PHYSICS-BASED SOUND RENDERING FOR COMPUTER ANIMATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Changxi Zheng

August 2012

PHYSICS-BASED SOUND RENDERING FOR COMPUTER ANIMATION

Changxi Zheng, Ph.D.

Cornell University 2012

The real world is full of sounds: a babbling brook winding through a tranquil forest, an agitated shopping cart plugging down a flight of stairs, or a falling piggybank breaking on the ground. Unfortunately virtual worlds simulated by current simulation algorithms are still inherently silent. Sounds are added as afterthoughts, often using "canned sounds" which have little to do with the animated geometry and physics. While recent decades have seen dramatic success of 3D computer animation, our brain still expects a full spectrum of sensations. The lack of realistic sound rendering methods will continue to cripple our ability to enable highly interactive and realistic virtual experiences as computers become faster.

This dissertation presents a family of algorithms for procedural sound synthesis for computer animation. These algorithms are built on physics-based simulation methods for computer graphics, simulating both the object vibrations for sound sources and sound propagation in virtual environments. These approaches make it feasible to automatically generate realistic sounds synchronized with animated dynamics.

Our first contribution is a physically based algorithm for synthesizing sounds synchronized with brittle fracture animations. Extending time-varying rigid-body sound models, this method first resolves near-audio-rate fracture events using a fast quasistatic elastic stress solver, and then estimates fracture patterns and resulting fracture impulses using an energy-based model. To make

it practical for a large number of fracture debris, we exploit human perceptual ambiguity when synthesizing sounds from many objects, and propose to use pre-computed sound proxies for reduced cost of sound-model generation.

We then introduce a contact sound model for improved sound quality. This method captures very detailed non-rigid sound phenomena by resolving modal vibrations in both collision and frictional contact processing stages, thereby producing contact sounds with much richer audible details such as micro-collisions and chattering. This algorithm is practical, enabled by a novel asynchronous integrator with model-level adaptivity built into a frictional contact solver.

Our third contribution focuses on another major type of sound phenomena, fluid sounds. We propose a practical method for automatic synthesis of bubble-based fluid sounds from fluid animations. This method first acoustically augments existing incompressible fluid solvers with particle-based models for bubble creation, vibration, and advection. To model sound propagation in both fluid and air domain, we weight each single-bubble sound by its bubble-to-ear acoustic transfer function value, which is modeled as a discrete Green's function of the Helmholtz equation. A fast dual-domain multipole boundary-integral solver is introduced for hundreds of thousands of Helmholtz solves in a typical babbling fluid simulation.

Finally, we switch gear and present a fast self-collision detection method for deforming triangle meshes. This method can accelerate deformable simulations and lead to faster sound synthesis of deformable phenomena. Inspired by a simple idea that a mesh cannot self collide unless it deforms enough, this method supports arbitrary mesh deformations while still being fast. Given a bounding volume hierarchy (BVH) for a triangle mesh, we operate on bounding-volume-related submeshes, and precompute Energy-based Self-

Collision Culling (ESCC) certificates, which indicate the amount of deformation energy required for the submesh to self collide. After updating energy values at runtime, many bounding-volume self-collision queries can be culled using the ESCC certificates. We propose an affine-frame Laplacian-based energy definition which sports a highly optimized certificate preprocess and fast runtime energy evaluation.

BIOGRAPHICAL SKETCH

Changxi Zheng was born in an isolated town in an eastern part of the city of Chongqing, China. He had no interaction with computers until his father bought him an early computer-learning system at his 5th class in the preliminary school. More like an entertainment toy, that system provided a very rudimentary Microsoft BASIC interpreter and a small set of APIs to program a simple NES game console. Changxi then started painstakingly learning to type his games with BASIC programs. It was those programs that planted a seed of computation in his mind.

In his high school, Changxi had a chance to participate national programming contests in China. While enjoying solving puzzles in Turbo Pascal, he also became obsessed about studying computer viruses and system exploitation, and sometimes even abandoned his academics.

Having enjoyed much fun of solving programming puzzles and studying dark arts, Changxi decided to study computer science systematically. In 2001, he attended Shanghai Jiaotong University in Shanghai, China, and completed with a Bachelor of Engineering degree in Computer Science in 2005. During his senior year in college, he was selected as a visiting student at Microsoft Research Asia in Beijing. As one of a few earliest undergraduate visiting students in that research lab, he was for the first time exposed to the scientific research frontier. This experience eventually translated his intellectual curiosity into an interest of scientific research, and motivated him to pursue a higher degree. Since 2006, Changxi has been studying for his doctorate degree in Computer Science at Cornell University.

This thesis is dedicated to my wife Xiaoying Zhou. Thank you for your loving and persistent support over the years. It is also dedicated to my parents, who dearly taxed themselves for my education and intellectual development.

ACKNOWLEDGEMENTS

As I am finally finishing this dissertation, I would like to thank and acknowledge many people for their help, support and encouragement over the years. First and foremost, my thanks to my advisor, Doug L. James for his vision and commitment to sound simulations, for his superb advice and guidance on not only specific research projects but also thinking science and engineering profoundly, and for the encouragement and generosity he has shown me along the way. Indeed, it was his enthusiastic teaching and presentation of challenging simulation problems that lured me to this area of research. He has been, and will continue to be a role model in my career.

During my graduate studies, I have been very lucky to also have had a chance to learn from some excellent mathematicians. I would like to sincerely thank Alexander Vladimirsky for guiding me approaching some simulation problems with an eye of applied mathematician and for his generous support and valuable advice. I would also like to thank John Guckenheimer for graciously serving as the advisor of my minor of applied mathematics and for giving me insightful advice. The conversation with him has always been inspiring, and nurtured me in numerous ways.

I am also grateful to the members of my thesis committee: Doug James, Steve Marschner, John Guckenheimer, Charles Van Loan, and Dexter Kozen. Their many constructive feedback and suggestions has helped me to rethink my work and improve it tremendously.

In my years at Cornell, the Computer Science Department has always been a friendly, supportive and creative environment. I have received helps from excellent students, faculty and staff. I would particularly like to thank Doug James, Steve Marschner, Joe Halpern, Emin Gün Sirer, Kavita Bala, John Hopcroft, Don

Greenberg, David Bindel, Noah Snavely, Charles Van Loan, Ashutosh Saxena and Nate Foster for their advice about presentation skills and career development.

I have been very fortunate to have had many excellent collaborators on papers related to my dissertation and even on papers that are very far from simulations. Each paper has been an ingredient that shaped my research style and vision. My thanks to Doug James, Jeff Chadwick, Yun Jiang, Marcus Lim, Ashutosh Saxena, Alexander Vladimirsky, Lusheng Ji, Dan Pei, Jia Wang, Paul Francis, Guobin Shen, Wei Pu, Shipeng Li, Scott Shenker, Ke Liang, Zaoyang Gong, and Qianni Deng.

Many thanks to my office mates and members in the computer graphics group and others in the department, who have offered a wealth of advice and enriched my wonderful experience at Cornell: Steven An, Jon Kaldor, John Moon, Shuang Zhao, Hu Fu, Wenzel Jakob, Cem Yuksel, Edgar Velazquez-Armendariz, Dustin Tseng, and Huijia Lin.

Finally, I have been deeply indebted to my family. To my parents and grandparents: Thanks for always being supportive, even though I have to be far away from you during the years. I would be nothing without you. And to my wife, Xiaoying: Thanks so much for your endless loving, encouraging and understanding. I have sacrificed much family time and left you alone at many days and nights. Thank you!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Over the past decades, computer graphics has been tremendously successful. Many endeavors in entertainment, design, education and medicine etc. have come to rely upon photo-realistic synthetic images and computer-simulated motions using sophisticated graphics techniques. This in turn motivated many researches on efficient generation of realistic visual effects.

However, our brains sense the physical world with not just our eyes but also our ears. The real world is full of sounds: a babbling brook winding through a tranquil forest, an agitated shopping cart plugging down a flight of stairs, or a falling piggybank breaking on the ground. They provide important clues to enhance human perception [VG00] of the physical world. Inevitably, computer-generated realities must incorporate a full spectrum of human sensations to enable highly immersive virtual realities. Unfortunately, current virtual worlds simulated by computer graphics models are still inherently silent. Sounds are added as afterthoughts, often using "canned sounds" which have little to do with the animated geometry and physics. Consequently, generating sounds synchronized with animated motions is expensive and laborious: production sounds rely on talented foley artists who manually record and process desired sound effects; and most interactive applications simply play prerecorded sounds triggered by specific events, but those sounds can be repetitive, expensive to store, and hard to be synchronized with user input. In the long term, the lack of realistic sound rendering methods will continue to cripple our ability to enable highly interactive and realistic virtual experiences even as the computer becomes faster.

Figure 1.1: **A piggybank breaking on the ground and producing sound**

In recent years, numerous computer simulation methods have been proposed for computer graphics applications, capturing highly detailed motions of many phenomena with affordable computation cost [OBH02, Sta99, CMT04, GHF+07, GBF03, KJM10]. The advancement of these simulation models also opens a door toward automatic procedural sound synthesis. Although researchers have started developing sound models built upon physical principles of object vibrations and sound propagation, previous work only addressed certain simple sound phenomena, and we will summarize the work in chapter 6.

In this thesis, we present a family of algorithms for physics-based sound synthesis of different types of sound phenomena, including sounds from rigid-body brittle fracture (Figure 1.1), modal contacts (Figure 1.2) and fluids (Figure 1.3). The sound phenomena we address in this thesis consider general and complex cases. Our goal is to develop computational methods practical on modern computers. We synthesize virtual sounds by capturing detailed motions that produce sound waves as well as simulating sound propagation in media. Our methods, based on physical principles, promise automatic generation of realistic sounds fully synchronized with animated geometry and physics.

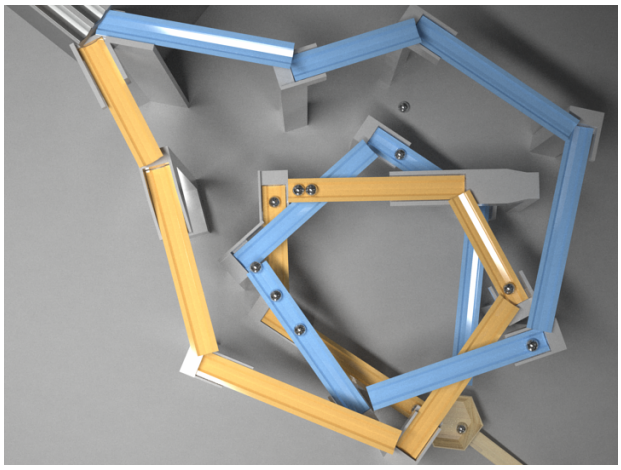


Figure 1.2: **A number of steel marbles rolling through a double helix of plastic chutes and producing rolling contact sounds**

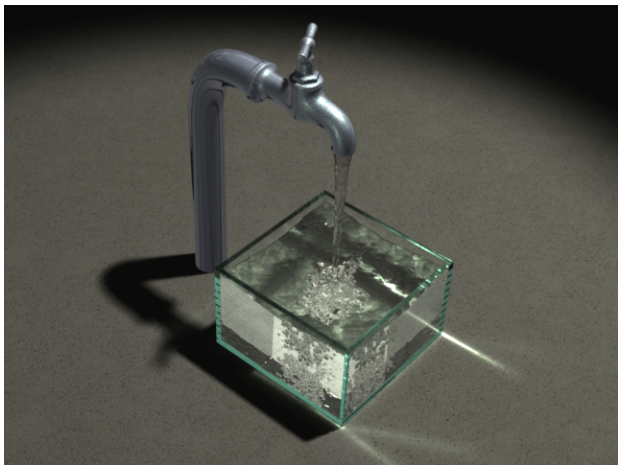


Figure 1.3: **A faucet pouring water into a water pool, producing thousands of air bubbles and generating familiar babbling sounds**

Generally speaking, a major portion of sound that we hear originates from object movement. These abrupt motions compress surrounding air or other media of objects, perturbing the medium pressure. Such disturbances then propagate as sound waves through the surrounding media. Possibly after some reflection, refraction or diffraction, those waves finally reach a listener. If those waves have frequency components in human hearing range ($20Hz$–$20kHz$), our

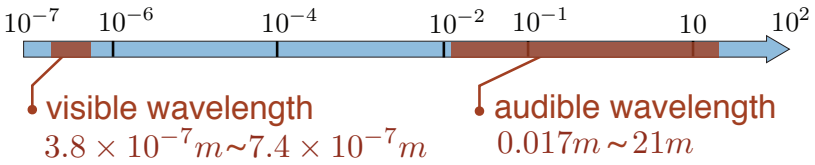
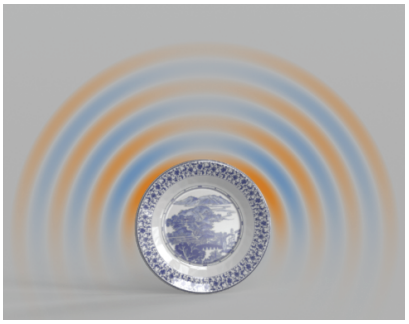


Figure 1.4: **The visible and audible wavelength ranges:** The audible wavelength is much larger than visible wavelength, and the audible wavelength range is much wider than visible wavelength range.

ear can perceive them. Computationally, in order to synthesize audible sounds, we need to model two steps of dynamics: sound *generation* and its *propagation*.

Sound Generation: To model sound sources, we need to simulate different types of motion that produce audible sounds: when we tap a coffee mug, a major part of sound comes from the mug's surface vibration excited by the tapping forces; physicists have revealed that bubbles generated when we pour water can vibrate and produce familiar water sounds; and even some air movement such as air vortices generated by swinging a sword produces audible aerodynamic sounds. Therefore, a physics-based sound rendering method must first resolve those audible motions. It is important to notice that simulating audible motions requires higher fidelity than the traditional simulations used in graphics applications. This is because human ear is capable of hearing motions up to $20kHz$ (see Figure 1.4), but those high frequency motions are often very tiny and invisible. Consequently, previous simulations for visual purposes took large integration time-steps for higher simulation performance, very often neglecting those invisible motions. When directly used for high-quality sound rendering, those simulation methods will miss audible sound details and lead to artifacts.

In this thesis, we will present high-fidelity simulation methods which are capable of capturing audible motions for sound synthesis In addition, as illustrated in chapter 3, 4 and 5, they can also improve the realism of visible motions.



Sound Propagation: Sound is a sequence of waves propagating through compressible media such as air or water[1]. When multiple waves are traveling at the same time, they interact with each other to produce a new wave. Because of the wave interaction, also called *interference*, sounding objects, such as a loud speaker, can emit sound waves with directionality, boosting sound magnitude along certain directions while suppressing it along others. When obstacles present, the sound propagation can be *reflected* or *diffracted*—that is, the fact that one can hear sounds around barriers while shadowed from the sound sources. Some medium properties affect the behavior of sound propagation as well, such as medium density, viscosity, and the motion of the medium itself. Sound waves are *refracted* when propagating through a medium which has spatially varying properties—for example, a medium that has non-uniform density distribution. In this thesis, we simulate sound propagation by solving the related partial differential equations, namely the Wave equation (2.9) and the Helmholtz equation (2.12). For practical simulation of sound propagation in complex environments, we exploit the inherent features of sound simulation, such as parallelism, pre-computation and insights from psychoacoustics. We present the details of all these methods in the following chapters.

[1]Sound can also propagate through solids, but there are additional propagation modes in solids. In this thesis, we will not consider the bulk waves propagating in solids.

1.1 Contributions and Outline

The organization and contributions of each subsequent chapter are as follows:

Chapter 2 overviews basic mathematical models that describe object vibrations and sound propagation. We first introduce linear model analysis that has been widely used to model surface vibrations of solid object. For sound propagation, we briefly introduce the governing equations: the time-domain acoustic wave equation is introduced, and its properties that have been exploited for numerical solves are presented; the frequency-domain Helmholtz equation is derived next, followed by an introduction of the multipole and far-field approximations. Finally, we also provide an overview of a pipeline for physics-based rigid-body sound synthesis.

Chapter 3 starts to present the sound rendering methods we have developed. In this chapter, we present a practical method to synthesize sounds from large-scale rigid-body fracture events [ZJ10]. Inspired by laboratory experiments, we approximate brittle fracture sounds using time-varying rigid-body sound models. A new energy-based model of fracture pattern generation is introduced to estimate "crack"-related fracture impulses. Moreover, to reduce the cost of generating sound models for complex fracture debris, we propose to replace certain fracture debris with simplified sound models. This approximation is based on the observation that sounds produced by small pieces of debris can be masked partially by large objects and become indiscernible. Those simplified sound models can be precomputed, stored in a database and quickly retrieved for runtime sound synthesis. Implemented in parallel, this approach leads to much faster runtime computation while retaining comparable sound quality.

Chapter 4 revisits the long-studied problem of frictional contacts simulation to resolve non-rigid sound details such as micro-collisions, chattering, squeaking and contact damping. Previous contact simulations were largely motivated by visual realism and thus failed to capture these details. In contrast, we simulate acoustic vibrations of objects to capture these sound details. Exploiting the transient nature of the small-scale dynamics, we propose an asynchronous simulation that adapts time-step sizes at runtime. Small time-steps are required only when micro-collisions have significant sound contributions, otherwise large time-steps are used. Consequently, we achieve a considerably faster simulation without sacrificing unprecedented small-scale sound details.

Chapter 5 switches to model sound phenomena from liquids like water. It introduces a computational model to produce liquid sounds synchronized with simulated fluid dynamics, largely based on our paper [ZJ09]. Acoustic research showed that the majority of sound from splashing liquids arises from harmonic vibrations resulting from the creation of tiny air bubbles [Lei94]. To practically simulate acoustic bubbles, We augment the existing incompressible fluid solvers for bubble creation and advection without explicitly simulating their acoustic vibrations. It is computationally expensive to accurately calculate sound propagation passing through time-varying water domain and air domain. Therefore, we approximate the associated two-phase scattering problems with one-way coupling: the sound propagation in water domain is solved first, and then the results on water surface are used as boundary conditions to solve for the air-domain propagation. These solves are implemented in parallel on distributed computing clusters. And we are able to simulate large-scale fluid

sound phenomena involving tens of thousands of bubbles and more than half a million two-phase Helmholtz solves.

Chapter 6 discusses the previous work on physics-based simulations and sound synthesis methods related to this thesis.

Chapter 7 is not directly related to sound. Instead, it presents a method of efficiently detecting self-collisions for arbitrarily deforming triangle meshes. This method can accelerate general deformable simulations, such as cloth, thin shell and elastic volumes. Provided that physics-based sound rendering algorithms desire high-fidelity simulation methods to model sound generation, this fast self-collision detection method can also improve the performance of deformable sound synthesis. Our method presented in this chapter achieves an order of magnitude speedup over traditional self-collision detection methods. Moreover, it is complementary to most of the existing self-collision detection methods, and can be used in combination with them.

Chapter 8 presents our conclusions and suggestions for future work.

Finally, we provide several appendices on the technical details and mathematical derivations appeared in our presentation in chapter 3, 5 and 7.

CHAPTER 2

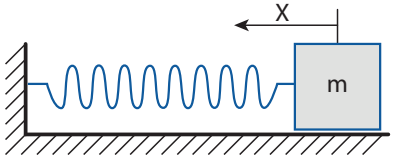
BASIC PHYSICAL MODELS OF SOUND PHENOMENA

In this chapter, we present basic physical models of sound generation and propagation. First, solid object vibration, a major type of motion that produces audible sounds, is introduced. We start from basic vibration equations that describe general object vibrations, and then introduce the linear modal vibration model widely used for rigid-body sound synthesis. In the second part of this chapter, we describe mathematical models for sound propagation, and briefly outline their computational methods. Finally, this chapter ends with an overview of the pipeline for rigid-body sound synthesis.

2.1 Modeling Sound Sources

2.1.1 Basic Vibration Model

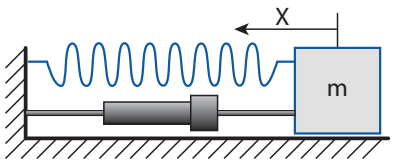
Perhaps the simplest vibration can be demonstrated using a spring-mass system, which has a single degree of freedom (DOF). And its position x is described by the equation



$$m\ddot{x} + kx = 0, \tag{2.1}$$

where k is the stiffness of the spring and m is the mass. If the spring is initially stretched by a distance of A and released, the spring starts vibrating following a sinusoid function $x(t) = A\cos(\omega t)$, where ω is the angular frequency with a value $\omega = \sqrt{k/m}$.

Damped Vibration: The spring-mass system (2.1) will vibrate forever, since the energy is conserved in this model. To model energy dissipation, one can add a "viscous" damper that outputs a damping force proportional to the velocity of the mass. Let c denote the strength of the damper, then this damped system follows an ordinary differential equation:



$$m\ddot{x} + c\dot{x} + kx = f. \tag{2.2}$$

where f, if existing, is the external force applied on the system. This is a second-order ODE, whose solution depends on the amount of damping c. If c is larger than the critical damping, $c_c = 2\sqrt{km}$, the system is over-damped; that is, no vibration motion will be observed. If $c < c_c$, then the system will vibrate, but eventually will stop vibrating over time. The 1-D vibration equation (2.2) can be numerically integrated using many different schemes. Among them, forward Euler method is probably the simplest one to implement, but suffers from being unstable with large time-step size. Higher-order methods, such as Runge-Kutta methods [But03], can be more suitable in many situations. In discrete time, the vibration can also be solved efficiently using an IIR digital filter [Ham83, JP02].

2.1.2 Elastic Vibration of Solid Object

Now we consider the vibration of an elastic object. This vibration will further perturb the pressure of surrounding air, producing propagating sound waves. Elastic vibration has been well understood using the models in *continuum mechanics*. In this section, we will not present the details of those models, but refer the readers to the books [Sha91, Fun77] for an introduction. We now directly

present the resulting equations of motion which describes the structural vibrations of an object.

The Equation of Motion: Using the finite element method (FEM), we discretize a solid object with a set of finite elements (e.g. a tetrahedral mesh). Under this discretization, the object vibration is described by the motions of n FE nodes. We represent the FE nodal positions using a vector $\boldsymbol{u} \in \mathbb{R}^{3n}$, in which each 3×1 sub-vector indicates the displacement of an FE node of the object. It satisfies the *Euler-Lagrange equation*,

$$\boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{C}(\boldsymbol{u}, \dot{\boldsymbol{u}}) + \boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{f}, \tag{2.3}$$

where $\boldsymbol{M} \in \mathbb{R}^{3n\times 3n}$ is the *mass matrix*, $\boldsymbol{C}(\boldsymbol{u}, \dot{\boldsymbol{u}}) \in \mathbb{R}^{3n}$ determines damping forces, describing the energy dissipation of the system, $\boldsymbol{R}(\boldsymbol{u}) \in \mathbb{R}^{3n}$ are internal deformation forces which resist the object from deformation, and $\boldsymbol{f} \in \mathbb{R}^{3n}$ are external forces, for example, the forces due to object collisions.

Now if we assume the considered object is rigid—in other words, the displacement $\boldsymbol{u}$ is infinitesimal, then it is sufficient to consider the linearize equation of (2.3), and we get

$$\boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{D}\dot{\boldsymbol{u}} + \boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}, \tag{2.4}$$

Here $\boldsymbol{K} = \nabla\boldsymbol{R}(\boldsymbol{0}_{3n}) \in \mathbb{R}^{3n\times 3n}$ denotes the Jacobian matrix of $\boldsymbol{R}$, evaluated at $\boldsymbol{u} = \boldsymbol{0}_{3n}$ ($\boldsymbol{0}_{3n}$ is a $3n \times 1$ zero vector). This matrix is called the *stiffness matrix*. $\boldsymbol{D}$ is the *damping matrix*. So far, there is no simple mathematical models to describe the damping mechanism in elastic materials. In many situations, one can use a *ad hoc* damping model; for example, *Rayleigh damping* is often used for ease of computation,

$$\boldsymbol{D} = \alpha\boldsymbol{M} + \beta\boldsymbol{K},$$

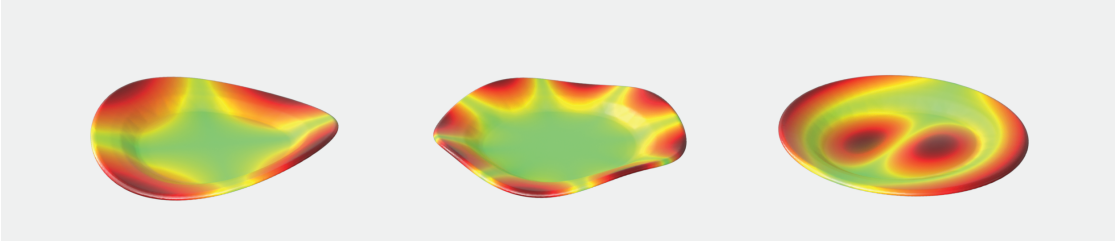


Figure 2.1: **Three vibration modes of a dinner plate:** We sampled three eigen-vectors from the eigen-matrix $\boldsymbol{U}$, and deforms the dinner plate using each eigen-vector. The magnitude of displacement of each vertex on the FE triangle mesh is colormapped.

where $\alpha \geq 0$ and $\beta \geq 0$ are two scalar parameters to weight the damping contribution from inertia and stiffness.

