Real-time Aerodynamic Sound Synthesis for Slender Objects

Jui-Hsien Wang*
Cornell University

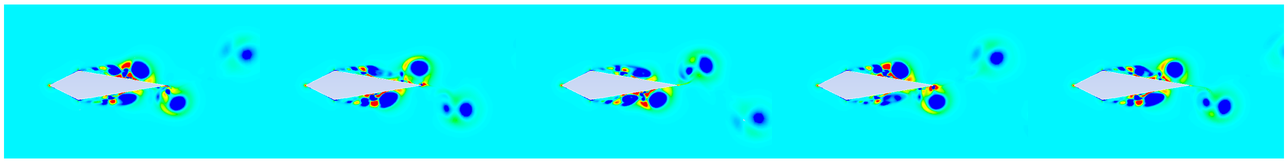


Figure 1: *The vortex shedding pattern of flow over a 2D sword determines its dipole characteristics and the aerodynamic sound when swung by, say, a virtual samurai.*

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 noticable audio-visual synchorinzation 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 acoustical 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 govenered 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 5.

2 Related Work

Fluid simulation has been used in the computer graphics community in realistic rendering of smoke, water, fire, and viscous fluid [Bridson 2008]. However, most of the work has focused on creating compelling visuals rather than audio content. A few fluid sound projects can be found: bubble sound synthesis was explored by making use of a particle-based model to augment existing incompressible fluid solvers [Zheng and James 2009]. The Minnaert model was used to mimic acoustical properties of bubbles [Doel 2005], and later extended to complement the use of fluid solvers [Moss et al. 2010]. However, the underlying physical principles of bubble sound generation are very different than those for aeolian sound. Bubble sounds are generated via individual or collective bubble oscillations, while aeolian tones are generated from the dynamical structure of vortices in the flow. Furthermore, numerical errors affect the bubble sound indirectly by altering its shape, whereas errors affect aeolian sound directly. Our approach make use of high-accuracy, mesh-based methods to minimize these errors.

Other physics-based sound synthesis work can be found for applications such as rigid-body motions [O’Brien et al. 2002; van den Doel et al. 2001], thin shells [Chadwick et al. 2009], and fractured materials [Zheng and James 2010; Chadwick et al. 2012]. However, these sounds are in principle generated by vibrating solids. The aerodynamic sound is generated by the oscillations of pressure field given a geometry.

Sound texture modeling and synthesis methods can be used to resynthesize an audio corpus. Chadwick *et al.* synthesized fire sound using a low frequency signal generated by physics-based simulation and then procedually added high-frequency content by means of bandwidth extension [Chadwick and James 2011]. An *et al.* used cloth simulation at graphics rates to drive a parametric model for both crumpling and friction events, and then applied a concatenative sound synthesis technique to map all features computed using the mel-frequency cepstral coefficients (MFCCs) to a experimentally constructed database [Steven S. An and Marschner 2012]. Other texture synthesis techniques such as inverse-FFT synthesis [**?**], and additive and subtractive synthesis also exist [Serra and Smith 1990]. Verron *et al.* built an additive synthesizer for environmental sounds, including wind sound [Verron 2010]. This model was based on an additive synthesis method, which performs analysis on an input signal to obtain a spectral envelope and a set of parameters to reconstruct the sound or, by tuning the parameters, synthesize a new sound.

Dobashi *et al.* presented a precomputation method to synthesize aerodynamic sound that is the most similar to our approach. They made use of computational fluid dynamics to sample fluid flows over a slender object to create a soundbank of aeolian sounds ("swoosh")[1], and resynthesize it at runtime based on an approach similar to the wavetable synthesis [van den Doel et al. 2001]. This is the main paper I am implementing.

3 Curle's Model

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

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

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

$\mathbf{q}$ 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.

$$sound(v_\theta) \approx sound(v_\parallel), \quad v_\parallel = v\cos\theta. \tag{3}$$

5 Runtime Considerations

5.1 Real-time motion capture

Position, speed, and orientation of the object are the metrics needed to compute the aerodynamic sound. To test the system, I first drove the sound model using mouse cursor (only speed was used), and then I configured Leap Motion to obtain the position, speed, and orientation information in real-time. Intertextural blending is introduced at the second stage.

Using mouse cursor To drive the sound model using mouse, the cursor position in screen space was queried at every audio callback, and the speed of the cursor was computed using first-order explicity Euler method. No orientation effect was introduced at this stage (therefore only one texture was considered).

