the error between filtered position and noiseless position is primarily due to lag when moving. With higher speeds, the filtered position lags farther and farther behind, increasing this distance (the small distances in the 0 mm/s interval are likely due to offset or overshooting). All filters introduce a similar amount of lag except for the 1€ filter which has less lag across all speed intervals.

As an overall comparison, we computed the Standard Error of the Mean (SEM) in mm for each filter for this data set. The 1€ filter has the smallest SEM (0.004) followed by LaViola's double exponential smoothing (0.013), the moving average and the Kalman filter (0.015), and single exponential smoothing (0.016). Our intention for this evaluation is to illustrate the performance of the 1€ filter in an intuitive way under realistic conditions. We are exploring alternative comparisons with user experiments, synthetic reference movements, different noise configurations, and examples of "noisy" hardware.

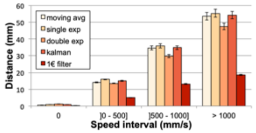


Figure 1. Mean distance between filtered and true cursor position for each speed interval and filter. Error bars represent 95% CI.

CONCLUSION
Human-Computer Interaction researchers and practitioners should stop filtering noisy input with a moving average. In most cases, they do not need to wrestle with low-level signal processing issues or with more complex techniques like Kalman filtering – which can be difficult to understand, tune, and implement. The 1€ filter is an intuitive and practical alternative since it is easy to understand, implement, and tune for low jitter and lag. Best of all, it produces better results.

REFERENCES
1. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. of UIST'11*, ACM (2011), 603–614.
2. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. of EGVE '03*, ACM (2003), 199–206.
3. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proc. of CHI '93*, CHI '93, ACM (1993), 488–493.
4. Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proc. of CHI '02*, 33–40.
5. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proc. of EICS '09*, ACM (2009), 187–196.
6. Vogel, D., and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution displays. In *Proc. of UIST '05*, ACM (2005), 33–42.
7. Ware, C., and Balakrishnan, R. Reaching for objects in vr displays: lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.
8. Welch, G., and Bishop, G. An introduction to the Kalman filter. SIGGRAPH course, ACM, Aug. 2001.
9. Wilson, A. D. Sensor- and recognition-based input for interaction. In *The Human Computer Interaction handbook*. CRC Press, 2007, 177–199.
10. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*, ACM (2007), 159–168.

APPENDIX A - 1€ FILTER

Algorithm 1: 1€ filter

```
EXT : First time flag: firstTime set to true
      Data update rate: rate
      Minimum cutoff frequency: mincutoff
      Cutoff slope: beta
      Low-pass filter: xfilt
      Cutoff frequency for derivate: dcutoff
      Low-pass filter for derivate: dxfilt
IN  : Noisy sample value: x
OUT : Filtered sample value

if firstTime then
    firstTime ← false
    dx ← 0
else
    dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 2: *Filter* method of Low-pass filter

```
EXT : First time flag: firstTime set to true
IN  : Noisy sample value : x
      Alpha value : alpha
OUT : Filtered value

if firstTime then
    firstTime ← false
    hatxprev ← x
end
hatx ← alpha * x + (1 - alpha) * hatxprev
hatxprev ← hatx
return hatx
```

Algorithm 3: Alpha computation

```
IN  : Data update rate in Hz: rate
      Cutoff frequency in Hz: cutoff
OUT : Alpha value for low-pass filter