Linear Modal Analysis: We now present how to efficiently solve the linearized vibration equation (2.4). Notice that in (2.4), the mass matrix $\boldsymbol{M}$ is constant in time, depending only on the object's geometry and density distribution; $\boldsymbol{K}$ is constant as well, depending on the object's geometry and material. Under FE discretization, both $\boldsymbol{M}$ and $\boldsymbol{K}$ are sparse and symmetric; $\boldsymbol{M}$ is positive-definite; and $\boldsymbol{K}$ is positive-semidefinite, having null space spanned by rigid translation and linearized rotation. Using Rayleigh damping, $\boldsymbol{D}$ is also a sparse symmetric positive-definite matrix if $\alpha > 0$. We refer the reader to [OSG02] for the details of computing these matrices.

First we solve a generalized eigenvalue problem,

$$\boldsymbol{K}\boldsymbol{U} = \boldsymbol{\Lambda}\boldsymbol{M}\boldsymbol{U}$$

to get the eigen-vector matrix $\boldsymbol{U}$ and diagonal eigen-value matrix Λ. The matrix $\boldsymbol{U}$ has the property that $\boldsymbol{U}^T\boldsymbol{M}\boldsymbol{U} = \mathbf{I}_{3n\times 3n}$ ($\mathbf{I}_{3n\times 3n}$ is a $3n \times 3n$ identity matrix). We write Λ as $\Lambda = \mathsf{diag}(\omega_i^2)$, where ω_i is the *undamped natural frequency*. Intuitively, each column of $\boldsymbol{U}$ describes a vibration mode with a fixed natural

frequency (see Figure 2.1). In practice of sound synthesis, we only compute eigenvalues and eigenvectors for frequencies ω_i up to $20kHz$, which can be done efficiently using implicitly restarted Arnoldi methods (with "shift-and-invert" spectral transformation) in the ARPACK library [GVL96a, LSY98]. Substituting the modal coordinate transformation, $\boldsymbol{u} = \boldsymbol{U}\boldsymbol{q}$ into (2.4) yields

$$\ddot{\boldsymbol{q}} + (\alpha\mathbf{I} + \beta\boldsymbol{\Lambda})\dot{\boldsymbol{q}} + \boldsymbol{\Lambda}\boldsymbol{q} = \boldsymbol{U}^T\boldsymbol{f}. \tag{2.5}$$

Now this is a system of independent vibration equations, each of which is a damped harmonic oscillator (2.2),

$$\ddot{q}_i + (\alpha + \beta\,\omega_i^2)\dot{q}_i + \omega_i^2 q = Q_i, \quad i = 1 \ldots n, \tag{2.6}$$

where Q_i is the *modal force*. These oscillators can be solved individually as introduced for (2.2). Finally, object vibrations are given by the linear superposition of all vibration modes, $\boldsymbol{u}(t) = \boldsymbol{U}\boldsymbol{q}(t)$.

2.2 Propagation of Sound

The object vibration causes pressure fluctuations of the surrounding media, producing sound waves which then propagate in the media. The mathematical model describing the propagation is the acoustic wave equation.

2.2.1 Acoustic Wave Equation

First, we present a brief derivation of the acoustic wave equation. Let $p(\boldsymbol{x}, t)$ denote the pressure value in a domain Ω of the media, in which the density field is $\rho(\boldsymbol{x}, t)$ and the air particle motion field is $\boldsymbol{v}(\boldsymbol{x}, t)$. The divergence of mass flow $\rho\boldsymbol{v}$

indicates the net flow of mass of a infinitesimally small region. Mathematically, it can be expressed as

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \rho \boldsymbol{v}. \tag{2.7}$$

A pressure gradient produces net forces in a local region, and causes air to flow in the opposite direction (because the gradient vector of a function points toward increasing direction). According to Newton's second law, we have

$$\nabla p = -\frac{\partial}{\partial t}(\rho \boldsymbol{v}).$$

Taking the divergence of the pressure gradient, and exchange the order of operators on the right-hand side, we get

$$\begin{aligned} \nabla \cdot \nabla p &= -\nabla \cdot \frac{\partial}{\partial t}(\rho \boldsymbol{v}) \\ &= -\frac{\partial}{\partial t}(\nabla \cdot \rho \boldsymbol{v}) \end{aligned}$$

Substituting (2.7) produces

$$\nabla^2 p = \frac{\partial^2 \rho}{\partial t^2}. \tag{2.8}$$

Now we need to connect the pressure with the density values. From the ideal gas law we have

$$p = \rho k T$$

where k is Boltzman's constant and T is the temperature in Kelvin. Substituting it into the right-hand side of (2.8) produces

$$\nabla^2 p = \left(\frac{1}{kT}\right) \frac{\partial^2 p}{\partial t^2}.$$

If we define the speed of sound wave as $c = \sqrt{kT}$, this equation is the standard acoustic wave equation,

$$\frac{1}{c^2} \frac{\partial^2 p(\boldsymbol{x}, t)}{\partial t^2} = \nabla^2 p(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \Omega \tag{2.9}$$

where c, the speed of sound in the medium, takes a value of 343.2m/s in air at standard temperature and pressure.

Now consider a vibrating object O in the domain, producing sound waves. The effect of O's surface motion on p is introduced via the boundary condition

$$\nabla p(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = -\rho a_n(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \partial\Omega, \tag{2.10}$$

where ρ refers to density of the surrounding medium, assumed constant (1.2041kg/m^3 for air at standard pressure and 20° C). $\boldsymbol{n}(\boldsymbol{x})$ and $a_n(\boldsymbol{x}, t)$ denote the surface normal and normal acceleration of position $\boldsymbol{x}$ on O's surface.

Some properties of the wave equation can be useful for a numerical solver. First, the solution of a wave equation is *translation invariant in time*. In particular, let $p(\boldsymbol{x}, t)$ be the solution of the wave equation with boundary condition

$$\nabla p(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = b(\boldsymbol{x}, t),$$

then $p(\boldsymbol{x}, t - C)$ is the solution of the wave equation with boundary condition

$$\nabla p(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = b(\boldsymbol{x}, t - C).$$

Next, the solution is *linear* in terms of boundary condition; namely, let $p_1(\boldsymbol{x}, t)$ and $p_2(\boldsymbol{x}, t)$ are respectively the solutions of wave equation with boundary condition

$$\nabla p(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = b_1(\boldsymbol{x}, t)$$

and

$$\nabla p(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = b_2(\boldsymbol{x}, t),$$

then given a linear combination of the two boundary conditions,

$$\nabla p(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = \alpha b_1(\boldsymbol{x}, t) + \beta b_2(\boldsymbol{x}, t),$$

the corresponding solution is $\alpha p_1(\boldsymbol{x}, t) + \beta p_2(\boldsymbol{x}, t)$. These properties enable a decomposition of complex boundary conditions. For example, as presented in §2.1.2, solid object vibration can be decomposed into a superposition of individual modal vibrations. For each modal vibration, its wave propagation can be solved independently and possibly in parallel, and the final sound wave is a superposition of individual waves.

Numerous numerical methods has been proposed to solve the wave equation. Usually they discretize the domain, and approximate both the spatial and time derivative using Finite Element, Finite Difference or Finite Volume methods. Then the evolution of sound waves is simulated by explicitly or implicitly time-stepping the discretized wave equation. We refer the reader to the book [LT09] for an overview of numerical solvers of wave equations and other types of PDEs. There are also open-source (e.g., openEMS) and commercial computer software packages (e.g., Abaqus and COMSOL) to solve wave propagation problems.

2.2.2 Helmholtz Equation

The Helmholtz equation is an equivalent model of wave equation to describe sound propagation. It results from applying the technique of separation of variables on the wave equation. Assuming the wave function $p(\boldsymbol{x}, t)$ can be separated as

$$p(\boldsymbol{x}, t) = \psi(\boldsymbol{x})q(t). \tag{2.11}$$

Suppose that the time function $q(t)$ represents a harmonic oscillation of the pressure, $q(t) = e^{i\omega t}$ at a fixed frequency ω. Substituting the expression (2.11) into the

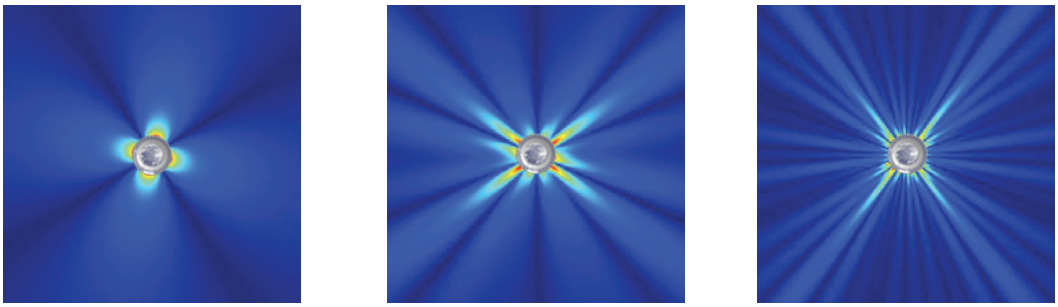


Figure 2.2: **Three transfer functions of a dinner plate:** We sample three vibration modes of a dinner plate, and compute their spatial transfer function $\psi(\boldsymbol{x})$ in free space. Their colormapped cross-sectional transfer values are shown here.

wave equation, and using a harmonic oscillation of $q(t)$, we reach the Helmholtz equation

$$\nabla^2\psi + k^2\psi = 0, \tag{2.12}$$

where $k = \frac{\omega}{c}$ is the wave number. The Helmholtz solution $\psi(\boldsymbol{x})$, called the *acoustic transfer function*, hence describes the spatially varying magnitude of a harmonic wave with a frequency ω (see Figure 2.2). Correspondingly, assuming the surface of object O is under a harmonic oscillation of frequency ω; namely the motion of a surface point $\boldsymbol{x}$ is assumed to be $\boldsymbol{v}(\boldsymbol{x})e^{i\omega t}$. Then we achieve a boundary condition for the Helmholtz equation

$$\nabla\psi(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) = -i\omega\rho\boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) \quad \boldsymbol{x} \in \partial\Omega. \tag{2.13}$$

Solution of the Helmholtz Equation

First we introduce the general solution of the Helmholtz equation, then we consider numerical solvers.

General Solution using Green's Identity: We briefly derive a general Helmholtz solution using Green's functions. A detailed derivation can be found in [Duf01]. Consider a position $\boldsymbol{y} \in \Omega$. A function ψ at that point can be written as

$$\psi(\boldsymbol{y}) = \int_{\Omega} \psi(\boldsymbol{x}) \delta(\boldsymbol{x} - \boldsymbol{y}) dV(\boldsymbol{x}), \tag{2.14}$$

where δ is the Dirac delta function. Now define $G(\boldsymbol{x}; \boldsymbol{y})$ as a Green's function solving the Helmholtz equation with a delta source term

$$\nabla^2 \psi + k^2 \psi = \delta(\boldsymbol{x} - \boldsymbol{y}).$$

Then we can substitute this equation into (2.14), and get

$$\psi(\boldsymbol{y}) = \int_{\Omega} \psi(\boldsymbol{x}) \left(\nabla^2 G(\boldsymbol{x}; \boldsymbol{y}) + k^2 G(\boldsymbol{x}; \boldsymbol{y}) \right) dV(\boldsymbol{x}), \tag{2.15}$$

Applying the Green's second integral theorem to the volume integral, we get

$$\begin{aligned} \psi(\boldsymbol{y}) = &- \int_{\Omega} [\nabla^2 \psi(\boldsymbol{x}) + k^2 \psi(\boldsymbol{x})] G(\boldsymbol{x}; \boldsymbol{y}) dV(\boldsymbol{x}) \\ &+ \int_{\partial\Omega} [G(\boldsymbol{x}; \boldsymbol{y}) \boldsymbol{n} \cdot \nabla \psi(\boldsymbol{x}) - \psi(\boldsymbol{x}) \boldsymbol{n} \cdot \nabla G(\boldsymbol{x}; \boldsymbol{y})] dS(\boldsymbol{x}) \end{aligned} \tag{2.16}$$

If we assume the pressure field ψ satisfies the homogeneous Helmholtz equation, the first term in (2.16) vanishes, and we get the *Kirchhoff integral formula* for a general Helmholtz solution

$$\psi(\boldsymbol{y}) = \int_{\partial\Omega} \left[G(\boldsymbol{x}; \boldsymbol{y}) \partial_{\boldsymbol{n}} \psi(\boldsymbol{x}) - \psi(\boldsymbol{x}) \partial_{\boldsymbol{n}} G(\boldsymbol{x}; \boldsymbol{y}) \right] dS(\boldsymbol{x}). \tag{2.17}$$

In this formula, the Green's function depends on the virtual environment to capture the effects such as reflection of ground. If we assume each sounding object is isolated from others, and ignore the scattering interaction between multiple sound objects, then we can use the free-space Green's function,

$$G(\boldsymbol{x}; \boldsymbol{y}) = \frac{e^{ik\|\boldsymbol{x} - \boldsymbol{y}\|}}{4\pi \|\boldsymbol{x} - \boldsymbol{y}\|}.$$

Evaluating (2.17) requires the acoustic transfer function values $\psi(\boldsymbol{y})$ and its outward-pointing normal derivative $\partial_{\boldsymbol{n}}\psi(\boldsymbol{y})$ on the object's surface $\boldsymbol{y} \in \Gamma$; the normal derivative is given by the Neumann boundary condition of (2.13), and the boundary transfer values are computed using the boundary element solver. In this thesis, we obtain $\psi(\boldsymbol{y})$ on Γ using the *FastBEM Acoustics* implementation (`www.fastbem.com`) of the fast multipole boundary element method [Liu09].

Multi-pole Expansion: The cost of evaluating the acoustic transfer $\psi(\boldsymbol{x})$ with (2.17) is linear in the number of surface elements, and becomes expensive for fine surface meshes. Unfortunately, breaking objects into tiny pieces almost inevitably leads to increased geometric complexity which will slow down our sound synthesis pipeline. We circumvent this issue by using a standard multipole expansion of $\psi(\boldsymbol{x})$. The Green's function can be expanded in a series of singular and regular basis functions using the identity [GD04],

$$G(\boldsymbol{x};\boldsymbol{y}) = ik \sum_{n=0}^{\infty} \sum_{m=-n}^{n} S_n^m(\boldsymbol{x}-\boldsymbol{x}_0)\, R_n^{-m}(\boldsymbol{y}-\boldsymbol{x}_0). \tag{2.18}$$

In this expansion, S_n^m is the singular spherical Helmholtz basis function

$$S_n^m(\boldsymbol{r}) = h_n^{(2)}(kr)\, Y_n^m(\theta,\phi), \tag{2.19}$$

where (r,θ,ϕ) are spherical coordinates of $\boldsymbol{r}$; $h_n^{(2)} \in \mathbb{C}$ are spherical Hankel functions of the second kind; and $Y_n^m \in \mathbb{C}$ are spherical harmonics. R_n^m in (2.18) is the regular counterpart of S_n^m, namely,

$$R_n^m(\boldsymbol{r}) = j_n(kr)\, Y_n^m(\theta,\phi), \tag{2.20}$$

where $j_n \in \mathbb{R}$ are the *spherical Bessel functions*. Finally, $\boldsymbol{x}_0$ in (2.18) is an arbitrary fixed point satisfying $\|\boldsymbol{x}-\boldsymbol{x}_0\| > \|\boldsymbol{y}-\boldsymbol{x}_0\|$ to ensure that (2.18) converges absolutely and uniformly; In practice, we place $\boldsymbol{x}_0$ at the object's center of mass.

Substituting (2.18) into (2.17), we can pull the S_n^m out of the surface integral to obtain the multipole expansion of $\psi(\boldsymbol{x})$:

$$\begin{aligned} p(\boldsymbol{x}) &= \int_\Gamma ik\Bigg[\sum_{n=0}^{\infty}\sum_{m=-n}^{n} S_n^m(\boldsymbol{x}-\boldsymbol{x}_0)\, R_n^{-m}(\boldsymbol{y}-\boldsymbol{x}_0)\,\frac{\partial\psi}{\partial\boldsymbol{n}}(\boldsymbol{y}) \\ &\qquad -\psi(\boldsymbol{y})\sum_{n=0}^{\infty}\sum_{m=-n}^{n} S_n^m(\boldsymbol{x}-\boldsymbol{x}_0)\,\frac{\partial R_n^{-m}}{\partial\boldsymbol{n}}(\boldsymbol{y}-\boldsymbol{x}_0)\Bigg] d\Gamma_{\boldsymbol{y}} \\ &= \sum_{n=0}^{\infty}\sum_{m=-n}^{n} S_n^m(\boldsymbol{x}-\boldsymbol{x}_0)\, M_n^m \end{aligned}$$

where the multipole coefficients, M_n^m, can be evaluated numerically using the formula

$$M_n^m = ik\int_\Gamma\left[R_n^{-m}(\boldsymbol{y}-\boldsymbol{x}_0)\,\frac{\partial\psi}{\partial\boldsymbol{n}}(\boldsymbol{y}) - \psi(\boldsymbol{y})\,\frac{\partial R_n^{-m}}{\partial\boldsymbol{n}}(\boldsymbol{y}-\boldsymbol{x}_0)\right] d\Gamma_{\boldsymbol{y}}, \tag{2.21}$$

with $\partial_{\boldsymbol{n}}\psi$ from the Neumann BC (2.13), and ψ from the BEM solver.

Far-field Approximation: If the distance from a sound source to a listener is large compared to the size of the object and the wave length, then the sound sources can be treated as a point source, making the far-field approximation applicable. According to the *Huyghens' integral formula*, the sound pressure contribution of a harmonic oscillation with a frequency ω is

$$p = -\frac{\rho}{2\pi}\frac{\partial}{\partial t}\int_{\partial\Omega}\frac{e^{-ikr}}{r}\frac{\partial\phi}{\partial n}ds,$$

where r is the distance from the sound source to the listener, and ϕ is the acoustic potential due to the object vibration, satisfying $\boldsymbol{v} = -\nabla\phi$. Therefore, for an object whose surface vibration is $\boldsymbol{v}(\boldsymbol{x})e^{i\omega t}$, $\boldsymbol{x}\in\partial\Omega$, the far-field approximation of acoustic transfer is

$$p = i\omega\rho\frac{e^{-ikr}}{2\pi r}\int_{\partial\Omega}\boldsymbol{v}(\boldsymbol{x})\cdot\boldsymbol{n}(\boldsymbol{x})ds$$

where the integral of surface normal velocity is called *volume velocity*.

2.3 Pipeline of Rigid-body Sound Synthesis

Provided the basic models of sound generation and propagation, we now outline the pipeline used in physics-based rigid-body sound synthesis:

1. First we simulate the motions of objects in a virtual environment, often using rigid-body simulation methods.[1] During the simulation, the time series of object positions as well as impulses applied on the objects are recorded. The object positions will be used to determine the listener's position in space relative to the object. The impulses, mostly produced by collisions between objects, will excite object vibration and produce sound.

2. The time series of impulses resulting from a rigid-body simulation then excite the object's elastic vibration introduced in §2.1.2. In particular, they are used to solve the linear vibration equation (2.4). Using linear modal analysis, each modal vibration can be solved individually using equation (2.6).

3. Next we solve for sound propagation excited by object's surface vibration. This can be done by solving the wave equation (2.9), for which the boundary condition (2.10) is determined by the object's vibrations from step 2. The position $\boldsymbol{x}$ to evaluate the sound pressure $p(\boldsymbol{x}, t)$ is the listener location relative to the sounding object. This position is time-varying, and can be determined from the time series of object positions recorded in step 1. More conveniently, we can compute sound propagation using the Helmholtz equation (2.12): Modal analysis in step 2 decomposes object vibrations into individual modes, each with a fixed natural frequency. Therefore, the sound propagation from each mode can be solved using the

[1] In chapter 4 we will show that rigid-body solver is insufficient to capture all the audible motion details.

Helmholtz equation. And all the resulting sound waves are summed together to produce final sound. This approach can easily generate stereo sounds as well: instead of evaluating sound pressure at a single listener position, we evaluate sound independently at the positions of two ears. Finally, post-processing filters might be used to capture the head-related transfer effects [Vor07].

CHAPTER 3

BRITTLE FRACTURE SOUND

In this chapter, we extend the rigid-body sound model introduced in chapter 2, and consider the sound generation for brittle fracture animations. Fracture sounds are usually important and loud parts of physically based animation and interactive virtual environments, for which an automatic and efficient sound synthesis method is desired. Current sound production methods have to rely on audio recordings of fracture events. These prerecording methods inherit shortcomings of implausibility, lack of physical synchronization, and large memory footprints to avoid repetition. Our method, motivated by laboratory experiments, approximates brittle fracture sounds using time-varying rigid-body sound models (see Chapter 2). We extend methods for fracturing rigid materials, and propose a fast quasistatic stress solver to resolve near-audio-rate fracture events. Fracture pattern and the resulting "crack"-related fracture impulses are estimated using an energy-based model. Moreover, we propose a multipole radiation model to enable level of detail control and scalable compuation of sound radiation for complex debris. Finally, to reduce sound-model generation costs for complex fracture debris, we propose Precomputed Rigid-Body Soundbanks comprised of precomputed ellipsoidal sound proxies.

3.1 Introduction

This chapter introduces a physically based approach for automatic synthesis of synchronized brittle fracture sounds for computer animation (see Figure 3.1). Despite the familiar complexity of 3D fracture animation, our fracture

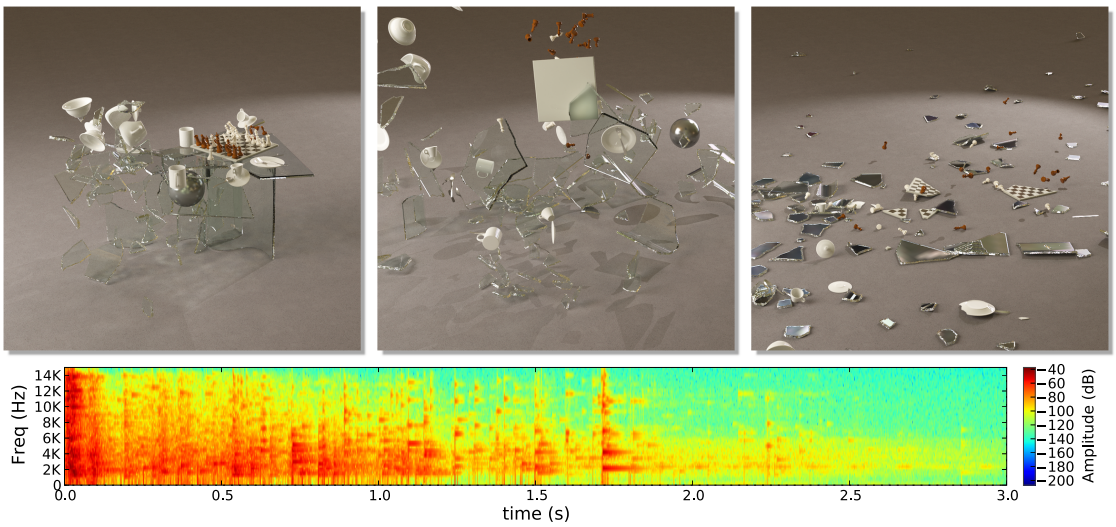


Figure 3.1: **SMASH!** We synthesize the violent fracture and impact sounds of a glass table setting smashed into over 300 pieces (see sound spectrogram). We use time-varying rigid-body sound models to approximate this brittle fracture sound by a superposition of 4046 modal vibrations (up to $14kHz$). To avoid thousand-mode modal analysis and acoustic transfer costs for complex fracture geometry, we use sound proxies sampled from Pre-computed Rigid-Body Soundbanks, here producing plausible fracture sound models at almost 500× speedup.

sound synthesis method inherits the simplicity of rigid-body sound synthesis. Based on observations from laboratory fracture experiments with high-speed video and sound recordings (see figure 3.2), we hypothesize that brittle fracture sounds can be efficiently and effectively approximated by time-varying rigid-body sound models (see Figure 3.3). Our rigid-body fracture sound synthesis has three parts: (i) a fracture preprocess which generates rigid-bodies with contact and "crack"-related fracture impulses; (ii) a parallel sound model generation phase consisting of modal and acoustic transfer analysis; and (iii) a sound synthesis phase where sounds are rendered at the listener's position.

