Game Technology

Lecture 13 – 30.01.2015

Dipl-Inf. Robert Konrad
Dr.-Ing. Florian Mehm

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Preliminary timetable

TECHNISCHE UNIVERSITÄT DARMSTADT

Lecture No.	Date	Topic
1	17.10.2014	Basic Input & Output
2	24.10.2014	Timing & Basic Game Mechanics
3	31.10.2014	Software Rendering 1
4	07.11.2014	Software Rendering 2
5	14.11.2014	Basic Hardware Rendering
6	21.11.2014	Animations
7	28.11.2014	Physically-based Rendering
8	05.12.2014	Physics 1
9	12.12.2014	Physics 2
10	19.12.2014	Procedural Content Generation
11	16.01.2015	Compression & Streaming
12	23.01.2015	Multiplayer
13	**30.01.2015**	**Audio**
14	06.02.2015	Scripting
15	13.02.2015	AI

Sound Waves

Sound waves

- Air compression
- Longitudinal Waves
- ~343 m/s
- 20 to 17000 Hz

Loudspeakers

Converts electrical signals to sound waves

- Using an acoustic membrane

Ears

Two identical audio sensors

Measures actual wave forms

- Using the eardrum

Computer -> Speaker

- **Small ring buffer**
- **Discretely sampled waveform**
- **Pointer to last sample written**
- **Pointer to next sample to read**

Sound Mixing

- **Superpositioning**
  - Adding waves

- **Again physically accurate**

- **Actual danger of superposition effects**
  - Avoid mixing identical sounds

Music

Music

Long files

- Played/Streamed in the background
- Mostly not influenced by gameplay

Sound Effects

Sound Effects

Short files

- Triggered by gameplay
- Modified according to position, environment,…

No Sound Effects

Speech

TECHNISCHE UNIVERSITÄT DARMSTADT

Sometimes more like sound effects

- „Ouch“

Sometimes more like music

- „lalalalalala“
- „blablablabla“

Music

Music

Pitch
- Frequency
- c d e f g a h c

Duration
- Duration

Loudness
- ~Amplitude

Tone Color
- Wave form
- Instrument

Early 80s Music

Early games used simple wave forms

- Square waves
- Triangle waves
- Sawtooth waves
- Plus noise

NES

Game Boy

Master System

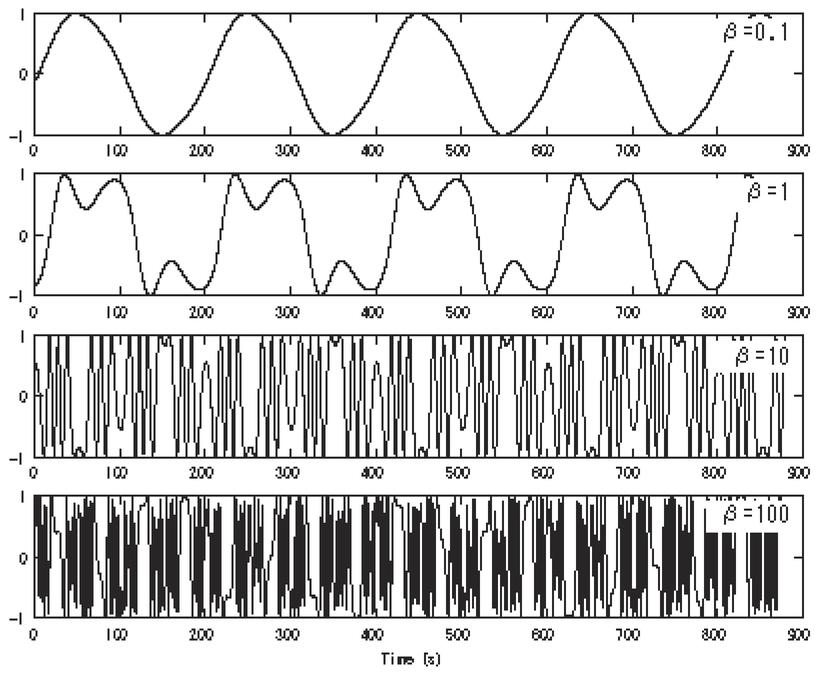
…

http://studio.substack.net

Late 80s Music

FM-Synthesis

$$s_{fm}(t) = A \cdot \cos\left(\omega_c t + \beta \cdot \cos(\omega_m t)\right)$$



AdLib

Mega Drive

Also 80s synthie music

Early 90s

„Tracker Music"

„Module Files"

Use short sound samples to represent instruments

- Change pitch as needed (or use more samples)

Amiga

SNES

(MT-32, MIDI)

Gravis Ultrasound

Dynamic Music

iMuse

Banjo Music

CDs

CDs

Plays music

No application control

- Apart from start/stop

No file loading while music plays

WAV, MP3,…

TECHNISCHE UNIVERSITÄT DARMSTADT

Play back large sound files manually

More flexible than CD audio

- But not as flexible as previous methods

Today audio compression is widely used

Orchestras

Sequencer

Sequencer

Basically works like old tracker programs

- But more and bigger samples, more effects,…

But almost always export only to wav

Sound effects back then

Originally based on square waves,…

[**http://www.bfxr.net**](http://www.bfxr.net)

Sound effects now

Recorded samples

Little to no hardware support

Number of simultaneously mixed samples usually limited

Sound Localization

Distance

- Increased distance -> Decreased amplitude (amplitude *= 1 / distance)
  - (and slightly decreased frequency)