tau ← 1.0 / (2*π*cutoff)
te ← 1.0 / rate
return 1.0 / (1.0 + tau/te)
```

[1]http://cm-bloggers.blogspot.com/2011/07/kinect-sdk-smoothing-skeleton-data.html

[2]This signal-to-noise ratio was estimated from Gametrak data using a zero phase shift filter and is consistent with numbers in [2]
[3]http://greg.czerniak.info/node/5

1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems

Géry Casiez[1,2,3], Nicolas Roussel[3] & Daniel Vogel[4]

[1]LIFL, [2]University of Lille & [3]Inria Lille, France
[4]Cheriton School of Computer Science, University of Waterloo, Canada
gery.casiez@lifl.fr, nicolas.roussel@inria.fr, dvogel@uwaterloo.ca

ABSTRACT

The 1€ filter ("one Euro filter") is a simple algorithm to filter noisy signals for high precision and responsiveness. It uses a first order low-pass filter with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to reduce lag. The algorithm is easy to implement, uses very few resources, and with two easily understood parameters, it is easy to tune. In a comparison with other filters, the 1€ filter has less lag using a reference amount of jitter reduction.

ACM Classification Keywords

H.5.2 [Information interfaces and presentation]: User interfaces - Input devices and strategies.

ABSTRACT
The 1€ filter ("one Euro filter") is a simple algorithm to filter noisy signals for high precision and responsiveness. It uses a first order low-pass filter with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to reduce lag. The algorithm is easy to implement, uses very few resources, and with two easily understood parameters, it is easy to tune. In a comparison with other filters, the 1€ filter has less lag using a reference amount of jitter reduction.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User interfaces - Input devices and strategies.

General Terms
Algorithms, Performance, Human Factors

Author Keywords
Signal; noise; jitter; precision; lag; responsiveness; filtering

INTRODUCTION
Noisy signals occur when an original time varying value undergoes undesirable and unpredictable perturbations. These may be caused by things like heat and magnetic fields affecting hardware circuitry, the limits of sensor resolution, or even unstable numerical computation. Noisy signals are a common problem when tracking human motion, particularly with custom sensing hardware and inexpensive input devices like the Kinect or Wiimote. In addition, even signals from established high-end sensing systems can become noisy when interaction techniques use large scaling effects. A common example is using a Vicon tracking system to implement ray casting with a wall display [6]: calibration problems and hand tremor add further perturbations to the ones amplified by the pointing technique.

Noise affects the quality of a signal in two primary ways [9]. It can reduce *accuracy*, by adding an *offset* between the observed values and the true ones. More often, it reduces *precision*, where repeated observations of values exhibit *jitter* – many different values are observed for a single true one. Jitter has a large effect on the way people perceive and act. For example noisy values are harder to read and unstable cursors hinder target acquisition [3, 7, 5]. One usually wants to filter noisy signals to reduce, and possibly remove, the unwanted parts of the signal. However, filtering inherently introduces time latency – commonly called *lag* – which reduces system *responsiveness*. Lag may not be an issue in domains like artificial perception and decision making, but with interactive feedback, it is very important. In fact, it is the combination of precision and responsiveness that are crucial: people can point accurately in spite of an offset, but only with minimal lag and jitter. The difficulty is that implementing and tuning a filter to minimize both jitter and lag is challenging, especially with little or no background in signal processing.

In this paper we describe the 1€ filter ("one Euro filter"), a tool to improve noisy signal quality with a tunable jitter and lag balance. It uses a low-pass filter, but the cutoff frequency changes according to speed: at low speeds, a low cutoff reduces jitter at the expense of lag, but at high speeds, the cutoff is increased to reduce lag rather than jitter. The intuition is that people are very sensitive to jitter and not latency when moving slowly, but as movement speed increases, people become very sensitive to latency and not jitter. We compare the 1€ filter to alternative techniques and show how it can reduce that same amount of jitter with less lag. It is also efficient and easy to understand, implement, and tune: the algorithm can be expressed in a few lines; it uses only basic arithmetic; and it has only two independent parameters that relate directly to jitter and lag. Other researchers and ourselves have already used variations of it in many projects. In fact, the "dynamic recursive low-pass filter" used by the third author in [6] established the basic principle, but it required four parameters and a fixed sample rate. The '1€' name is an homage to the $1 recognizer [10]: we believe that the 1€ filter can make filtering input signals simpler and better, much like the $1 recognizer did for gestures.

After a review of the jitter, lag, and alternative filtering techniques, we describe the 1€ filter in detail with an implementation, discuss tuning with different applications, and conclude with an illustrative comparison.

JITTER, LAG, AND FILTERING
Several studies show jitter and lag have a negative impact on performance. MacKenzie et al. found mouse movement times increased 16% with 75 ms lag, and up to 64% with 225

CHI'12, May 5–10, 2012, Austin, Texas, USA.

ms lag [3]. With 3D hand tracking, Ware and Balakrishnan found that only 50 ms lag reduced performance by more than 8% [7]. Pavlovych and Stuerzlinger found no performance degradation below 58 ms lag using a mouse or Wiimote, but increasing jitter from 4 to 8 pixels doubled error rates for small targets [5]. Assuming a 100 PPI screen, 4 pixels corresponds to 1mm of jitter mean-to-peak: close to the 0.4 mm of jitter they found with the established Optitrack system.

Although the precision of an input device may be very good, it does not take into account scaling effects introduced by interaction techniques. Device input is often scaled up, so people can cover more display distance with less device movement. For example, default operating system mouse transfer functions can be scaled up 12× [1] and factors as high as 90× have been used when ray casting on wall sized displays [6]. Regardless of native device precision, scaling amplifies even small sensing perturbations, increasing jitter.

These results highlight the importance of balancing jitter and lag. Jitter should be less than 1mm mean-to-peak, but lag should be below 60 ms. As we shall see, any filter introduces some lag and considering 40-50 ms of inherent system lag [5], that leaves less than 10-20 ms for the filter.

Moving average

By the Central Limit Theorem and reasonable assumptions, averaging enough successive values of a noisy signal should produce a better estimate of the true one [9]. As a result, a *moving average* of the last n data values is commonly used by computer scientists as a kind of filter. For example, Myers et al. [4] used one for laser pointers and reduced hand tremor jitter from ± 8 pixels to between ± 2 and ± 4 pixels using a 0.5s window ($n = 10$). Since all n values are weighted equally, this creates a lag up to n times the sampling period.

Low-pass filters and exponential smoothing

With human movements, noise typically forms high frequencies in the signal while actual limb movements have lower frequencies. A low-pass filter is designed to let these desired low frequency portions pass through, while attenuating high frequency signals above a fixed cutoff frequency. The *order* of a low-pass filter relates to how aggressively it attenuates each frequency: first order filters reduce the signal amplitude by half every time the frequency doubles, while higher order variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i = \alpha X_i + (1 - \alpha)\hat{X}_{i-1} \tag{1}$$

The first term of the equation is the contribution of new input data value, and the second term adds inertia from previous values. As α decreases, jitter is reduced, but lag increases since the output responds more slowly to changes in input. Since the contribution of older values exponentially decreases, a low-pass filter will have less lag than a high n moving average filter.

Smoothing techniques used in business and economic forecasts are similar in approach to a low-pass filter. The equation for *single exponential smoothing* is very similar to Equation 1. As the name suggests, *double exponential smoothing* uses two of these equations to handle trends in the signal. Although not formally documented, the Microsoft Kinect skeleton filters appear to be a variant of this type of smoothing[1]. LaViola extended double exponential smoothing for predictive tracking [2], building on Equations 1 and 2 to predict positions τ time steps in the future (Equation 3):

$$\hat{X}_i^{[2]} = \alpha \hat{X}_i + (1 - \alpha)\hat{X}_{i-1}^{[2]} \tag{2}$$

$$P_{t+\tau} = \left(2 + \frac{\alpha\tau}{1 - \alpha}\right)\hat{X}_i - \left(1 + \frac{\alpha\tau}{1 - \alpha}\right)\hat{X}_i^{[2]} \tag{3}$$

Kalman filters

Unlike the techniques above, Kalman filters make assumptions about the system generating the signal. Typically used for navigation and tracking, they work well when combining data from different sensors (e.g. a GPS and a speedometer) or when the system can be modeled by equations (e.g. determining vehicle acceleration from accelerator pedal position). Kalman filters rely on a *process model* and a *measurement model*. The standard Kalman filter uses a discrete-time linear stochastic difference equation for the process model and assumes that process and measurement noise are independent of each other, white, and are normally distributed [8]. When estimating the true position of a moving object, the process model is typically a linear function of the speed and the previous estimated position. With additional complexity, Extended and Unscented variants of Kalman filters can also model non-linear processes and observations [8].

In the frequent case where the process and measurement noise covariances are not known, one must determine them empirically. This task can be challenging, and an improperly tuned filter can increase and even degrade the signal [9], by creating artificial "overshooting" movements for example. Moreover, understanding Kalman filters requires mathematical knowledge beyond basic linear algebra such as statistics, random signals, and stochastic methods. Implementing them requires a language or library with matrix operations. And, as demonstrated by LaViola for predictive tracking, they can be considerably slower to compute than double exponential smoothing predictors (approximately 135×) with similar jitter and lag performance [2].

THE 1€ FILTER

The 1€ filter is an adaptive first-order low-pass filter: it adapts the cutoff frequency of a low-pass filter for each new sample according to an estimate of the signal's speed, or more generally, its derivative value. Even though noisy signals are often sampled at a fixed frequency, filtering can not always follow the same pace, especially in event-driven systems. To accommodate possible fluctuations, we rewrite equation 1 to take into account the actual time interval between samples. Using a direct analogy with an electrical circuit, where a resistor in series with a capacitor defines a first order low-pass filter, α can be computed as a function of the sampling period T_e and a time constant τ, both expressed

[1] http://cm-bloggers.blogspot.com/2011/07/kinect-sdk-smoothing-skeleton-data.html

equation for *single exponential smoothing* is very similar to Equation 1. As the name suggests, *double exponential smoothing* uses two of these equations to handle trends in the signal. Although not formally documented, the Microsoft Kinect skeleton filters appear to be a variant of this type of smoothing[1]. LaViola extended double exponential smoothing for predictive tracking [2], building on Equations 1 and 2 to predict positions τ time steps in the future (Equation 3):

$$\hat{X}_i^{[2]} = \alpha \hat{X}_i + (1-\alpha)\hat{X}_{i-1}^{[2]} \tag{2}$$

$$P_{t+\tau} = \left(2 + \frac{\alpha\tau}{1-\alpha}\right)\hat{X}_i - \left(1 + \frac{\alpha\tau}{1-\alpha}\right)\hat{X}_i^{[2]} \tag{3}$$

Kalman filters

Unlike the techniques above, Kalman filters make assumptions about the system generating the signal. Typically used for navigation and tracking, they work well when combining data from different sensors (e.g. a GPS and a speedometer) or when the system can be modeled by equations (e.g. determining vehicle acceleration from accelerator pedal position). Kalman filters rely on a *process model* and a *measurement model*. The standard Kalman filter uses a discrete-time linear stochastic difference equation for the process model and assumes that process and measurement noise are independent of each other, white, and are normally distributed [8]. When estimating the true position of a moving object, the process model is typically a linear function of the speed and the previous estimated position. With additional complexity, Extended and Unscented variants of Kalman filters can also model non-linear processes and observations [8].

In the frequent case where the process and measurement noise covariances are not known, one must determine them empirically. This task can be challenging, and an improperly tuned filter can increase and even degrade the signal [9], by creating artificial "overshooting" movements for example. Moreover, understanding Kalman filters requires mathematical knowledge beyond basic linear algebra such as statistics, random signals, and stochastic methods. Implementing them requires a language or library with matrix operations. And, as demonstrated by LaViola for predictive tracking, they can be considerably slower to compute than double exponential smoothing predictors (approximately 135×) with similar jitter and lag performance [2].

THE 1€ FILTER

The 1€ filter is an adaptive first-order low-pass filter: it adapts the cutoff frequency of a low-pass filter for each new sample according to an estimate of the signal's speed, or more generally, its derivative value. Even though noisy signals are often sampled at a fixed frequency, filtering can not always follow the same pace, especially in event-driven systems. To accommodate possible fluctuations, we rewrite equation 1 to take into account the actual time interval between samples. Using a direct analogy with an electrical circuit, where a resistor in series with a capacitor defines a first order low-pass filter, α can be computed as a function of the sampling period T_e and a time constant τ, both expressed in seconds (Equation 4). The resistor and capacitor values define the time constant ($\tau = RC$) and the corresponding cutoff frequency f_c, in Hertz, of the circuit (Equation 5).

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{4}$$

$$\tau = \frac{1}{2\pi f_c} \tag{5}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1 + \frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \tag{7}$$

The sampling period T_e (or its inverse, the sampling rate) can be automatically computed from timestamps, so the cutoff frequency f_c is the only configurable parameter in equation 6. As with any low-pass filter, decreasing f_c reduces jitter, but increases lag. Finding a good trade-off between the two is difficult since people are more sensitive to jitter at low speeds, and more sensitive to lag at high speeds. This is why an adaptive cutoff frequency works well. To reduce jitter, a low f_c is used at low signal speeds, and to reduce lag, f_c is increased as speed increases. We found that a straightforward linear relationship between cutoff frequency f_c and the absolute speed works well (Equation 7). The speed (i.e the derivative $\dot{\hat{X}}_i$) is computed from raw signal values using the sampling rate and then low-pass filtered with a cutoff frequency chosen to avoid high derivative bursts caused by jitter. Our implementation uses a fixed value of 1 Hz, leaving only two configurable parameters: the intercept $f_{c_{min}}$ and the slope β shown in Equation 7. Details of the algorithm are provided in the Appendix.

Tuning and Applications

To minimize jitter and lag when tracking human motion, the two parameters can be set using a simple two-step procedure. First β is set to 0 and $f_{c_{min}}$ to a reasonable middle-ground value such as 1 Hz. Then the body part is held steady or moved at a very low speed while $f_{c_{min}}$ is adjusted to remove jitter and preserve an acceptable lag during these slow movements. Next, the body part is moved quickly in different directions while β is increased with a focus on minimizing lag. Note that parameters $f_{c_{min}}$ and β have clear conceptual relationships: if high speed lag is a problem, increase β; if slow speed jitter is a problem, decrease $f_{c_{min}}$. Rotational input uses a similar tuning process, but rotation axis and angle are filtered separately.

Another application of the 1€ filter is displaying noisy numerical values, such as an unsteady frame rate used to monitor graphical application performance. The goal is to reduce jitter to make the numerical output legible while minimizing lag so the value remains timely. Tuning is similar to above: adjust $f_{c_{min}}$ until the text becomes stable, then increase β until just before the text become unstable.

COMPARISON WITH OTHER FILTERS

To compare the 1€ filter with other techniques, we created a Python application that periodically samples the XY position of the system cursor, adds noise, and displays filtered in seconds (Equation 4). The resistor and capacitor values define the time constant ($\tau = RC$) and the corresponding cutoff frequency f_c, in Hertz, of the circuit (Equation 5).

$$\alpha = \frac{1}{1+\frac{\tau}{T_e}} \tag{4}$$

$$\tau = \frac{1}{2\pi f_c} \tag{5}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1+\frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \tag{7}$$

The sampling period T_e (or its inverse, the sampling rate) can be automatically computed from timestamps, so the cutoff frequency f_c is the only configurable parameter in equation 6. As with any low-pass filter, decreasing f_c reduces jitter, but increases lag. Finding a good trade-off between the two is difficult since people are more sensitive to jitter at low speeds, and more sensitive to lag at high speeds. This is why an adaptive cutoff frequency works well. To reduce jitter, a low f_c is used at low signal speeds, and to reduce lag, f_c is increased as speed increases. We found that a straightforward linear relationship between cutoff frequency f_c and the absolute speed works well (Equation 7). The speed (i.e the derivative $\dot{\hat{X}}_i$) is computed from raw signal values using the sampling rate and then low-pass filtered with a cutoff frequency chosen to avoid high derivative bursts caused by jitter. Our implementation uses a fixed value of 1 Hz, leaving only two configurable parameters: the intercept $f_{c_{min}}$ and the slope β shown in Equation 7. Details of the algorithm are provided in the Appendix.

Tuning and Applications

To minimize jitter and lag when tracking human motion, the two parameters can be set using a simple two-step procedure. First β is set to 0 and $f_{c_{min}}$ to a reasonable middle-ground value such as 1 Hz. Then the body part is held steady or moved at a very low speed while $f_{c_{min}}$ is adjusted to remove jitter and preserve an acceptable lag during these slow movements. Next, the body part is moved quickly in different directions while β is increased with a focus on minimizing lag. Note that parameters $f_{c_{min}}$ and β have clear conceptual relationships: if high speed lag is a problem, increase β; if slow speed jitter is a problem, decrease $f_{c_{min}}$. Rotational input uses a similar tuning process, but rotation axis and angle are filtered separately.

Another application of the 1€ filter is displaying noisy numerical values, such as an unsteady frame rate used to monitor graphical application performance. The goal is to reduce jitter to make the numerical output legible while minimizing lag so the value remains timely. Tuning is similar to above: adjust $f_{c_{min}}$ until the text becomes stable, then increase β until just before the text become unstable.

COMPARISON WITH OTHER FILTERS

To compare the 1€ filter with other techniques, we created a Python application that periodically samples the XY position of the system cursor, adds noise, and displays filtered cursor positions. Each filter can be tuned interactively and all filters can be shown simultaneously making it possible to visually compare jitter reduction and lag across parameter settings and filters. Once tuned, timestamped positions can be logged for the system cursor (with and without noise) and filtered positions of all filters. We used a MacBook Pro with a 1440 × 900 pixel display (109 PPI).

In our comparison, we used independent Gaussian white noises for X and Y with a 50 dB SNR[2], a public implementation of the Kalman filter[3], and custom implementations of a moving average, single exponential, and LaViola's double exponential smoothing. We tuned moving average first and used its performance as a baseline. We found that averaging more than 14 data values did not reduce jitter further and only increased lag, so we used n=14. Then we interactively tuned the other filters to primarily match the jitter reduction of moving average, and secondarily attempting to reduce lag.

Tuning single exponential smoothing to match the reference jitter requires a low alpha value (α=0.11) which introduces lag. This highlights the difficulty of tuning with only a single parameter. For LaViola's double exponential smoothing filter, the reference jitter is obtained with a lower alpha value (α=0.06) and with lower lag. However, this causes overshooting when the pointer abruptly decelerates. For the Kalman filter, we set the measurement noise covariance to the variance of the introduced noise (18.06) as in [2], and adjusted the process noise covariance until we obtained the reference jitter reduction (at a value of 0.3). The amount of lag for this setting was comparable to the moving average and single-exponential. For the 1€ filter, we matched the reference jitter and optimized lag using the tuning procedure described above. In the first tuning step, setting $f_{c_{min}}$ = 1 Hz and β = 0 matched the reference jitter and lag was similar to single exponential smoothing. In the second tuning step, increasing β to 0.007 made the lag almost imperceivable yet maintained the reference jitter when stationary or moving slowly. A supplementary video demonstrates this tuning process and visualizes filter performance.

For a quantitative comparison, we logged the system cursor at 60 Hz for about 1 hour during regular desktop use, then added white noise and applied the filters using the settings above. Figure 1 shows the distance from each filtered cursor position to the true one, binned into four speed intervals. Note that since we tuned the filters to match a reference jitter when not moving, the error between filtered position and noiseless position is primarily due to lag when moving. With higher speeds, the filtered position lags farther and farther behind, increasing this distance (the small distances in the 0 mm/s interval are likely due to offset or overshooting). All filters introduce a similar amount of lag except for the 1€ filter which has less lag across all speed intervals.

As an overall comparison, we computed the Standard Error of the Mean (SEM) in mm for each filter for this data set. The 1€ filter has the smallest SEM (0.004) followed by LaViola's double exponential smoothing (0.013), the moving average and the Kalman filter (0.015), and single exponential smoothing (0.016). Our intention for this evaluation is to illustrate the performance of the 1€ filter in an intuitive way under realistic conditions. We are exploring alternative comparisons with user experiments, synthetic reference movements, different noise configurations, and examples of "noisy" hardware.

[2]This signal-to-noise ratio was estimated from Gametrak data using a zero phase shift filter and is consistent with numbers in [2]

[3]http://greg.czerniak.info/node/5

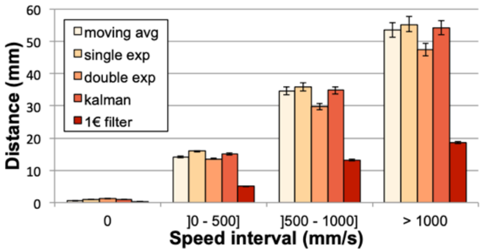


Figure 1. Mean distance between filtered and true cursor position for each speed interval and filter. Error bars represent 95% CI.

CONCLUSION

Human-Computer Interaction researchers and practitioners should stop filtering noisy input with a moving average. In most cases, they do not need to wrestle with low-level signal processing issues or with more complex techniques like Kalman filtering – which can be difficult to understand, tune, and implement. The 1€ filter is an intuitive and practical alternative since it is easy to understand, implement, and tune for low jitter and lag. Best of all, it produces better results.

REFERENCES

1. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. of UIST'11*, ACM (2011), 603–614.
2. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. of EGVE '03*, ACM (2003), 199–206.
3. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proc. of CHI '93*, CHI '93, ACM (1993), 488–493.
4. Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proc. of CHI '02*, 33–40.
5. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proc. of EICS '09*, ACM (2009), 187–196.
6. Vogel, D., and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution

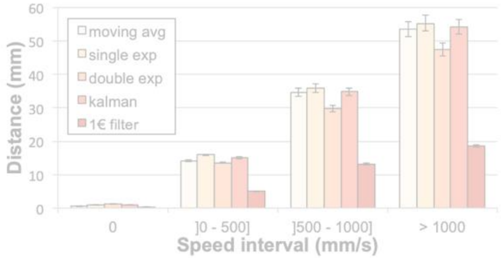


Figure 1. Mean distance between filtered and true cursor position for each speed interval and filter. Error bars represent 95% CI.

set. The 1€ filter has the smallest SEM (0.004) followed by LaViola's double exponential smoothing (0.013), the moving average and the Kalman filter (0.015), and single exponential smoothing (0.016). Our intention for this evaluation is to illustrate the performance of the 1€ filter in an intuitive way under realistic conditions. We are exploring alternative comparisons with user experiments, synthetic reference movements, different noise configurations, and examples of "noisy" hardware.

CONCLUSION

Human-Computer Interaction researchers and practitioners should stop filtering noisy input with a moving average. In most cases, they do not need to wrestle with low-level signal processing issues or with more complex techniques like Kalman filtering – which can be difficult to understand, tune, and implement. The 1€ filter is an intuitive and practical alternative since it is easy to understand, implement, and tune for low jitter and lag. Best of all, it produces better results.

REFERENCES

1. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. of UIST'11*, ACM (2011), 603–614.
2. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. of EGVE '03*, ACM (2003), 199–206.
3. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proc. of CHI '93*, CHI '93, ACM (1993), 488–493.
4. Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proc. of CHI '02*, 33–40.
5. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proc. of EICS '09*, ACM (2009), 187–196.
6. Vogel, D., and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution displays. In *Proc. of UIST '05*, ACM (2005), 33–42.
7. Ware, C., and Balakrishnan, R. Reaching for objects in vr displays: lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.
8. Welch, G., and Bishop, G. An introduction to the Kalman filter. SIGGRAPH course, ACM, Aug. 2001.
9. Wilson, A. D. Sensor- and recognition-based input for interaction. In *The Human Computer Interaction handbook*. CRC Press, 2007, 177–199.
10. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*, ACM (2007), 159–168.

APPENDIX A - 1€ FILTER

Algorithm 1: 1€ filter

EXT: First time flag: *firstTime* set to *true*
Data update rate: *rate*
Minimum cutoff frequency: *mincutoff*
Cutoff slope: *beta*
Low-pass filter: *xfilt*
Cutoff frequency for derivate: *dcutoff*
Low-pass filter for derivate: *dxfilt*
IN : Noisy sample value: *x*
OUT: Filtered sample value