We leverage prior work on fracturing rigid materials [BHTF07], and propose a sparse, direct, least-squares solver for the rank-deficient elastostatic problem

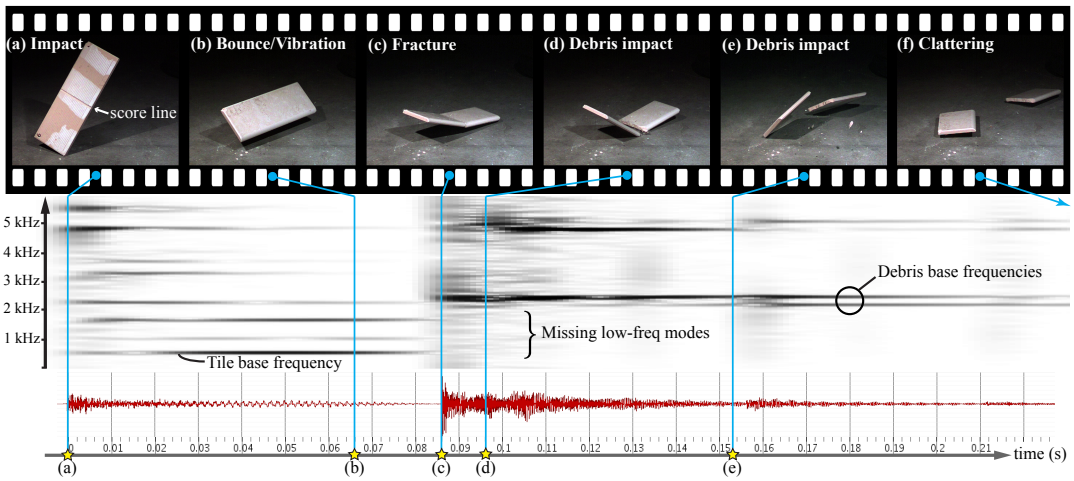


Figure 3.2: **Laboratory experiments** reveal the time-varying modal sound structure of brittle fracture events, as shown by (Top) high-speed video at 1200 Hz, (Bottom) 96 kHz sound recordings, and (Middle) frequency spectrograms (2048 bins & 93% overlap for frequency clarity). In this experiment, a pre-scored ceramic tile dropped from two feet onto a concrete floor (a) impacts without breaking; (b) bounces and vibrates with the spectrogram showing distinct tile vibration frequencies; (c) a second impact fractures the tile halfway, and produces louder vibrations with higher relative half-tile frequencies; (d-f) additional post-fracture collision events further excite the half-tile frequencies.

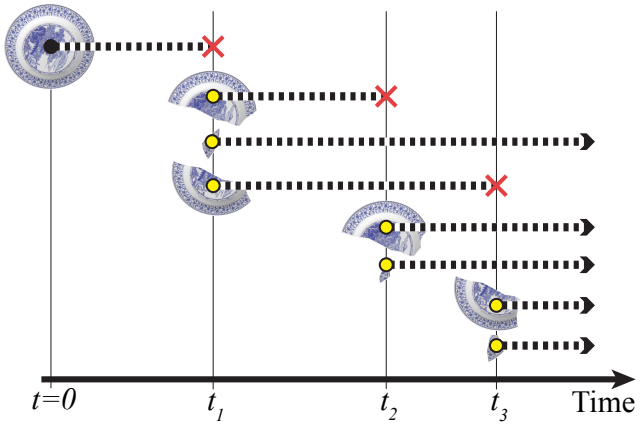


Figure 3.3: **Time-varying rigid-body sound models** are used to approximate fracture sounds. Discrete fracture events result in spectral discontinuities due to modal sound model destruction (✳) and creation (○). Fracture impulses excite created sound models.

($K\boldsymbol{u} = \boldsymbol{f}$) to resolve fracture sound events at near audio rates. An energy-based fracture model is used to model plausible fracture energy and sound generation, and also to estimate stress-based fracture impulses which excite the initially silent sound models of rigid-body debris to produce characteristically loud "crack" sounds (see Figure 3.2). The fracture simulation's fracture and contact impulses are used to excite each rigid-body's modal sound model. Frequency-domain acoustic transfer functions are computed for all vibration modes, and represented using high-order Helmholtz multipole expansions for efficient on-the-fly model generation and level of detail control. Real-time auralization and visualization of fracture simulations is possible, with GPU-accelerated transfer evaluation.

Unfortunately significant sound-model generation costs (due to modal analysis and acoustic transfer) can be a bottleneck for violent fracture processes generating hundreds of rigid-body sound models. Sadly, many small debris sounds are difficult to discern, and can be masked by larger objects or loud fracture events. We propose Precomputed Rigid-Body Soundbanks to efficiently replace many rigid-body sound models with simpler ellipsoidal sound proxies. We sample the space of material-specific ellipsoidal sound models, exploiting fundamental scale dependencies to reduce the problem to a 2D lookup table. During sound synthesis, any rigid-body can retrieve its precomputed ellipsoid proxy (indexed by its inertia matrix), and scale it to have matching fundamental frequency. Rigid-body fracture and contact impulses (from the correct geometry) are then applied to the ellipsoid proxy to produce sound. Plausible fracture sounds involving hundreds of rigid bodies can be synthesized at a fraction of the cost (see Figure 3.1).

3.2 Rigid Fracture Simulation

Our sound synthesis pipeline uses rigid-body fracture simulation to generate and animate fracture debris. Estimated contact and fracture impulses are used to excite rigid-body sound models—readers not familiar with rigid-body sound can find background on modal analysis and acoustic transfer in chapter 2. Our fracture simulator is closely related to the method for fracturing rigid materials of Bao et al. [BHTF07]. We also use an impulse-based rigid-body simulator [GBF03] since contact iteration costs amortize well over small near-audio-rate timesteps, e.g., $\Delta t = 1/10000s$ to $1/5000s$ in our examples. We propose three extensions for rigid-fracture sound synthesis: (i) a sparse, direct, least-squares solver for fast quasistatic stress analysis at high near-audio rates, (ii) an energy-based method for generating plausible fracture patterns, which we used to (iii) estimate stress-based fracture impulses that produce fracture-related "crack" sounds.

3.2.1 Fast Quasistatic Stress Analysis

Quasistatic stress analysis is commonly used to approximate brittle fracture [MDM01, BHTF07]. Given an N node tetrahedral mesh, it involves solving the elastostatic equation

$$\mathsf{K}\boldsymbol{u} = \boldsymbol{f}, \tag{3.1}$$

for the quasistatic displacement, $u \in \mathbb{R}^{3N}$, resulting from external contact forces, $\boldsymbol{f}$, in order to compute element stresses using standard methods [BW97]—we also compute the elastic strain energy, E_S, for energy-based fracture (§3.2.2). In contrast to computer animation where (3.1) is solved near or at graphics

rates [MDM01, BHTF07, SSF09], for fracture sound synthesis it is desirable to timestep the system at much higher rates, e.g., 5 kHz, to resolve micro-collisions and fracture events. In practice, large fracture simulations can request solves for hundreds of thousands of micro-impact problems.

It is well known that K is sparse, symmetric, and also rank deficient: its rank is always $3N - 6$, due to a rank-6 null space associated with translation and linearized rotation of the unconstrained rigid body. To calculate the linearized quasistatic stress distribution, we can either compute the min-norm least-squares solution to (3.1), or just solve the least-residual problem (3.1) since the rigid component of $\boldsymbol{u}$ (or $\boldsymbol{f}$) produces no stress. The iterative MINRES algorithm [PS75] can solve the least-residual problem, but suffers from slow convergence. Müller et al. [MDM01] approximated it by anchoring a number of points in the objects, thereby enforcing extraneous constraints that break momentum conservation. Bao et al. [BHTF07] solved (3.1) using a modified Conjugate Gradient method with an additional projection at each iteration to remove the null space. Unfortunately this iterative method converges slower than we would like, in part due to the difficulty preconditioning the rank-deficient K matrix.

We propose a sparse, direct least-squares solver that exploits temporal coherence, and is faster and more robust for sound applications. After a one-time setup/factorization cost per object, solutions can be obtained essentially via back substitution. We briefly sketch the method here, and defer solver details (on $\boldsymbol{\mathcal{P}}$ and V) to Appendix A.1. First, we project out the linearized rigid-body motion using an orthogonal projection $\boldsymbol{\mathcal{P}}$ to ensure that the system is compatible, i.e., that $\boldsymbol{\mathcal{P}}\boldsymbol{f} \in \mathsf{range}(\mathsf{K})$. Next we construct a special $3N \times (3N - 6)$ sparse orthogonal matrix V, then premultiply (3.1) by V^T and substitute $\boldsymbol{u} = \mathsf{V}\boldsymbol{r}$, to ob-

tain a sparse, symmetric and *full-rank* $(3N-6)\times(3N-6)$ system which can be solved directly with Cholesky factorization [GVL96a]:

$$\mathsf{V}^T\mathsf{K}\mathsf{V}\,\boldsymbol{r} = \mathsf{V}^T\boldsymbol{\mathcal{P}}\boldsymbol{f}. \tag{3.2}$$

Given that $\mathsf{V}\boldsymbol{r}$ is a least-residual solution of (3.1) for compatible RHS (see Appendix A.2 for a proof), we simply project out the particular translation and rotation chosen by V, to obtain the min-norm least-squares displacement solution, $\boldsymbol{u}^* = \boldsymbol{\mathcal{P}}\mathsf{V}\boldsymbol{r}$. In summary, each time a rigid object is created we compute and cache its sparse Cholesky factorization, $\mathsf{L}\mathsf{L}^T = \mathsf{V}^T\mathsf{K}\mathsf{V}$ and data for $\boldsymbol{\mathcal{P}}$. For each simulation timestep with nonzero contact forces, $\boldsymbol{f}$, we evaluate the least-squares quasistatic displacement incrementally (from right to left),

$$\boldsymbol{u}^* = \boldsymbol{\mathcal{P}}\mathsf{V}(\mathsf{V}^T\mathsf{K}\mathsf{V})^{-1}\mathsf{V}^T\boldsymbol{\mathcal{P}}\,\boldsymbol{f}, \tag{3.3}$$

with the primary cost being the Cholesky backsubstitution for $(\mathsf{V}^T\mathsf{K}\mathsf{V})^{-1}$; for our examples, one-time factorization costs were 0.18s–1.30s, whereas (3.3) solves cost only 0.007s–0.23s.

3.2.2 Energy-based Debris Generation

We use an energy-based method for generating plausible fracture patterns, with energy also used later for plausible sound generation (§3.2.3). To determine when brittle fracture occurs, we use the *Rankine hypothesis* [GS06] wherein material breaks when any principal stress value exceeds a given threshold. We use a Voronoi-based fracture pattern method similar to [Rag02, BHTF07], however, in contrast, we incrementally construct the fracture pattern to maintain bounded fracture energy. We estimate the energy required to generate fracture surfaces

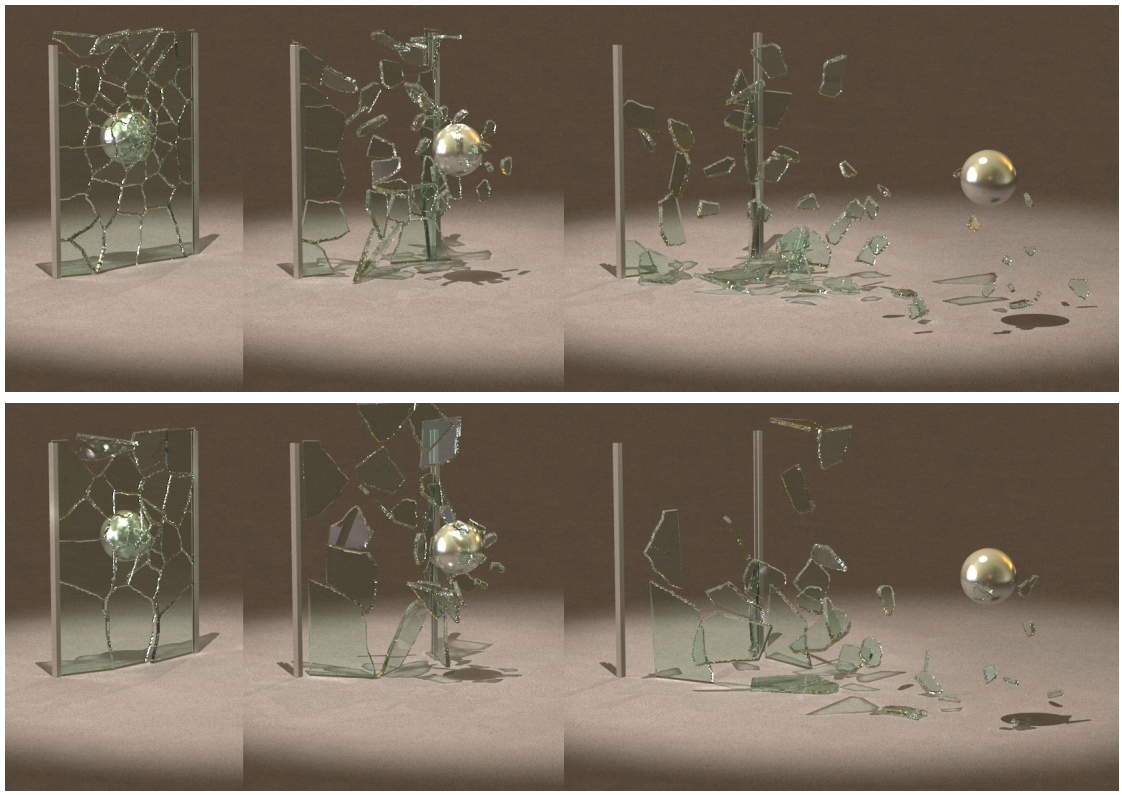


Figure 3.4: **Fracture toughness dependent sound:** (Top) A window simulated with low fracture toughness (G_c = 50 J/m^2) produces smaller debris with overall higher pitch than (Bottom) a window with higher fracture toughness ($G_c = 120\ J/m^2$).

of total area A_F by

$$E_F = G_c\, A_F,$$

where G_c is the *fracture toughness* material parameter (e.g., the *c*ritical strain energy release rate for mode-I fracture) which describes the material's ability to resist fracture; G_c values we used in examples are given in Table 3.1. To avoid excessive fracturing and uncontrolled sound generation, we require the consumed fracture energy E_F to be less than the quasistatic strain energy,

$$E_F \leq \eta\, E_S,$$

where the parameter $\eta \in (0, 1)$ controls how much strain energy is converted into fracture energy; we use η = 0.8 in our examples. The impact of fracture

toughness on sound generation is illustrated in Figure 3.4.

We generate Voronoi-like fracture patterns by incrementally sampling region "seed" points $\boldsymbol{p}_i$ (with probability proportional to strain energy density) then using a region-growing strategy. Each unassigned tetrahedral node $\boldsymbol{x}$ has prority-queue values for each region i given by the weighted distance $k_i\,\mathsf{dist}(\boldsymbol{x}, \boldsymbol{p}_i)$ where k_i is the strain energy density at $\boldsymbol{p}_i$ raised to the power $\alpha > 0$; we use $\alpha = 0.15$. Tetrahedral nodes are captured by adjacent regions with minimal queue values, and tend to produce smaller pieces in regions of high strain energy density. To ensure regions are connected, we use geodesic distances with edge-based approximations computed with Dijkstra's algorithm. After each point insertion, we estimate E_F, then continue adding points until $E_F \leq \eta\, E_S$ can not be satisfied. Multi-region elements are split [BGTG04]. Fine region meshes are generated for rendering, modal analysis and Helmholtz sound radiation analysis; our mesh edge lengths h satisfy a 20 kHz sampling condition, $h < \lambda/6 \approx 5mm$ [Liu09].

3.2.3 Fracture Impulse Estimation

Previous rigid-body fracture simulations in computer animation ignore stress-based impulses introduced by fracture events, relying instead on contact forces to push debris apart [MDM01, BHTF07]. While this can be sufficient for animation, these additional "fracture impulses" can be important contributors to debris sound. For example, snapping a candy cane in two can produce audible fracture sound even when the pieces separate cleanly without subsequent contact. We propose an energy-based model to estimate impulses from fracture events so that

Figure 3.5: **Bigger "cracks" with fracture impulses:** Our quasistatic fracture impulses produce more explosive fractures, thereby producing louder and more characteristic "crack" sounds. In contrast, simulations without fracture impulses applied (see inset) lack fracture-released kinetic energy. This ceramic chessboard ($38cm \times 38cm$) was dropped from one meter high onto the ground.

(i) explosive effects are introduced by the strain energy release, and (ii) impulses excite the modal sound models of the initially silent debris (see Figure 3.5).

Our fracture-impulse model assumes that unused quasistatic strain energy is converted into kinetic energy

$$\Delta E_K = E_S - E_F$$

and introduced by stress-based fracture impulses. For each fracture-surface triangle i, the exerted stress force $\boldsymbol{s}_i$ and torque $\boldsymbol{j}_i$ are

$$\boldsymbol{s}_i = a_i \boldsymbol{P}_i \boldsymbol{n}_i; \quad \boldsymbol{j}_i = \boldsymbol{r}_i \times a_i \boldsymbol{P}_i \boldsymbol{n}_i \tag{3.4}$$

where a_i is the triangle area, $\boldsymbol{n}_i$ is unit direction perpendicular to the triangle, $\boldsymbol{r}_i$ is the distance vector from the object's center of mass to the center of the

questioned triangle, and $\boldsymbol{P}$ is the Piola-Kirchhoff stress tensor of the tetrahedron on which the triangle is contained [BW97].

Let τ denote the effective duration to exert fracture forces (3.4). Let $\boldsymbol{v}_d^-$ and ω_d^- respectively denote the pre-fracture linear and angular velocity of the material associated with the d-th piece of debris. The post-fracture linear and angular velocity are

$$\boldsymbol{v}_d^+ = \boldsymbol{v}_d^- + \frac{\tau}{m_d} \sum_{\text{triangle } i} \boldsymbol{s}_i; \quad \omega_d^+ = \omega_d^- + \tau \mathsf{I}_d^{-1} \sum_{\text{triangle } i} \boldsymbol{j}_i, \tag{3.5}$$

where m_d and I_d are the d^{th} rigid body's total mass and angular inertia, respectively. We estimate τ by equating ΔE_K with the system's change in kinetic energy,

$$\Delta E_K = \frac{1}{2} \sum_{\text{debris } d} \left(m_d \|\boldsymbol{v}_d^+\|^2 - m_d \|\boldsymbol{v}_d^-\|^2 + (\boldsymbol{\omega}_d^+)^T \mathsf{I}_d \boldsymbol{\omega}_d^+ - (\boldsymbol{\omega}_d^-)^T \mathsf{I}_d \boldsymbol{\omega}_d^- \right),$$

which is a quadratic equation in τ, where only the smallest positive solution is physically relevant. Each time an object is fractured, we solve for τ and apply the fracture impulses to the debris.

3.3 Parallel All-Frequency Multipole Sound

We approximate each object's acoustic transfer function, $p(\boldsymbol{x})$, using a single-point multipole expansion for reliable estimates of far-field sound [GD04]. This simple representation also has three major benefits: (i) runtime level of detail control for complex fracture sound simulations (unlike [JBP06]), (ii) an efficient parallel GPU implementation, and (iii) convenient support for our Rigid-Body Soundbanks (in §3.4) to enable scalable fracture sound synthesis.

Multipole Radiation Model: The spherical multipole wave expansion of each mode's $p(\boldsymbol{x})$ takes the form

$$p(\boldsymbol{x}) \approx \sum_{n=0}^{\bar{n}} \sum_{m=-n}^{n} S_n^m(\boldsymbol{x} - \boldsymbol{x}_0)\, M_n^m \tag{3.6}$$

where $M_n^m \in \mathbb{C}$ are multipole expansion coefficients, and S_n^m are multipole basis functions (singular, radiating solutions to the Helmholtz equation). A derivation of this multipole expansion is introduced in §2.2.2 of chapter 2. In practice, we can precompute the multipole coefficients, M_n^m, for each object since they are independent of listening position $\boldsymbol{x}$. At runtime, only $(\bar{n}+1)^2$ summation terms need be computed, which is independent of the object's geometric complexity.

In our implementation, we place the multipole expansion point, $\boldsymbol{x}_0$, at the object's center of mass. Since the distance between the listener's position and the center of mass, $\|\boldsymbol{x} - \boldsymbol{x}_0\|$, is much larger than the object's diameter, the series approximation typically converges quickly to the accurate transfer function. However, convergence is frequency dependent with higher $\bar{n}$ required at higher frequencies; we use the empirical formula, $\bar{n} = \max(\frac{1}{4}kL, 4)$ [Liu09], where the length scale, L, is the object diameter; fewer terms can be used to reduce transfer-evaluation costs.

Multipole Coefficient Solver: The M_n^m coefficients can be precomputed in various ways. For example, source simulation or equivalent source methods [Och95, JBP06] directly estimate M_n^m by requiring that (3.6) matches the Neumann boundary condition in a least squares sense (note that this requires that $\boldsymbol{x}_0$ lies inside the object). In our implementation we use the fast multipole boundary element method to first solve the associated Helmholtz boundary integral problem, and then evaluate the multipole coefficients via their integral representations; we refer the reader to §2.2.2 and the book [GD04] for implementation

details. We exploit mode-level parallelism by computing transfer models on a cluster.

Parallel Sound Evaluation: Evaluating the sound pressure from all modes of all objects involves significant transfer evaluation costs, but is a pleasantly parallel computation. The sound contribution from an object's vibration mode, i, is estimated using $p_i(\boldsymbol{x}, t) = |p_i(\boldsymbol{x})|\, q_i(t)$. The total fracture sound is the superposition of all object mode contributions:

$$\begin{aligned} \mathsf{sound}(\boldsymbol{x}, t) &= \sum_{\text{mode } i} |p_i(\boldsymbol{x})|\, q_i(t) \\ &= \sum_{\text{mode } i} \left| \sum_{n=0}^{\bar{n}_i} \sum_{m=-n}^{n} S_n^m(\boldsymbol{x} - \boldsymbol{x}_{0i}; i)\, M_n^m(i) \right| q_i(t) \end{aligned} \tag{3.7}$$

where $M_n^m(i)$ are precomputed for mode i using (2.21). In our parallel implementation, we evaluate $\{p_i(\boldsymbol{x})\}$ on the GPU (using NVIDIA's CUDA API) sampled in time at high graphics rates (200 Hz), then copy it back to the CPU, and (optional) apply an HRTF filter. Modal amplitudes $q_i(t)$ are computed using an IIR filter in parallel on multi-core CPUs at audio sample rates (although GPU implementations are possible [TO09]), then each q_i is multiplied by $|p_i(\boldsymbol{x})|$ (with p interpolated up to audio rates) to obtain the mode's sound contribution, $|p_i(\boldsymbol{x})|q_i(t)$. The GPU evaluation of $p_i(\boldsymbol{x})$ values exploits thread-level parallelism across modes (i), and in p_i's (m, n)-summation using parallel reduction [Har07]. Furthermore, all modes' multipole basis functions, $S_n^m = h_n^{(2)}(kr)Y_n^m(\theta, \phi)$ are first computed in parallel using a multi-pass GPU algorithm:

- In a first pass, one thread per mode computes the $h_n^{(2)}(kr)$ values for $n = 0 \ldots \bar{n}_i$ (with $kr = k_i r_i$) using the recurrence relation, $h_{n+1}^{(2)}(kr) = \frac{2n+1}{kr} h_n^{(2)}(kr) - h_{n-1}^{(2)}(kr)$.

- Next $Y_n^m(\theta, \phi)$ is computed in two passes: (1) one thread per mode computes $Y_m^m(\theta, \phi), m = 0 \ldots p_i$ using the recurrence relation between Y_m^m and Y_{m-1}^{m-1} [PTVF07], then (2) one thread per mode per m value computes $Y_n^m, n = m + 1 \ldots \bar{n}_i$ using the recurrence relation between Y_n^m and Y_{n-1}^m.