Using Leap Motion Next, I sought for low latency, stable sensors to compute runtime object orientation. I have tried Nintendo Wii's controller but the bluetooth interface was not stable, and the field-of-view (FOV) of its infrared sensor was too small to resolve swinging motion. I then found Leap Motion, which is a low-cost commerical sensor released in 2013. The mature API makes it quite easy to use. The Leap Motion system recognizes and tracks hands, fingers and finger-like tools. The device operates in an intimate proximity with high precision and tracking frame rate (on my machine the peak sampling rate can reach 150Hz when lighting condition is good) and reports discrete positions, gestures, and motion. The Leap Motion controller uses optical sensors and infrared light. The sensors have FOV of about 150 degrees, making it a good candidate for the swinging application. Please see Fig. 2 for more detailed description of this system.

Knowing the direction of the hand, $\mathbf{h}$, and the normal direction of the palm, $\mathbf{p}$, the relative wind direction in the world space can be transformed to the object space. Using the identity for the slender object Eq. 3, the sound source function can then be fetched by projecting this wind direction onto the plane of the cross-section and quantized to an integer number that represents the index of the texture (5 degree interval for all cases). This process is illustrated in Fig. 3.

5.2 Clicking noise and audio underrun

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

[1]This paper showed a cavity tone demo, but contains no details of how it was done. The main focus of it was to simulate the swooshing sound.

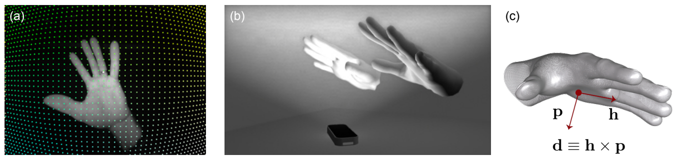
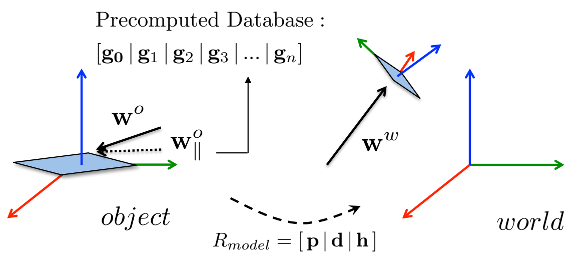


Figure 2: *Leap Motion sensor: (a) The system use IR sensors to detect the motion and gestures of the hand. The raw IR images can be used for debugging; (b) The system features a 150 degree FOV, making it a good candidate for our swinging motion; (c) The orientation of the object can be uniquely determined by transforming the wind vector into the coordinate system defined by the direction of the hand and the palm normal.*



Precomputed Database :
$[\mathbf{g}_0 \,|\, \mathbf{g}_1 \,|\, \mathbf{g}_2 \,|\, \mathbf{g}_3 \,|\, \ldots \,|\, \mathbf{g}_n]$
$\mathbf{w}^o$
$\mathbf{w}^o_{\|}$
$\mathbf{w}^w$
object
world
$R_{model} = [\,\mathbf{p} \,|\, \mathbf{d} \,|\, \mathbf{h}\,]$

Figure 3: *The wind direction, determined by the swinging motion, is first transformed into the object space, and then projected onto the plane of the cross-section. This vector, $\mathbf{w}^o_{\|}$, is then quantized into a texture index for this object to fetch the sound database, which is preloaded when the program starts.*

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 infrequenctly 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 happens when the buffer is not properly filled in the requested time frame, and portaudio has no choice but to substitute it with zeros. This problem can be severe if the implementation is not handling it explicitly. By requesting high-priority threads to the audio callback and to the motion-capture callback, the underrun problem can be alleviated if the space of sound source function for the object has reasonable size. 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 Result

6.1 Implementation Details

The project was implemented in C++. PortAudio[2] was used for audio control. Leap Motion API[3] was used to control the device and interface with the sound system developed based on Dobashi's model. All the fluid simulation data was obtained using Ansys Fluent, a proprietary finite-volume solver. The simulation ran on 32 core desktop equipped with Intel Xeon processors. More statistics of the precomputation is given in Table 1. All the simulation geometry is given in Fig. 4.