```
if firstTime then
    firstTime ← false
    dx ← 0
else
    dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 2: *Filter* method of Low-pass filter

EXT: First time flag: *firstTime* set to *true*
IN : Noisy sample value : *x*
Alpha value : *alpha*
OUT: Filtered value

```
if firstTime then
    firstTime ← false
    hatxprev ← x
end
hatx ← alpha * x + (1 - alpha) * hatxprev
hatxprev ← hatx
return hatx
```

Algorithm 3: Alpha computation

IN : Data update rate in Hz: *rate*
Cutoff frequency in Hz: *cutoff*
OUT: Alpha value for low-pass filter

```
tau ← 1.0 / (2*π*cutoff)
te ← 1.0 / rate
return 1.0 / (1.0 + tau/te)
```

Kalman filters

Unlike the techniques above, Kalman filters make assumptions about the system generating the signal. Typically used for navigation and tracking, they work well when combining data from different sensors (e.g. a GPS and a speedometer) or when the system can be modeled by equations (e.g. determining vehicle acceleration from accelerator pedal position). Kalman filters rely on a *process model* and a *measurement model*. The standard Kalman filter uses a discrete-time linear stochastic difference equation for the process model and assumes that process and measurement noise are independent of each other, white, and are normally distributed [8].

REFERENCES

7. Ware, C., and Balakrishnan, R. Reaching for objects in vr displays: lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.
8. Welch, G., and Bishop, G. An introduction to the Kalman filter. SIGGRAPH course, ACM, Aug. 2001.
9. Wilson, A. D. Sensor- and recognition-based input for interaction. In *The Human Computer Interaction handbook*. CRC Press, 2007, 177–199.
10. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*, ACM (2007), 159–168.

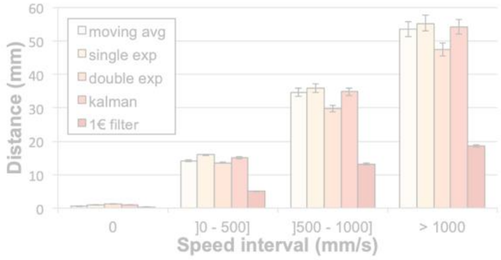


Figure 1. Mean distance between filtered and true cursor position for each speed interval and filter. Error bars represent 95% CI.

set. The 1€ filter has the smallest SEM (0.004) followed by LaViola's double exponential smoothing (0.013), the moving average and the Kalman filter (0.015), and single exponential smoothing (0.016). Our intention for this evaluation is to illustrate the performance of the 1€ filter in an intuitive way under realistic conditions. We are exploring alternative comparisons with user experiments, synthetic reference movements, different noise configurations, and examples of "noisy" hardware.

CONCLUSION

Human-Computer Interaction researchers and practitioners should stop filtering noisy input with a moving average. In most cases, they do not need to wrestle with low-level signal processing issues or with more complex techniques like Kalman filtering – which can be difficult to understand, tune, and implement. The 1€ filter is an intuitive and practical alternative since it is easy to understand, implement, and tune for low jitter and lag. Best of all, it produces better results.

REFERENCES

1. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. of UIST'11*, ACM (2011), 603–614.
2. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. of EGVE '03*, ACM (2003), 199–206.
3. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proc. of CHI '93*, CHI '93, ACM (1993), 488–493.
4. Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proc. of CHI '02*, 33–40.
5. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proc. of EICS '09*, ACM (2009), 187–196.
6. Vogel, D., and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution displays. In *Proc. of UIST '05*, ACM (2005), 33–42.
7. Ware, C., and Balakrishnan, R. Reaching for objects in vr displays: lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.
8. Welch, G., and Bishop, G. An introduction to the Kalman filter. SIGGRAPH course, ACM, Aug. 2001.
9. Wilson, A. D. Sensor- and recognition-based input for interaction. In *The Human Computer Interaction handbook*. CRC Press, 2007, 177–199.
10. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*, ACM (2007), 159–168.

APPENDIX A - 1€ FILTER

Algorithm 1: 1€ filter

EXT: First time flag: *firstTime* set to *true*
Data update rate: *rate*
Minimum cutoff frequency: *mincutoff*
Cutoff slope: *beta*
Low-pass filter: *xfilt*
Cutoff frequency for derivate: *dcutoff*
Low-pass filter for derivate: *dxfilt*
IN : Noisy sample value: *x*
OUT: Filtered sample value

```
if firstTime then
  | firstTime ← false
  | dx ← 0
else
  | dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 2: *Filter* method of Low-pass filter

EXT: First time flag: *firstTime* set to *true*
IN : Noisy sample value : *x*
Alpha value : *alpha*
OUT: Filtered value

```
if firstTime then
  | firstTime ← false
  | hatxprev ← x
end
hatx ← alpha * x + (1 - alpha) * hatxprev
hatxprev ← hatx
return hatx
```

Algorithm 3: Alpha computation

IN : Data update rate in Hz: *rate*
Cutoff frequency in Hz: *cutoff*
OUT: Alpha value for low-pass filter