3.4 Precomputed Rigid-Body Soundbanks

Complex fracture sounds can involve very large numbers of rigid-body sound sources, each of which requires computing an expensive, object-specific sound model. For complex fracture scenarios, the simulation bottleneck quickly becomes rigid-body sound model generation (modal analysis, and multipole radiation analysis). Unfortunately, the opportunities for precomputation are limited by the unpredictable nature of fractured geometry generation.

Ironically, for complex fracture scenarios, it can be difficult to fully discern each individual rigid-body sound. Sounds produced by small objects can be masked partially by large pieces of debris. We exploit this perceptual ambiguity by augmenting debris with simple precomputed sound models of similar frequency content. Specifically, we propose to use ellipsoid-shaped sound proxies during sound synthesis (see Figure 3.6) since (i) ellipsoids provide the smoothest shape matching the rigid-body inertial mass, and (ii) the smooth surface allows us to parameterize the proxy contact location using contact normal. While the rigid-body fracture simulation and contact impulses are based on the original fractured geometry, external forces can be applied to the proxy's sound model, thereby avoiding the bottleneck of model-specific sound model generation.

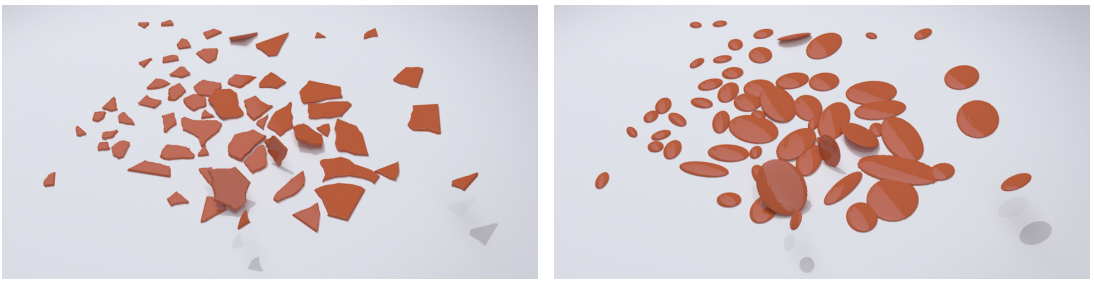


Figure 3.6: **Fracture debris and ellipsoidal sound proxies**

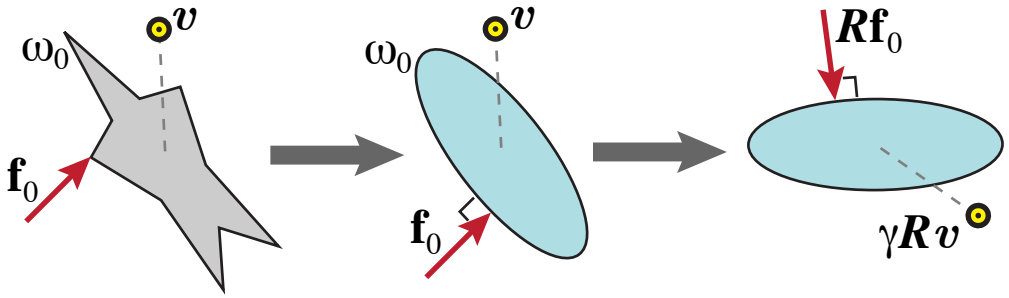


Figure 3.7: **Ellipsoidal sound proxies** are indexed by each object's inertia matrix, and scaled γ to have matching base frequency, ω_0. Contact forces $\boldsymbol{f}_0$ and relative listening positions $\boldsymbol{v}$ are rotated R to the precomputed proxy frame for sound synthesis.

For each material we precompute a Rigid-Body Soundbank, where each sound model has ellipsoidal geometry, mode shapes U, frequencies ω, and precomputed multipole coefficients M_n^m. To generate object sounds, we first retrieve its closest soundbank proxy based on an inertia tensor metric, and scale it to match our object's base frequency. Next we map the external forces to the proxy, load precomputed eigendata and multipole M_n^m values, and synthesize the sound at the relative position. This process is illustrated in Figure 3.7, and summarized in Algorithm 1.

Algorithm 1: **Runtime use of Rigid-Body Soundbanks**

input: The soundbank S; external forces $\boldsymbol{f}$
foreach *sounding object obj* **do**
  if `use_proxy(`obj`)` **then** `// §3.4.4`
    $\boldsymbol{a} \leftarrow$`find_best_proxy(`$obj$`,`S`)` `// §3.4.4`
    $\gamma \leftarrow$`scale_factor(`obj`,`$\boldsymbol{a}$`)` `// §3.4.4`
    R $\leftarrow$`rotation(`obj`,`$\boldsymbol{a}$`)` `// §3.4.2`
    $\boldsymbol{f}' \leftarrow$`map_forces(`$\gamma$`,`R`,`$\boldsymbol{f}$`)` `// §3.4.2`
    $(\boldsymbol{\omega}, \mathsf{U}) \leftarrow$`load_eigen(`$\boldsymbol{a}$`,`S`)`
    $(\boldsymbol{\omega}', \mathsf{U}') \leftarrow$`scale_eigen(`$\boldsymbol{\omega}$`,`U`,`$\gamma$`)` `// §3.4.1`
    $M \leftarrow$`load_moments(`$\boldsymbol{a}$`,`S`)`
    $M' \leftarrow$`scale_moments(`M, γ`)` `// §3.4.1`
    `generate_proxy_sound(`$\boldsymbol{\omega}$, U′,M',$\boldsymbol{f}'$`)`
  else
    `directly_generate_sound(`obj`)`
  end
end

3.4.1 Exploiting Scale Dependence

Naïve sampling of sound models over the 3D space of ellipsoidal shapes will lead to significant memory requirements and/or poor shape resolution. We therefore exploit the fact that uniformly scaling a rigid-body model does not fundamentally alter its modal vibration and multipole radiation models, but only induces power-law scalings. By precomputing a sound model for one rigid-body shape, we obtain sound models for all scaled versions. In particular, if the geometry of an object is scaled by γ, then the following scalings result (see Appendix A.3 for derivations):

$$\begin{array}{cccccc} \boldsymbol{x} & \omega & k\boldsymbol{x} & \alpha+\beta\omega^2 & \mathsf{U} & M_n^m \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \gamma\,\boldsymbol{x} & \gamma^{-1}\omega & k\boldsymbol{x} & \alpha+\beta\gamma^{-2}\omega^2 & \gamma^{-3/2}\mathsf{U} & \gamma^{-5/2}M_n^m \end{array} \tag{3.8}$$

In order to guarantee all-frequency sound up to a high-frequency cutoff, e.g., 16 kHz, we note that shrinking models ($\gamma < 1$) increases their frequency range

since $[\omega_{min}, \omega_{max}] \rightarrow [\frac{\omega_{min}}{\gamma}, \frac{\omega_{max}}{\gamma}]$, and therefore will preserve the all-frequency property. In contrast, enlarging ($\gamma > 1$) may not since $\frac{\omega_{max}}{\gamma}$ may drop below the high-frequency cutoff. We therefore precompute all-frequency models which are as large as needed to ensure $\gamma < 1$.

3.4.2 Ellipsoidal Sound Proxies

Normalized ellipsoids: Our soundbank is based on precomputing sound models for axis-aligned ellipsoids,

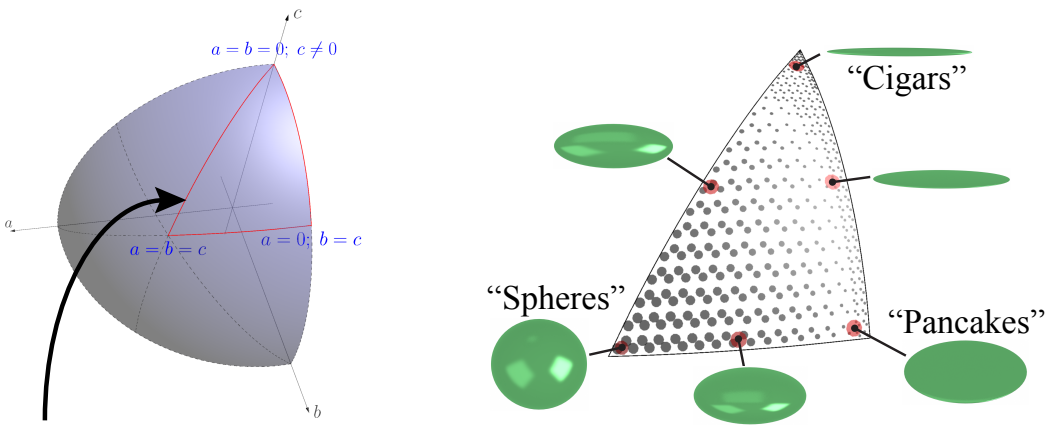
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1, \tag{3.9}$$

where the ellipsoid's shape is parameterized by the lengths of the principal axes, $\hat{\boldsymbol{a}} = (a, b, c)^T$. To avoid sampling this three-dimensional (a, b, c) parameter space, we exploit scaling dependences (§3.4.1) to eliminate scale via the normalization,

$$\|\hat{\boldsymbol{a}}\|_2 = 1\, meter \quad \Leftrightarrow \quad a^2 + b^2 + c^2 = 1, \tag{3.10}$$

effectively reducing the dimensionality from three to two. Furthermore, we can assume that $a \leq b \leq c$, to reduce the parameter space to a small triangular patch on the unit sphere (see Figure 3.8(Left)). For each ellipsoid, we conduct the modal analysis, and solve the boundary integral problems to precompute M_n^m values; the eigen-modes, frequencies, and M_n^m values are stored in the soundbank.

Inertia-matrix parameterization: Given a rigid body's symmetric inertia matrix, $\mathbb{I} \in \mathbb{R}^{3\times3}$, we identify the rigid-body's ellipsoidal sound proxy using the principal moments of inertia. These are obtained from the eigenvalue decomposition, $\mathbb{I} = \mathsf{V}_I \Lambda_I \mathsf{V}_I^T$, where V_I are the orthogonal eigenvectors (specifying the principal axes of inertia), and the principal moments of inertia are the diagonal



(a) Spherical patch (b) Ellipsoid sample space

Figure 3.8: **Ellipsoid sampling on the unit sphere**

values of $\Lambda_I = \mathsf{diag}(I_1, I_2, I_3)$, with $I_1 \geq I_2 \geq I_3$. The Λ_I are directly related to the normalized ellipsoid parameters, $\hat{\boldsymbol{a}}$: the corresponding unit ellipsoid proxy is parameterized by [1]

$$\hat{\boldsymbol{a}} = \frac{1}{\sqrt{I_1 + I_2 + I_3}} \begin{pmatrix} \sqrt{-I_1 + I_2 + I_3} \\ \sqrt{+I_1 - I_2 + I_3} \\ \sqrt{+I_1 + I_2 - I_3} \end{pmatrix} \quad \in \mathbb{R}^3. \tag{3.11}$$

Proxy contact forces are estimated by mapping rigid-body contact forces to the proxy as follows. Let $\boldsymbol{f}_0$ denote an external rigid-body force defined in the object's material frame. The equivalent *proxy contact force* is $\boldsymbol{f}_p = \mathsf{R}\boldsymbol{f}_0$, where the precomputed rotation matrix $\mathsf{R} = \mathsf{V}_I^T$ maps forces from the object's material frame to the proxy's material frame. The *proxy contact point* is estimated uniquely by assuming that the contact force is applied in the normal direction, which is easily computed by solving a 3×3 linear equation (see Appendix A.4).

[1] This follows from the ellipsoid's principal moments of inertia,
$I_1 = \frac{m}{5}(b^2 + c^2)$, $I_2 = \frac{m}{5}(a^2 + c^2)$, $I_3 = \frac{m}{5}(a^2 + b^2)$.

Material	Density	Young's mod.	Poisson	G_c
Glass	2600 kg/m^3	6.2×10^{10} GPa	0.2	80 J/m^2
Ceramic	2700 kg/m^3	7.2×10^{10} GPa	0.19	200 J/m^2

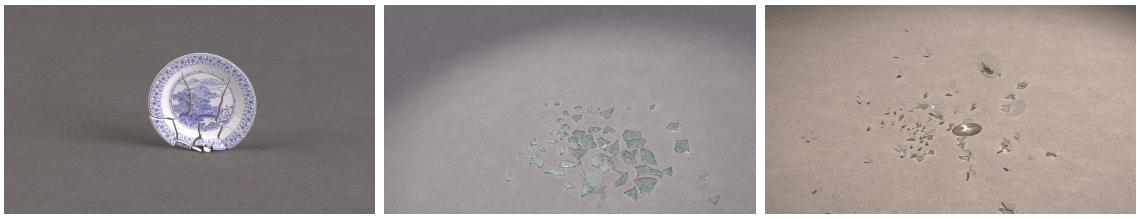
Table 3.1: Material Parameters

Proxy sound mapping: We evaluate each proxy's sound in its axis-aligned frame. Given the listening position vector in the rigid-body's center-of-mass frame, $\boldsymbol{v}$, we simply rotate and scale the vector into the proxy frame, $\boldsymbol{v} \rightarrow \gamma \mathsf{R} \boldsymbol{v}$, and evaluate the proxy sound there (see Figure 3.7).

3.4.3 Soundbank Sampling

In practice it is desirable to precompute and store as little soundbank information as possible. To avoid uniformly sampling the spherical patch in ellipsoid parameter space, we use a simple adaptive strategy to resolve faster modal frequency variations in ellipsoid samples near shape singularities at the "pancake" and "cigar" vertices (see Figure 3.8(Right)).

Material-specific Soundbanks: In theory, a different soundbank must be precomputed for each homogeneous material, e.g., for our glass and ceramic material parameters (see Table 3.1). Runtime adjustments can still be made to scale, and Rayleigh damping parameters, α and β. However, often plausible material parameters are similar, e.g., glass and ceramic, and we precompute a single "glass" soundbank and scale the models appropriately.



Dinner Plate Glass Slab Wine Glass

Figure 3.9: **Fracture simulation images**

3.4.4 Proxy Retrieval and Scaling

Proxy use criterion: In practice, we use sound proxies for sufficiently small debris with base frequencies above a user-specified threshold, ω_{proxy}. Given a rigid-body candidate for proxy replacement, we first compute the lowest eigen-frequency ω_0 of the object. A rigid body will only use the proxy when $\omega_0 > \omega_{proxy}$ is satisfied. Note that this single-frequency ω_0 calculation is far cheaper than computing the object's modal sound model.

Proxy retrieval: Given a rigid body for proxy replacement, we first compute its normalized parameter vector, $\hat{\boldsymbol{a}}$, then select the soundbank ellipsoid, $\boldsymbol{a}'$, with the minimum Euclidean distance, $\|\boldsymbol{a}' - \hat{\boldsymbol{a}}\|_2$. We use a kd-tree to accelerate this closest-point query.

Equi-frequency scaling: The retrieved ellipsoid proxy is unscaled. Instead of using inertia to scale the object, which can be a poor indicator of sound frequency, we select the proxy scale γ such that the proxy and rigid body have the same base eigen-frequency, ω_0, since it is an important perceptual attribute to match [MCR04]. Specifically, if the unscaled proxy's lowest eigen-frequency is ω_{p0}, then the desired scale is $\gamma = \omega_{p0}/\omega_0$. We then scale the proxy's mode shapes, frequencies, multipole coefficients, M_n^m, etc., as described in section 3.4.1.

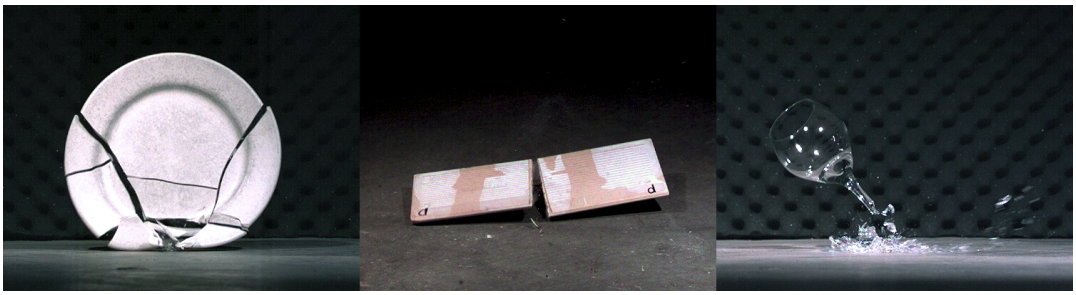


Figure 3.10: **Fracture experiments** were recorded using high-speed video (1200 FPS) and stereo audio recordings (96 kHz).

3.5 Results

For information on simulation cost versus fracture debris complexity, please see example statistics in Table 3.3. Representative stills are provided in Figure 3.9. Please see our video for all sound and animation results, including comparisons to reference laboratory fracture experiments (see Figure 3.10). Rigid-Body Soundbank performance improvements are described in Table 3.2.

Ground sound model: Ground vibrations can play an important role in sound generation when smashing objects onto it, especially given the ground's large size and wide frequency range. For example, a small piece of glass with extremely high pitch can produce low-frequency sound responses when dropped on the ground. We approximate ground sounds by first precomputing a modal model of a large concrete slab ($9m \times 9m \times 0.9m$) with 1000 modes with frequencies from $700Hz$ to $4kHz$, and each modes' corresponding multipole coefficients. Next we estimate the ground vibration response $\boldsymbol{q}(t)$ due to a unit impulse applied at the center of the slab. To synthesize ground sounds for simulations, we simply convolve the ground response with ground contact forces, and emanate multipole radiation from the appropriate contact position.

Example	ω_{proxy}	%Modes Replaced	CPU Sound Time (min)	
			Direct	With Proxies
Glass Slab	4000 Hz	36.4%	5631	3944
	2000 Hz	64.8%		2323
	0 Hz	100%		5.2
Window (low G_c)	3400 Hz	49.1%	4262	2202
	0 Hz	100%		9.2
Window (high G_c)	2000 Hz	45.2%	6023	3029
	0 Hz	100%		12.2
Table	2000 Hz	22.8%	17792	12980
	600 Hz	62.7%		7103
	140 Hz	95.7%		36.2

Table 3.2: **Rigid-Body Soundbank Performance:** Increasing ellipsoid proxy use (by lowering ω_{proxy}) leads to dramatic reductions in expensive sound model generation costs (modal+transfer+audio) over direct computation (serial timings). By replacing all fragment sound models by proxies, sound differences were barely audible, and roughly 500× speedups were observed in some cases—limited by disk I/O in our implementation.

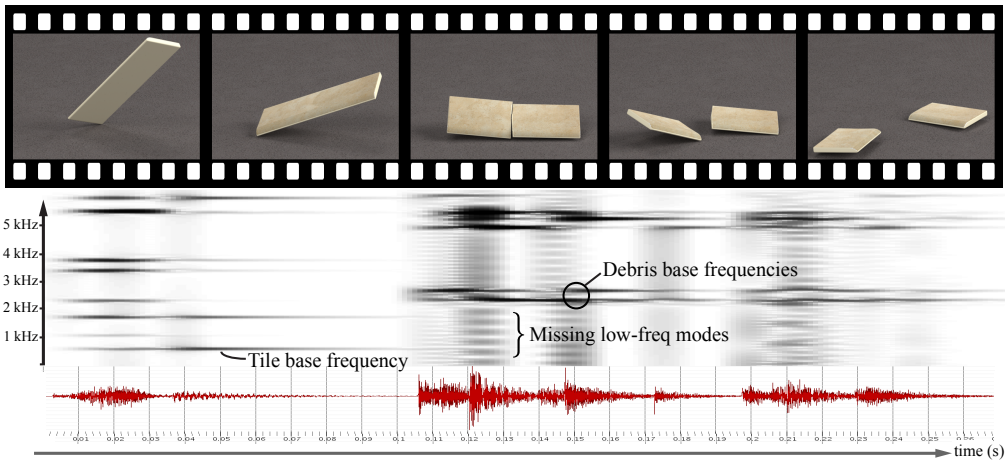


Figure 3.11: **Simulated tile fracture experiments**

Example (Ceramic dinner plate): We simulated the fracture of a ceramic plate (see Figure 3.9). Laboratory experiments of plates dropped on the ground produced qualitatively similar fracture patterns and sounds. For reference, we also provide a fracture-free simulation of a spolling (spinning + rolling) plate.

Model	Complexity								Timings (min)			
	time	$1/\Delta t$	Objects	Triangles	Tetrahedra	Modes	Solves	freqRange	Frac	Modal*	Trnsfr*	Audio
Plate	5s	10kHz	1→17	100k→342k	404k→967k	76	91k	20-20k	1.41*hr*	19.8	8.4	1.2
Tile	0.9s	5 kHz	1→3	47k→109k	201k→419k	81	1.2k	20-20k	0.05*hr*	12.2	6.2	0.1
WindowHiG_c	5s	5 kHz	1→57	50k→287k	120k→467k	611	252k	20-20k	3.1*hr*	18.6	31.2	3.1
WindowLoG_c	5s	5 kHz	1→89	50k→344k	120k→528k	586	302k	20-20k	3.4*hr*	20.4	50.4	3.3
Glass Slab	2s	10kHz	1→72	50k→407k	121k→634k	377	237k	20-20k	3.6*hr*	24.7	38.4	2.4
Wine Glass	3.2s	10kHz	1→83	101k→674k	372k→1391k	57	538k	20-20k	4.7*hr*	40.6	55.2	4.6
Poor Piggy	3.2s	10kHz	17→64	405k→752k	1628k→2190k	240	384k	20-20k	6.4*hr*	33.6	49.2	8.2
Rich Piggy	2s	10kHz	301→358	6743k→7131k	27191k→27801k	1411	413k	20-20k	26.4*hr*	37.2	56.6	12
Table	5s	5 kHz	62→328	1972k→6048k	6816k→15781k	4046	1138k	20-14k	12.2*hr*	66.3	73.8	20

Table 3.3: **Example Statistics** for simulation duration, rigid-body timestep size, and input and total geometric complexity; total number of scene modes; number of quastistatic stress solves. Serial timings are provided for fracture dynamics simulation and audio synthesis, whereas parallel timings ($*$) are given for modal analysis and acoustic transfer computations on a 16-node cluster of 8-core Xeons. Fortunately, most of the latter costs can be avoided using Rigid-Body Soundbanks (see Table 3.2).

Figure 3.12: **Breaking the bank!** A piggy bank is smashed into 58 pieces, and releases 300 coins.

Example (Ceramic tile): To compare against our laboratory tile experiments (see Figure 3.2), we synthesized sounds for a pre-scored virtual tile fracture experiment (see Figure 3.11). Our rigid-body restitution model produced faster bouncing behavior and our tile had slightly higher pitch, however qualitatively similar time-varying rigid-body sounds were observed.

Example (Glass slab): We dropped a glass slab which shattered into about 70 pieces (see Figure 3.9). This example was particularly well approximated by Soundbank proxies, and sounded similar even for 100% replacement.

Example (Glass window): We smashed two thick glass windows of differing fracture toughness (see Figure 3.4), which also fared well with proxy replacement.

Example (Wine glass): We dropped a wine glass onto its bowl, which exploded and left its stem behind (see Figure 3.9). Similar laboratory experiments produced qualitatively similar sounds.

Example (Piggy bank): We simulated two piggy banks: "Poor Piggy" contained only a few coins, whereas "Rich Piggy" contained many coins (see Figure 3.12). The latter example had the most expensive fracture dynamics due primarily to the impulse-based solver's handling of hundreds of coins stacked

inside.

Example (Table): The smashed table setting (see Figure 3.1) is our largest example, with over 4000 modes, and over a million quasistatic stress solves. Given the high model-generation costs, the Rigid-Body Soundbank is particularly useful on this example.

Comparison (With/without fracture impulses): Without fracture impulses (§3.2.3), excitations are only due to rigid-body contact forces (recall Figure 3.5). We synthesized the sound of the glass slab falling onto the ground with and without fracture impulses, to demonstrate the more distinct "crack" obtained when using fracture impulses.

Example (Interactive GPU-accelerated demonstrations): Real-time demonstrations were performed for the dinner plate and the piggy bank. For the spolling and smashing dinner plate (with $\sum_{\text{mode}i} \bar{n}_i = 2176$ expansions) our OpenMP 8-core CPU implementation required 0.156s to compute all-mode transfer, whereas our GPU-based transfer implementation (§3.3) only required 0.0003s (3.3kHz) on an NVIDIA Tesla card—**a 520× speedup**.

