Digital Signal Processing for Music

Part 17: real-time and blocking

alexander lerch

Georgia Tech | Center for Music Technology

real-time systems
introduction

- many audio processing systems are real-time systems
- this includes
  - most audio plugins,
  - studio hardware effects etc.

real-time systems
introduction

real-time system (wikipedia)

"In a real-time digital signal processing (DSP) process, the analyzed (input) and generated (output) samples can be processed (or generated) continuously in the time it takes to input and output the same set of samples independent of the processing delay"

- "processing delay and resources must be bounded even if the processing continues for an unlimited time"
- "mean processing time per sample is no greater than the sampling period, which is the reciprocal of the sampling rate"

⇒ "perform all computations continuously at a fast enough rate that the output (...) keeps up with changes in the input signal"

real-time systems
introduction

real-time system (wikipedia)

"In a real-time digital signal processing (DSP) process, the analyzed (input) and generated (output) samples can be processed (or generated) continuously in the time it takes to input and output the same set of samples independent of the processing delay"

- "processing delay and resources must be bounded even if the processing continues for an unlimited time"
- "mean processing time per sample is no greater than the sampling period, which is the reciprocal of the sampling rate"

⇒ "perform all computations continuously at a fast enough rate that the output (...) keeps up with changes in the input signal"

real-time systems
introduction

real-time system (wikipedia)

"In a real-time digital signal processing (DSP) process, the analyzed (input) and generated (output) samples can be processed (or generated) continuously in the time it takes to input and output the same set of samples independent of the processing delay"

- "processing delay and resources must be bounded even if the processing continues for an unlimited time"
- "mean processing time per sample is no greater than the sampling period, which is the reciprocal of the sampling rate"

⇒ "perform all computations continuously at a fast enough rate that the output (...) keeps up with changes in the input signal"

real-time systems
introduction

real-time system (wikipedia)

"In a real-time digital signal processing (DSP) process, the analyzed (input) and generated (output) samples can be processed (or generated) continuously in the time it takes to input and output the same set of samples independent of the processing delay"

- "processing delay and resources must be bounded even if the processing continues for an unlimited time"
- "mean processing time per sample is no greater than the sampling period, which is the reciprocal of the sampling rate"

⇒ "perform all computations continuously at a fast enough rate that the output (...) keeps up with changes in the input signal"

real-time systems
environment

- digital audio playback/recording: **constant stream** of audio samples to/from your sound device
  - distance of samples is defined by (constant!) sample rate
  - **hard timing** — no room for pauses
- if sound device (e.g., DAC) does not receive a sample in time, **audio will glitch**

- typical operating systems work with **blocks of samples**
- typically, audio API **(quasi-)periodically** requests a block of samples

real-time systems
environment

- digital audio playback/recording: **constant stream** of audio samples to/from your sound device
  - distance of samples is defined by (constant!) sample rate
  - **hard timing** — no room for pauses
- if sound device (e.g., DAC) does not receive a sample in time, **audio will glitch**

- typical operating systems work with **blocks of samples**
- typically, audio API **(quasi-)periodically** requests a block of samples

real-time systems

properties

- **performance**:
  - processing time for one block $\leq$ block length
  - real-time computing does not necessarily mean high performance computing!
- **causality**:
  - system output/state depends only on current and prior values
  - *no* knowledge of future samples
- **latency**:
  - delay of a system between the stimulus and the response to this stimulus
    - *algorithmic delay*: (FFT-Processing, Look-Ahead, ...)
    - *interface delay*: (block length, ad/da conversion)

real-time systems
properties

- **performance**:
  - processing time for one block $\leq$ block length
  - real-time computing does not necessarily mean high performance computing!

- **causality**:
  - system output/state depends only on current and prior values
  - *no* knowledge of future samples

- **latency**:
  - delay of a system between the stimulus and the response to this stimulus
    - *algorithmic delay*: (FFT-Processing, Look-Ahead, ...)
    - *interface delay*: (block length, ad/da conversion)

real-time systems
properties

- **performance**:
  - processing time for one block $\leq$ block length
  - real-time computing does not necessarily mean high performance computing!