Direction

- Interpolate between speakers
- Better quality -> add more speakers

Sound Direction

Can also be simulated using headphones

Brain analyzes sounds to infer directions

https://www.youtube.com/watch?v=8IXm6SuUigI

Sound Localization Left/Right

TECHNISCHE UNIVERSITÄT DARMSTADT

Measure time differences between ears

Loudness differences between ears

- Because of the head
- Depends highly on frequency
  - Partly used for frequencies > 800 Hz
  - Exclusively used for frequencies > 1600 Hz

Sound Localization Front/Back

Analyzes frequency differences caused by the ear forms

- Also to a lesser degree head, shoulders,…
- Sadly somewhat individual

Analyzes changes due to head movements

Doppler Effect

Frequency increases/decreases
 when sound source/receiver moves

For increasing/decreasing distance

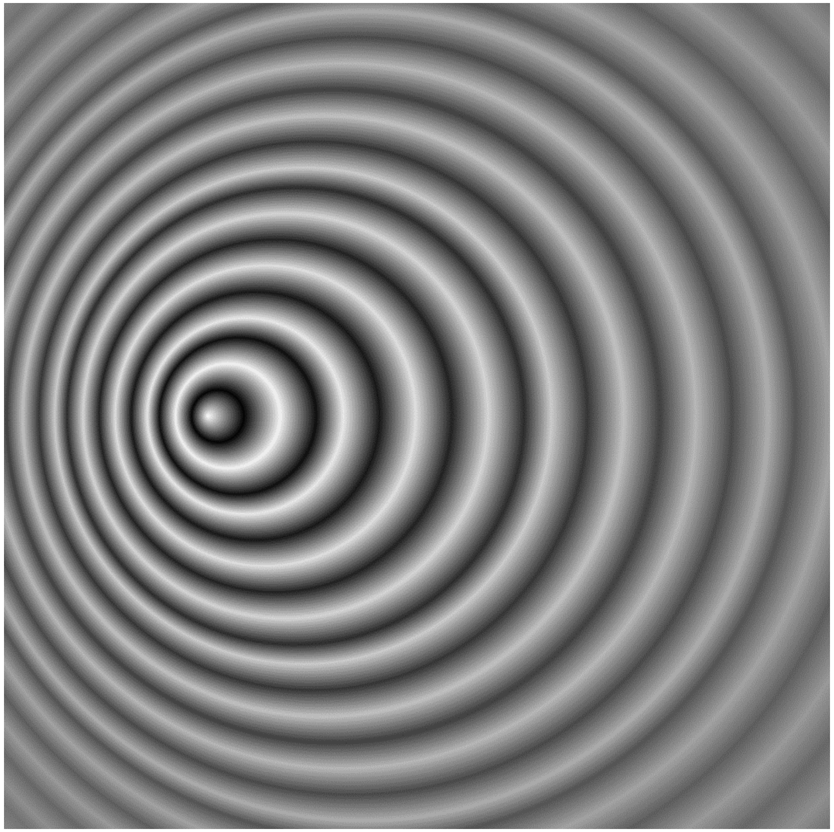
$$f_{\mathrm{B}} = f_{\mathrm{S}} \cdot \frac{c + v_{\mathrm{B}}}{c - v_{\mathrm{S}}}$$

$$f_{\mathrm{B}} = f_{\mathrm{S}} \cdot \frac{c - v_{\mathrm{B}}}{c + v_{\mathrm{S}}}$$

B: destination

S: source

c: speed of sound



Sound reflections

Highly dependent on surface structure and wavelength

Large surface, small wavelength

- Direct reflection

Rough surface

- Scatters sound

Effects

Echo, Reverb

- Direct reflections
- Replay sound with reduced amplitude

Damping

- Occluders

Calculating Reflections

Requires 3D model of the scene

- Geometry
- Surface properties

Kind of like 3D graphics

A3D

3D audio API from Aureal

Supports modelling and simulating 3D environments for sound

Only supported on special hardware

- From the late 90s

A3D today

Creative Labs bought Aureal in 2000

- After they lost a patent war in court

A3D functionality mostly integrated in EAX

EAX deprecated

EAX functionality integrated in OpenAL (EFX)

OpenAL kind of deprecated -> OpenAL Soft

Sound Effects and Music today

Mostly primitive

- Streaming prerecorded music
- Basic sound effects playback
  - Maybe some simple environmental effects

Data Oriented Design

Focus on structure of data in memory

Because computations are fast but fetching data is slow

- Game engines mostly transform data
- Object oriented programming tends to spread data all over
  - Also makes multithreading hard

OOP

```
class Bot {
  vec3 position;
  float mod;
  float aimDirection;
  // lots of additional data
  void updateAim(vec3 target) {
        aimDirection = dot(position, target) * mod;
  }
};
foreach (bot) {
  bot.updateAim();
  // more sruff
}
```

OOP

```
class Bot {
  vec3 position;
  float mod;
  float aimDirection;
  // lots of additional data <- loaded into cache line
  void updateAim(vec3 target) { <- loaded into instruction cache
          aimDirection = dot(position, target) * mod;
  }
};
foreach (bot) {
  bot.updateAim();
  // more stuff
}
```

DOD

```
struct AimingData {
  vec3* positions;
  float* mod;
  float* aimDir;
};

void updateAims(float* aimDir, const AimingData* aim, vec3 target, int
  count) {
  for (int i = 0; i < count; ++i) {
          aimDir[i] = dot(aim->positions[i], target) * aim->mod[i];
  }
}
```

DOD

Splits code and data

Focuses on optimization for memory

Makes multithreading easy

OOP

Good fit for high level code

- Aka the actual game

Mostly terrible for low level code

- Aka the engine