```
tau ← 1.0 / (2*π*cutoff)
te ← 1.0 / rate
return 1.0 / (1.0 + tau/te)
```

1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems

Géry Casiez[1,2,3], Nicolas Roussel[3] & Daniel Vogel[4]
[1]LIFL, [2]University of Lille & [3]Inria Lille, France
[4]Cheriton School of Computer Science, University of Waterloo, Canada
gery.casiez@lifl.fr, nicolas.roussel@inria.fr, dvogel@uwaterloo.ca

ABSTRACT
The 1€ filter ("one Euro filter") is a simple algorithm to filter noisy signals for high precision and responsiveness. It uses a first order low-pass filter with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to reduce lag. The algorithm is easy to implement, uses very few resources, and with two easily understood parameters, it is easy to tune. In a comparison with other filters, the 1€ filter has less lag using a reference amount of jitter reduction.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User interfaces - Input devices and strategies.

General Terms
Algorithms, Performance, Human Factors

Author Keywords
Signal; noise; jitter; precision; lag; responsiveness; filtering

INTRODUCTION
Noisy signals occur when an original time varying value undergoes undesirable and unpredictable perturbations. These may be caused by things like heat and magnetic fields affecting hardware circuitry, the limits of sensor resolution, or even unstable numerical computation. Noisy signals are a common problem when tracking human motion, particularly with custom sensing hardware and inexpensive input devices like the Kinect or Wiimote. In addition, even signals from established high-end sensing systems can become noisy when interaction techniques use large scaling effects. A common example is using a Vicon tracking system to implement ray casting with a wall display [6]: calibration problems and hand tremor add further perturbations to the ones amplified by the pointing technique.

Noise affects the quality of a signal in two primary ways [9]. It can reduce *accuracy*, by adding an *offset* between the observed values and the true ones. More often, it reduces *precision*, where repeated observations of values exhibit *jitter* – many different values are observed for a single true one. Jitter has a large effect on the way people perceive and act. For example noisy values are harder to read and unstable cursors hinder target acquisition [3, 7, 5]. One usually wants to filter noisy signals to reduce, and possibly remove, the unwanted parts of the signal. However, filtering inherently introduces time latency – commonly called *lag* – which reduces system *responsiveness*. Lag may not be an issue in domains like artificial perception and decision making, but with interactive feedback, it is very important. In fact, it is the combination of precision and responsiveness that are crucial: people can point accurately in spite of an offset, but only with minimal lag and jitter. The difficulty is that implementing and tuning a filter to minimize both jitter and lag is challenging, especially with little or no background in signal processing.

In this paper we describe the 1€ filter ("one Euro filter"), a tool to improve noisy signal quality with a tunable jitter and lag balance. It uses a low-pass filter, but the cutoff frequency changes according to speed: at low speeds, a low cutoff reduces jitter at the expense of lag, but at high speeds, the cutoff is increased to reduce lag rather than jitter. The intuition is that people are very sensitive to jitter and not latency when moving slowly, but as movement speed increases, people become very sensitive to latency and not jitter. We compare the 1€ filter to alternative techniques and show how it can reduce that same amount of jitter with less lag. It is also efficient and easy to understand, implement, and tune: the algorithm can be expressed in a few lines; it uses only basic arithmetic; and it has only two independent parameters that relate directly to jitter and lag. Other researchers and ourselves have already used variations of it in many projects. In fact, the "dynamic recursive low-pass filter" used by the third author in [6] established the basic principle, but it required four parameters and a fixed sample rate. The '1€' name is an homage to the $1 recognizer [10]: we believe that the 1€ filter can make filtering input signals simpler and better, much like the $1 recognizer did for gestures.

After a review of the jitter, lag, and alternative filtering techniques, we describe the 1€ filter in detail with an implementation, discuss tuning with different applications, and conclude with an illustrative comparison.

JITTER, LAG, AND FILTERING
Several studies show jitter and lag have a negative impact on performance. MacKenzie et al. found mouse movement times increased 16% with 75 ms lag, and up to 64% with 225 ms lag [3]. With 3D hand tracking, Ware and Balakrishnan found that only 50 ms lag reduced performance by more than 8% [7]. Pavlovych and Stuerzlinger found no performance degradation below 58 ms lag using a mouse or Wiimote, but increasing jitter from 4 to 8 pixels doubled error rates for small targets [5]. Assuming a 100 PPI screen, 4 pixels corresponds to 1mm of jitter mean-to-peak: close to the 0.4 mm of jitter they found with the established Optitrack system.

Although the precision of an input device may be very good, it does not take into account scaling effects introduced by interaction techniques. Device input is often scaled up, so people can cover more display distance with less device movement. For example, default operating system mouse transfer functions can be scaled up 12× [1] and factors as high as 90× have been used when ray casting on wall sized displays [6]. Regardless of native device precision, scaling amplifies even small sensing perturbations, increasing jitter.

These results highlight the importance of balancing jitter and lag. Jitter should be less than 1mm mean-to-peak, but lag should be below 60 ms. As we shall see, any filter introduces some lag and considering 40-50 ms of inherent system lag [5], that leaves less than 10-20 ms for the filter.

Moving average
By the Central Limit Theorem and reasonable assumptions, averaging enough successive values of a noisy signal should produce a better estimate of the true one [9]. As a result, a *moving average* of the last n data values is commonly used by computer scientists as a kind of filter. For example, Myers et al. [4] used one for laser pointers and reduced hand tremor jitter from ± 8 pixels to between ± 2 and ± 4 pixels using a 0.5s window ($n = 10$). Since all n values are weighted equally, this creates a lag up to n times the sampling period.

Low-pass filters and exponential smoothing
With human movements, noise typically forms high frequencies in the signal while actual limb movements have lower frequencies. A low-pass filter is designed to let these desired low frequency portions pass through, while attenuating high frequency signals above a fixed cutoff frequency. The *order* of a low-pass filter relates to how aggressively it attenuates each frequency: first order filters reduce the signal amplitude by half every time the frequency doubles, while higher order variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i = \alpha X_i + (1-\alpha)\hat{X}_{i-1} \quad (1)$$

The first term of the equation is the contribution of new input data value, and the second term adds inertia from previous values. As α decreases, jitter is reduced, but lag increases since the output responds more slowly to changes in input. Since the contribution of older values exponentially decreases, a low-pass filter will have less lag than a high n moving average filter.

Smoothing techniques used in business and economic forecasts are similar in approach to a low-pass filter. The equation for *single exponential smoothing* is very similar to Equation 1. As the name suggests, *double exponential smoothing* uses two of these equations to handle trends in the signal. Although not formally documented, the Microsoft Kinect skeleton filters appear to be a variant of this type of smoothing[1]. LaViola extended double exponential smoothing for predictive tracking [2], building on Equations 1 and 2 to predict positions τ time steps in the future (Equation 3):

$$\hat{X}_i^{[2]} = \alpha \hat{X}_i + (1-\alpha)\hat{X}_{i-1}^{[2]} \quad (2)$$

$$P_{t+\tau} = \left(2 + \frac{\alpha\tau}{1-\alpha}\right)\hat{X}_i - \left(1 + \frac{\alpha\tau}{1-\alpha}\right)\hat{X}_i^{[2]} \quad (3)$$

Kalman filters
Unlike the techniques above, Kalman filters make assumptions about the system generating the signal. Typically used for navigation and tracking, they work well when combining data from different sensors (e.g. a GPS and a speedometer) or when the system can be modeled by equations (e.g. determining vehicle acceleration from accelerator pedal position). Kalman filters rely on a *process model* and a *measurement model*. The standard Kalman filter uses a discrete-time linear stochastic difference equation for the process model and assumes that process and measurement noise are independent of each other, white, and are normally distributed [8]. When estimating the true position of a moving object, the process model is typically a linear function of the speed and the previous estimated position. With additional complexity, Extended and Unscented variants of Kalman filters can also model non-linear processes and observations [8].

In the frequent case where the process and measurement noise covariances are not known, one must determine them empirically. This task can be challenging, and an improperly tuned filter can increase and even degrade the signal [9], by creating artificial "overshooting" movements for example. Moreover, understanding Kalman filters requires mathematical knowledge beyond basic linear algebra such as statistics, random signals, and stochastic methods. Implementing them requires a language or library with matrix operations. And, as demonstrated by LaViola for predictive tracking, they can be considerably slower to compute than double exponential smoothing predictors (approximately 135×) with similar jitter and lag performance [2].

THE 1€ FILTER
The 1€ filter is an adaptive first-order low-pass filter: it adapts the cutoff frequency of a low-pass filter for each new sample according to an estimate of the signal's speed, or more generally, its derivative value. Even though noisy signals are often sampled at a fixed frequency, filtering can not always follow the same pace, especially in event-driven systems. To accommodate possible fluctuations, we rewrite equation 1 to take into account the actual time interval between samples. Using a direct analogy with an electrical circuit, where a resistor in series with a capacitor defines a first order low-pass filter, α can be computed as a function of the sampling period T_e and a time constant τ, both expressed in seconds (Equation 4). The resistor and capacitor values define the time constant ($\tau = RC$) and the corresponding cutoff frequency f_c, in Hertz, of the circuit (Equation 5).

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \quad (4)$$

$$\tau = \frac{1}{2\pi f_c} \quad (5)$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1+\frac{\tau}{T_e}} \quad (6)$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \quad (7)$$

The sampling period T_e (or its inverse, the sampling rate) can be automatically computed from timestamps, so the cutoff frequency f_c is the only configurable parameter in equation 6. As with any low-pass filter, decreasing f_c reduces jitter, but increases lag. Finding a good trade-off between the two is difficult since people are more sensitive to jitter at low speeds, and more sensitive to lag at high speeds. This is why an adaptive cutoff frequency works well. To reduce jitter, a low f_c is used at low signal speeds, and to reduce lag, f_c is increased as speed increases. We found that a straightforward linear relationship between cutoff frequency f_c and the absolute speed works well (Equation 7). The speed (i.e the derivative $\dot{\hat{X}}_i$) is computed from raw signal values using the sampling rate and then low-pass filtered with a cutoff frequency chosen to avoid high derivative bursts caused by jitter. Our implementation uses a fixed value of 1 Hz, leaving only two configurable parameters: the intercept $f_{c_{min}}$ and the slope β shown in Equation 7. Details of the algorithm are provided in the Appendix.

Tuning and Applications
To minimize jitter and lag when tracking human motion, the two parameters can be set using a simple two-step procedure. First β is set to 0 and $f_{c_{min}}$ to a reasonable middle-ground value such as 1 Hz. Then the body part is held steady or moved at a very low speed while $f_{c_{min}}$ is adjusted to remove jitter and preserve an acceptable lag during these slow movements. Next, the body part is moved quickly in different directions while β is increased with a focus on minimizing lag. Note that parameters $f_{c_{min}}$ and β have clear conceptual relationships: if high speed lag is a problem, increase β; if slow speed jitter is a problem, decrease $f_{c_{min}}$. Rotational input uses a similar tuning process, but rotation axis and angle are filtered separately.

Another application of the 1€ filter is displaying noisy numerical values, such as an unsteady frame rate used to monitor graphical application performance. The goal is to reduce jitter to make the numerical output legible while minimizing lag so the value remains timely. Tuning is similar to above: adjust $f_{c_{min}}$ until the text becomes stable, then increase β until just before the text become unstable.

COMPARISON WITH OTHER FILTERS
To compare the 1€ filter with other techniques, we created a Python application that periodically samples the XY position of the system cursor, adds noise, and displays filtered cursor positions. Each filter can be tuned interactively and all filters can be shown simultaneously making it possible to visually compare jitter reduction and lag across parameter settings and filters. Once tuned, timestamped positions can be logged for the system cursor (with and without noise) and filtered positions of all filters. We used a MacBook Pro with a 1440 × 900 pixel display (109 PPI).

In our comparison, we used independent Gaussian white noises for X and Y with a 50 dB SNR[2], a public implementation of the Kalman filter[3], and custom implementations of a moving average, single exponential, and LaViola's double exponential smoothing. We tuned moving average first and used its performance as a baseline. We found that averaging more than 14 data values did not reduce jitter further and only increased lag, so we used n=14. Then we interactively tuned the other filters to primarily match the jitter reduction of moving average, and secondarily attempting to reduce lag.

Tuning single exponential smoothing to match the reference jitter requires a low alpha value (α=0.11) which introduces lag. This highlights the difficulty of tuning with only a single parameter. For LaViola's double exponential smoothing filter, the reference jitter is obtained with a lower alpha value (α=0.06) and with lower lag. However, this causes overshooting when the pointer abruptly decelerates. For the Kalman filter, we set the measurement noise covariance to the variance of the introduced noise (18.06) as in [2], and adjusted the process noise covariance until we obtained the reference jitter reduction (at a value of 0.3). The amount of lag for this setting was comparable to the moving average and single-exponential. For the 1€ filter, we matched the reference jitter and optimized lag using the tuning procedure described above. In the first tuning step, setting $f_{c_{min}}$ = 1 Hz and β = 0 matched the reference jitter and lag was similar to single exponential smoothing. In the second tuning step, increasing β to 0.007 made the lag almost imperceivable yet maintained the reference jitter when stationary or moving slowly. A supplementary video demonstrates this tuning process and visualizes filter performance.

For a quantitative comparison, we logged the system cursor at 60 Hz for about 1 hour during regular desktop use, then added white noise and applied the filters using the settings above. Figure 1 shows the distance from each filtered cursor position to the true one, binned into four speed intervals. Note that since we tuned the filters to match a reference jitter when not moving, the error between filtered position and noiseless position is primarily due to lag when moving. With higher speeds, the filtered position lags farther and farther behind, increasing this distance (the small distances in the 0 mm/s interval are likely due to offset or overshooting). All filters introduce a similar amount of lag except for the 1€ filter which has less lag across all speed intervals.

As an overall comparison, we computed the Standard Error of the Mean (SEM) in mm for each filter for this data set. The 1€ filter has the smallest SEM (0.004) followed by LaViola's double exponential smoothing (0.013), the moving average and the Kalman filter (0.015), and single exponential smoothing (0.016). Our intention for this evaluation is to illustrate the performance of the 1€ filter in an intuitive way under realistic conditions. We are exploring alternative comparisons with user experiments, synthetic reference movements, different noise configurations, and examples of "noisy" hardware.

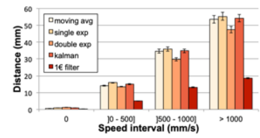


Figure 1. Mean distance between filtered and true cursor position for each speed interval and filter. Error bars represent 95% CI.

CONCLUSION
Human-Computer Interaction researchers and practitioners should stop filtering noisy input with a moving average. In most cases, they do not need to wrestle with low-level signal processing issues or with more complex techniques like Kalman filtering – which can be difficult to understand, tune, and implement. The 1€ filter is an intuitive and practical alternative since it is easy to understand, implement, and tune for low jitter and lag. Best of all, it produces better results.

REFERENCES
1. Casiez, G., and Roussel, N. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proc. of UIST'11*, ACM (2011), 603–614.
2. LaViola, J. J. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proc. of EGVE '03*, ACM (2003), 199–206.
3. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proc. of CHI '93*, CHI '93, ACM (1993), 488–493.
4. Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. Interacting at a distance: measuring the performance of laser pointers and other devices. In *Proc. of CHI '02*, 33–40.
5. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proc. of EICS '09*, ACM (2009), 187–196.
6. Vogel, D., and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution displays. In *Proc. of UIST '05*, ACM (2005), 33–42.
7. Ware, C., and Balakrishnan, R. Reaching for objects in vr displays: lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.
8. Welch, G., and Bishop, G. An introduction to the Kalman filter. SIGGRAPH course, ACM, Aug. 2001.
9. Wilson, A. D. Sensor- and recognition-based input for interaction. In *The Human Computer Interaction handbook*. CRC Press, 2007, 177–199.
10. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*, ACM (2007), 159–168.

APPENDIX A - 1€ FILTER

Algorithm 1: 1€ filter

```
EXT : First time flag: firstTime set to true
      Data update rate: rate
      Minimum cutoff frequency: mincutoff
      Cutoff slope: beta
      Low-pass filter: xfilt
      Cutoff frequency for derivate: dcutoff
      Low-pass filter for derivate: dxfilt
IN  : Noisy sample value: x
OUT : Filtered sample value

if firstTime then
  | firstTime ← false
  | dx ← 0
else
  | dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 2: *Filter* method of Low-pass filter

```
EXT : First time flag: firstTime set to true
IN  : Noisy sample value : x
      Alpha value : alpha
OUT : Filtered value

if firstTime then
  | firstTime ← false
  | hatxprev ← x
end
hatx ← alpha * x + (1 - alpha) * hatxprev
hatxprev ← hatx
return hatx
```

Algorithm 3: Alpha computation

```
IN  : Data update rate in Hz: rate
      Cutoff frequency in Hz: cutoff
OUT : Alpha value for low-pass filter

tau ← 1.0 / (2*π*cutoff)
te ← 1.0 / rate
return 1.0 / (1.0 + tau/te)
```

[1]http://cm-bloggers.blogspot.com/2011/07/kinect-sdk-smoothing-skeleton-data.html

[2]This signal-to-noise ratio was estimated from Gametrak data using a zero phase shift filter and is consistent with numbers in [2]
[3]http://greg.czerniak.info/node/5

A typical paper

- Abstract
- Introduction
- Background
- The “big idea”
- Evaluation and metrics
- Conclusion
- References
- Appendices

Crash course in mathematical notation

$x \quad y \quad z \quad i \quad j \quad k \quad p \quad q \quad s \quad t$

$\alpha \quad \beta \quad \gamma \quad \delta \quad \epsilon \quad \theta \quad \lambda \quad \mu \quad \pi \quad \rho \quad \sigma \quad \tau \quad \phi \quad \varphi \quad \omega$

$$\alpha \quad \beta \quad \gamma \quad \delta \quad \epsilon \quad \theta \quad \lambda \quad \mu \quad \pi \quad \rho \quad \sigma \quad \tau \quad \phi \quad \varphi \quad \omega$$

alpha gamma epsilon lambda pi sigma phi omega

beta delta theta mu rho tau phi

α β γ δ ϵ θ λ μ π ρ σ τ ϕ φ ω

alpha gamma epsilon lambda pi sigma phi omega

beta delta theta mu rho tau phi

It can be seen in Eq. 2 that the odd-order lower-side frequencies, sin(α-β), sin(α-3β), etc., are preceded by a a b d e l m p r s t

$\alpha \quad \beta \quad \gamma \quad \delta \quad \epsilon \quad \theta \quad \lambda \quad \mu \quad \pi \quad \rho \quad \sigma \quad \tau \quad \phi \quad \varphi \quad \omega$