Soundbank Comparisons (Varying proxy-replacement threshold): To evaluate results produced by varying degrees of rigidbody soundbank use, we simulated the glass window, glass slab, and table examples with varying proxy-replacement thresholds—as controlled by the frequency-based ω_{proxy} proxy-use criterion. Surprisingly, replacing even all the object sound models with the ellipsoidal proxies (for large reductions in sound model computations) still produced plausible sounds. See the video for these comparisons, and Table 3.2 for model-generation speedups.

3.6 Limitations and Future Work

Significant challenges remain for physically based fracture sound synthesis. One open problem is the approximation of fracture sounds for very large objects, such as buildings, which pose many difficulties: expensive stress/modal analyses, complex debris, dense frequency spectra with huge numbers of eigenmodes, and more complex sound radiation, e.g., simple multipole sound models are invalid for large objects where the listener is inside the object.

Rigid-body sound is a convenient abstraction of brittle fracture, but not all fracture processes are so instantaneous or conveniently modeled. Gradual cracking, such as cracking lake ice or tearing, can require time-domain modeling of crack propagation which complicates modal sound modeling. Our quasistatic fracture impulse is a simple approximation of the effective fracture impulse resulting from complex time-dependent fracture processes. Since crack propagation speeds in brittle materials can be close to the Rayleigh wave speed (typically several km/s) the physical fracture impulses are determined by an extremely rapid process [GFM+93], e.g., brittle fracture of 10 cm scale objects can occur on the time-scale of a 20 kHz wave period. Due to the ill-posed nature of fracture-impulse estimation with a quasistatic model, we instead proposed an energy-based fracture-impulse model using quasistatic stress.

Ductile objects, such as metals, can undergo visible deformation during the fracture process [OBH02] and require additional investigation. Quasistatic fracture modeling works well for contact forces, but dynamic resonance-based fracture and fluid-solid coupling can also be important, e.g., a soprano smashing a wine glass with her voice. Contact coupling can significantly affect vibration-based sounds, and was only approximated by *ad hoc* contact damping. "Hair-

line" fractures may not fully separate the object, and can affect vibration-based sounds. In general, modal vibrations should also be coupled with frictional contact mechanics, and pose collision detection challenges [JP04].

Thin objects require special care for fracture simulation [BHTF07], as well as for sound modeling. For example, our wine glass involved both thin shell-like regions, and volumetric regions which proved difficult to mesh using tetrahedra. Thin-shell models also require special treatment for multipole radiation evaluation. Thin objects, such as glass panes, might also exhibit nonlinear vibrations during violent fracture processes [CAJ09]. Our fracture pattern generation does not produce fine debris and dust [IJN09], however precomputed soundbanks could be used to model their sounds.

We have modeled debris radiation using a linear superposition of non-interacting rigid-body sources, but real simulations can involve significant inter-object scattering effects. Our frequency-domain radiation model simplifies the time-domain nature of fracture. For example, far-field sound involves significant time-delay effects, and is also more complicated for large and also fast-moving and spinning objects [MI86].

Our rigid-body soundbank is based on homogeneous ellipsoidal primitives for simplicity and convenience, but audible differences can occur for nonconvex geometry (see video). However, it remains to be seen if other geometric primitives, e.g., fracture shards, provide more realistic results. In general, understanding the perceptible shape space for sound proxies, and the extent to which precomputed soundbanks can be used to replace general geometry are open problems. Finally, far more agressive speed-accuracy trade-offs can be made for interactive applications.

CHAPTER 4

MODAL CONTACT SOUND

As introduced in previous chapters, sounds from solid object contacts can be efficiently generated based on linear modal analysis (See Chapter 2) with rigid body dynamics: rigid body simulations resolve contact events, and estimate resulting impulses, which then drive the linear modal vibrations of the associated objects; these vibrations then produce sound waves propagating in the environments.

Unfortunately, treating vibrating objects as "rigid" during collision and contact processing fundamentally limits the range of sounds that can be computed, and contact solvers for rigid body animation can be ill-suited for modal contact sound synthesis, producing various sound artifacts. In this chapter, we resolve modal vibrations in both collision and frictional contact processing stages, thereby enabling non-rigid sound phenomena such as micro-collisions, vibrational energy exchange, and chattering. We propose a frictional multibody contact formulation and modified Staggered Projections solver [KSJP08] which is well-suited to sound rendering and avoids noise artifacts associated with spatial and temporal contact-force fluctuations which plague prior methods. To enable practical animation and sound synthesis of numerous bodies with many coupled modes, we propose a novel asynchronous integrator with model-level adaptivity built into the frictional contact solver. Finally, vibrational contact damping is modeled to approximate contact-dependent sound dissipation.

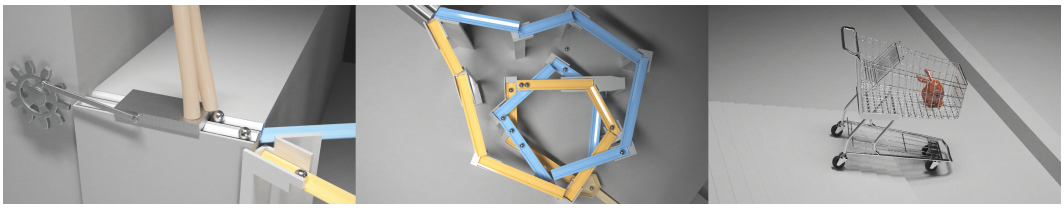


Figure 4.1: **A Rube-Goldberg contraption** that demonstrates many challenging multibody contact sounds. A noisy block feeder (Left) with flexible tubes ejects marbles into a double helix of plastic chutes (Middle), which causes a cup to fill up, lifting a lever that drops a bunny into a runaway shopping cart (Right) producing familiar clattering and clanging sounds due to deformable micro-collisions. Our approach can accurately resolve modal vibrations and contact sounds using an asynchronous, adaptive, frictional contact solver.

4.1 Introduction

Sound models based on linear modal vibrations are widely used to efficiently synthesize plausible contact sounds for so-called rigid bodies in computer animation and interactive virtual environments. For speed and simplicity, linear modal vibrations are usually just excited by using contact force impulses from rigid body contact solvers. However, in reality, there is no such thing as a "rigid" object, and the same small vibrations that produce sound also play an important role in producing rich contact events: micro-collisions, chattering, squeaking, coupled vibrations, contact damping, etc. Ignoring contact-level vibrations is the source of many sound-related deficiencies, as these small vibrations can be visually inconsequential but aurally significant. For example, pounding on a seemingly "rigid" dinner table can shake dishes—and may also upset your friends (see Figure 4.2). Frictional contact and deformation coupling is also important for sound; for instance, slip-stick phenomena is responsible for many familiar squeaking and scraping sounds, e.g., fingernails scraping on a chalk-

Figure 4.2: **Vibrational coupling makes a racket!** A rigid bunny dropped onto a table causes the table to deform rapidly, which in turn causes the resting dishes to receive contact impulses and go flying—with noticeable sound. In contrast, a traditional rigid body simulation (not shown) bounces the bunny off the perfectly rigid table without disturbing any dishes or causing much sound.

board. Resolving these vibrational contact effects is challenging due to the need to resolve deformable collisions and contact at high temporal rates.

Even in seemingly rigid scenarios, such as an object resting on a plane, current contact solver implementations can generate temporally incoherent contact impulses which lead to sound artifacts, such as resting objects that strangely humm or buzz when integrated at near-audio rates. These artifacts are a consequence of the fundamental non-uniqueness of rigid body contact forces (e.g., static indeterminacy) which can lead to point-like and nonphysical contact force (traction) distributions. Additionally, rigid-body contact impulses can exhibit nonphysical temporal fluctuations, which lead to noise-related sound artifacts (especially with iterative contact solution techniques) that must be dissipated artificially.

Moreover, the sound of a resting object should also depend on its contact state, and how contacts oppose surface vibrations. As an example, a coffee mug exhibits distinctive vibrational damping when placed in different orientations

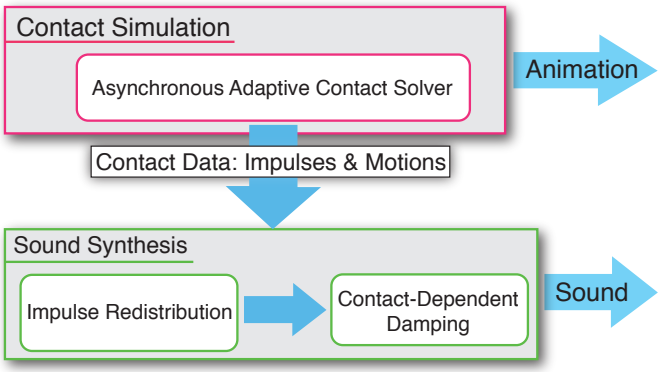


Figure 4.3: **Overview:** Multibody simulation is performed using an asynchronous adaptive contact solver (Top). This simulation yields animation data as well as impulses and object motion data which are fed to the sound synthesis algorithm running in a separated pass (Bottom). When synthesizing modal sound, input impulses are redistributed to enhance temporal coherence and eliminate noise, and contact-dependent damping is applied.

on surfaces (see Figure 4.4). This contact damping phenomena involves complex vibrational and contact coupling effects, and is ignored in current sound models or handled in *ad hoc* ways, e.g., "increase damping when in contact."

This chapter introduces an approach we developed to address all of these concerns and enable richer contact sounds (see Figure 4.3). We adopt a flexible multibody dynamics formulation, wherein each seemingly rigid object is allowed to deform with linear modal vibrations. Deformable collision processing at near audio rates is used to generate frictional contact problems to resolve perceptually important coupling and micro-collisions (see Figures 4.1 & 4.2). We use a modified Staggered Projections algorithm to solve the frictional contact problems [KSJP08], with modifications to avoid solver noise due to spatial and temporal incoherence in both the generation of contact points and the computation of contact forces. Vibrational contact damping phenomena are approximated using a time-varying lumped contact damping matrix to attenuate

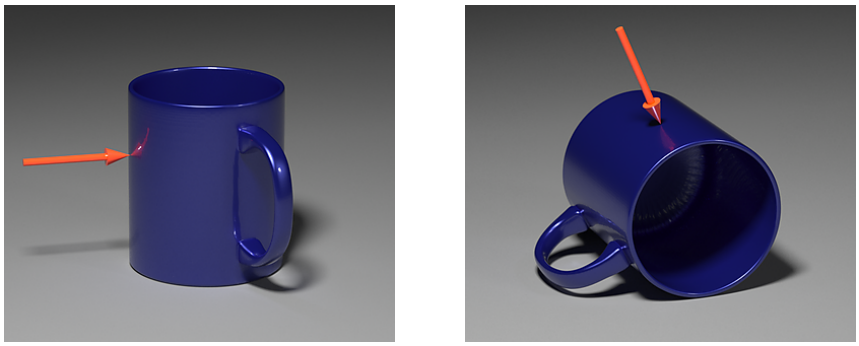


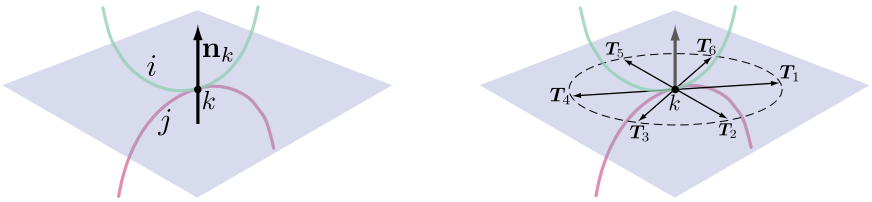
Figure 4.4: **Contact-dependent vibrational damping** is readily apparent by tapping a coffee mug at the same location while in different contact configurations. Our contact sounds differ because contact damping opposes modal vibrations differently.

vibrations during sound synthesis.

Since multibody frictional contact solves with thousands of coupled modes at near-audio rates are computationally intractable for sound synthesis, in this chapter, we propose a novel mode-adaptive asynchronous integrator which is capable of identifying and integrating low-frequency vibration modes in the contact solver (see Figure 4.8). We will describe methods for adding and removing modes from the contact-level simulation, as well as noise-free methods for simulating all audible modes during subsequent sound synthesis. Our implementation is able to handle dozens of bodies with thousands of audible modes in a practical manner for high-quality offline sound synthesis.

4.2 Low-Noise Contact Resolution

In this section we briefly review multibody contact and the Staggered Projections (SP) solver [KSJP08], then discuss its noise-related limitations for sound synthesis (in §4.2.2), and propose a modified SP solver which pro-



(a) (b)

Figure 4.5: **Contact geometry and friction cone sampling**

duces low-noise contact impulses efficiently (in §4.2.3). We refer the reader to [Bro99, KSJP08] for more details.

4.2.1 Background on Contact Problems

Following the notation of Kaufman et al. [KSJP08], we consider a multibody contacting system represented by the generalized position coordinate q, generalized velocity $\dot{\mathsf{q}}$, and mass matrix M. The world-frame position of point i on some object is given by the mapping $\boldsymbol{x}_i(\mathsf{q})$ which describes rigid body motion and linear modal deformations. Contacts in the system are described by the set C. Given a contact $k \in C$ between two points i and j with a normal direction $\boldsymbol{n}_k$ (Figure 4.5(a)), the corresponding generalized normal direction is given by $\mathsf{n}_k = \Gamma_k^T \boldsymbol{n}_k$ where Γ_k is the relative velocity Jacobian defined as $\Gamma_k = \nabla \boldsymbol{x}_i - \nabla \boldsymbol{x}_j$. The generalized normal contact impulses can be represented by the vector $\mathsf{c} = \mathsf{N}\alpha$, where $\mathsf{N} = [\mathsf{n}_1 \mathsf{n}_2 \dots \mathsf{n}_{|C|}]$ and α is the vector of magnitudes of normal contact impulses.

The isotropic Coulomb friction on the tangential plane can be linearized using $\boldsymbol{f}_k = \boldsymbol{T}_k \beta_k$, where $\boldsymbol{T}_k$ is the matrix whose column vectors uniformly sample the friction disk (Figure 4.5(b)), and β_k is the vector of impulse magnitudes

along each of the sampled directions. This polyhedral friction cone simplifies the Coulomb friction inequality into

$$e^T \beta_k \leq \mu_k \alpha_k, \quad s.t. \quad \beta_k \geq 0, \tag{4.1}$$

where μ_k is the coefficient of friction, and α_k is the normal contact impulse magnitude at k and $e = [1 \dots 1]^T$ [ST96]. The generalized friction impulses on all of the contacts can be written as $\mathsf{f} = \mathsf{D}\beta$, where $\mathsf{D} = [\Gamma_1^T \boldsymbol{T}_1 \dots \Gamma_{|C|}^T \boldsymbol{T}_{|C|}]$ and $\beta = [\beta_1^T \dots \beta_{|C|}^T]^T$. Using this notation, we can discretize the Euler-Lagrange equation (with timestep size h) as follows,

$$\mathsf{M}\left(\dot{\mathsf{q}}^{t+1} - \dot{\mathsf{q}}^{t}\right) = h\mathsf{g}(\mathsf{q}^t, \dot{\mathsf{q}}^t) + h\mathsf{f}_{ext}^t + \mathsf{N}\alpha^{t+1} + \mathsf{D}\beta^{t+1}\ , \tag{4.2}$$

where g, the *quadratic velocity vector* function, provides the Coriolis and centrifugal forces, and f_{ext} describes the external forces. The SP method solves this equation using a predictor-corrector method. First, it computes a velocity prediction $\dot{\mathsf{q}}^p$ by solving

$$\mathsf{M}\left(\dot{\mathsf{q}}^{p} - \dot{\mathsf{q}}^{t}\right) = h\mathsf{g}(\mathsf{q}^t, \dot{\mathsf{q}}^t) + h\mathsf{f}_{ext}^t. \tag{4.3}$$

Next, it solves

$$\mathsf{M}\left(\dot{\mathsf{q}}^{t+1} - \dot{\mathsf{q}}^{p}\right) = \mathsf{N}\alpha^{t+1} + \mathsf{D}\beta^{t+1} \tag{4.4}$$

to correct the velocity, which involves estimating $\dot{\mathsf{q}}^{t+1}$ by simultaneously solving the following two quadratic programming problems iteratively,

$$\begin{aligned} \dot{\mathsf{q}}^{t+1} &= \arg\min_{\mathsf{v}} \left(\frac{1}{2}\mathsf{v}^T\mathsf{M}\mathsf{v} - \mathsf{v}^T(\mathsf{M}\dot{\mathsf{q}}^p + \mathsf{D}\beta^{t+1})\right), \\ &s.t. \quad \mathsf{N}^T\mathsf{v} \geq 0 \end{aligned} \tag{4.5a}$$

$$\begin{aligned} \beta^{t+1} &= \arg\min_{\beta} \left(\frac{1}{2}\beta^T\mathsf{D}^T\mathsf{M}^{-1}\mathsf{D}\beta + \beta^T\mathsf{D}^T(\dot{\mathsf{q}}^p + \mathsf{M}^{-1}\mathsf{N}\alpha^{t+1})\right), \\ &s.t. \quad E^T\beta < \mathrm{diag}(\mu)\alpha^{t+1},\ \beta \geq 0, \end{aligned} \tag{4.5b}$$

where (4.5a) is the contact problem for determining the normal impulse, and (4.5b) is the friction problem that determines the frictional impulse. Here E is

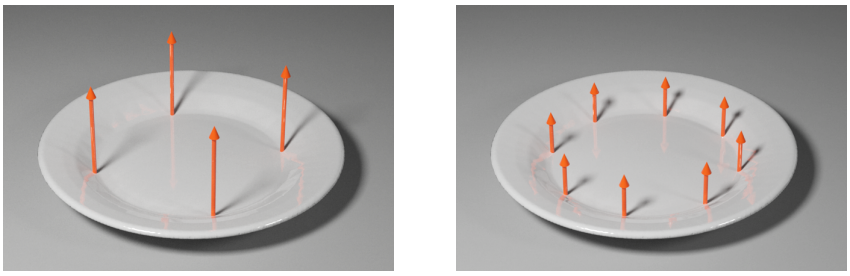


Figure 4.6: **Non-unique contact impulses can produce noise** since fluctuations in resting contact forces, while causing no motions, produce changing modal forces which can produce noise, e.g., resting objects that "hum."

the matrix form of all the vectors e in (4.1) such that each column k has ones in rows corresponding to entries in the subvector $\beta_k \in \beta$ and zeros in all other elements; α^{t+1} in the second problem (4.5b) is a vector of normal contact impulse magnitudes, which is the Lagrange multipliers of the constraints of the first problem (4.5a). These QP problems are essentially the dual form of the LCP formulation modeling the contacting systems. We discuss methods for solving these large sparse QP problems later (§4.5).

4.2.2 Non-unique Contact Impulses and Noise

When solving (4.5) [Bro99], fundamental difficulties exist for sound synthesis:

1. First, given a fixed β^{t+1}, it can be proved that equation (4.5a) has a unique solution, since the mass matrix M is positive definite [1]. However, the Lagrange multipliers of the constraints, i.e., the contact impulses, are not necessarily unique because, according to the Karush-Kuhn-Tucker (KKT)

[1] Cottle et al. [CPS92] proved that the LCP form of (4.5a) has a unique solution if M is a *P-matrix*—a matrix where all principal minors have positive determinants. A positive definite matrix is a P-matrix, however, P-matrices are not necessarily positive definite.

condition of (4.5a), the Lagrange multipliers should satisfy

$$\mathsf{N}\alpha^{t+1} = \mathsf{M}(\dot{\mathsf{q}}^{t+1} - \dot{\mathsf{q}}^{p}) - \mathsf{D}\beta^{t+1}. \tag{4.6}$$

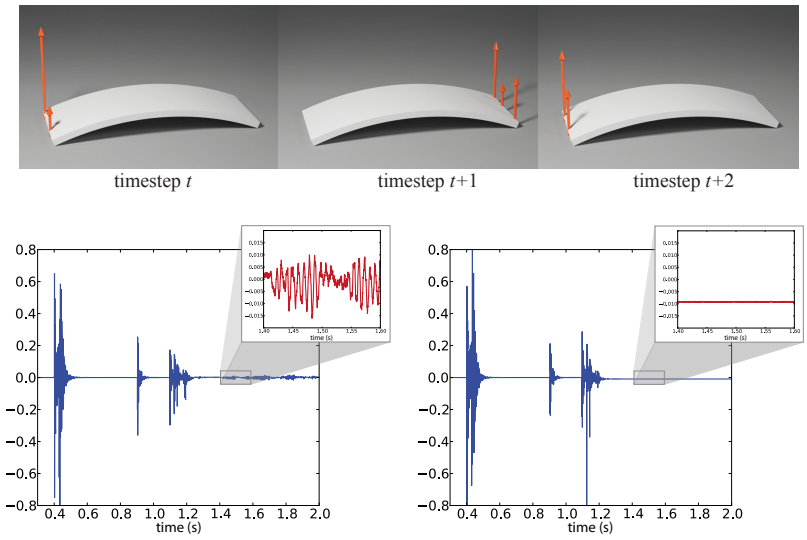
This equation could have non-unique solutions when N is rank deficient, and rank-deficient N is almost inevitable when the number of contacts is large enough. As a result of this non-uniqueness, the contact impulses can be temporally incoherent, and using them to excite the sound model can lead to audible noise artifacts (see Figure 4.6).

2. The second difficulty comes from the friction problem (4.5b), in which the Hessian matrix $\mathsf{D}^T\mathsf{M}^{-1}\mathsf{D}$ tends to be singular when the number of contacts is large—note that the size of the problem is proportional to the number of contacts. Solving such large rank-deficient quadratic programming problems will quickly slow down the simulation.

We therefore seek to estimate a temporally coherent set of active contacts of minimal size.

4.2.3 Estimating Temporally Coherent Active Contacts

Contact Generation: In standard multi-body simulation, a contact is generated when (i) collision between two objects is detected and (ii) their relative normal velocity at the collision point is negative. These criteria can produce plausible motion effects; however, they tend to also generate noisy contact forces which is problematic for sound synthesis. For example, consider the curved slab resting on the ground with multiple contact points (Figure 4.7). The traditional contact generation produces non-zero forces which cycle among multiple contact points as the simulation proceeds: contact forces generated to satisfy



(a) Velocity-based Contact Sounds (b) Our Position-based Contact Sounds

Figure 4.7: **Noisy contact forces on a static curved slab:** (Top) At timestep t, the non-zero contact forces appear at the left end of the slab; at timestep $t+1$, they are at the right end of the slab; at the next time step, the non-zero forces come back to the left end, and so on so forth. This microscopic contact cycling leads to noise (a) generated rather than pure silence (b) as we expected in this case.

constraints at the current contact points at the current timestep can cause other contact constraints to be violated at subsequent timesteps.

To avoid contact generation noise, we propose to generate contacts whenever intersections are detected, irrespective of the relative contact velocity. This criteria can generate more contacts than the standard one, leading to more expensive contact problems. In particular, the friction solve (4.5b) becomes harder since the problem size is proportional to the number of contacts. We therefore propose the following contact filtering scheme to reduce the number of contacts and maintain temporal coherence.

Contact Reduction: We maintain active contacts by applying the *active set method* [GMW81] when solving the contact problem (4.5a). When solutions are found, the active set method also identifies the active constraints for that problem. The resulting constraint Lagrange multipliers, which correspond to contact forces, become quite sparse. Since zero contact force always leads to zero friction force, we can turn off all inactive contacts to reduce the size of any subsequent friction problem (4.5b). To ensure temporally coherent active-contact selection (as well as faster SP convergence), we "warm start" the SP solver with the solution from the previous timestep [KSJP08]. To initialize the contact filtration process, we can first solve (4.5a) using an interior point method, then use that as an initial guess for the active set method. On average, we observe about one order of magnitude speed up over the simulation without contact reduction.

4.3 Asynchronous Adaptive Contact Solver

By simulating flexible objects we can resolve more interesting dynamic behaviors, such as contact coupling and chattering. These behaviors can enable more realistic visual effects and richer sounds. However, deformable simulation is much more expensive than purely rigid simulation. Furthermore, the simulation timestep size is restricted by the highest modal vibration frequency due to the stability condition—the higher frequency, the smaller the timestep has to be. For high-quality sound synthesis where all audible modes are desired, this simulation cost can be prohibitive.