- **causality**:
  - system output/state depends only on current and prior values
  - *no* knowledge of future samples

- **latency**:
  - delay of a system between the stimulus and the response to this stimulus
    - *algorithmic delay*: (FFT-Processing, Look-Ahead, ...)
    - *interface delay*: (block length, ad/da conversion)

real-time systems
properties

- **performance**:
  - processing time for one block $\leq$ block length
  - real-time computing does not necessarily mean high performance computing!
- **causality**:
  - system output/state depends only on current and prior values
  - *no* knowledge of future samples
- **latency**:
  - delay of a system between the stimulus and the response to this stimulus
    - *algorithmic delay*: (FFT-Processing, Look-Ahead, ...)
    - *interface delay*: (block length, ad/da conversion)

real-time systems
examples

which of the following effects/processors are capable of real-time processing

real-time systems
examples

which of the following effects/processors are capable of real-time processing

- level normalization (normalize highest amplitude to max)
- biquad EQ
- reverb
- pitch shifting
- time stretching

real-time systems
examples

which of the following effects/processors are capable of real-time processing

- level normalization (normalize highest amplitude to max)
+ biquad EQ
+ reverb
+ pitch shifting
- time stretching

real-time systems
clock

- **synchronization**
  - different components of a real-time systems have to be synchronized to work together

  ⇒ **master clock**
  - otherwise: multiple clocks have slight deviations and run out of sync (→ buffer overflow/underflow → clicks)

- **master clock**
  - on a computer: usually the sound card
  - timers etc. that are not based on the sound card clock will **never** work

real-time systems
clock

- **synchronization**
  - different components of a real-time systems have to be synchronized to work together

  ⇒ **master clock**
  - otherwise: multiple clocks have slight deviations and run out of sync (→ buffer overflow/underflow → clicks)

- **master clock**
  - on a computer: usually the sound card
  - timers etc. that are not based on the sound card clock will **never** work

real-time systems
clock

- **synchronization**
  - different components of a real-time systems have to be synchronized to work together

  ⇒ **master clock**
  - otherwise: multiple clocks have slight deviations and run out of sync (→ buffer overflow/underflow → clicks)

- **master clock**
  - on a computer: usually the sound card
  - timers etc. that are not based on the sound card clock will **never** work

real-time systems
clock

- **synchronization**
  - different components of a real-time systems have to be synchronized to work together

  $\Rightarrow$ **master clock**
  - otherwise: multiple clocks have slight deviations and run out of sync ($\rightarrow$ buffer overflow/underflow $\rightarrow$ clicks)

- **master clock**
  - on a computer: usually the sound card
  - timers etc. that are not based on the sound card clock will **never** work

real-time systems
blocking

- **operating system/sound device**
  - block size settings depend on *latency and interactivity requirements*
  - modern driver models and systems can achieve *block sizes down to a few samples* at typical audio sample rates (that is not necessarily true for mobile devices)
  - for some drivers the block size can vary
  - *typical range*: 32 – 4096 samples, higher on Android
  - despite the block size setting, there *might be additional buffering* taking place in one of the layers between hardware and audio software
- **plugin host**
  - often: simply the system block size for efficiency reasons
  - but: block sizes *may* vary (cf. automation)
- **plugin**
  - needs to process/buffer *any* input/output block size
  - is likely to use a different block size and overlapping blocks internally (beware of workload peaks!)

real-time systems
blocking

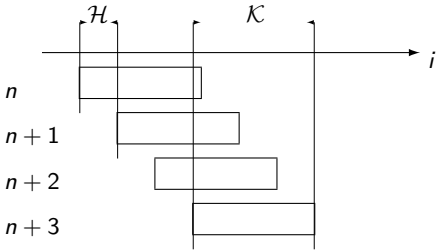
- **operating system/sound device**
  - block size settings depend on *latency and interactivity requirements*
  - modern driver models and systems can achieve *block sizes down to a few samples* at typical audio sample rates (that is not necessarily true for mobile devices)
  - for some drivers the block size can vary
  - *typical range*: 32 – 4096 samples, higher on Android
  - despite the block size setting, there *might be additional buffering* taking place in one of the layers between hardware and audio software
- **plugin host**
  - often: simply the system block size for efficiency reasons
  - but: block sizes *may* vary (cf. automation)
- **plugin**
  - needs to process/buffer *any* input/output block size
  - is likely to use a different block size and overlapping blocks internally (beware of workload peaks!)

real-time systems
blocking

- **operating system/sound device**
  - block size settings depend on *latency and interactivity requirements*
  - modern driver models and systems can achieve *block sizes down to a few samples* at typical audio sample rates (that is not necessarily true for mobile devices)
  - for some drivers the block size can vary
  - *typical range*: 32 – 4096 samples, higher on Android
  - despite the block size setting, there *might be additional buffering* taking place in one of the layers between hardware and audio software
- **plugin host**
  - often: simply the system block size for efficiency reasons
  - but: block sizes *may* vary (cf. automation)
- **plugin**
  - needs to process/buffer *any* input/output block size
  - is likely to use a different block size and overlapping blocks internally (beware of workload peaks!)