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \qquad \text{for } i = 1, ..., k.$$

arxiv.org/abs/1611.01144

$$x \qquad X \qquad \mathcal{X} \qquad \hat{x} \qquad x^* \qquad \bar{x} \qquad \tilde{x} \qquad x'$$

$$X = \{x_1, x_2, x_3, \ldots, x_n\}$$

$$x_i \qquad x_{i,j,k} \qquad x^i \qquad x^{(i)} \qquad x_j^{(i)} \qquad x[i]$$

$$\sum$$

$$\sum$$

$$\sum_{n=0}^{N-1} a_n \sin(n\omega t)$$

$$\sum$$

$$\sum_{n=0}^{N-1} a_n \sin(n\omega t)$$

```
float sum = 0.0f;
for (int n = 0; n < N; ++n) {
    sum += a[n] * std::sin(n * omega * t);
}
return sum;
```

$$\sum_{i=1}^{m}\sum_{j=1}^{n} i^2 + j$$

```
float outerSum = 0.0f;
for (int i = 1; i <= m; ++i) {
    float innerSum = 0.0f;
    for (int j = 1; j <= n; ++j) {
        innerSum += (i*i + j);
    }
    outerSum += innerSum;
}
return outerSum;
```

$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

$$\int_0^T x(t)\cos(n\omega t)dt$$

$$\int_0^T x(t) \cos(n\omega t) dt$$

$$\sum_{t=0}^{T} x[t] \cos(n\omega t)$$

$$f'(x) = \ldots$$

$$\frac{df(x)}{dx} = \ldots$$

$$\frac{d^2 f}{dx^2} = \ldots$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = \ldots$$

$$\Re\{\mathrm{e}^{i\omega_{\mathrm{c}}t+k\cos(\omega_{\mathrm{m}}t)}\}$$

$$||x||_2 \quad ||x||_2^2$$

$$\exp\left(-\frac{\sum_{j=1}^{n}\left(x_j - l_j^{(i)}\right)^2}{2\sigma^2}\right)$$

$$\boldsymbol{p}(\boldsymbol{x},t) = \begin{bmatrix} e^{st+v_1x_1} \\ \dots \\ e^{st+v_mx_m} \end{bmatrix} \triangleq e^{st\boldsymbol{I}+\boldsymbol{V}\boldsymbol{X}} \cdot \mathbf{1}$$

$$\sum_{i=1}^{k}\log\left|\det\left(\frac{\partial f_i(\mathbf{h}_{i-1})}{\partial\mathbf{h}_{i-1}}\right)\right|$$

$$y_n(t) = \begin{cases} H_0\Delta(n) + \Delta(n)\displaystyle\sum_{m=1}^{M_{\text{out}}}\frac{r_{\text{out},m}}{p_{\text{out},m}}e^{p_{\text{out},m}(t-t_n)} & \text{if } t \geq t_n \\ 0 & \text{otherwise.} \end{cases}$$

https://en.wikipedia.org/wiki/Glossary_of_mathematical_symbols

We can now write an expression for the output $y[n]$:

$$y[n] = \tilde{y}(n) = \int_{-\infty}^{\infty} h_{\text{rect}}(u) y(n-u) du$$
$$= \int_{0}^{1} y(n-u) du$$
$$= \int_{0}^{1} f(\tilde{x}(n-u)) du$$

using (3), and noticing that over this interval $u = \tau$, we can write:

$$y[n] = \int_{0}^{1} f(\tilde{x}) d\tau \tag{7}$$
$$= \int_{0}^{1} f(x_n + \tau(x_{n-1} - x_n)) d\tau$$

From integration by substitution, we can write:

$$\int_{0}^{1} f(\tilde{x}) \frac{d\tilde{x}}{d\tau} d\tau = \int_{x_n}^{x_{n-1}} f(\tilde{x}) d\tilde{x}$$

The piecewise linear nature of $\tilde{x}$ now becomes useful as it means that $\frac{d\tilde{x}}{d\tau}$ is constant over the extent of the τ integration, and can be factored out of the integral to produce:

$$y[n] = \int_{0}^{1} f(\tilde{x}) d\tau = \frac{d\tau}{d\tilde{x}} \int_{x_n}^{x_{n-1}} f(\tilde{x}) d\tilde{x}$$
$$= \frac{1}{x_{n-1} - x_n} \int_{x_n}^{x_{n-1}} f(\tilde{x}) d\tilde{x} \tag{8}$$

Finally, by applying the fundamental theorem of calculus, we produce:

$$y[n] = \frac{F_0(x_n) - F_0(x_{n-1})}{x_n - x_{n-1}} \tag{9}$$

where F_0 is the antiderivative of f.

Reducing the Aliasing of Nonlinear Waveshaping Using Continuous-time Convolution (Parker, Zavalishin, Le Bivic, 2016)

Implementing the 1€ Filter

$$
\begin{aligned}
\alpha &= \frac{1}{1 + \frac{\tau}{T_e}} && (4) \\
\tau &= \frac{1}{2\,\pi\, f_c} && (5) \\
\hat{X}_i &= \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \frac{1}{1 + \frac{\tau}{T_e}} && (6) \\
f_c &= f_{c_{min}} + \beta \, |\dot{\hat{X}}_i| && (7)
\end{aligned}
$$

THE 1€ FILTER

The 1€ filter is an adaptive first-order low-pass filter: it adapts the cutoff frequency of a low-pass filter for each new sample according to an estimate of the signal's speed, or more generally, its derivative value. Even though noisy signals are often sampled at a fixed frequency, filtering can not always follow the same pace, especially in event-driven systems. To accommodate possible fluctuations, we rewrite equation 1 to take into account the actual time interval between samples. Using a direct analogy with an electrical circuit, where a resistor in series with a capacitor defines a first order low-pass filter, α can be computed as a function of the sampling period T_e and a time constant τ, both expressed in seconds (Equation 4). The resistor and capacitor values define the time constant ($\tau = RC$) and the corresponding cutoff frequency f_c, in Hertz, of the circuit (Equation 5).

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{4}$$

$$\tau = \frac{1}{2\pi f_c} \tag{5}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \frac{1}{1 + \frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta \, |\dot{\hat{X}}_i| \tag{7}$$

variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i = \alpha X_i + (1 - \alpha) \hat{X}_{i-1} \tag{1}$$

variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i \quad = \quad \alpha \, X_i + (1 - \alpha) \, \hat{X}_{i-1} \tag{1}$$

filtered data

input data

previous filtered data

variants reduce the signal amplitude at a greater rate. A discrete time realization of a first order low-pass filter is given by Equation 1 where X_i and $\hat{X}_i$ denote the raw and filtered data at time i and α is a smoothing factor in $]0, 1]$:

$$\hat{X}_i = \alpha X_i + (1 - \alpha) \hat{X}_{i-1} \quad (1)$$

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{4}$$

$$\tau = \frac{1}{2\,\pi\, f_c} \tag{5}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \frac{1}{1 + \frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta\, |\dot{\hat{X}}_i| \tag{7}$$

$$
\begin{aligned}
\alpha &= \frac{1}{1+\frac{\tau}{T_e}} && (4)\\
\tau &= \frac{1}{2\,\pi\, f_c} && (5)\\
\hat{X}_i &= \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1+\frac{\tau}{T_e}} && (6)\\
f_c &= f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| && (7)
\end{aligned}
$$

$$
\begin{aligned}
\alpha &= \frac{1}{1 + \frac{\tau}{T_e}} && (4) \\
\tau &= \frac{1}{2\,\pi\, f_c} && (5) \\
\hat{X}_i &= \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \frac{1}{1 + \frac{\tau}{T_e}} && (6) \\
f_c &= f_{c_{min}} + \beta \, |\dot{\hat{X}}_i| && (7)
\end{aligned}
$$

$$
\begin{aligned}
\alpha &= \frac{1}{1+\frac{\tau}{T_e}} && (4)\\
\tau &= \frac{1}{2\,\pi\, f_c} && (5)\\
\hat{X}_i &= \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1+\frac{\tau}{T_e}} && (6)\\
f_c &= f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| && (7)
\end{aligned}
$$

is why an adaptive cutoff frequency works well. To reduce jitter, a low f_c is used at low signal speeds, and to reduce lag, f_c is increased as speed increases. We found that a straightforward linear relationship between cutoff frequency f_c and the absolute speed works well (Equation 7). The speed (i.e the derivative $\dot{X}_i$) is computed from raw signal values using the sampling rate and then low-pass filtered with a cutoff frequency chosen to avoid high derivative bursts caused by jitter. Our implementation uses a fixed value of 1 Hz, leaving

$$\alpha = \frac{1}{1+\frac{\tau}{T_e}} \tag{4}$$

$$\tau = \frac{1}{2\,\pi\, f_c} \tag{5}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e}\hat{X}_{i-1}\right)\frac{1}{1+\frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \tag{7}$$

$$f_c = f_{c_{min}} + \beta \, |\dot{\hat{X}}_i| \tag{7}$$

$$\tau = \frac{1}{2 \, \pi \, f_c} \tag{5}$$

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{4}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \frac{1}{1 + \frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta \, |\dot{\hat{X}}_i| \tag{7}$$

$$\tau = \frac{1}{2\,\pi\, f_c} \tag{5}$$

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{4}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \frac{1}{1 + \frac{\tau}{T_e}} \tag{6}$$

$$f_c = f_{c_{min}} + \beta \, |\dot{\hat{X}}_i| \tag{7}$$

$$\tau = \frac{1}{2\,\pi\, f_c} \tag{5}$$

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{4}$$

$$\hat{X}_i = \left(X_i + \frac{\tau}{T_e} \hat{X}_{i-1} \right) \alpha \tag{6}$$

$$f_c = f_{c_{min}} + \beta\,|\dot{\hat{X}}_i| \tag{7}$$

$$\tau = \frac{1}{2\,\pi\,f_c} \tag{5}$$

$$\alpha = \frac{1}{1+\frac{\tau}{T_e}} \tag{4}$$

$$\hat{X}_i = \alpha\,X_i + (1-\alpha)\,\hat{X}_{i-1} \tag{6}$$

$$\hat{X}_i = \alpha X_i + (1 - \alpha)\hat{X}_{i-1}$$

```
float filter(float x)
{



    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}}$$

```
float filter(float x)
{




    float alpha = 1.0f / (1.0f + tau / Te);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

```
float filter(float x)
{




    float alpha = 1.0f / (1.0f + tau * sampleRate);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

$$\tau = \frac{1}{2\,\pi\,f_c}$$

```
float filter(float x)
{




    float tau = 1.0f / (2.0f * PI * cutoff);
    float alpha = 1.0f / (1.0f + tau * sampleRate);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

```
float filter(float x)
{




    float r = 2.0f * PI * cutoff;
    float alpha = r / (r + sampleRate);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

```cpp
float filter(float x)
{




    float cutoff = min_cutoff + beta * std::abs(derivative);
    float r = 2.0f * PI * cutoff;
    float alpha = r / (r + sampleRate);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

```
float filter(float x)
{
    float dx = ???
    float dalpha = 2.0f * PI / (2.0f * PI + sampleRate);
    derivative = dx * dalpha + derivative * (1.0f - dalpha);

    float cutoff = min_cutoff + beta * std::abs(derivative);
    float r = 2.0f * PI * cutoff;
    float alpha = r / (r + sampleRate);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

Algorithm 1: 1€ filter

EXT: First time flag: *firstTime* set to *true*
Data update rate: *rate*
Minimum cutoff frequency: *mincutoff*
Cutoff slope: *beta*
Low-pass filter: *xfilt*
Cutoff frequency for derivate: *dcutoff*
Low-pass filter for derivate: *dxfilt*

IN : Noisy sample value: *x*

OUT: Filtered sample value

```
if firstTime then
    firstTime ← false
    dx ← 0
else
    dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 1: 1€ filter

```
EXT:  First time flag: firstTime set to true
      Data update rate: rate
      Minimum cutoff frequency: mincutoff
      Cutoff slope: beta
      Low-pass filter: xfilt
      Cutoff frequency for derivate: dcutoff
      Low-pass filter for derivate: dxfilt
IN :  Noisy sample value: x
OUT:  Filtered sample value

if firstTime then
    firstTime ← false
    dx ← 0
else
    dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 1: 1€ filter

```
EXT : First time flag: firstTime set to true
      Data update rate: rate
      Minimum cutoff frequency: mincutoff
      Cutoff slope: beta
      Low-pass filter: xfilt
      Cutoff frequency for derivate: dcutoff
      Low-pass filter for derivate: dxfilt
IN  : Noisy sample value: x
OUT : Filtered sample value

if firstTime then
    firstTime ← false
    dx ← 0
else
    dx ← (x - xfilt.hatxprev()) * rate
end
edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
cutoff ← mincutoff + beta * |edx|
return xfilt.filter(x, alpha(rate, cutoff))
```