Fortunately higher frequency vibrations tend to damp more quickly, and are often perceptually important for only a short time (see inset time-series). Af-

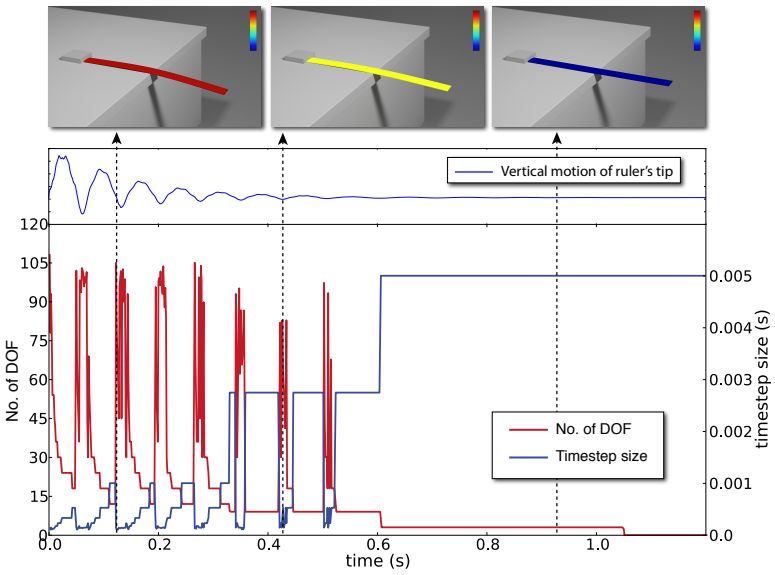
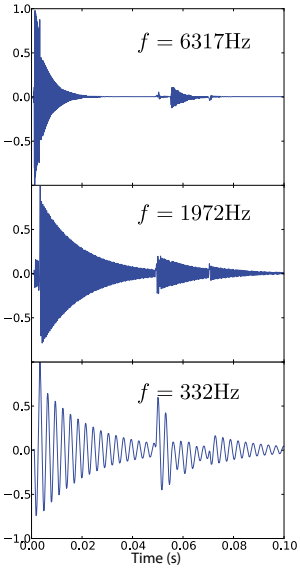


Figure 4.8: **Mode-adaptive contact simulation** (ruler example)

terwards, simulating only lower frequency modes (or even rigid objects) can suffice to capture perceptually important dynamics. We propose to exploit this transient nature of higher frequency vibrations by adaptively selecting the simulated modes and asynchronously integrating the system using the largest timestep possible (see Figure 4.8). To avoid diminishing returns, we can also limit the maximum mode frequency used in contact simulation, leaving higher frequency modes to be considered only during a subsequent sound synthesis phase (§4.4). An overview of our asynchronous adaptive integration scheme is outlined in Algorithm 2. Next we will describe each of its steps.



Asynchronous Integration: First, consider a simulation without contacting bodies. To evolve the objects independently, each of them has a private timestamp indicating its local simulated time. This timestamp is similar to the concept of *local virtual time* in the rigid timewarp method [Mir00]; it is used to sched-

ule the objects in a priority queue. At each simulation step, the object at the top of the queue has the earliest timestamp and is popped to advance to its next timestep. If it is not in contact, we simply reschedule it into the priority queue after advancing its state and increasing its timestamp. Otherwise, we need to resolve collisions before rescheduling it.

Collision Detection: Let us refer the object currently popped from the priority queue the *active object*. To detect collisions, all the other objects need to synchronize to it. These objects are all advanced no less than the active object, since they are deeper in the priority queue. We therefore synchronize them by linearly interpolating their states at the time of the active object. While requiring a little extra memory since each object now needs to maintain two states from last two consecutive timesteps, linear interpolation introduces almost no performance overhead in the simulation. Next the standard discrete-time collision detection is performed using oriented bounding box (OBB) hierarchies for rigid objects [GLM96]; for modal deformation, we use Bounded Deformation Tree ideas to quickly update the OBB hierarchy (with fixed bounding-box orientation) [JP04].

Contact Groups: Based on the detected intersections, objects are grouped into *contact groups* [Mir00]; the component objects of a contact group intersect with each other and are separated from other contact groups. Therefore they must be integrated as a unit. We identify these contact groups by detecting independent connected sets in the contact graph [GBF03].

State Rollback: For any object that is more advanced than the active object, if it is not in contact at all its interpolated state is simply discarded, and it remains at its current advancement; otherwise, it must belong to some contact group and

Algorithm 2: Overview of asynchronous adaptive integration

```
begin
    while simulation is not over do
        ao ← queue.pop()
        foreach obj ≠ ao do
            interpolate_state(obj, ao.timestamp)
        end
        detect_collision()
        identify_contact_groups()
        foreach contact group cg do
            advance_to_next_step(cg);              // Algorithm 3
        end
        foreach contacting object obj do
            increase_timestamp(obj)
            reschedule(obj)
        end
    end
end
```

has to be integrated within that group. Therefore, we roll back its current state by replacing it with its interpolated state (see Figure 4.9 for an illustration). Then each contact group is integrated independently using an adapted timestep size according to the algorithm described in the rest of the section (See Algorithm 3).

4.3.1 Contact Group Advancement

We solve the contact-friction problem described in section 4.2.1 to advance a contact group. In particular, for velocity prediction (Equation 4.3), we use explicit forward Euler method and the Newmark integrator [Wri06] to integrate rigid motion and modal deformation respectively; velocities are then corrected using the SP iterations. The simulated vibration modes, i.e. the simulated degree of freedom (DOF), are determined based on the current modal vibrating state–a mode that is largely damped can be safely ignored from simulation.

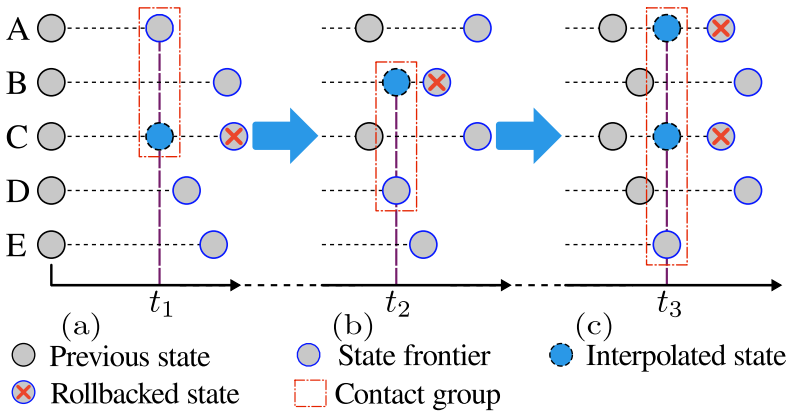


Figure 4.9: **Asynchronous Advancement of Objects:** (a) At time t_1, object A is selected as the current active object. It is in contact with a more advanced object C. Then C synchronizes with A using its interpolated state at t_1, and advance from there together with A; (b) Next D is at the top of the queue, and is in contact with B at t_2. *B* rolls its state back to t_2, and advances its state with *D*; (c) Then E is processed, which is in contact with A and C. Therefore E, A and C advance together to the next timestep.

However, the current vibrating state of a mode cannot be determined without solving the contract-friction problem. Therefore, we first estimate the current normal contact impulses (line 2 of Algorithm 3), then use the estimated impulses to determine the simulated DOF (line 3), and finally solve the adapted problem (line 5-6).

Normal Contact Impulse Estimation: The normal contact impulses are estimated by solving a single QP problem (4.5a), wherein we use the friction impulses from the last timestep assuming temporal coherence. This leads to one extra QP solve at each timestep. Fortunately, this QP is well-posed and is much cheaper to solve compared to the full SP algorithm. The timestep size used in the solve is determined as follows.

Timestep Determination: Due to the integration stability condition, the timestep size of integrating a contact group is restricted by the highest simu-

Algorithm 3: **Adaptive timestep for contact group** C : advance_to_next_step(contact$_g$roup)

```
   begin
1      ts ←determine_timestep_size(C)
2      f_c ←contact_solve(C, ts)
3      adapt_dof(C, f_c)
4      ts ←determine_timestep_size(C)
       if cull_friction_solve(C) then
5          frictionless_contact_solve(C, ts)
       else
6          staggered_projections(C, ts)
       end
   end
```

lated vibration frequency $\mathtt{f}_h$ of its component objects. In practice, we found that a timestep size $\Delta t = 1/(6.5\mathtt{f}_h)$ produces stable modal deformation. In addition, a default timestep size Δt_{max} is used for rigid bodies, giving us the timestep size of a contact group as $\Delta t = \min(1/(6.5\mathtt{f}_h), \Delta t_{max})$. For contact impulse estimation mentioned above, we use the $\mathtt{f}_h$ value from last timestep (line 1). To finally advance the contact group, $\mathtt{f}_h$ is updated (line 4) after the simulated DOF is adapted.

Adaptation (DOF Increase): In the following description, we denote the maximum number of simulated modes of an object as N_m. Assume these modes are labeled from 1 to N_m, ordered from lower to higher frequencies; and let s denote the current number of simulated modes (modes $1 \ldots s$). To adapt the simulated DOF of an object, we first check if s should be increased. Large external impulses can excite high-frequency modes, thus activating them in the simulation. We measure the excitement of modes by considering the time derivative of the modal vibration equation, i.e.

$$\ddot{\mathsf{v}} + \mathsf{C}\dot{\mathsf{v}} + \mathsf{K}\mathsf{v} = \mathsf{U}^T \dot{f} \ .$$

The impulse response of this equation measures the velocity excitement of modes due to external impulses, and has the magnitude proportional to $\mathsf{U}^T \dot{f}$. The time derivative $\dot{f}$ is estimated using $(f_c - f^{n-1})/\Delta t$, where f_c is the estimation of normal contact impulse described above, and f^{n-1} is the normal contact impulse from last timestep. Then we approximate the velocity excitement vector using

$$\hat{\boldsymbol{v}} = \mathsf{U}^T \frac{f_c - f^{n-1}}{\Delta t}, \tag{4.7}$$

and find the highest mode j such that $\hat{\boldsymbol{v}}_j \geq \delta_u$, where δ_u is a parameter to control the upgrade criterion. In practice, we use $\delta_u = 1E-5$ for all the adaptive simulations. We increase the simulated modes up to j if $j > s$.

Adaptation (DOF Decrease): If s is not increased, we further check if it could be decreased to enable faster simulation. A mode could be deactivated if its modal vibration energy has been largely dissipated. The modal energy of mode i is computed using

$$\mathsf{E}_i = \frac{1}{2}(\dot{\mathsf{q}}_i^2 + \mathsf{k}_i \mathsf{q}_i^2) \,, \tag{4.8}$$

where q and $\dot{\mathsf{q}}$ are its modal vibrational displacement and velocity, and k_i is its mass-normalized modal stiffness. We then find the highest mode j such that $\mathsf{E}_j \geq \delta_d$, where δ_d is a parameter to control the downgrade criterion. $\delta_d = 1E-7$ is used in our examples. We decrease the simulated modes down to j if $j < s$. If none of the modes has modal energy larger than δ_d, then the modal deformation is deactivated, and only rigid motion is considered.

Culling Friction Solves: We note that the contact-friction problem gets largely simplified for frictionless contacts. Not only is this because the friction problem (4.5b) is fundamentally harder to solve than the contact problem (4.5a) but also no iteration is needed at all in this case. When objects are resting without any

static friction, ignoring friction introduces no error. In simulations, this tends to happen frequently, since objects always tend to become static due to energy dissipation. We detect such frictionless contact groups by applying the following heuristics: given a contact group possessing a set of contacts denoted by C and a set of objects denoted by O, its friction solve can be turned off if it simultaneously satisfies two conditions:

$$\sum_{i \in C} |\boldsymbol{v}_i^{(t)}| < \delta_t, \tag{4.9}$$

where $\boldsymbol{v}_i^{(t)}$ is the predicted relative tangential velocity (after integrating using (4.3)) at contact i, and δ_t is a positive value close to zero (1E-8 in practice); and

$$\sum_{j \in O} \|\mathbf{f}_j^{(n-1)}\| < \delta_f, \tag{4.10}$$

where $\mathbf{f}_j^{(n-1)}$ is the generalized friction impulse of object j from last timestep, and δ_f is also a small value (1E-10 in practice). Note that these heuristics are conservative: (4.9) guarantees no relative movement in the contact group, while (4.10) ensures no friction force applied in the last timestep. In our experiments, we observe 6%-24% additional performance improvement.

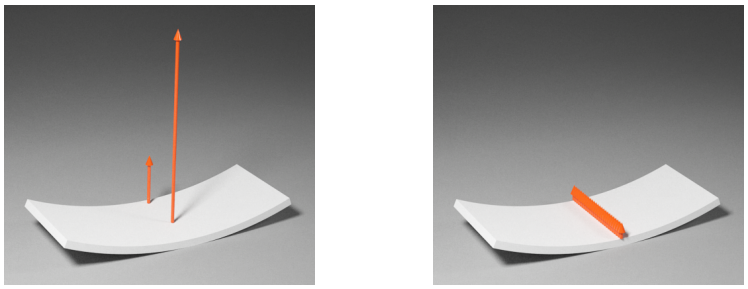
4.4 Sound Synthesis

This section describes (i) how the contact simulation fits into our two-pass sound synthesis pipeline (§4.4.1), (ii) how impulses generated during contact resolution are spatially redistributed prior to exciting all-frequency modal sounds (§4.4.2), and then (iii) how the redistributed impulses are used to parameterize our modal contact damping model (§4.4.3).

4.4.1 Sound Synthesis Pipeline

Aside from the sound-aware contact resolution approach, our sound synthesis pipeline is similar to the approach used in [ZJ10]. Specifically, solid objects are represented using tetrahedral meshes; modal vibration models are precomputed using the finite element method; the modal sound model is excited by contact impulses obtained from dynamic simulation; sound radiation is approximated using Helmholtz acoustic transfer models represented via precomputed multipole expansions, and used with *head related transfer functions* (HRTF) for sound rendering.

While our method could generate sound as the simulation advances, in practice we use a *two-pass implementation*. The first pass simulates the dynamics while recording the time series of contact impulses to disk. The second pass synthesizes sound by integrating the modal vibration equations of excited objects. Each object's recorded contact impulses are first spatially redistributed (§4.4.2) before determining modal excitations, and then used to determine each mode's contact damping (§4.4.3). To support integration of numerous modes, mode-level parallelism is exploited. Overall the second pass is much faster than the first dynamics pass (See Table 4.2). Once the expensive dynamics is computed, we can quickly resynthesize the sound, e.g., to efficiently tune sound-related damping parameters, such as α, β and γ in (4.18) which are perceptually important [KPK00].



(a) Without refinement (b) With refinement

Figure 4.10: **Impulse Refinement:** A curved slab is sitting on the ground. The orange arrow bars indicate contact force distribution with their length proportional to the magnitude of the forces. (a) The quadratic programming solve somewhat arbitrarily selects only a few contacts to be active; (b) our impulse refinement algorithm produces more uniformly distributed forces among all the contacts.

4.4.2 Impulse Redistribution

The algorithms of §4.2 and §4.3 described how to generate temporally coherent low-noise force impulses; however, for efficiency, these impulses are applied only at a minimal number of active contacts selected by the active set method (see Figure 4.10(a)). Applying these nonphysically sparse impulses to the modal sound model leads to increased noise in the synthesized sound which we would like to avoid. Fortunately we can again exploit the non-uniqueness of the impulse solution to redistribute contact impulses after each simulation timestep to obtain a more uniform spatial distribution (see Figure 4.10). We emphasize that these redistributed impulses are only used for sound synthesis, and do not affect the simulated motion–which uses sparse contact impulses for speed.

At the end of each timestep, let $\bar{\alpha}$ and $\bar{\beta}$ denote the normal contact and friction impulse magnitudes, respectively, as a result of solving (4.5); the associated

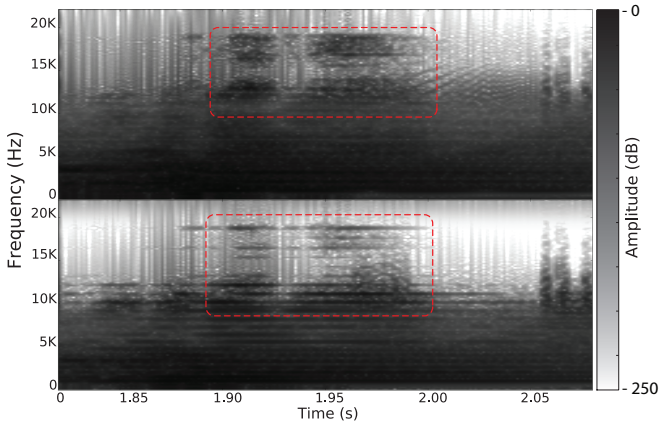


Figure 4.11: **Clearer sounds with impulse refinement:** Contact-impulse noise can result in muddier spectral responses (Top), whereas the spectrogram of sound generated using impulse refinement (Bottom) exhibits less noise, e.g., compare modal frequency lines in highlighted region (See smooth rocking example from Figure 4.10).

contact and friction impulses are

$$\mathsf{c} = \mathsf{N}\bar{\alpha} \quad \text{and} \quad \mathsf{f} = \mathsf{D}\bar{\beta}. \tag{4.11}$$

Our adjusted impulse distribution can be written as

$$\alpha = \bar{\alpha} + \boldsymbol{N}_{\mathsf{N}} k_\alpha \quad \text{and} \quad \beta = \bar{\beta} + \boldsymbol{N}_{\mathsf{D}} k_\beta \tag{4.12}$$

where $\boldsymbol{N}_{\mathsf{N}}$ and $\boldsymbol{N}_{\mathsf{D}}$ are the orthonormal matrices spanning the null space of N and D, respectively; we obtain these matrices using the LAPACK [ABB+99] routine geqp3. Since both $\mathsf{N}\boldsymbol{N}_{\mathsf{N}} = 0$ and $\mathsf{D}\boldsymbol{N}_{\mathsf{D}} = 0$, the impulse redistribution does not change the total impulses c and f, and thus does not affect the simulated dynamics. To make these more uniformly distributed, we solve a QP problem

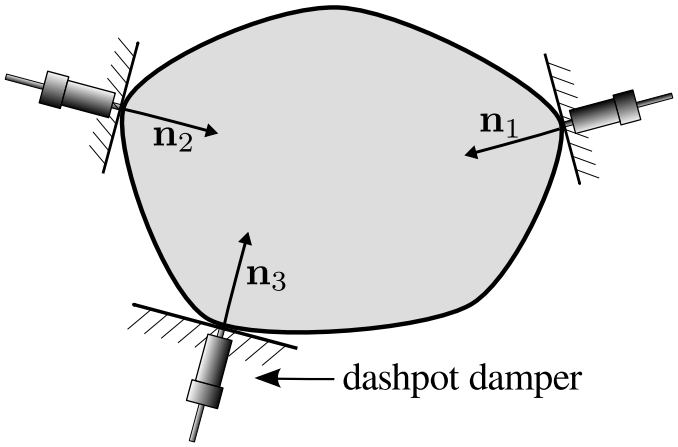


Figure 4.12: **Modified viscous contact dampers** are used to approximate mode-dependent contact-damping phenomenon.

for k_α and k_β:

$$\begin{aligned} &\arg\min_{k_\alpha, k_\beta} \frac{1}{2}\left(\|\bar{\alpha} + \boldsymbol{N}_{\mathsf{N}} k_\alpha - \alpha^{t-1}\|^2 + \|\bar{\beta} + \boldsymbol{N}_{\mathsf{D}} k_\beta - \beta^{t-1}\|^2\right) \\ &\qquad\qquad \bar{\alpha} + \boldsymbol{N}_{\mathsf{N}} k_\alpha \geq 0 \\ &\textit{s.t.} \qquad \bar{\beta} + \boldsymbol{N}_{\mathsf{D}} k_\beta \geq 0 \\ &\qquad\qquad E\left(\bar{\beta} + \boldsymbol{N}_{\mathsf{D}} k_\beta\right) \leq \operatorname{diag}(\mu)\left(\bar{\alpha} + \boldsymbol{N}_{\mathsf{N}} k_\alpha\right) \end{aligned} \tag{4.13}$$

where α^{t-1} and β^{t-1} are the contact and friction impulse magnitudes from the previous timestep. Note that this QP problem is defined for a single object, and therefore each can be solved independently. The problem size is much smaller than in (4.5), and these solves add only a little overhead to the overall simulation, e.g., only 2%-10% extra cost is observed. However, it results in computed sounds with much less noise (see Figure 4.11).

4.4.3 Contact-dependent Modal Damping

Given the distributed contact forces for sound synthesis (from §4.4.2), we now proceed to efficiently approximate contact-dependent damping forces (recall

Figure 4.4). Prior sound synthesis methods typically use a velocity-proportional Rayleigh damping model where the damping matrix is a linear combination of the mass and stiffness matrices, $\alpha \boldsymbol{M} + \beta \boldsymbol{K}$, and α and β are material-dependent positive scalars [Sha91]. Our contact damping model begins by considering simple viscous dampers at each contact point (see Figure 4.12) where the viscous coefficient is proportional to the contact force magnitude estimated from dynamic simulation. Specifically, given point contact k with normal direction $\boldsymbol{n}_k$ and contact force magnitude c_k, its vibrational velocity along both normal and tangential directions are

$$\boldsymbol{v}_k^{(n)} = \boldsymbol{n}_k \boldsymbol{n}_k^T \mathsf{U}_k \dot{\mathsf{q}}; \quad \boldsymbol{v}_k^{(t)} = (\boldsymbol{I} - \boldsymbol{n}_k \boldsymbol{n}_k^T) \mathsf{U}_k \dot{\mathsf{q}} \tag{4.14}$$

respectively, where $\mathsf{U}_k \in \mathbb{R}^{3\times r}$ is the r-mode displacement submatrix corresponding to point k. The viscous contact damping force can be modeled as

$$\boldsymbol{d}_k = \gamma c_k (\boldsymbol{v}_k^{(n)} + \mu \boldsymbol{v}_k^{(t)}) = \gamma c_k \left(\mu \boldsymbol{I} + (1-\mu) \boldsymbol{n}_k \boldsymbol{n}_k^T \right) \mathsf{U}_k \dot{\mathsf{q}} \,, \tag{4.15}$$

where μ is the coefficient of friction, and the positive scalar γ is a material-dependent parameter controlling the strength of damping forces. With this model, the generalized damping force for contact k can be written as

$$\mathsf{U}_k^T \boldsymbol{d}_k = \gamma c_k \mathsf{U}_k^T \left(\mu \boldsymbol{I} + (1-\mu) \boldsymbol{n}_k \boldsymbol{n}_k^T \right) \mathsf{U}_k \dot{\mathsf{q}} \ \in \mathbb{R}^r. \tag{4.16}$$

To express in matrix form, we define

$$\mathsf{G} \equiv \sum_{k \in C} c_k \mathsf{U}_k^T \left(\mu \boldsymbol{I} + (1-\mu) \boldsymbol{n}_k \boldsymbol{n}_k^T \right) \mathsf{U}_k \,. \tag{4.17}$$

where C denotes the set of point contacts at the current simulation step. Then the total effective modal damping of the vibration is

$$\mathsf{C} = \mathsf{U}^T (\alpha \boldsymbol{M} + \beta \boldsymbol{K}) \mathsf{U} + \gamma \mathsf{G}. \tag{4.18}$$

Material	Density (kg/m^3)	Young's modulus(GPa)	Poisson's ratio	Damping		
				α	β	γ
Ceramic	2700	7.4E+10	0.19	6	1E-7	3E-2
Polystyrene	1050	3.5E+9	0.34	30	8E-7	4E-4
Steel	7850	2E+11	0.29	5	3E-8	3E-1
MDF	615	4E+9	0.32	35	5E-6	9E-3
Wood	750	1.1E+10	0.25	60	2E-6	5E-4

Table 4.1: **Material Parameters:** The table was made with medium density fiberboard (MDF).

Here α, β and γ are all material dependent constants (see Table 4.1 for values used in the examples). Note that the contact damping term above is also symmetric positive definite, which guarantees it will dissipate energy.

Unfortunately, due to G this damping matrix is non-diagonal, and couples all vibration modes together thereby leading to vastly more expensive modal dynamics integration costs for sound synthesis, e.g., $O(r^2)$ versus $O(r)$ costs for r modes. Fortunately, we observe that we can still obtain phenomenologically similar results by simply ignoring off-diagonal elements of G, and thus preserve a linear-time modal sound synthesis phase. Therefore the contact damping of mode m is simply $\gamma G_{mm}\dot{q}_m$, where the diagonal matrix coefficient is

$$G_{mm} = \gamma \sum_{k \in C} c_k \mathsf{U}_{k,m}^T \left(\mu \boldsymbol{I} + (1-\mu)\boldsymbol{n}_k \boldsymbol{n}_k^T\right) \mathsf{U}_{k,m}, \tag{4.19}$$

where $\mathsf{U}_{k,m} \in \mathbb{R}^3$ is the displacement at contact k of mode m. Note that this damping term depends on the current contact force c_k and modal displacement $\mathsf{U}_{k,m}$, and therefore captures spatial and temporal dependencies. In practice, we compute the contact damping coefficient G_{mm} at each simulation timestep, and cubically interpolate them for use in an IIR filter for modal sound synthesis.