real-time systems
block based processing

processing of *blocks of samples* vs. individual samples

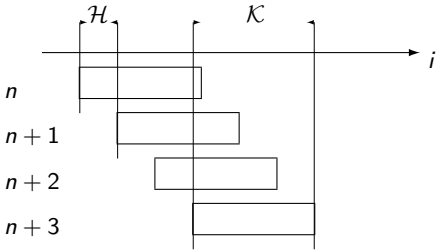


reasons:

- block based algorithms (FFT, ...)
- audio hardware characteristics
- efficiency (SIMD, memory allocation)

real-time systems
block based processing

processing of *blocks of samples* vs. individual samples



reasons:

- block based algorithms (FFT, ...)
- audio hardware characteristics
- efficiency (SIMD, memory allocation)

real-time systems
blocking

- **time-stamps**
  - blocking can be considered similar to down-sampling
  - ⇒ *what time stamps to assign to each block?*
    - begin of each block
    - center of each block

- **initialization**
  - real-time systems are designed to work for infinite input stream
  - ⇒ *how to initialize internal buffers?*
    - usually zeros, but other initializations may make sense in specific scenarios

- **performance** issues due to blocking
  - plugin gets stream of samples split into small blocks (e.g., 32 samples)
  - internally, STFT with large hopsize (e.g., 2048 samples) is used
  - ⇒ *what is the potential performance problem here?*
    - each hop requires data from 64 input blocks
    - ⇒ no processing can be done for 63 blocks
    - ⇒ processing of huge FFT has to be done during the 64th block (32 samples)

real-time systems
blocking

- **time-stamps**
  - blocking can be considered similar to down-sampling
  - ⇒ *what time stamps to assign to each block?*
    - begin of each block
    - center of each block
- **initialization**
  - real-time systems are designed to work for infinite input stream
  - ⇒ *how to initialize internal buffers?*
    - usually zeros, but other initializations may make sense in specific scenarios
- **performance** issues due to blocking
  - plugin gets stream of samples split into small blocks (e.g., 32 samples)
  - internally, STFT with large hopsize (e.g., 2048 samples) is used
  - ⇒ *what is the potential performance problem here?*
    - each hop requires data from 64 input blocks
    - ⇒ no processing can be done for 63 blocks
    - ⇒ processing of huge FFT has to be done during the 64th block (32 samples)

real-time systems
blocking

- **time-stamps**
  - blocking can be considered similar to down-sampling
  - ⇒ *what time stamps to assign to each block?*
    - begin of each block
    - center of each block
- **initialization**
  - real-time systems are designed to work for infinite input stream
  - ⇒ *how to initialize internal buffers?*
    - usually zeros, but other initializations may make sense in specific scenarios
- **performance** issues due to blocking
  - plugin gets stream of samples split into small blocks (e.g., 32 samples)
  - internally, STFT with large hopsize (e.g., 2048 samples) is used
  - ⇒ *what is the potential performance problem here?*
    - each hop requires data from 64 input blocks
    - ⇒ no processing can be done for 63 blocks
    - ⇒ processing of huge FFT has to be done during the 64th block (32 samples)

real-time systems
blocking

- **time-stamps**
  - blocking can be considered similar to down-sampling
  - ⇒ *what time stamps to assign to each block?*
    - begin of each block
    - center of each block
- **initialization**
  - real-time systems are designed to work for infinite input stream
  - ⇒ *how to initialize internal buffers?*
    - usually zeros, but other initializations may make sense in specific scenarios
- **performance** issues due to blocking
  - plugin gets stream of samples split into small blocks (e.g., 32 samples)
  - internally, STFT with large hopsize (e.g., 2048 samples) is used
  - ⇒ *what is the potential performance problem here?*
    - each hop requires data from 64 input blocks
    - ⇒ no processing can be done for 63 blocks
    - ⇒ processing of huge FFT has to be done during the 64th block (32 samples)

real-time systems
inplace processing

what is "inplace processing"

real-time systems
inplace processing

what is "inplace processing"

- samples of the input block are replaced with the output block

real-time systems
inplace processing

what is "inplace processing"

- samples of the input block are replaced with the output block
  - \+ resource friendly: memory allocation for output buffer
  - − original input data cannot be used anymore

summary
real-time systems

real-time systems have the following properties:

- hard **performance** requirements
  - processing of input block has to be faster then time span of this block **for each block, not only on average**
- **causality**
  - future samples cannot taken into account (or only by increasing the latency: look-ahead)
- **latency**
  - time between input and system response, usually intended to be minimal