```
float filter(float x)
{
    float dx = (x - y) * sampleRate;
    float dalpha = 2.0f * PI / (2.0f * PI + sampleRate);
    derivative = dx * dalpha + derivative * (1.0f - dalpha);

    float cutoff = min_cutoff + beta * std::abs(derivative);
    float r = 2.0f * PI * cutoff;
    float alpha = r / (r + sampleRate);
    y = x * alpha + y * (1.0f - alpha);
    return y;
}
```

https://gery.casiez.net › 1euro

1€ Filter - Géry Casiez

1€ **Filter** Géry Casiez 1,2,3 Nicolas Roussel 3 and Daniel Vogel 4 1 LIFL, 2 University of Lille, 3 Inria Lille and 4 Cheriton School of Computer Science, University of Waterloo. This page implements the "1€ **Filter**" in different languages. See our CHI 2012 paper (PDF). The name "1€" is an homage to the $1...

https://github.com › casiez › OneEuroFilter

Algorithm to filter noisy signals for high precision and ... - GitHub

The 1€ **filter** ("**one Euro filter**") is a simple algorithm to **filter** noisy signals for high precision and responsiveness. It uses a first order low-pass **filter** with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to...

https://jaantollander.com › post › noise-filtering-using-one-euro-filter

Noise Filtering Using 1€ Filter | Jaan Tollander de Balsch

The 1€ **Filter** is a low pass **filter** for filtering noisy signals in real-time. It is also a simple **filter** with only two configurable parameters. The signal at time T i is denoted as value X i and the filtered signal as value X ^ i. The **filter** uses exponential smoothing. X ^ 1 = X 1 (1) X ^ i = α X i + (1 − α) X ^ i − 1, i ≥ 2.

audiodev.blog/krunch/

Krunch Saturator

Combination low-pass filter and saturation plug-in based on the 1€ Filter

Some time ago I came across the 1 Euro Filter, an adaptive filter designed to balance jitter and lag in noisy input for interactive systems. I was curious what it would sound like when applied to audio. **Pretty good actually!** The filter adds harmonics in an interesting way. So I turned it into a free plug-in named Krunch.

In this blog post I will talk a bit about how the 1 Euro Filter works and why it sounds like a saturation effect when used to process audio.

You can grab the code and VST3/AU files from GitHub.

How to use Krunch

Krunch combines low-pass filtering with saturation. It has the following controls:

- **KRUNCH** The higher this is dialed up, the more the sound will be filtered. But even at low values it will already add crunch.

audiodev.blog/krunch/

Patents

United States Patent [19]

Ishibashi, deceased

[11] **Patent Number: 4,658,691**

[45] **Date of Patent: Apr. 21, 1987**

[54] **ELECTRONIC MUSICAL INSTRUMENT**

[75] Inventor: **Masanori Ishibashi, deceased,** late of Oume, Japan, by Masayuki Ishibashi, legal representative

[73] Assignee: **Casio Computer Co., Ltd.,** Tokyo, Japan

[21] Appl. No.: **788,669**

[22] Filed: **Oct. 17, 1985**

Related U.S. Application Data

[63] Continuation of Ser. No. 561,180, Dec. 14, 1983, abandoned.

[30] **Foreign Application Priority Data**

Dec. 17, 1982 [JP] Japan 57-221266
Dec. 22, 1982 [JP] Japan 57-225582

[51] **Int. Cl.**[4] **G10H 1/02**
[52] **U.S. Cl.** **84/1.19;** 84/1.28
[58] **Field of Search** 84/1.01, 1.03, 1.19, 84/1.21, 1.23, 1.24, 1.25, 1.28

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,175,464 11/1979 Deutsch 84/1.19
4,183,275 1/1980 Niimi et al. 84/1.19
4,223,582 9/1980 Kato et al. 84/1.19
4,249,447 2/1981 Tomisawa 84/1.01

FOREIGN PATENT DOCUMENTS

A10015424 9/1980 European Pat. Off. .
WO8103236 11/1981 World Int. Prop. O. .
1404559 9/1975 United Kingdom .
2027250A 2/1980 United Kingdom .
A2032159 4/1980 United Kingdom .
A2087621 5/1982 United Kingdom .

Primary Examiner—F. W. Isen
Attorney, Agent, or Firm—Frishauf, Holtz, Goodman & Woodward

[57] **ABSTRACT**

An electronic musical instrument includes circuitry for modifying an ordinary address signal which changes at a uniform rate over one cycle of a waveform, into a modified address signal whose rate varies in one cycle of the waveform by the use of a modification signal.

The modified address signal accesses a storage device such as a ROM in which waveform data is stored, thereby producing the modified waveform data from the storage device. The modification signal is obtained from the ordinary address signal through a predetermined logic circuit.

46 Claims, 68 Drawing Figures

ELECTRONIC MUSICAL INSTRUMENT

This application is a continuation of application Ser. No. 561,180, filed Dec. 14, 1983, now abandoned.

2. BACKGROUND OF THE INVENTION

The present invention relates to a waveform generator circuit which generates a waveform with digital circuitry, and more particularly to an electronic musical instrument in which the rate of accessing a waveform changes in one cycle of the waveform.

With the progress of digital technology, it has become possible to generate waveform data by means of digital circuitry and to convert the digital waveform data into an analog signal by means of a digital-to-analog converter, thereby to produce an analog signal waveform. Such waveform generation by digital circuitry is also applied to electronic musical instruments, and the products of electronic musical instruments capable of generating waveforms of various tone colors are implemented.

Until now, the musical sound generating systems of the electronic musical instruments based on digital circuitry as stated above have included (i) a sinusoidal wave synthesis system, (ii) a variable filter system, (iii) a waveform memory readout system, (iv) a frequency modulation system, etc.

The sinusoidal wave synthesis system (i) is a system wherein the sinusoidal wave signals of a fundamental wave and higher harmonics are generated by a digital circuit, and these digital waveform signals are synthesized to produce a musical sound of desired tone color. In case of producing musical sounds in desired harmonic overtone forms, this system needs computing channels which are equal in number to the sorts of required harmonic overtones. Further, in case of changing a spectrum with time, higher harmonics control signals equal in number to the sorts of harmonic overtones are needed, for varying amplitude levels for the respective harmonic overtones. This system has the problems that the generator circuit becomes large in size because the aforementioned computing channels and higher harmonics control signals necessitate circuits equal in number to the sorts of harmonic overtones, and that the generation control of the higher harmonic control signals becomes complicated.

The variable filter system (ii) is a system wherein a digital filter is used, and the frequency characteristic of the filter is changed by a variable signal. This system has the problem that the circuit of the digital filter becomes large in size. Further, in a case where a waveform is generated at a fixed sampling rate, that is, where the fundamental tone to be inputted to the digital filter is generated at a fixed sampling rate, a waveform having a large number of higher harmonics is difficult to obtain, resulting in the problem that the effect of the digital filter in a higher harmonics region decreases to half. This system also has the problem that folded distortion arises.

The waveform memory readout system (iii) is a system wherein waveform data stored in a memory or the like in advance is sequentially read out in correspondence with a phase angle, thereby to generate a waveform. Since the aforementioned waveform data stored in the waveform memory is the data of a musical sound waveform to be produced as a musical sound, the spectrum of the waveform has been fixed. In order to change the spectrum, therefore, waveform data corresponding to the change of the spectrum must be stored in the memory, and moreover, a control circuit for reading out the data successively in correspondence with the change of the spectrum is needed. This system accordingly has the problems that the capacity of the memory is large and that the control circuit is complicated.

The system (iv) is an application of frequency modulation, and is a system wherein, using the two sinusoidal waves of a carrier wave and a modulating wave, the frequency ratio and the modulation depth are changed thereby to change a harmonic overtone. This system can control the harmonic overtone to some extent. Since, however, each harmonic overtone changes according to a Bessel function, it has been difficult to obtain a musical sound whose spectrum has a smoothly changing envelope, for example, whose amplitude value decreases as the waveform changes from the fundamental wave toward the higher harmonics.