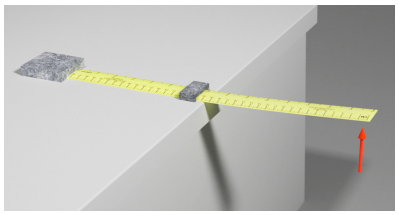
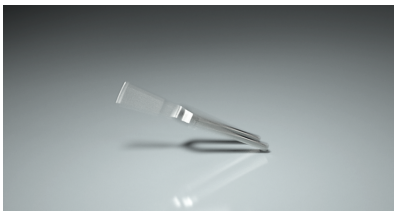



Figure 4.13: **Ruler and tuning fork examples**

Example	Surface Vertices	tetrahe-dra	DOF	Time (s)	Dynamics Cost (hr)	Sound Cost (s)	Sound Modes
Ruler	5760	18860	108	2.0	0.08	7.2	305
Table	362550	2836215	132	2.2	4.8	22.2	1005
Mug	29718	161543	6	3.0	0.12	5.3	64
Tuning Fork	6601	44076	6	2.0	0.08	4.3	30
Pipes	185912	910765	138	12.2	6.3	15.7	842
Marble Tracks	188864	310844	426	35	9.6	32.4	3822
Shopping Cart	215150	992355	186	4.0	4.7	14.2	2060

Table 4.2: **Example Statistics:** Maximum degrees of freedom (DOF) for rigid and modal dynamics. Adaptive dynamics simulations used a highest simulated vibration frequency of f_h = 5000 Hz. Consequently the minimum timestep size is around 3.07E-5s, except for the two purely rigid examples (mug and tuning fork) which used a timestep size of 0.001s. Sound synthesis uses all modes below a 20 kHz cutoff, except for the shopping cart where 12 kHz was used.

4.5 Results

We list all material related parameters used in our examples in Table 4.1. Simulation statistics are given in Table 4.2. All reported timings are based on our implementation on a Linux system with an 8-core Intel Xeon X5570 CPU. Please see our video for all sound and animation results, and related comparisons.

Ruler Twang: We simulated a comedic ruler "twang" sound. We fixed one end of a ruler near the edge of a table (see Figure 4.13). When excited by a single impulse, the ruler starts chattering against the table, producing a distinc-

tive "twang" sound. Next we move the ruler as it vibrates, changing its contact positions against the table. The changing contact configuration and movement cause varying contact coupling between the modes which leads to a characteristic (and funny) rising-pitch ruler twang. This example highlights the strength of our simulation, since these sound effects could not be synthesized without resolving micro-collision events accurately. Despite the short timescale and rapid collisions, the method is still able to make use of mode adaptation (see Figure 4.8).

Table: We dropped a heavy rigid body bunny onto a table (fixed to the ground) which had a stack of dinner plates and bowls on it. This example demonstrates the perceptual importance of resolving vibrational coupling for both visual and sound realism (see Figure 4.2). Using a traditional rigid-body simulation, the table does not deform and thus the kitchenware stays static since no elastic energy can be transmitted by the rigid-body model. In our simulation, however, the stiff table deforms a tiny amount, and consequently all the kitchenware on the table is excited, and even bounced into the air when a heavy bunny is dropped on the table. The audible difference is clear and dramatic. Despite our asynchronous adaptive simulator often only simulating a small number of total modes, it can still resolve these rich vibrational coupling effects (see Figure 4.15); comparing to our non-adaptive simulation, we observed about 2.6X speedup. This example also illustrates the low-noise sound generation of our method and contact filtering (see Figure 4.14)

Coffee mug: To demonstrate our contact damping model, we simulated a coffee mug being tapped at a fixed position with different contact orientations (see Figure 4.4). For validation, we compared our synthesized sounds with record-

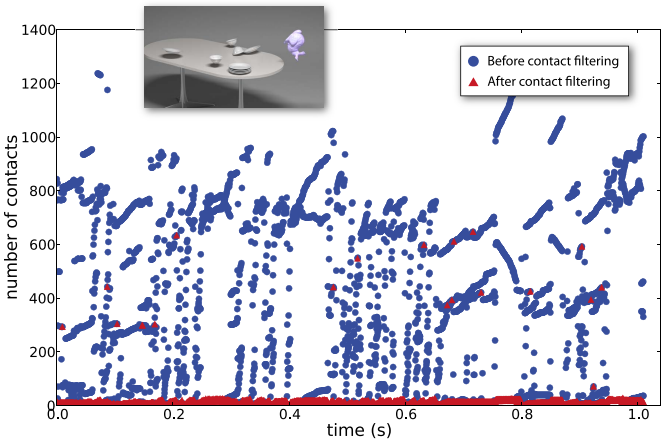


Figure 4.14: **Contact filtering for fast low-noise contact sound:** The stack of plates and other objects generate a large number of contacts at each timestep. Our contact filtering method (§4.2.3) can effectively cull most contacts, producing fewer and more temporally coherent contacts.

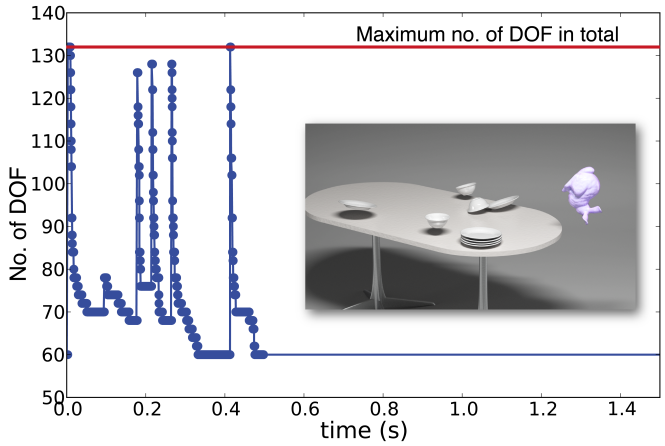


Figure 4.15: **Mode-adaptive contact simulation** of a table (72 modes max) and 10 rigid objects (dishes and bunny) greatly reduces the number of active table modes (max 72) except for impact events.

ings of a real mug experiment to demonstrate qualitatively similar damping effects.

Tuning fork: We also demonstrate contact damping using a tuning fork dropped on the ground (see Figure 4.13). The tuning fork can ring for a long

time when excited by an impulse and not in contact. However, the ringing rapidly dissipates when the tuning fork is in contact with the ground.

Rube-Goldberg Contraption: This example (see Figure 4.1) demonstrates the ability of our simulator to handle more complicated animations. It consists of three parts: (i) a block feeder with flexible marble-loaded tubes that eject marbles; (ii) a myriad of plastic marble tracks that guide marbles to fall into a wood cup, which is attached to a raised lever, which will eventually lift a bunny and let it fall into a shopping cart; (iii) the shopping cart then descends a bumpy incline and hits a curb. To aid in timing and construction, we simulated the three parts separately, using ending conditions from the previous stage as initial conditions for the next stage.

The flexible tubes barely touch the block feeder. Because of the friction forces, the tubes are deformed as the feeder moves. When the tubes deform sufficiently, the friction forces cannot maintain the deformation, and interesting squeaking sounds are produced. To model the friction sounds of the block feeder against the table, we perturb the normal directions on the bottom of the feeder using Gaussian noise, analogous to the normal maps used in [RYL10]. The double-helix marble runs illustrate the efficiency of our adaptive simulation: we observed a 6.8× speedup over non-adaptive simulation due to the fewer active modes (see Figure 4.16 and 4.17). The small metal balls have lowest modal frequencies well above 20 kHz, so all ball impact sounds ("clicks") were approximated using a recorded metal-ball sound.

Solving large-scale sparse QP problems: We use third-party QP solvers in our implementation. Unlike [KSJP08], we can not use the robust QP solver referred to as "QL" [Sch05] since it only supports *dense* QP which become im-

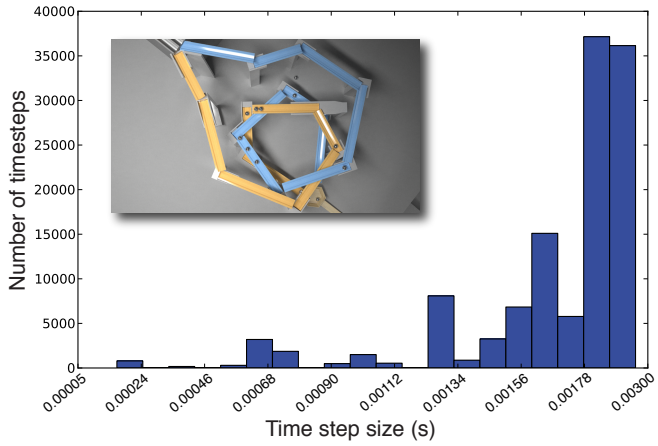


Figure 4.16: **Histogram of timestep size** ("Rube-Goldberg" example) demonstrates that very small timesteps are used rarely, e.g., to resolve transient high-frequency modes.

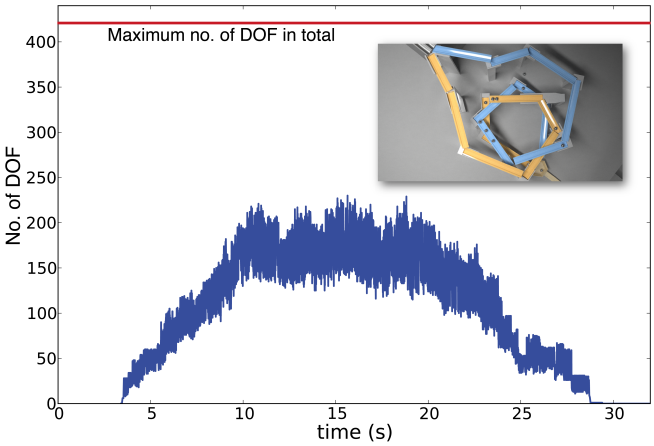


Figure 4.17: **Mode-adaptive contact simulation**

practical for larger problems involving numerous modes. Unfortunately, in our experience, none of the large-scale *sparse* QP solvers are robust enough to successfully solve all of the problems generated by our contact simulations: they can fail to find optimal solutions on feasible problems. Since their failure cases are often rare and different, our practical solution was to use two solvers: the *GALAHAD* package's *QPC* routine (`galahad.rl.ac.uk`), and *KNITRO* (`www.ziena.com/knitro.htm`). Both of them implement the ac-

tive set method and the interior point method. In our implementation, we first try to solve using the QPC routine, then, if it fails, we switch to the active set method in KNITRO. In very rare cases, the second try also fails, then we change to the interior point method implemented in the *QPB* routine in GALAHAD. Using this scheme, we could find optimal (or occasionally only near optimal) solutions to all contact problems.

4.6 Limitations and Future Work

In this chapter, we have proposed a detailed approach for contact resolution which is able to synthesize a new range of challenging sound phenomena, such as vibration-induced chattering and contact damping, as well as to avoid deficiencies in current practice, such as contact-solver-induced noise. We have also proposed an adaptive asynchronous contact solver which makes it possible to efficiently simulate coupling of rigid and modal objects via frictional contact. As a result, we have enabled high-quality sound synthesis for a range of previously unexplored sound phenomena.

Our approach and implementation are not without limitations, and there are many opportunities for future work. While it is clear that speed-accuracy trade-offs can always be made, we are interested in faster methods which retain accuracy. Robustness of the frictional contact solver is dependent on our ability to solve large-scale sparse quadratic programs robustly, and current QP solvers need improvement[2]. Our post-dynamics impulse refinement step generates friction impulses for the sound phase which are not guaranteed to satisfy the

[2]In order to spur algorithmic developments in the QP solver community, a range of challenging QP test problems arising from our simulations will be made available on our website.

maximum dissipation optimality conditions. Our method involves several user-specific parameters, such as for adaptivity, which although straightforward to set in our examples, may be difficult to tune in more complex cases. Further studies are needed into the simulation of squeaking and other high-frequency coupling phenomena. Modal analysis provides only a partial theory for contact sounds, since very small objects, such as marbles, have vibration frequencies which can far exceed our hearing limits, yet they still produce audible contact "clicks." Generalizing contact, sound and radiation models to support larger deformation scenarios is a major unsolved problem for modal sound synthesis. Our current implementation exhibits triangle faceting artifacts, which can be a problem for rolling phenomena [vdDKP01]. Finally there is more to computing multibody contact sounds than just simulating vibrations: multibody radiation effects can also play a significant role in the resulting sound. For example, the sound of a spoon falling into a mug is strongly affected by the mug's cavity resonance, but current single-body acoustic transfer models [JBP06, CAJ09, ZJ10] and large-scale room acoustics techniques [RSM+10] do not entirely suffice.

CHAPTER 5

FLUID SOUND

In previous chapters, we present the physics-based algorithms for synthesizing solid object sounds. In addition to those sound phenamena, another class of sound sources which are ubiquitous in our physical world is liquids: splash, splatter, babble, sploosh, drip, drop, bloop and ploop! Liquids are everywhere and noisy. This chapter presents a practical method for automatic procedural synthesis of synchronized harmonic bubble-based sounds from 3D fluid animations. First, to avoid audio-rate time-stepping of compressible fluids, we acoustically augment existing incompressible fluid solvers with particle-based models for bubble creation, vibration, advection, and radiation. Next, sound radiation from harmonic fluid vibrations is modeled using a time-varying linear superposition of bubble oscillators. We weight each oscillator by its bubble-to-ear acoustic transfer function, which is modeled as a discrete Green's function of the Helmholtz equation. To solve potentially millions of 3D Helmholtz problems, we propose a fast dual-domain multipole boundary-integral solver, with cost linear in the complexity of the fluid domain's boundary. We demonstrate this method by synthesizing different fluid sound phenomena, including water drops, pouring, babbling and splashing.

5.1 Introduction

Computer graphics have seen enormous success of physically based fluid simulation; however, these simulations remain inherently silient movies. For most fluid applications, sound is an afterthought, added using stock recordings. While replaying "canned fluid sounds" is cheap and sometimes plausible, it can

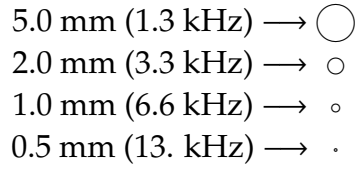


Figure 5.1: **Tiny bubbles** (drawn to scale) are responsible for producing the characteristic high-frequency sounds produced by harmonic fluids. Bubble diameters and vibration frequencies (ω_d) are given.

lack synchronization and physical consistency with observed dynamics, and may appear repetitive and perhaps irritating. Furthermore, while offline applications can rely on talented foley artists to "cook up" plausible sounds at their leisure, future interactive applications and virtual environments will demand algorithms for automatic procedural sound synthesis. Realistic physically based sound methods have appeared for vortex-based fluid sounds [DYN03] and solid bodies [OCE01, JBP06], but we still do not know how to simulate synchronized physics-based sounds for familiar splashes and splatters.

What causes fluid sounds? Perhaps surprisingly, the majority of sound from a splashing droplet of water arises from harmonic vibrations resulting from the entrainment (creation) of millimeter-scale air bubbles (see Figure 5.1). Basically, the bubble oscillator stores potential energy as compressed air and surface tension, and kinetic energy as surrounding fluid vibrations. The important role of these tiny "acoustic bubbles" in water sound generation has been recognized for nearly a century since pioneering work by Minnaert [Min33], and large texts have since been written about them [Lei94]. Recently, van den Doel [vdD05] proposed bubbles as primitives for fluid sound synthesis, and synthesized compelling sounds using stochastically excited modal sound banks.

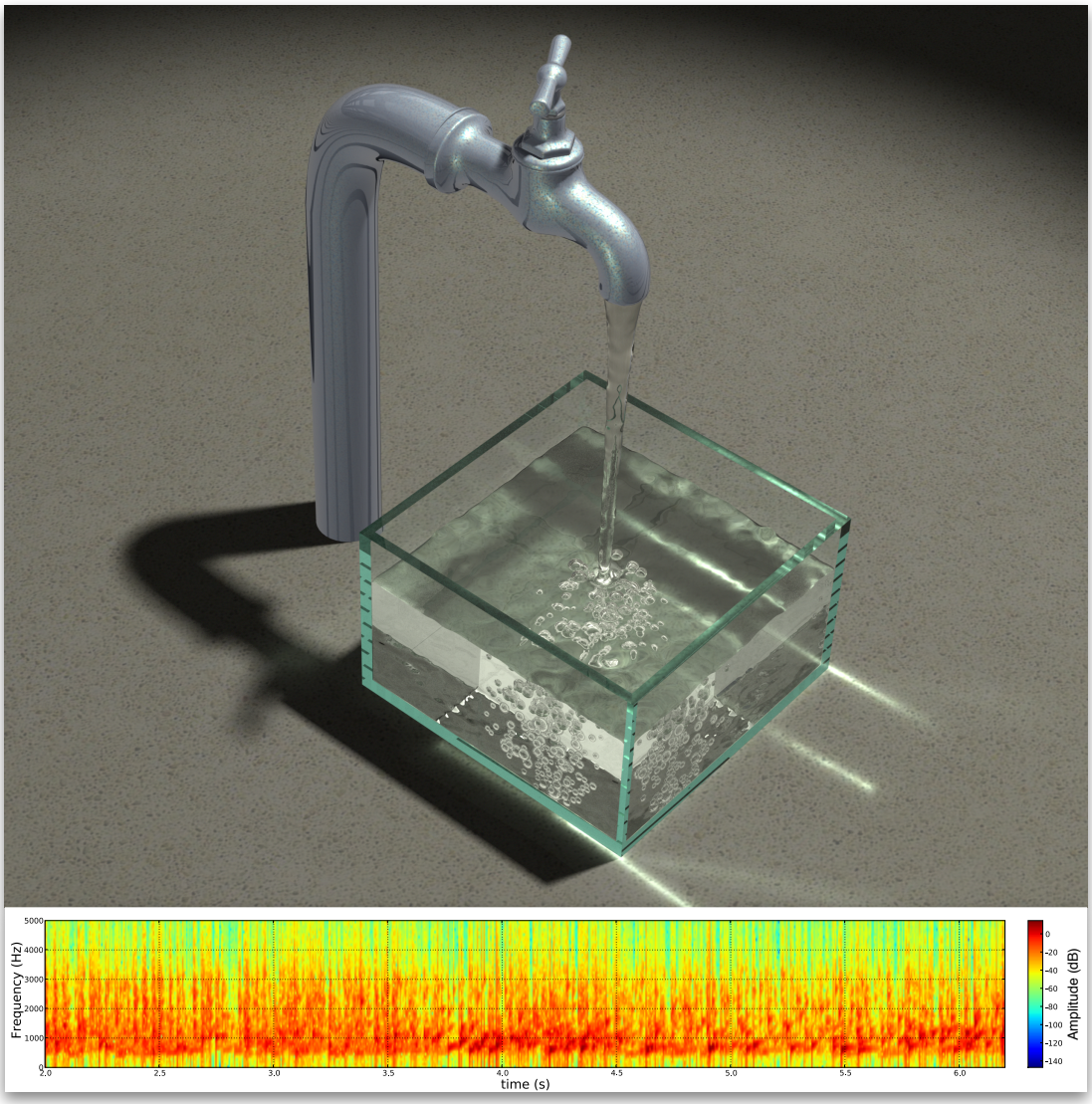


Figure 5.2: **Synthesizing the sound of pouring water** via the linear superposition of acoustic radiation from 7900 vibrating acoustic bubbles.

Exploiting multiple timescales: Ironically, the complex visible motion of the air-fluid interface causes relatively little sound, in part because visible surface motions are inefficient radiators of sound waves at audible frequencies [Bra20]. Instead, the fluid shape vibrates harmonically at audio frequencies due to the microscopic oscillations induced by internal air bubbles, and acts like a shape-changing 3D loudspeaker. For example, consider visible fluid movements oc-

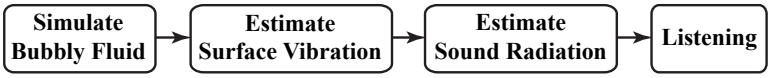


Figure 5.3: **Overview:** (1) We first simulate an incompressible fluid flow with bubbles. For each vibrating bubble we (2) estimate the induced fluid-air surface vibration and (3) resulting air-domain sound pressure. (4) Finally, the linear superposition of bubble sound fields are rendered to the listener.

curring at graphics rates: a water splash on a $15cm$-sized domain might occur over a 10^{-1} second timescale, i.e., a few frames, whereas enormous water sound speeds ($c_{water} \approx 1450m/s$) allow water sound waves to cross the $15cm$ domain in only 10^{-4} seconds. This thousand-fold difference in animation and sound wave timescales is why sound waves can propagate through small fluid bodies almost as if they were standing still. Therefore, we choose to model sound wave propagation and radiation in fluids by assuming they are a sequence of static problems. Given the harmonic nature of bubbles, we can efficiently model sound waves in the frequency-domain using the Helmholtz wave equation.

Our approach: We propose the first practical physically based method for synthesizing synchronized harmonic fluid sounds for computer animation (see Figure 5.2 for a preview). We model the creation of bubbles by air entrainment at the fluid surface; the advection of these bubbles with the fluid flow; the surface vibrations induced by the bubbles' vibrations; and the radiation of these vibrations into the air, producing sound (see Figure 5.3). Our method augments an existing incompressible fluid flow solver with a particle-based acoustic bubble model that models bubble entrainment, advection, vibration, and radiation. By avoiding audio-rate time-stepping of 3D compressible fluid sound waves (which are expensive, and difficult to parallelize), we can extend existing graphics fluid simulators with a pleasantly parallel sound model.

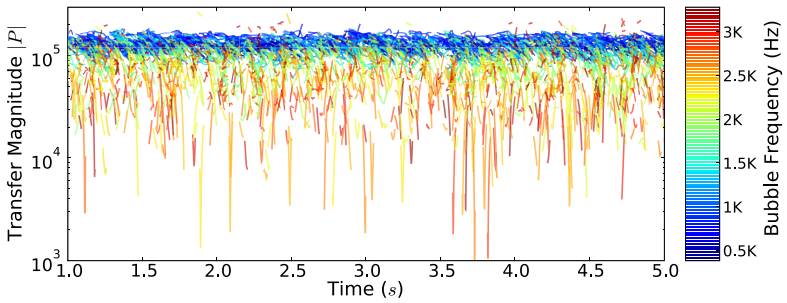


Figure 5.4: **Observed transfer magnitudes** $|P|$ illustrate the complex bubble-dependent temporal structure, and significant hundred-fold variations in magnitude. Frequency colors illustrate that transfer magnitude is not just a function of frequency, but rather has other complex spatial and temporal dependencies. (Data: "water step" example.)

Our main contribution is a parallel algorithm for estimating sound radiation. Spherical bubble vibrations induce harmonic vibrations of the fluid-air interface, which leads to acoustic radiation[1] which we approximate by a time-varying linear superposition of harmonic bubble contributions. The amplitude of each bubble oscillation is effectively multiplied by the bubble-to-ear acoustic transfer function, which we model in the frequency domain as the bubble-located Green's function of the Helmholtz wave equation for the instantaneous fluid geometry. These transfer functions can exhibit complex hundredfold variations which we believe are key to capturing the tonal character of harmonic fluids (see Figure 5.4). Enabling inexpensive Helmholtz Green's function evaluations is achieved by a novel dual-domain multipole approximation based on a two-stage fast linear-time boundary-integral solver. In the first stage, we solve a fluid-domain problem to estimate the normal velocity of the vibrating air-fluid interface. In the second stage, we estimate a multipole approximation of the air-

[1] A graphics analogy: a point light source of specific frequency (the acoustic bubble) radiates light out of a tiny air-filled void (the water) into a highly refractive solid (the air) where it is observed (by the listener). Interestingly, because of the large difference in the speed of sound in water (1497m/s) and air (343m/s), the effective index of refraction is $\eta = 4.4$!

domain acoustic radiation for sound rendering. Key benefits are that the transfer functions need only be updated at fluid simulation rates (or slower), and the only audio rate calculation required is the cheap integration of nonlinear bubble vibrations—thus subsequent sound synthesis can be achieved potentially in real time. We demonstrate harmonic fluid animations involving thousands of acoustic bubble sound sources, with parallelized sound computation times comparable to fluid simulation.

