GlassReverb Report

Zilin Zeng
Audio and Music Engineering Department
University of Rochester
Rochester, USA
zzeng12@u.rochester.edu

Tianwei Jiang
Audio and Music Engineering Department
University of Rochester
Rochester, USA
tjiang14@u.rochester.edu

• Abstract

We designed an audio plug-in called “GlassReverb” that digitally simulated the effect of playing audio through a liquid-filled wine glass. The mechanical impulse response was measured using an accelerometer and an inertial exciter attached to the surface of the glass. The measured impulse response was modified using the previously established model for the harmonics of wine glasses to allow a user to control the physical properties of the wine glass system in real-time.

• Introduction

Inspired by the glass harmonica, we intended to implement it in the software. Our system strictly followed a Linear Time-Invariant (LTI) system, so the audio effect was achieved by convolving the impulse response of a designed wine glass with the user’s audio input. The default impulse response was measured using the Impulse Response Measurer app from MATLAB, as shown in Figure 1.

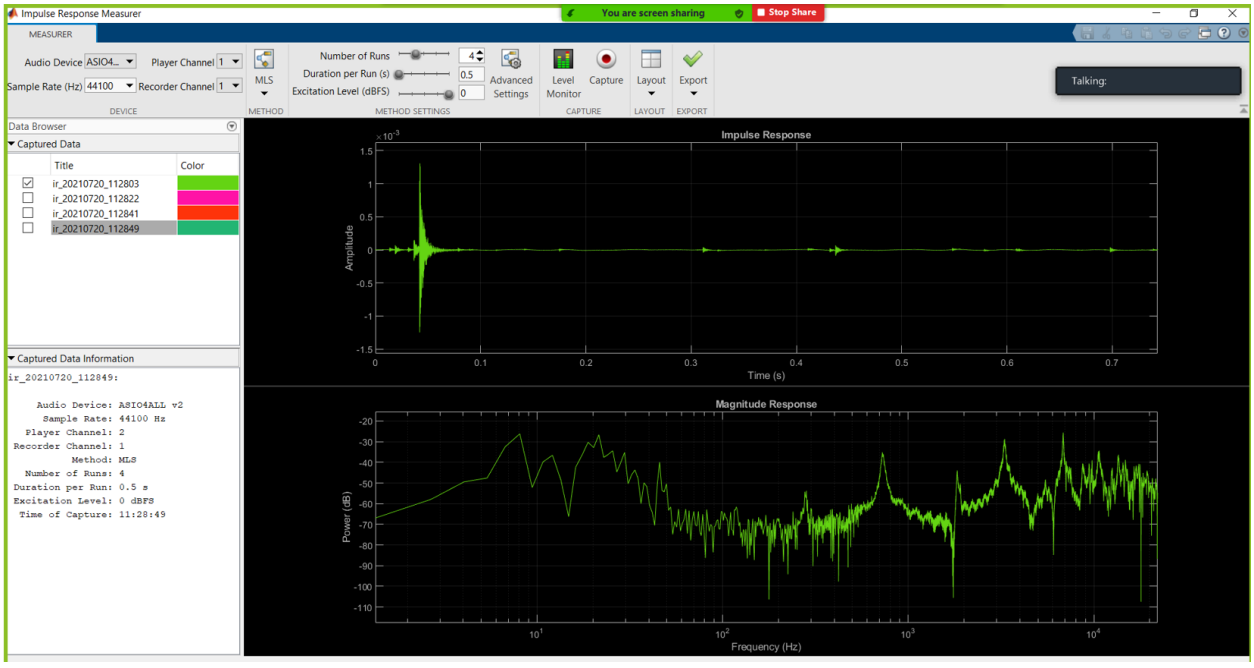

Figure 1 - Default impulse response for an empty universal-shape wine glass

Figure 2 shows our measurement setup:

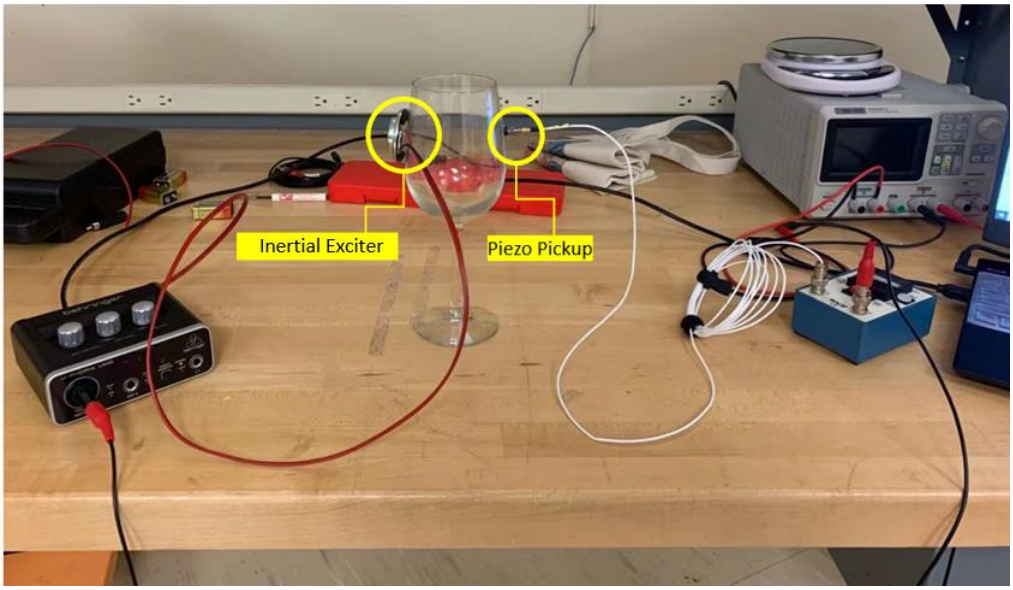

Figure 2 – measurement set up for the impulse response of an empty universal-shape wine glass

An inertial exciter was attached to the body of the wine glass. We used the Maximum Length Sequence (MLS) method that excited the wine glass with white noises played through the inertial exciter. A piezo pickup was attached to the opposite side of the glass to record the default impulse response from the wine glass due to noise excitation.

As shown in Figure 3, the user can design his (her) intended wine glass impulse response by changing the following wine glass parameters: wine glass height, wine glass rim thickness, wine glass rim radius, liquid height, and liquid density.

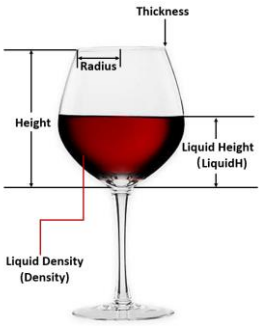

Figure 3 – user-controlled plug-in parameters

The default impulse response retrieved was the response for an empty universal-shaped wine glass. We calculated the fundamental frequency vo' for its response using this equation [1]:

$$\nu_0' = \frac{1}{2\pi}\left(\frac{3Y}{5\rho_g}\right)^{1/2}\frac{a}{R^2}.$$

Y is the Young's Modulus for glass. ρg is the density for the wine glass. a is the rim thickness for the wine glass. R is the rim radius for the wine glass.

Based on the user's input parameters on the physical properties of a wine glass, we calculated the fundamental frequency vh for a user-designed wine glass using the following equation [1]:

$$\left(\frac{\nu_0}{\nu_h}\right)^2 \approx 1 + \frac{\alpha}{5}\frac{\rho_l R}{\rho_g a}\left(\frac{h}{H}\right)^4.$$

α is a value determined by the shape of the wine glass, which typically equals 1.4. ρl is the density for the liquid. h is the height of the liquid, and H is the height for the body of the wine glass.

We then modified the default impulse response based on the ratio between calculated vh and vo'. We resampled the default impulse response using that ratio to shift the pitch of the wine glass response based on the user design. We used MATLAB to design our audio plug-in. Figure 4 shows the Graphical User Interface (GUI) for our "GlassReverb."