Further, there is a system wherein a peak (hereinbelow, termed the "formant peak") is possessed in the higher frequency region of the spectrum of a musical sound waveform, and the formant peak frequency is changed with time, thereby to bestow a change on a musical sound. An example is to utilize the resonance effect of a voltage control filter VCF in an analog synthesizer. Methods of generating the aforementioned formant peak by means of a digital circuit include (a) a method wherein the coefficient of a harmonic overtone synthesized by adding sinusoidal waves is changed with time so as to give rise to a filter effect, and to generate a peak value in the amplitude values of higher harmonics of higher orders, and (b) a method wherein a resonance effect as attained with an analog filter is produced by a digital low-pass filter. The method (a) is the same as the foregoing system (i). It requires computing channels corresponding to the higher-order frequencies in order to generate the higher harmonics, and besides, it needs to set amplitudes for the respective higher harmonics, namely, harmonic overtones, so that a complicated circuit is necessitated and has been difficult to fabricate. With the method (b), the circuit of the digital filter becomes larger in size and has similarly been difficult in realization.

3. SUMMARY OF THE INVENTION

The present invention has been made in order to solve the problems of the prior art, and has for its first object to provide a waveform generating system which permits the spectrum of a waveform to change smoothly.

A second object of the present invention is to provide a waveform generating system which generates the waveforms of a rectangular wave, a sawtooth wave, etc. free from the higher frequency components of the signals thereof.

A third object of the present invention is to provide a musical sound generating system for an electronic musical instrument in which the spectrum of a waveform is changed by a digital circuit.

A fourth object of the present invention is to provide a musical sound generating system for an electronic musical instrument which generates a musical sound having a peak value in the higher frequency region of a spectrum, namely, in harmonic overtones.

According to the present invention, there is provided an electronic musical instrument comprising storage means to store waveform information; address signal production means to produce an address signal which changes at a uniform rate over one cycle of a waveform, in order to read out the waveform information stored in said storage means; modification means to modify the address signal produced from said address signal production means, into a modified address signal whose changing rate varies in one cycle of the waveform; and means to access said storage means by the use of the modified address signal delivered from said modification means.

Another feature of the present invention is to provide an electronic musical instrument comprising storage means to store waveform information; address signal production means to successively produce address signals for reading out the waveform information stored in said storage means; modification means to modify each of the address signals into a modified address signal which appoints an address of more than one cycle of a waveform while said each address signal appoints an address of one cycle of the waveform; and means to access said storage means by the use of the modified address signal delivered from said modification means.

4. BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing an embodiment of the present invention;

FIG. 2 is a block diagram showing the first arrangement of a waveform synthesizer circuit in FIG. 1;

FIGS. 3 and 12 are circuit diagrams each showing the arrangement of FIG. 2more in detail;

FIGS. 4(a)-4(d) are diagrams for explaining symbols used in FIG. 3;

FIGS. 5, 8, 10, 13, 14, 18, 19, 20, 21, 23 and 25 are waveform diagrams for explaining the formation of waveforms in the present invention;

FIGS. 6(A), 7(A), 9(A) and 11(A) show output waveforms in an embodiment of the present invention, while FIGS. 6(B), 7(B), 9(B) and 11(B) show corresponding spectra;

FIG. 15 is a circuit diagram of a read only memory and peripheral circuits thereof showing a modified embodiment of the present invention;

FIG. 16 is a block diagram showing the second arrangement of the waveform synthesizer circuit in FIG. 1;

FIGS. 17, 22 and 24 are circuit diagrams each showing the arrangement of FIG. 16 more in detail; and

FIGS. 26(A1), 26(B1), . . . 26(F1), FIGS. 27(A1), 27(B1), . . . 27(F1) and FIGS. 28(A1), 28(B1), . . . 28(F1) show waveforms generated by the respective embodiments of the present invention in FIGS. 17, 22 and 24, while FIGS. 26(A2), 26(B2), . . . 27(F2), FIGS. 27(A2), 27(B2), . . . 27(F2) and FIGS. 28(A2), 28(B2), . . . 28(F2) show corresponding spectra.

5. PREFERRED EMBODIMENTS OF THE INVENTION

FIG. 1 is a circuit block diagram showing an embodiment of the present invention. In the illustrated embodiment of FIG. 1, the present invention is applied to an electronic musical instrument.

The first output of keyboard 1 is applied to a frequency information generator circuit 2, while the second output is applied to a higher harmonics control signal generator circuit 4 as well as an envelope control signal generator circuit 5. The output of the frequency information generator circuit 2 enters the first input terminal of a phase angle computing circuit 3. The output terminal of the phase angle computing circuit 3 is connected to the second input terminal thereof and the input terminal A of a waveform synthesizer circuit 8. The output terminal of the higher harmonics control signal generator circuit 4 is connected to the first input terminal of an adder circuit 6. The second input terminal of the adder circuit 6 is supplied with a control signal from another circuit (not shown). The output of the adder circuit 6 enters the input terminal B of the waveform synthesizer circuit 8. The output terminal C of the waveform synthesizer circuit 8 is connected to the first input terminal of an envelope multiplier circuit 7, the second input terminal of which has the output terminal of the envelope control signal generator circuit 5 connected thereto. The output terminal of the envelope multiplier circuit 7 is connected to a digital-to-analog converter circuit DAC (not shown). The keyboard 1 is a circuit which generates the positional information of a depressed key and the timing signal of the key. The positional information of the key is applied to the frequency information generator circuit 2, and the timing signal of the key to the higher harmonics control signal generator circuit 4 and the envelope control signal generator circuit 5. The frequency information generator circuit 2 is a circuit which generates frequency information, namely, phase angle information corresponding to the depressed key on the basis of the aforementioned positional information of the key. By way of example, it delivers the phase angle information in succession in accordance with specified clock pulses. The phase angle computing circuit 3 adds the information applied to the first and second input terminals thereof, and delivers the result. Since the output of the phase angle computing circuit 3 enters the second input terminal thereof, the phase angle information items produced from the frequency information generator circuit 2 are successively added to the contents of the phase angle computing circuit 3 in accordance with the specified clock pulses. That is, the phase angle information items produced from the frequency information generator circuit 2 are accumulated by the phase angle computing circuit 3. The cumulation is executed in single-cycle units, and when a phase angle of above one cycle has been reached, the phase of one cycle is subtracted. In the embodiment of FIG. 1, the phase angle of one cycle (corresponding to 2π) is set at, e.g., 2^{12}. When this value has been exceeded, a carry ought to be provided. Since, however, no carry is used, the operation of the embodiment results in the subtraction of the phase angle corresponding to one cycle. The output of the phase angle computing circuit 3 is applied to the input terminal A of the waveform synthesizer circuit 8. The higher harmonics control signal generator circuit 4 is supplied with the timing signal, and converts it into, e.g., a tone color control signal for changing a higher harmonic component with time. The resulting output of the tone color control signal is added in the adder circuit 6 with the external control signal, for example, a control signal for changing a tone color by means of an actuator disposed outside. The adder circuit 6 can be omitted in a case where the control signal is not externally applied. The output of the adder circuit 6 is applied to the input terminal B of the waveform synthesizer circuit 8. The waveform synthesizer circuit 8 is a circuit for accessing a waveform after the phase angle or address signal changing at a uniform rate as received from the input terminal A is converted into a modified address signal whose one cycle is equal to one cycle of the received address signal, but in which the first half of such one cycle has a higher rate and the latter half a lower rate by way of example, or into a modified address signal which addresses more than one cycle while the received address signal appoints one cycle. The extent of the modification changes, depending upon the control signal received from the input terminal B.

