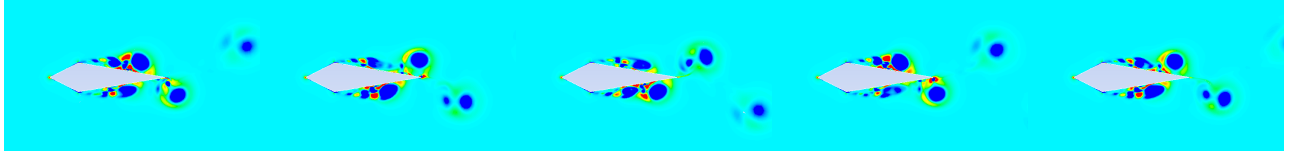


# Real-time Aerodynamic Sound Synthesis for Slender Objects

Jui-Hsien Wang\*  
Cornell University



**Figure 1:** The vortex shedding pattern of flow over a 2D sword determines its dipole characteristics and the aerodynamic sound when swung by, say, a virtual samurai.

## Abstract

In this project, I explored the problem of real-time, physics-based aerodynamic sound synthesis for slender objects. It is largely inspired by the paper by Dobashi *et al.* [Dobashi et al. 2003]. The aerodynamic sound when we swing a slender object, such as a stick, is originated from the complex interaction between the air flow and the stick. Sufficient spatial/temporal resolution was regarded to be essential to capture the physics and thus the characteristics of the sound generated. However, the required fluid simulation is too expensive to run at run-time audio stepping rate. To avoid such computation, I first precomputed a comprehensive database that contains relevant sound textures evaluated from high-quality grid-based fluid simulation, and then at runtime, this database is fetched and textures are blended to effectively resynthesize the aerodynamic swinging sound. Next, to increase the interactivity of the project, I interfaced the sound system with Leap Motion sensor to give real-time motion capture data. The system is proven to be quite reliable and can run at real-time even on a low-end laptop, and create realistic swinging sound.

**Keywords:**

## 1 Introduction

In games and virtual environments, accurately representing the sound of moving a slender object quickly, such as a character swinging a club, is important for story telling and the realism of the environment. However, most of the time, this sound is faked by either playing back experimentally recorded “canned” sound, or by using random parametric models. These models are cheap to evaluate, but they lack physical basis of how the sound is generated, and therefore can cause noticeable audio-visual synchronization problem or need a lot of hand-tuning to ensure the quality. Dobashi [Dobashi et al. 2003] presented the first automated, physically-based aerodynamic sound synthesis pipeline.

\*e-mail: jw969@cornell.edu

This class of aerodynamic sound is generated by the fluid flow around the object, and can be described by Curle’s model [?; ?]. Given two critical assumptions: (1) the object is acoustically compact, and (2) the listener is placed at a far-field position, the aerodynamic sound can be approximated by a dipole source, whose magnitude is governed by the unsteady forces generated by the object placed in the flow field. Section 3 specifies more details on this model.

Using this model, we computed fluid flow around the object of interest and the dipole sound source function, and cached them in a database for that object for runtime evaluation. At runtime, the propagation is modeled by use of free-space Green’s function, and the sound source function was used to evaluate the sound at listener’s position given the motion of object. Amplitude scaling based on Curle’s model and frequency scaling based on Strouhal number were implemented. Section 4 and Section 5 discuss these parts.

Leap Motion was configured to give a run-time motion capture data, and used as the input of our sound system. The basics of this interface will be described and demonstrated in Section 6.

## 2 Related Work

## 3 Curle’s Model

For acoustically compact object and far-field listener, the sound pressure is governed by the equation [?]

$$p(\mathbf{q}, t) = \frac{1}{4\pi c_0 r^2} (\mathbf{q} - \mathbf{o}) \cdot \mathbf{g}(t - \frac{r}{c_0}), \quad (1)$$

$$\mathbf{g}(t) = \frac{\partial}{\partial t} \mathbf{F}(t) dS. \quad (2)$$

$\mathbf{q}$  is the listener position, and  $\mathbf{o}$  is the object center position.  $c_0$  is the speed of sound,  $r$  is the distance between listener and object center, and  $\mathbf{F}$  is the aerodynamic forces of the object in the texture domain, precomputed in the fluid simulation. Given this model, we can statically sample and cache the function  $\mathbf{g}$  at different inflow speed and directions for a given object, and only evaluate the sound propagation at runtime. The runtime cost would be a data-fetching process, and a few flops to decide the position and direction of the object with respect to the listener in order to scale the sound amplitude and time delay.