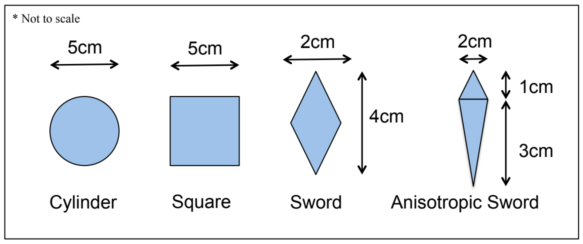


Figure 4: *Cross-section of the geometry used in the precomputation.*

6.2 UI design

The UI interface includes an OpenGL window to render the instantaneous hand position, orientation, and speed for debugging purposes. There are options to toggle on/off options, for example frequency shift. Sliders are used to adjust sensitivity and amplitude scaling exponent at runtime to speed up the iterations. Demonstration please see Fig. 5.

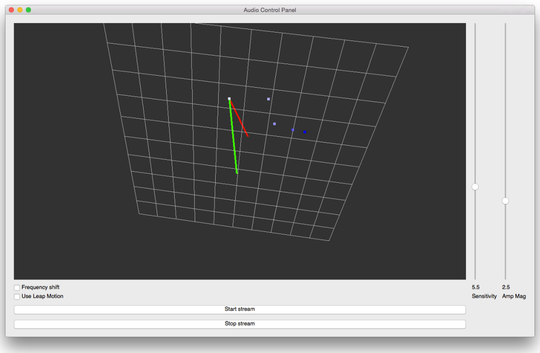


Figure 5: *UI for the real-time control.*

6.3 Motion capture data

My system run in real-time. However, for comparison purposes, one of the data sequence was recorded in order to use repeatedly to test different textures and geometries. One particular sequence was used throughout the rendering. The detailed tracking data for

[2]http://www.portaudio.com/
[3]https://developer.leapmotion.com

Table 1: *Precomputation statistics.*

Shape	Dimension	Sampling resolution (deg)	# Orientations	Cell number	Processing time / sample (mins)
Cylinder	2	5	1	18816	104.27
Square	2	5	10	42308	154.51
Sword	2	5	19	42308	123.46
Anisotropic Sword	2	5	37	31920	101.21

this sequence is shown in Fig. 6. Only the motion of the centroid of hand and tip of index finger are tracked in order to increase the frame rate as much as possible. From the figure, one can notice that the Leap Motion sensor is very reliable when palm direction is roughly aligned with table normal (that is, the silhoulette is the clearest); however, when fingers are overlapped or palm is rotated, the tracking can be noisy.

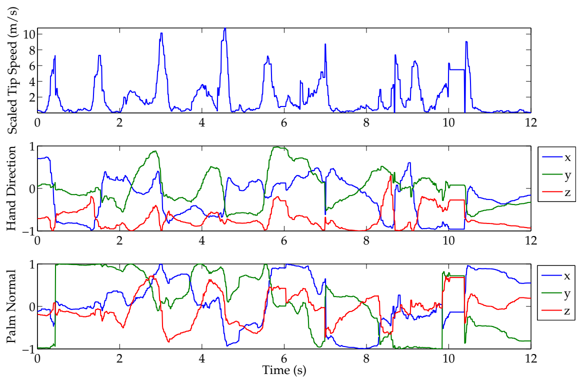


Figure 6: *Example of motion capture data. Tip speed is scaled because my finger is too short to produce any interesting sound. X: point to my right when facing the computer monitor; Y: up direction; Z: towards me.*

6.4 2D cylinder

2D cylinder is the simplest case. Because of its symmetry, in total only 1 sampling is needed if all three sampling properties are applied (introduced in Section 4). To validate the model, I ran a few more speed cases, and observe almost linear scaling in the peak frequency of the obtained sound clips (Fig. 7).

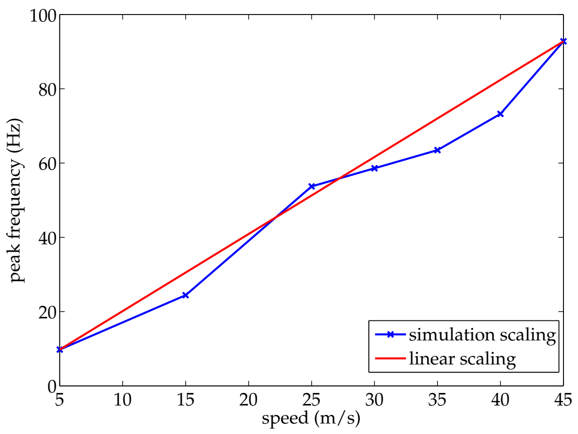


Figure 7: *The frequency scaling for cylinder introduced in Section 4 is validated by running "redundant" speed sampling.*

6.5 2D square

2D square has certain rotational symmetry so we can sample only 45(deg) of it. The sound changes only slightly when rotated but I observed increasing peak frequency (by only a little) when rotated away from the rest position shown in Fig. 4.