The timing signal of the keyboard 1 is further applied to the envelope control signal generator circuit 5. The envelope control signal generator circuit 5 generates control data for changing the amplitude of a musical sound to-be-produced in correspondence with the depressed key. The output or envelope signal of the circuit 5 enters the envelope multiplier circuit 7. On the other hand, waveform data delivered from the output terminal C of the waveform synthesizer circuit 8 enters the envelope multiplier circuit 7. The envelope multiplier circuit 7 multiplies the waveform data and the envelope signal, and delivers the result. The output of the envelope multiplier circuit 7 is applied to the digital-to-analog converter circuit DAC (not shown), by which it is converted into an analog signal.

By way of example, the waveform synthesizer circuit 8 is composed of a divider circuit 9 and a waveform memory 10 as shown in FIG. 2. The divider circuit 9 executes an operation in which the phase angle received from the input terminal A is divided by the tone color control signal, namely, higher harmonics control signal received from the input terminal B, in a specified phase angle range and is further divided by a different value in another specified range. That is, in the waveform synthesizer circuit 8, the advancing way of the phase angle is not held constant over one cycle, but is changed. The divided result accesses the waveform memory 10 within the waveform synthesizer circuit 8, and waveform data is delivered from the output terminal C. The access to the memory at this time is not fixed over one cycle, but is changed within one cycle, so that the waveform data obtained by distorting the phase of a waveform stored in the waveform memory 10 is provided from the output terminal C.

FIG. 3 is a detailed circuit diagram illustrative of the first arrangement of the waveform synthesizer circuit 8 corresponding to the embodiment of the present invention shown in FIG. 2. Symbols in FIG. 3 are informal, and the respective symbols (a) and (c) denote setups depicted at (b) and (d) in FIGS. 4(a)-4(d). As seen from FIGS. 4(a)-4(d), FIG. 4(a) expresses the gate circuit (FIG. 4(b)) of a FET, the source and drain of which correspond to the input and output of the gate circuit and the gate of which corresponds to the control input terminal of the gate circuit. FIG. 4(a) shows the exclusive logic OR gate (FIG. 4(d)) for an input. A group of input terminals N is connected to a group of gates G1 and a group of gates G2. The ends of the groups of gates G1, G2 remote from the input terminals N are connected to a group of exclusive logic OR gates EOR1, the output signals of which are applied to the inputs A0-A11 of a divider DIV through a group of exclusive logic OR gates EOR2. The group of gates G1 are connected so that the respective bit positions N0-N11 of the input terminals N may be shifted by one bit toward the upper bits, and the least significant bit thereof is connected so that a low level (ground level) may be received. A control terminal SAT is directly connected to the control input terminals of the group of gates G2, and it is connected to the control input terminals of the

group of gates G1 through an in
input of an AND gate AND1 has a
connected thereto, the second inpu
N11 of the input terminals N conne
output thereof is connected to the
exclusive logic OR gates EOR1 in

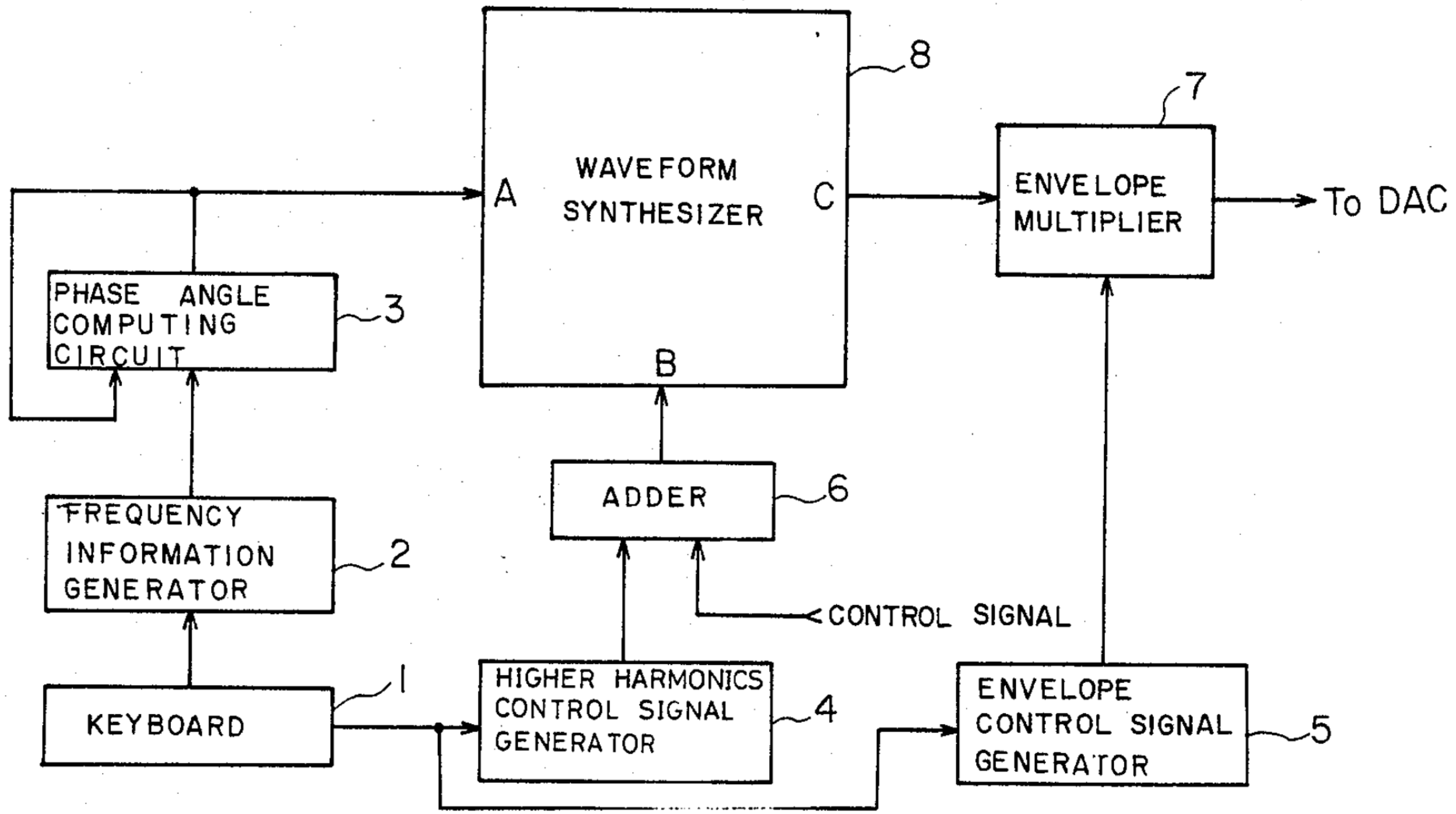
The bits M0-M10 and bit M11
terminals M are connected to the i
divider DIV through a group of
gates EOR3 and through a gate G3
sive logic OR gate EOR3, respectiv
exclusive logic OR gate EOR3 cor
M11 has a gate G4 connected ther
gate G4 remote from the exclusive
3 is grounded, and the control input
the control terminal SAT connec
while, the control input terminal of
control terminal SAT connected
inverter I2. The first inputs A11-
COMP are supplied with the outp
exclusive logic OR gates EOR1,
puts B11-B0 are supplied with the
entering the group of exclusive log
The comparison output of the co
connected to the first input of an
The control terminal SAT is conn
input of the AND gate AND2, t
enters the second inputs of the r
exclusive logic OR gates EOR2 and

The operated outputs D0-D11
enter the address inputs of a read
through groups of gates G5, G6. T
tude values of the half wavelength
sine waves are stored in the read on
corresponds to −1 when all the o
level, and to +1 that they are at a h
terminal SQU is directly connected
terminals of the group of gates G5,
to the control input terminals of th
through an inverter I3. The outputs
only memory ROM are delivered
exclusive logic OR gates EOR4. T
SQU and the bit N11 are respective
inputs of an AND gate AND3, t
enters the inputs of the group of
gates EOR4 in common.

In the embodiment of the present
FIG. 3, the input terminals N and
inputs A and B of the waveform sy
FIG. 1, respectively. The input te
with the output or phase angle data
bits from the phase angle computin
while the input terminal M is sup
color control data or modulation
of, e.g., 12 bits from the adder circ

This circuit includes the three co
SIP and SQU as stated above. By
aforementioned control terminals,
high level to one of them, a wav
ously depending upon the signals
input terminal M.

First, when the high level sign
control terminal SAT and low leve
to the control terminals SIP, SQU,
generated. When the control term
supplied with the low level signals
AND gates AND1 and AND3 be
nals, and the groups of exclusive lo



WAVEFORM SYNTHESIZER
A
B
C
8
ENVELOPE MULTIPLIER
7
To DAC
PHASE ANGLE COMPUTING CIRCUIT
3
ADDER
6
FREQUENCY INFORMATION GENERATOR
2
CONTROL SIGNAL
KEYBOARD
1
HIGHER HARMONICS CONTROL SIGNAL GENERATOR
4
ENVELOPE CONTROL SIGNAL GENERATOR
5

Fig. 3

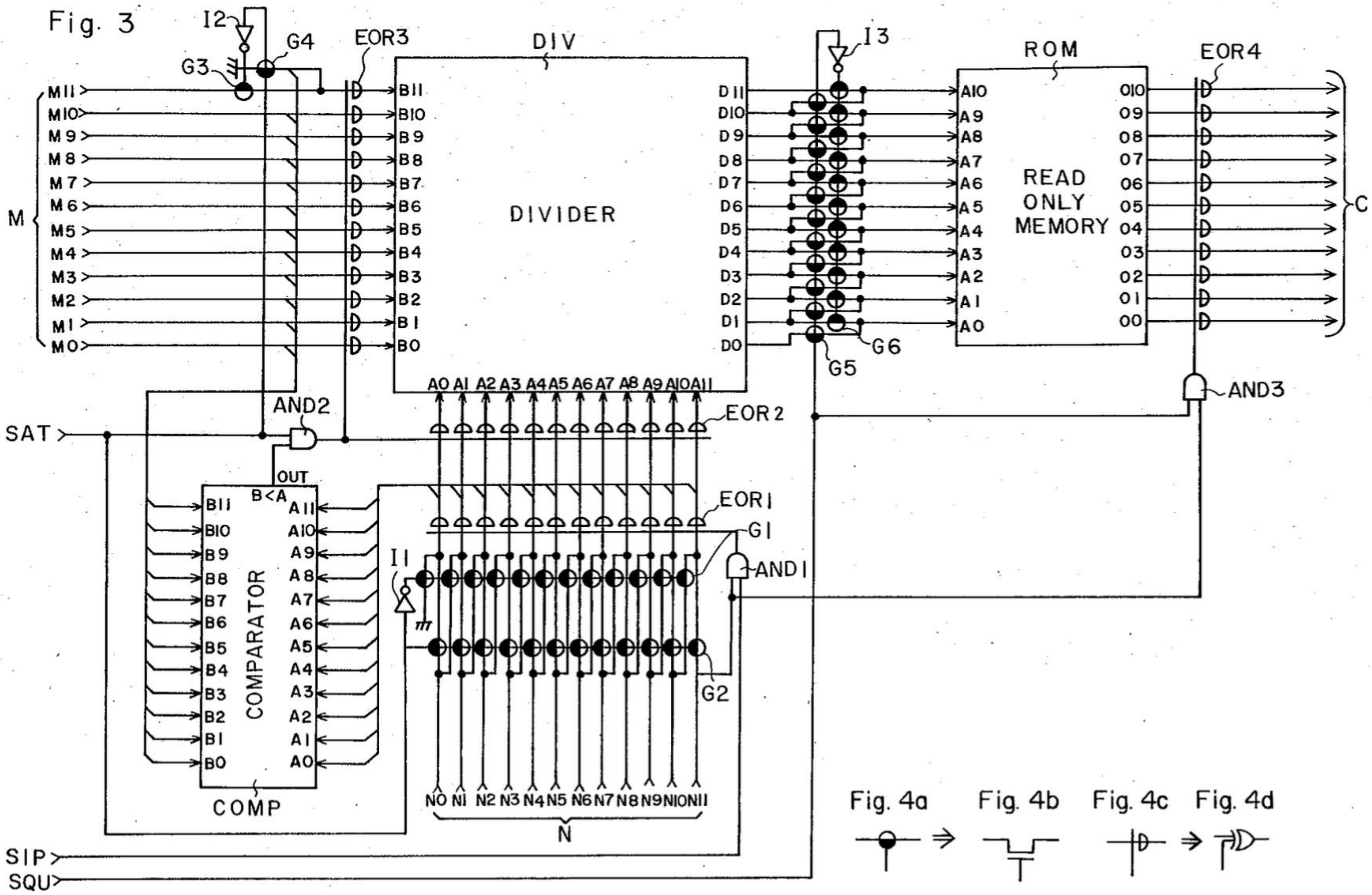


Fig. 4a Fig. 4b Fig. 4c Fig. 4d

Fig. 28 (B 1)

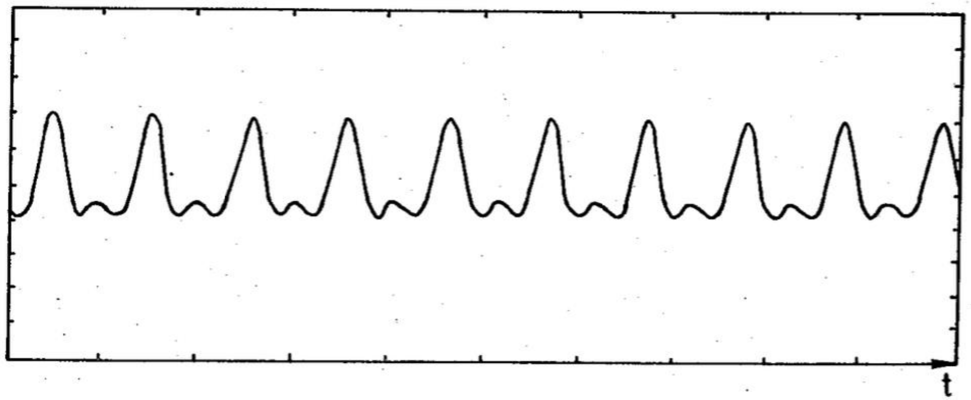
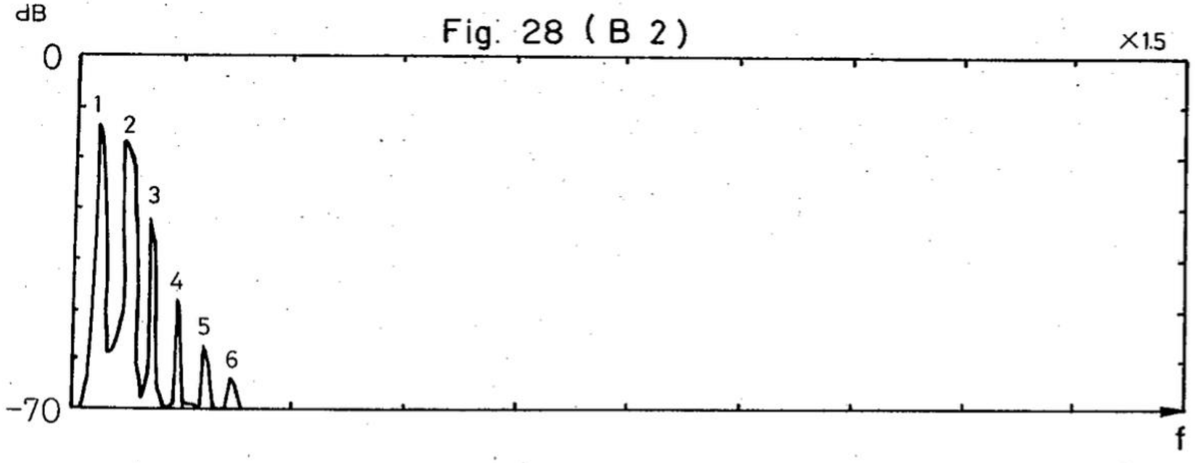


Fig. 28 (B 2)



What is claimed is:

1. An electronic musical instrument, comprising:

storage means for storing waveform information;

address signal production means for producing a single address signal which changes at a uniform rate corresponding to a frequency of the waveform to be produced over one cycle of a waveform, to read out the waveform information stored in said storage means;

modulating signal production means for producing a modulating signal;

modification means coupled to said address signal production means and to said modulating signal production means for modifying the single address signal produced from said address signal production means, into a modified address signal according to the modulating signal supplied from said modulating signal production means without using a feedback loop from said storage means, the changing rate of said modified address signal varying in one cycle of the waveform; and

accessing means coupled to said modification means for accessing said storage means by the use of the modified address signal delivered from said modification means to generate a waveform signal which has a distorted waveform according to the modulating signal produced by the modulating signal production means, and has the frequency determined by the single address signal generated by the address signal production means.

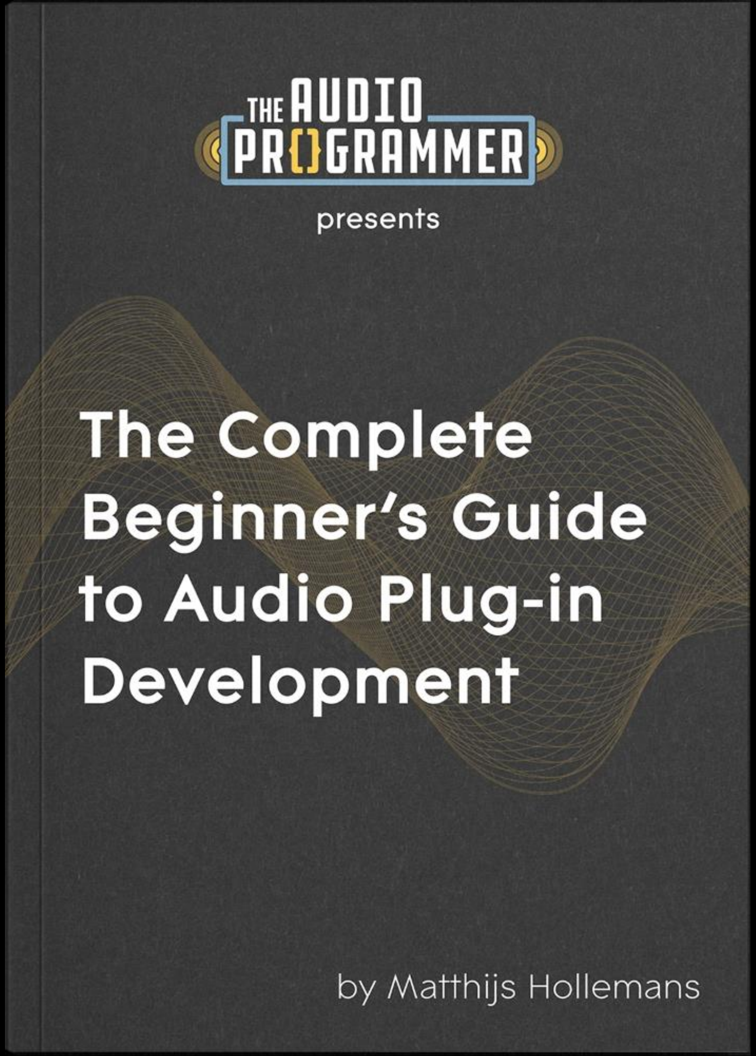
2. The electric musical instrument according to claim 1, wherein said modification means includes means for modifying said single address signal by switching said single address signal and an inverted value of said single address signal within said one cycle of the waveform.

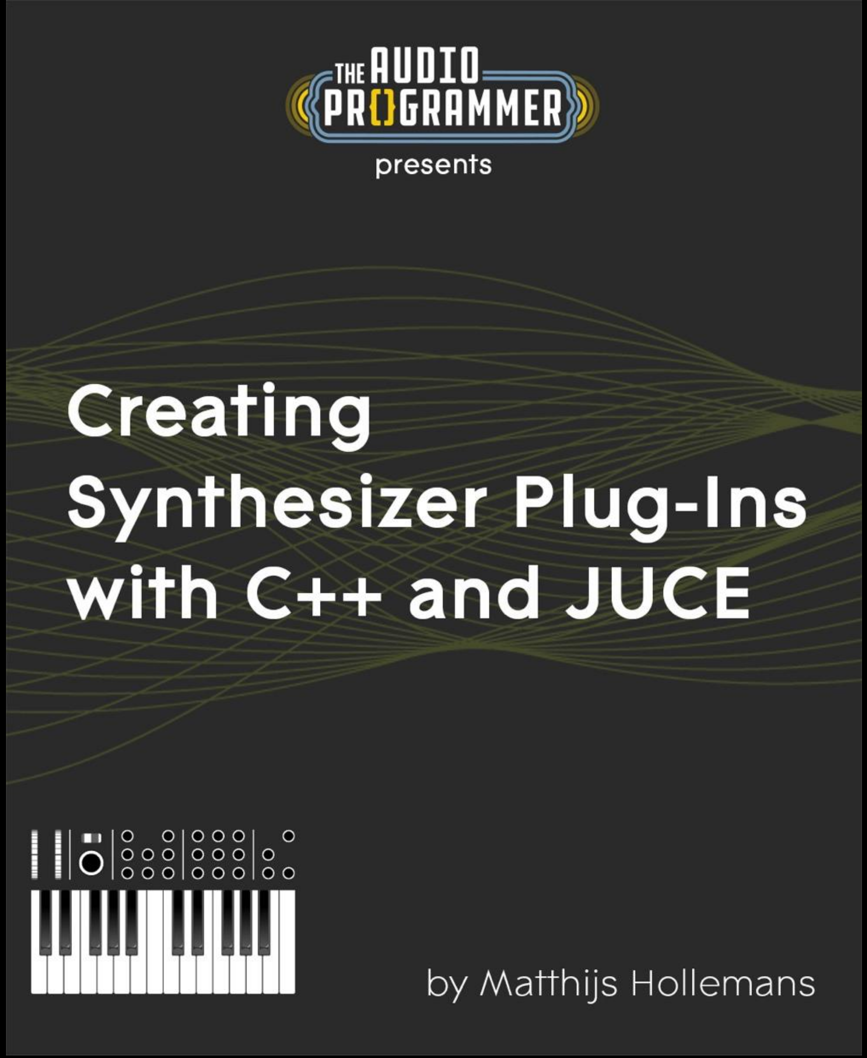
3. An electronic musical instrument according to claim 1, wherein said modulating signal production means produces a modulating signal which changes with the lapse of time.

4. An electronic musical instrument according to claim 1, wherein said address signal production means delivers at the uniform rate, phase angle information which defines a phase angle of the waveform.

5. An electronic musical instrument according to claim 1, wherein said storage means stores sine waves or cosine waves as the waveform information.

6. An electronic musical instrument according to claim 1, wherein said storage means stores waveforms which correspond to half-cycles or quarter-cycles of cosine waves.





Website: audiodev.blog

Code: github.com/hollance

Discord: @matthijs

Email: mail@hollance.com