DIGITAL MUSIC WORKSHOP / 02 / DIGITAL AUDIO SIGNAL PROCESSING

DIGITAL MUSIC WORKSHOP / 02 / DIGITAL AUDIO SIGNAL PROCESSING

- writing the audio buffer
- noise
- amplification
- sampler
- oscillator
- echo

WRITING THE AUDIO BUFFER

- blocks of *sample data* are requested by the *audio system*
- application has to writes values to the block
- value range: `FLOAT(-1.0, 1.0)`

WRITING THE AUDIO BUFFER

- block size defines the *latency* (small block == low latency)
- e.g update rate: 44100 Hz / 512 BLOCKSIZE ≈ 86 Hz

NOISE
-----

noise is produced by generating random values:

```
mSample = noise(-1.0, 1.0f);
```

AMPLIFICATION
---

amplification is achieved by multiplying sample values with an *amplification factor*:

```
float mAmplification = 2.0f;
mSample = mSample * mAmplification;
```

- *amplification factor* greater than `1.0` increases the volume
- *amplification factor* smaller than `1.0` and greater than `0.0` decreases volume
- *amplification factor* smaller than `0.0` invertes the signal

----
SAMPLER
----

a *sampler* can play back *pre-recorded* chunks of data:

```
float[] mSampleData = { 0.1f, -0.56f, 0.44f, 0.16f, … }
mSample = mSampleData[i];
```

SAMPLER

----

sample data can be exported as *raw* data from other applications (i.e *Audacity*).

note, that files are usually stored in `byte` format, i.e 4 bytes are combined into a single `float` sample value (see `Sampler.load(byte[])` and `Sampler.bytesToFloat32(byte[])`).

if the sampler produces weird sounds the `endianess` might be wrong (see `Sampler.bytesToFloat32(byte[], boolean)`).

see `ExampleDSP07Sampler`

OSCILLATOR
----

oscillators can be produced by creating alternating values from functions:

```
mSample = sin(r);
```

see `ExampleBasics04DigitalSignalProcessing`

ECHO
----

an echo (or delay) can be created by buffering sample data and adding it back into newer samples.

see `SketchExampleDSP03Echo`

REFERENCES

- Musicdsp.org
- VCV Rack Manual / DSP
- More Awesome Music DSP*