6.6 2D sword

2D sword needs to be sampled for 90(deg) in order to cover all its orientations. The geometry was recreated using 2D square one and therefore the mesh count is the same in Table 1. The intertextural blending is also quite challenging since many textures need to be blended. What I did was to allocate a fraction of the audio buffer to first blend between textures, and then have the rest of the buffer used on blending single texture amplitude/frequency scaling.

6.7 2D anisotropic sword

I spent the most simulation time on this test case. Unfortunately, the sampling for this object wasn't quite successful. The reason was because I changed a few simulation parameters for computing speed optimization but forgot to turned them back when it didn't work out. In the video, clear artifacts are audible when the sword is swung at certain angle.

7 Conclusion and Limitation

I found the 6th power amplitude scaling does not work quite well because of the high dynamic range of amplitude variations, instead exponent of $2 \sim 3$ seems to work the best for my dataset. The linear frequency scaling was validated in additional simulation cases for cylinder geometry. Several geometries was sampled and compared using the same motion capture data (see video). Perceptually there is not a lot of interesting variation, but it could be due to (1) the motion-capture sequence did not properly excite the textures; (2) the geometries I picked were too similar in size such that the pitch shift (based on Strouhal scaling) is not obvious enough. Leap Motion sensor is great for its cost, however, the sampling rate and stability can be further improved using (the much more expensive) Vicon system, which can reach sampling rate of 1000Hz; retroreflective markers would also help improving the tracking results instead of IR markers.

This turned out to be a very fun project as I get to play with fluid simulation, real-time sound rendering, OpenGL, motion capture hardware and some aspects of parallel computing. I didn't have time to try the 3D object simulation, the description of which in the paper is actually quite unclear. In specific, how to discretize the 3D object and represent its force oscillations in different part was not mentioned at all. One alternative is to use the unsteady force for the entire object directly for each point source, but it is really hacky. Overall, I think this is a good paper given the difficulty of the problem and the computing capability back then, this method is cheap and can produce compelling results.

References

BRIDSON, R. 2008. *Fluid Simulation for Computer Graphics*. A K Peters, Ltd.

CHADWICK, J. N., AND JAMES, D. L. 2011. Animating fire with sound. *ACM Transactions on Graphics 30*, 4 (Aug.).

CHADWICK, J. N., AN, S. S., AND JAMES, D. L. 2009. Harmonic shells: A practical nonlinear sound model for near-rigid thin shells. *ACM Transactions on Graphics* (Aug.).

CHADWICK, J. N., ZHENG, C., AND JAMES, D. L. 2012. Faster acceleration noise for multibody animations using precomputed soundbanks. *ACM/Eurographics Symposium on Computer Animation*.

DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2003. Real-time rendering of aerodynamic sound using sound textures based on computational fluid dynamics. *ACM Transactions on Graphics 22*, 3 (July), 732.

DOEL, K. V. D. 2005. Physically based models for liquid sounds. *ACM Trans. Appl. Percept. 2*, 4 (Oct.), 534–546.

MOSS, W., YEH, H., HONG, J.-M., LIN, M. C., AND MANOCHA, D. 2010. Sounding liquids: Automatic sound synthesis from fluid simulation. *ACM Trans. Graph. 29*, 3.

O'BRIEN, J. F., SHEN, C., AND GATCHALIAN, C. M. 2002. Synthesizing sounds from rigid-body simulations. In *The ACM SIGGRAPH 2002 Symposium on Computer Animation*, ACM Press, 175–181.

SERRA, X., AND SMITH, JULIUS, I. 1990. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal 14*, 4, pp. 12–24.

STEVEN S. AN, D. L. J., AND MARSCHNER, S. 2012. Motion-driven concatenative synthesis of cloth sounds. *ACM Transactions on Graphics (SIGGRAPH 2012)* (Aug.).

VAN DEN DOEL, K., KRY, P. G., AND PAI, D. K. 2001. Foleyautomatic: Physically-based sound effects for interactive simulation and animation. 537–544.

VERRON, C. 2010. *Synthèse Immersive de Sons d'Environnement (Immersive Synthesis of Environmental Sounds)*. PhD thesis, Université de Provence.

ZHENG, C., AND JAMES, D. L. 2009. Harmonic fluids. *ACM Transactions on Graphics (SIGGRAPH 2009) 28*, 3 (Aug.).

ZHENG, C., AND JAMES, D. L. 2010. Rigid-body fracture sound with precomputed soundbanks. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010) 29*, 3 (July).