## 4 Sound Texture Database Construction

The sound source function,  $\mathbf{g}$ , can be precomputed in a fluid simulation. I used proprietary finite-volume solver Ansys Fluent to do

the computation. The object being sampled was placed in a virtual wind tunnel and rotated with 5 degree interval to sample the sound source function for every inflow direction. To shorten this preprocess, the following identities are used

- (1) The amplitude of the dipole sound scales linearly with inflow speed in the log-space. That is,  $|p_v| = (v/v_0)^\alpha |p_{v_0}|$ .  $\alpha = 6$  is the coefficient Dobashi used, but I found that  $\alpha \approx 2$  works better because the high coefficient gives too wide the dynamic range.
- (2) The frequency of the dipole sound scales linearly with inflow speed. The physical argument here is based on the Strouhal frequency,  $St = fD/v$ , where  $St$  is the Strouhal number,  $D$  is the diameter of the cylinder, and  $f$  is the frequency. For a wide range of Reynold's number ( $Re = 100 \sim 1E6$ ), the Strouhal number stays roughly constant for cylinder ( $St_{cylinder} \sim 0.21$ ) and therefore for the same object, the frequency scales linearly with inflow velocity. Linear interpolation was applied to sound textures to first upsample it to avoid artifacts when we scale the textures.
- (3) For inflow with non-zero incidence angle, the sound is approximately equal to the sound created by inflow speed scaled by the geometric factor. Therefore, all the inflow directions at runtime were projected to the normal direction of the object before the sound source function was evaluated.  $sound(v_\theta) \approx sound(v_\parallel)$ ,  $v_\parallel = v \cos \theta$ .

## 5 Runtime Considerations

One of the big problems I encountered when implementing this paper was how to do the real-time audio rendering properly, the discussion of which is missing from the paper. There were two main problems associated to it: (1) the time-scale mismatch in motion capture refresh rate and audio stream sampling rate can cause severe “clicking” sound when scaling the textures or jumping between textures. (2) the audio underrun caused by improper thread priority for audio callback and high motion capture latency.

The clicking problem (either caused by resampling the texture in the amplitude/frequency shifting or jumping between textures) can be resolved by requesting an optimum buffer size. To avoid audio glitch, the audio interface (I use portaudio) reduces the number of audio callbacks by requesting an audio buffer when the audio stream is opened. The size of the buffer can be fixed or adaptive, depending on the hardware implementation and the application (for example, the CoreAudio framework in MacOS is well written, and can largely reduce the latency). What I did to resolve the clicking was to enforce the motion-capture data to be sent only when audio callback happens, and then linearly blend between the two states of motion data (sound amplitude/frequency). I found that the buffer size of  $\sim 100$  works the best. If the buffer is too large, the motion-capture data is updated infrequently and the playback will sound sluggish; whereas if its too small, then the buffer doesn't have enough time to blend the textures and clicking will occur. Note that this number is roughly the ratio between audio sampling rate (for most of them is 10000Hz) and motion-capture update refresh rate (at around 100Hz).

The audio underrun can happen when the buffer cannot be properly filled in the requested time frame, and portaudio has no choice but to fill the buffer with zeros. This problem can be severe, but can be resolved by requesting a high-priority thread to the audio callback and another high-priority thread for motion-capture. Also, unbounded time operations such as file I/O should be completely avoided or minimized in the audio callback. For the same reason,

the sound textures should all be loaded when the program starts, to avoid unnecessary underrun at the cost of higher memory footprint.

## 6 Real-time Motion Capture

### 6.1 Integration with the sound system

The motion capture system should input the state of the object in order to fetch the database. Position, speed and orientation are the required metrics to determine the sound state.

### 6.2 Using mouse cursor

### 6.3 Using Leap Motion

## 7 Result

### 7.1 Implementation Details

## Acknowledgements

## References

- DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2003. Real-time rendering of aerodynamic sound using sound textures based on computational fluid dynamics. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 22, 3.