Student Name: Emil Sønderskov Hansen
Student ID: 20196042

Sound Processing
Mini-Project: Rain Drops on a Uniform Metal Bar

AALBORG UNIVERSITY

Introduction

This mini-project aims to synthesize the sound of raindrops falling on a uniform metal bar. A real-time implementation was created for the Unity game engine, including a plugin for the audio engine FMOD using C++. In the 3D scene, raindrops trigger the synthesized sound based on the collision position on a metal bar. Lastly, third-party tools for room modeling (geometrical acoustics) were added to enhance the realism of the synthesized raindrops.

Raindrop Model

The synthesis of liquid (raindrops) in this project is based on the paper *"Physically Based Models for Liquid Sounds"* by Doel (2005) [2]. Water is nearly soundless; however, trapped air in bubbles is audible [2]. The bubbles can be synthesized as sinusoidal sounds that decay as energy releases. The paper presents the following equation for a single bubble:

$$x(t) = asin(2\pi ft)e^{-dt}$$

The frequency f depends on the bubble's radius: $f = 3/r$, and the energy loss is given by $0.13/r + 0.0072r^{-3/2}$. Additionally, a rising pitch, which forms close to the surface, provides another perceptual cue from a raindrop. The paper proposes a time-dependent equation for frequency:

$$f(t) = f_0(1 + \sigma t)$$

The rising pitch is affected by the controllable parameter ε, as: $\sigma = \varepsilon d$. A suitable value for ε is approximately 0.1 for raindrop sounds [2].

A Matlab method was implemented based on the paper:

```
function[out] = bubbleSound(fs, maxLength, a, r)
T = 1/fs; % time step
t = (0:T:maxLength); % time vector
f0 = 3/r; % frequency
d = 0.13/r+ 0.0072*r^(-3/2); % decay
eps = 0.1; % rising pitch
sigma = eps*d;
ft = f0*(1+sigma*t); % frequency over time
x = a*sin(2*pi*ft.*t).*exp(-d*t); % bubble synthesis
out = x;
```

Uniform Metal Bar

The implementation of the uniform metal bar used Finite-Difference Time-Domain Methods from the book *"Numerical Sound Synthesis"* by S. Bilbao (2009) [1]. The uniform thin bar is based on the 1D wave equation, and a partial difference equation (PDE) is presented in the book as:

$$u_{tt} = -\kappa^2 u_{xxxx}$$

Here, κ is a stiffness parameter defined as $\kappa = \sqrt{(EI)/(\rho A L^4)}$, where ρ is the material density, and E is Young's Modulus. For this implementation, κ is simplified to a controllable parameter.

The PDE is an equation for a continuous system over space (x) and time (t). To implement this, the equation must be discretized and solved for u_l^{n+1}, where n is the next sample, and l is a grid point along the metal bar. First, a discrete approximation for time (u_{tt}) is presented as:

$$u_{tt} \approx \frac{1}{k^2}(u_l^{n+1} - 2u_l^n + u_l^{n-1})$$

Here, k is the time step ($k = 1/fs$). Second, discrete approximation for space (u_{xxxx}):

$$u_{xxxx} \approx \frac{1}{h^4}(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n - 4u_{l-1}^n + u_{l-2}^n)$$

Here, $h = ck$ represents the space traveled over a time step, where c is the speed of sound in the medium.

Finally, the discrete space and time definitions can be placed in the PDE, resulting in the final discretized equation for u_l^{n+1}:

$$u_l^{n+1} = 2u_l^n - u_l^{n-1} - \frac{k^2\kappa^2}{h^4}(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n - 4u_{l-1}^n + u_{l-2}^n)$$

The total number of grid points (N) along the metal can be found using $N = L/h$, where L is the length of the metal bar. For stability reasons, h is recalculated as $h = \sqrt{2 * \kappa * k}$.[1]

A Matlab method was developed based on this equation, incorporating a excitation vector as an input and added damping:

```
function[out] = metalBar(fs, exciter, L, lengthSoundS, excitePoint, damping)

k = 1/fs; % time step
density = 2.7000e-06; % density of aluminium
E = 68; % Youngs Module for aluminium
c = sqrt(E/density); % speed of sound for solid
h = c*k; % Get space travelled per time step
N = floor(L/h); % divide length with h
kappa = 1; % stiffness
h = sqrt(2*kappa*k);  % recalculate for stability
lengthSound = fs * lengthSoundS;

%Exite point is divided in to 10 possible points no matter what for simplicity

if(excitePoint > 10)
    excitePoint = 10;
end
if(excitePoint < 0)
```