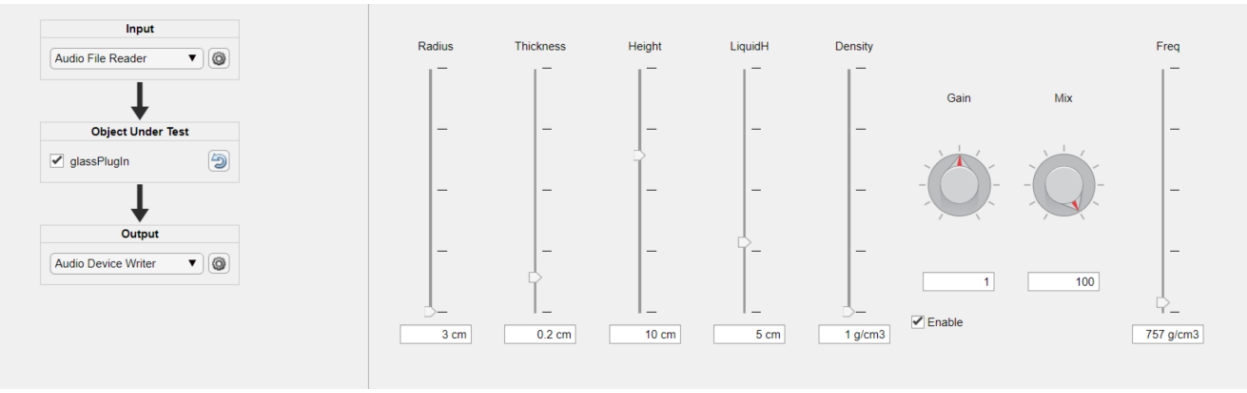

Figure 4 – plug-in GUI configuration

• MATLAB Source Code for Signal Processing:

```
%Signal Processing Part
function out = process(glassPlugIn,in)
  if glassPlugIn.Enable
    %load the recorded impulse response for an empty glass cup
    IRSelected = load('measuredIRData.mat','IRData');
    IRDataSelected = IRSelected.IRData;
    IRInput = zeros(1001,1);
    IRInput = transpose(IRDataSelected(1,1800:2800));

    Pg = 2.889; % g/cm^3 glass density
    Y  = 7*10^11; %in dyne/cm^3 glass Young's Modulus
    alpha = 1.4;
```

```
        %calculating the resonant frequency for a liquid filled glass wine cup
        FundFreq = (1/(2*pi))*sqrt((3*Y)/(5*Pg)) * (glassPlugIn.Thickness/(glassPlugIn.Radius^2)) *
sqrt(1+((4/3)*((glassPlugIn.Radius/glassPlugIn.Height)^4)));

          CalculatedFreq =
FundFreq/(sqrt(1+(alpha/5)*((glassPlugIn.Density*glassPlugIn.Radius)/(Pg*glassPlugIn.Thickness))*((glassPlugIn.WaterH/glassPlugIn.Height)^
4)));

        Sr = zeros(round(round(glassPlugIn.Freq)/757)*1001,1);

        Sr = resample(IRInput(:,1),757, CalculatedFreq)  %shift the resonant frequency of the loaded impulse response based on the calculation

        Sr = [Sr ; zeros(size(in,1)-length(Sr),1)];

        out = zeros(size(in,1),size(in,2));

         %audio signal processing part: convolve the impulse
         %response with audio input
         for i = 1 : size(in,1)
            if i == 1
               glassPlugIn.CircularBuffer(1,:) = zeros(1,size(in,2));
               glassPlugIn.CircularBuffer(1,:) = in(1,:);
            end

            if i<= (size(in,1)-1)
               out(i,1) = sum(glassPlugIn.CircularBuffer(1:size(in,1),1).*Sr(:,1));
               out(i,2) = sum(glassPlugIn.CircularBuffer(1:size(in,1),2).*Sr(:,1));
               glassPlugIn.CircularBuffer(:,1) = circshift(glassPlugIn.CircularBuffer(:,1),1);
               glassPlugIn.CircularBuffer(:,2) = circshift(glassPlugIn.CircularBuffer(:,2),1);
               glassPlugIn.CircularBuffer(1,:) = in(i+1,:);
            end

            if i == size(in,1)
               out(i,1) = sum(glassPlugIn.CircularBuffer(1:size(in,1),1).*Sr(:,1));
               out(i,2) = sum(glassPlugIn.CircularBuffer(1:size(in,1),2).*Sr(:,1));
               glassPlugIn.CircularBuffer(:,1) = circshift(glassPlugIn.CircularBuffer(:,1),1);
               glassPlugIn.CircularBuffer(:,2) = circshift(glassPlugIn.CircularBuffer(:,2),1);
            end
         end

        % mix and volume setup
        boostCoeff1 = 0.2/max(out(:,1));
        boostCoeff2 = 0.2/max(out(:,2));
        out(:,1) = out(:,1).*boostCoeff1;
        out(:,2) = out(:,2).*boostCoeff2;
        out = out.*glassPlugIn.Gain;
        out = (1-glassPlugIn.Mix/100).*in + (glassPlugIn.Mix/100).*out;
      else
        out = in;
      end
    end
```

• Conclusion:

We have demonstrated the eligibility for designing the “GlassReverb” in MATLAB. We are currently working on transforming the MATLAB code into the JUCE code to implement a stand-alone audio plug-in. We are also working on improving the plug-in’s audio quality by optimizing our codes and seeking better ways to manipulate the input signal.

Reference

[1] French, A. P. “In Vino Veritas: A Study of Wineglass Acoustics.” American Journal of Physics 51, no. 8 (1983): 688–94. https://doi.org/10.1119/1.13147.