[1] The recalculation of h is taken from Silvins slides from the lecture

```
    excitePoint = 0;
end

excitePoint = floor((N/10)*floor(excitePoint));

% create a vector with zeros for each needed sample (n, n-1,n+1) in the equation, with
    the size of total grid points

uNext = zeros(N+1, 1); % i.e., u(n+1,l)
u = zeros(N+1,1);
uPrev = u;

for n = 1:lengthSound
    if(n < length(exciter)) % excite the grid point point
        u(excitePoint) = u(excitePoint) + exciter(n);
    end

    % go through grid points and calculate next sample
    for l = 3:N-2
        uNext(l) = (2 * u(l) - uPrev(l) - ((kappa^2*k^2)/h^4) * (u(l+2)-4*u(l+1)+6*u(l)
            -4*u(l-1)+u(l-2)) + damping*k*uPrev(l))/(1+damping*k);
    end

    % take output of sample at the excited point
    y(n) = u(excitePoint);

    % current sample becomes previous, and next sample becomes current
    uPrev = u;
    u = uNext;
end

out = y;
```

The complete derivations can be found in the appendix.

Real-time Implementation for FMOD/Unity and Room Acoustics

A real-time implementation was developed for Unity, where a 3D environment controls the physical model proposed in this mini-project. The Matlab methods were rewritten in C++, and by utilizing callback methods from the FMOD API, along with initializing FMOD parameters, a dynamic library for FMOD Studio[2] (and Unity[3]) was exported and integrated into a 3D environment. In this setup, FMOD serves as the audio engine, and triggering one raindrop results in a corresponding FMOD sound event with the plugin playing (see FMOD plugin on figure 1).

For the Unity implementation, a 3D apartment environment was imported, and a 3D cylinder looking like aluminum was added. A visual particle system simulating raindrops was implemented, and these raindrops trigger the physical model when they collide with the cylinder's surface (see figure 2). The cylinder is divided into 10 excitation points with colliders, and the excitation points in the physical model is triggered based on the point of collision. The length of the 3D cylinder

[2]https://www.fmod.com/
[3]https://unity.com/

Figure 1: *The bubble generator attached to a FMOD sound event with controllable parameters*

corresponds to the length of the metal bar model in real-time. The radius of the raindrop is randomized between 0.004 and 0.007, and ε is randomized between 0.01 an 0.1.

Figure 2: *The Unity scene with the cylinder in an apartment being hit with rain drops. A collision instantiates an object with a FMOD sound event with the bubble generator, and parameters are adjusted based on the point of collision, and the length of the cylinder.*

To enhance the model's realism, the Steam Audio plugin[4] for FMOD was added. This plugin enables specialization using a generic head-related transfer function (HRTF). Additionally, by tagging all geometry in the 3D scene with Steam Audio materials, the plugin generates an arbitrary room impulse response adjusted by the listener's position in the room in real-time. This is achieved through the use of ray-based geometrical acoustics.

An attached video demonstrates the implementation, along with the C++ and C# code for Unity.

[4]https://valvesoftware.github.io/steam-audio/

References

[1] Stefan D. Bilbao. *Numerical sound synthesis finite difference schemes and simulation in musical acoustics*. John Wiley amp; Sons, 2009.

[2] Kees van den Doel. "Physically based models for liquid sounds". In: *ACM Transactions on Applied Perception (TAP)* 2.4 (2005), pp. 534–546.

1 Appendix

1.1 Derive discrete equation

$\frac{1}{k^2}(u_l^{n+1}-2u_l^n+u_l^{n-1}) = -\kappa^2\frac{1}{h^4}(u_{l+2}^n-4u_{l+1}^n+6u_l^n-4u_{l-1}^n+u_{l-2}^n)$ replace with discrete time and space

$k^2\frac{1}{k^2}(u_l^{n+1}-2u_l^n+u_l^{n-1}) = k^2-\kappa^2\frac{1}{h^4}(u_{l+2}^n-4u_{l+1}^n+6u_l^n-4u_{l-1}^n+u_{l-2}^n)$ multiply with k^2 on both sides

$$u_l^{n+1} - 2u_l^n + u_l^{n-1} = \frac{k^2\kappa^2}{h^4}(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n - 4u_{l-1}^n + u_{l-2}^n) \text{ simplify}$$

$$u_l^{n+1} = 2u_l^n + u_l^{n-1} - \frac{k^2\kappa^2}{h^4}(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n - 4u_{l-1}^n + u_{l-2}^n) \text{ solve for } u_l^{n+1}$$