5.2 Background: Incompressible Fluid Solver

Our acoustic bubble simulation is designed to augment existing incompressible liquid solvers familiar to the graphics community [FM96, Sta99, FF01, EMF02]. In this paper, we employ the Euler equations governing inviscid flow [OF03],

$$0 = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho} \nabla p \qquad \text{subject to} \qquad 0 = \nabla \cdot \boldsymbol{u}, \tag{5.1}$$

which relate the liquid's velocity ($\boldsymbol{u}$), pressure (p) and density (ρ). Our approach does not depend critically on any particular fluid simulation method. However, in our implementation we use the FLIP/PIC method [ZB05], since its fluid particles are convenient markers to track bubble creation. We compute the level set function, $\phi(\boldsymbol{x})$ (negative in fluid, and positive in air), using the method proposed in Adams et al. [APKG07], with redistancing performed at each time step using a fast marching method [OF03]. We update the particle-based bubble simulation after each fluid time step using a one-way coupling approximation.

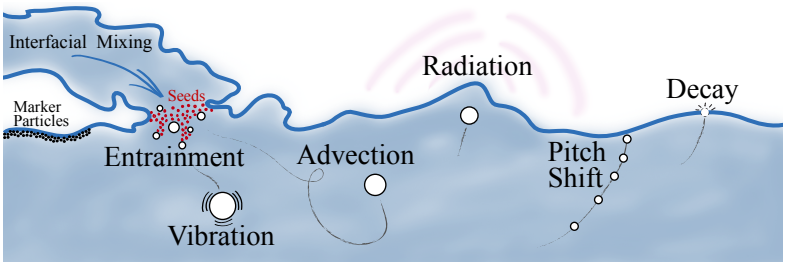


Figure 5.5: **Life of an acoustic bubble**

5.3 Modeling Acoustic Bubbles

Acoustic bubbles have received significant attention, and we refer the reader to the text by Leighton [Lei94] for a comprehensive introduction. Unfortunately their integration into 3D fluid simulators leads to a number of modeling details which need to be addressed (see Figure 5.5). We now summarize the acoustic bubble model used to implement Harmonic Fluids.

5.3.1 The Spherical Acoustic Bubble

Bubbles that generate audible sounds are typically quite small ($\approx$ 1 mm), and in that limit, surface tension forces are strong enough to make the bubble essentially spherical. The spherical air bubble is an excellent oscillator, pulsating after an initial entrainment-related impulse. The simplest linear vibration model assumes an ideal, spherically pulsating, mono-frequency bubble, and was proposed originally by Minnaert [Min33]. It models a spring-bob system where the restoring "spring" force is due to air pressure and surface tension, and inertia is due to the effective mass of the surrounding liquid. Consider a pulsating

spherical bubble with radius

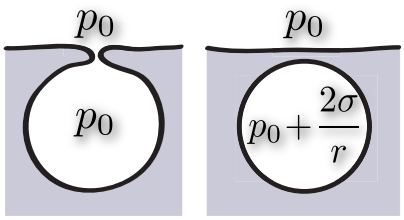
$$r(t) = r_0 + q(t), \tag{5.2}$$

where r_0 is the static radius (which may change slowly), and $q(t)$ is a small fluctuation ($|q| \ll r_0$) due to rapid spherical pulsations. An established linear model of the harmonic pulsations is the simple harmonic oscillator [Lei94]:

$$\ddot{q} + 2\beta\dot{q} + \omega_0^2 q = \frac{F_b}{m_b^{rad}}, \tag{5.3}$$

where ω_0 is the bubble's resonant frequency (in radians/sec); β is the damping rate; F_b is the external forcing due to liquid pressure fluctuations and entrainment; and $m_b^{rad} = 4\pi r_0^3 \rho$ is the bubble's *effective radiative mass*. In practice, we hear the damped natural frequency $\omega_d = \sqrt{\omega_0^2 - \beta^2}$; sample values were given in Figure 5.1. Formulae for ω_0 and $\beta = \beta(\omega_0, r_0)$ are provided in Appendix B.1.

5.3.2 Exciting Bubble Vibrations



At the moment of bubble entrainment, the fluid-trapped air is subjected to an additional pressure: pressure jumps from just air pressure, p_0, to one also involving surface tension, $p_0 + p_\sigma$, where $p_\sigma = \frac{2\sigma}{r_0}$ is the extra surface tension ("Laplace") pressure [Lei94]. For tiny acoustic bubbles, the surface tension pressure jump can be enormous. We model bubble vibration forcing using an initial pressure-jump impulse, and ignore later forcing. This corresponds to forcing the bubble vibration equation (5.3) with the right-hand side given by $\frac{2\sigma}{r_0 m_b^{rad}}\delta(t)$ (for $t = 0$ entrainment), and would yield the oscillator response,

$$q(t) = \frac{2\sigma}{r_0 m_b^{rad} \omega_d} e^{-\beta t} \sin \omega_d t.$$

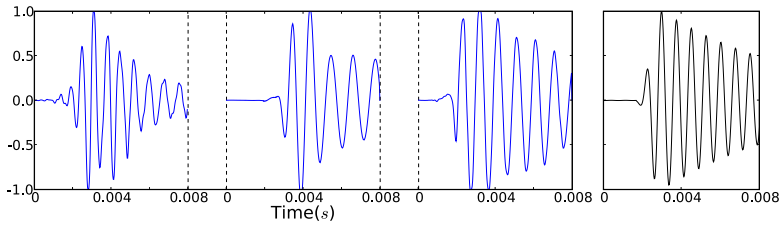


Figure 5.6: **Comparison of bubble excitations:** (Left) Three recorded bubble sounds (in blue) illustrating typical bubble excitation responses and variations; and (Right) the response of our bubble model. Recordings were obtained from individual droplets falling 0.5m from a faucet (at ≈ 2 droplets/sec) into a water-filled container (roughly $30cm \times 30cm \times 15cm$).

Since the frequency and damping coefficients of (5.3) are time dependent, in practice we integrate the vibrations numerically using the mid-point method. To soften the attack (c.f. [vdDKP01]), we smoothly blend the sound in over a $\Delta t = \frac{C}{\beta}$ window. We use $C = \ln(0.85)$ to blend until the amplitude decays to 0.85 of its initial amplitude; our blending function is given by (B.4) in Appendix B.1. By tuning these parameters via comparisons to recordings, our vibration responses appear plausible (see Figure 5.6).

5.3.3 Particle-based Bubble Advection

Identical to previous works, we model bubbles as buoyant particles advected in the incompressible flow [GH04, CPPK07]. Each tiny bubble is advected independently, ignoring complex bubble-bubble interactions. We model the bubble motion as a particle of effective mass $m_b = \frac{4}{3}\pi r_0^3 \rho$ (of the liquid hole), with applied pressure, gravity and drag forces,

$$\boldsymbol{f}_p = -K_p V_b \nabla p_i + m_b \boldsymbol{g} \tag{5.4}$$

$$\boldsymbol{f}_d = \frac{1}{2} C_d \rho A_b (\boldsymbol{u} - \boldsymbol{v}_b) \|\boldsymbol{u} - \boldsymbol{v}_b\| \tag{5.5}$$

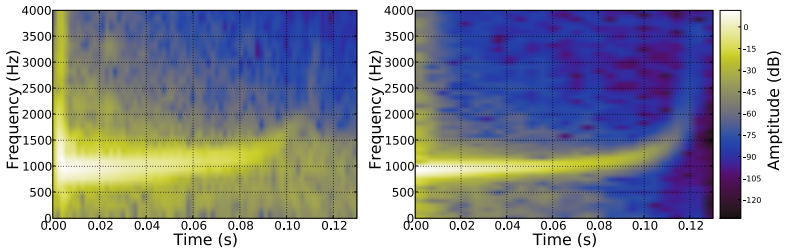


Figure 5.7: **Water drop spectra:** (Left) A recording of a single-bubble water drop experiment; (Right) the spectra of a single-bubble fluid sound synthesized from a digital mockup. Each water droplet fell from a height of ≈ 0.5 meters into a pool of water. Both spectra exhibit qualitatively similar structures, with clear evidence of rising pitch.

where μ_f is fluid viscosity, $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}_b)$ is the fluid velocity at bubble's location, $\boldsymbol{x}_b$, the drag coefficient is $C_d = 0.2$, the bubble surface area is $A_b = 4\pi r_0^2$, its volume is $V_b = \frac{4}{3}\pi r_0^3$, and $K_p = 0.8$ in our examples. The drag force model is suitable for tiny acoustic bubbles which have Reynolds number, $Re = 2\rho\|\boldsymbol{u} - \boldsymbol{v}_b\| r_0/\mu_f \gg 1$. After each fluid time step, we integrate the particle's motion using the mid-point method.

5.3.4 Time-dependent Bubble Frequency

The simple acoustic bubble model (5.3) uses a fixed frequency ω_0 (and damping β). However, a perceptually important feature of moving bubbles is that their frequency can vary significantly over time, with bubble sounds often having a characteristic rising pitch, e.g., the familiar "bloooop?" of a water drop (see Figure 5.7). Phenomenologically speaking, as the bubble approaches the fluid surface, the effective vibrational mass m_b^{rad} of the surrounding fluid decreases (since there is less of it to move), whereas the stiffness k_b (due to surface tension and air compression) is relatively unchanged, thus producing an increase in

the resonant frequency, $\omega_0^2 = k_b/m_b^{rad}$. Models exist for rising bubble pitch as a function of distance to a planar liquid surface [Str53], but can only provide about a $\sqrt{2}$ frequency multiplier—in our experiments, we often observed 2–3 times frequency multipliers likely due to complex local fluid geometry (see Figure 5.7).

To support nonplanar interface geometry and larger pitch shifts[2] we propose an *ad hoc* model based on the bubble's level set values, ϕ. Our frequency-change model captures two behaviors: (i) the bubble frequency does not change until it approaches the surface, and (ii) the faster the bubble travels to the surface, the faster its pitch changes. At each sound synthesis time step we increment the natural frequency, ω_0, by

$$\Delta\omega = K_\Delta\, \omega_d\, e^{-\eta\left(\frac{\phi}{\phi_0}-1\right)}\, \Delta\phi, \tag{5.6}$$

where $\Delta\phi$ is the distance change since the last timestep, $\phi = \phi(t)$ is the (negative) distance to the fluid surface (from the fluid simulator); ϕ_0 is a distance parameter controlling how close to the interface the bubble must be to undergo pitch shift, and η controls the spatial rapidity of change; and K_Δ controls the magnitude of frequency changes. Following ω_0 modification, we update dependent parameters (such as ω_d and β). In our results, we always use $K_\Delta = 72.95$ and $\eta = 1.24$, but adjust ϕ_0 (between -0.008 and -0.025 meters). Our parameters (K_Δ and η) were tuned manually by performing numerous comparisons to real-world experiments. Qualitatively similar results can be obtained (see Figure 5.7). Finally, since ϕ is only evaluated at fluid time-stepping rates, for sound synthesis we temporally interpolate ϕ values to audio rates using a cubic spline; a low-pass filter is also used to remove temporal noise artifacts introduced by

[2] and to avoid estimating the bubble frequency as an eigenvalue of a fluid-bubble interaction problem [Oha04]

the fluid discretization.

5.3.5 Modeling Acoustic Bubble Entrainment

To compute plausible fluid sounds, the entrainment of bubbles by estimated fluid-air mixing must be done so as to produce bubbles with appropriate distributions of radii (frequencies), amplitudes, and spatial and temporal structure. Once a bubble is created and an initial impulse applied, it can be simulated and sonified. Unfortunately, the bubble entrainment process is terribly complex [Lei94] and computationally difficult to resolve spatially and temporally. Therefore we propose a simplified model of the acoustic bubble creation process. Similar to prior work [GH04], we use marker particles to track where bubbles should be created, and our spherical acoustic bubbles are driven by one-way coupling to the fluid simulator. Primary differences in our work ("bubble seeds," bubble creation rates, and modeling of radii and spectra) result from our attempts to make the bubbles sound more plausible. We defer the interested reader to Appendix B.2 for the details of our acoustic bubble entrainment model, and now proceed with sound radiation modeling.

5.4 Modeling Fluid Sounds

Sound radiation is modeled as a superposition of individual bubble sounds. For each harmonic bubble, we first estimate the induced fluid-air interface vibration, then next estimate the radiated air-domain sound waves that travel to the listener. Our multiple timescale approximation models these waves in the fre-

quency domain. We now describe how to estimate the time-harmonic acoustic pressure field, $P(\boldsymbol{x}, t) = P(\boldsymbol{x})e^{+i\omega t}$, where $P(\boldsymbol{x})$ is the (slowly time varying) spatial part satisfying the Helmholtz equation, and ω is the frequency of an acoustic bubble.

Listening to Helmholtz Green's functions: Given a bubble of frequency ω at position $\boldsymbol{x}_b$ in the fluid domain Ω_f, we use the harmonic Green's function $P(\boldsymbol{x}; \boldsymbol{x}_b) \in \mathbb{C}$ of the Helmholtz wave equation on the unbounded region comprised of both fluid and air domains, $\Omega = \Omega_f \bigcup \Omega_a$:

$$(\nabla^2 + k^2(\boldsymbol{x}))\, P(\boldsymbol{x}; \boldsymbol{x}_b) = S_b\, \delta(\boldsymbol{x} - \boldsymbol{x}_b), \quad \boldsymbol{x} \in \Omega, \tag{5.7}$$

where the spatially varying wavenumber is

$$k(\boldsymbol{x}) = \begin{cases} \omega/c_f, & \boldsymbol{x} \in \Omega_f \\ \omega/c_a, & \boldsymbol{x} \in \Omega_a \end{cases} \tag{5.8}$$

and c is the speed of sound (see Table 5.1). The Green's function is subject to an homogeneous Neumann boundary condition on the solid interface,

$$\partial_n P \equiv \frac{\partial P}{\partial n} = 0, \quad \boldsymbol{x} \in \Gamma_s \tag{5.9}$$

which corresponds to a "no vibration" boundary condition of zero surface normal velocity, $v_n = \mathbf{n} \cdot \mathbf{v}(\mathbf{x})$, since $\partial_n P \equiv -i\omega\rho\, v_n$; and the Sommerfeld radiation condition at infinity [How98]. The bubble's source strength S_b (for unit vibration amplitude) is

$$S_b = -4\pi\rho\, \omega^2 r_0^2 \tag{5.10}$$

(see derivation in Appendix B.3). We will often refer to P as the bubble-to-ear acoustic transfer function, or simply "transfer." With this definition, we could approximate the sound contribution at the listening position, $\boldsymbol{x} \in \Omega_a$, due to a

single bubble via

$$|P(\boldsymbol{x};\boldsymbol{x}_b)|\, q(t), \quad \boldsymbol{x}\in\Omega_a \tag{5.11}$$

(or more sophisticated auralizations (see §5.6)). Unfortunately, in practice we desire to solve the Helmholtz PDE (5.7) *for every bubble in the scene at each fluid time step*. To make matters worse, since Ω is the unbounded region, efficient computation and evaluation of this function (for audio rendering) is a practical concern.

5.5 Dual-domain Multipole Radiation Solver

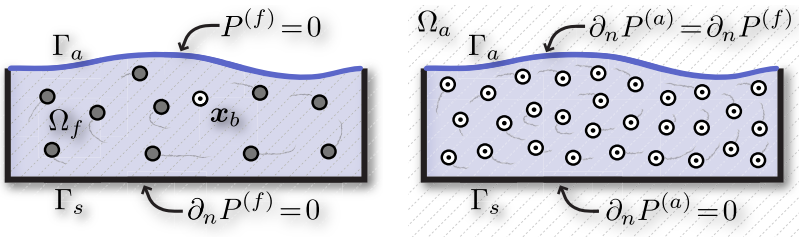
We now describe a novel Helmholtz boundary integral solver for rapid evaluation of the acoustic pressure $P(\boldsymbol{x};\boldsymbol{x}_b)$ to enable sound synthesis from harmonic fluids. We use an efficient two-stage approximation to any bubble's Helmholtz Green's function that exploits the common case wherein fluid vibrations are affected by the surrounding air only weakly. The solver is summarized in Figure 5.8. Readers wishing to skip this section's heavier mathematical details can proceed to §5.6 "Sound Synthesis Pipeline."

5.5.1 Dual-domain Helmholtz Approximation

Let us now consider how to break the computation of the fluid-air Helmholtz Green's function $G(\boldsymbol{x};\boldsymbol{x}_b)$ into two Helmholtz problems with one-way coupling.

Pass #1: First, we compute a fluid-domain Green's function, $P^{(f)} = P^{(f)}(\boldsymbol{x};\boldsymbol{x}_b)$,

$$(\nabla^2 + k_f^2)\, P^{(f)} = S_b\delta(\boldsymbol{x}-\boldsymbol{x}_b), \quad \boldsymbol{x}\in\Omega_f, \tag{5.12}$$



Fluid pressure $P^{(f)}$ problem Air pressure $P^{(a)}$ problem

Figure 5.8: **Overview of dual-domain Helmholtz formulation:** (Left) Pass #1 solves the fluid-domain problem to estimate the fluid's acoustic pressure $P^{(f)}$ assuming that fluid on the solid boundary Γ_s can not vibrate ($\partial_n P^{(f)} = 0$), and that fluid at the air interface is free to move ($P^{(f)} = 0$). In addition to the singular bubble source at $\boldsymbol{x}_b$, numerous advected regular sources also contribute to $P^{(f)}$; their expansion coefficients $\mathtt{c}_f$ are least-squares estimated to match the boundary conditions. (Right) Pass #2 solves the air-domain problem to estimate the air's acoustic pressure $P^{(a)}$ assuming that air on the solid boundary Γ_s can not vibrate ($\partial_n P^{(a)} = 0$), but that vibrations on the air interface Γ_a match the computed fluid vibrations ($\partial_n P^{(a)} = \partial_n P^{(f)}$). The air pressure $P^{(a)}$ is described by a larger number of advected singular multipole sources, whose expansion coefficients $\mathtt{c}_a$ are least-squares estimated to match the Neumann boundary conditions.

subject to the homogeneous boundary conditions on the fluid-air boundary, Γ_a, and the solid-air boundary, Γ_s,

$$P^{(f)} = 0, \quad \boldsymbol{x} \in \Gamma_a, \tag{5.13}$$

$$\partial_n P^{(f)} = 0, \quad \boldsymbol{x} \in \Gamma_s. \tag{5.14}$$

Pass #2: Second, given an approximation of the fluid-domain Green's function pressure, $P^{(f)}$, we can evaluate its normal derivative on the fluid-air interface (which specifies the surface normal velocity, v_n), and use that as an input to estimate the radiation into the surrounding air. The resulting air-domain Green's function, $P^{(a)}(\boldsymbol{x}; \boldsymbol{x}_b)$ satisfies the unforced Helmholtz equation in air,

$$(\nabla^2 + k_a^2)\, P^{(a)}(\boldsymbol{x}; \boldsymbol{x}_b) = 0, \quad \boldsymbol{x} \in \Omega_a, \tag{5.15}$$

subject to the Sommerfeld radiation condition at infinity, and

$$\partial_n P^{(a)} = \partial_n P^{(f)}, \qquad \boldsymbol{x} \in \Gamma_a, \tag{5.16}$$

$$\partial_n P^{(a)} = 0, \qquad \boldsymbol{x} \in \Gamma_s, \tag{5.17}$$

which are derivative (velocity) boundary conditions, with the all important nonzero values on the vibrating fluid interface, Γ_a, and just zero values on any supporting rigid interface, Γ_s. Finally, our $P^{(a)}$ model is used to evaluate $P(\boldsymbol{x}; \boldsymbol{x}_b)$ in Ω_a for sound rendering.

5.5.2 Pass #1: Interior Fluid-domain Solver

We approximate the Helmholtz problems using Trefftz-style equivalent source methods [KK95, Och95, JBP06]. In each pass, the domain PDE is satisfied using a series expansion of fundamental solutions to the Helmholtz equation, and the boundary conditions are approximated in a least-squares sense to estimate expansion coefficients.

Pressure Expansion: To satisfy (5.12) for the fluid-domain Helmholtz Green's function, we introduce the pressure expansion

$$P^{(f)}(\boldsymbol{x}; \boldsymbol{x}_b) = s(\boldsymbol{x}; \boldsymbol{x}_b) + \mathbf{U}^{(f)}(\boldsymbol{x})\, \mathsf{c}_f, \tag{5.18}$$

where s is the singular free-space Helmholtz Green's function,

$$s(\boldsymbol{x}; \boldsymbol{x}_b) = -\frac{e^{-ik_f R}}{4\pi R} S_b, \qquad (R = \|\boldsymbol{x} - \boldsymbol{x}_b\|) \tag{5.19}$$

satisfying the fluid Helmholtz equation

$$(\nabla^2 + k_f^2)\, s(\boldsymbol{x}; \boldsymbol{x}_b) = S_b \delta(\boldsymbol{x} - \boldsymbol{x}_b), \tag{5.20}$$

and the Sommerfeld radiation condition; the second part of (5.18) is a weighted combination of n_f nonsingular functions,

$$\mathbf{U}^{(f)}(\boldsymbol{x})\ \mathsf{c}_f = \left[\psi_1^{(f)}\ \psi_2^{(f)}\ \ldots\ \psi_{n_f}^{(f)}\right]\mathsf{c}_f \tag{5.21}$$

where $\mathsf{c}_f \in \mathbb{C}^{n_f}$ are weights, and $\mathbf{U}^{(f)}$ is a row matrix of functions, $\psi^{(f)}$, each satisfying the fluid Helmholtz equation (without regard for Γ boundary conditions)

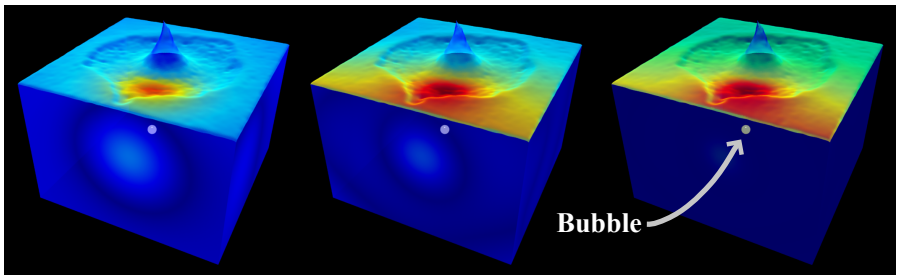
$$(\nabla^2 + k_f^2)\,\psi_j^{(f)}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Omega_f. \tag{5.22}$$

Since the $P^{(f)}$ expansion (5.18) satisfies the Green's function PDE in (5.12), it remains only to select a sufficiently complete basis, $\mathbf{U}^{(f)}$, then find coefficients, c_f, to satisfy the homogeneous boundary conditions (5.13-5.14). We propose using the *regular* spherical Helmholtz solutions [GD05] (other choices are possible),

$$\psi_j^{(f)}(\boldsymbol{x}; \boldsymbol{x}_j^{(f)}) = j_\ell(k_f R)\ Y_\ell^m(\theta, \phi), \tag{5.23}$$

where $\psi_j^{(f)}$ is positioned at $\boldsymbol{x}_j^{(f)}$ (we describe point-source selection later in §5.5.4), $R = \|\boldsymbol{x} - \boldsymbol{x}_j^{(f)}\|_2$, $Y_\ell^m \in \mathbb{C}$ are the spherical harmonics, and $j_\ell(k_f R)$ are spherical Bessel functions of the 1^{st} kind, e.g., $j_0(z) = \frac{\sin z}{z}$, $j_1(z) = \frac{\sin z}{z^2} - \frac{\cos z}{z}$, $j_2(z) = (\frac{3}{z^2} - 1)\frac{\sin z}{z} - \frac{3\cos z}{z^2}$. In our implementation, we use basis functions up to and including quadrupoles ($\ell = 0, 1, 2$), so our n-point multipole expansions have $n_f = 9n$ unknown complex-valued coefficients.

Collocated Least-Squares Estimation: Given the homogeneous boundary conditions on $P^{(f)}$ (5.13-5.14), we collocate the boundary condition equations at N boundary points to obtain N equations involving the $\mathsf{c}_f \in \mathbb{C}^{n_f}$ unknowns, then estimate c_f using weighted least squares. Collocation points are chosen as mesh vertices (discussed in §5.5.5), and each sample point, $\boldsymbol{x}_i$, has normal, $\boldsymbol{n}_i$,



5 quadrupoles 10 quadrupoles 20 quadrupoles

Figure 5.9: **Estimated surface velocity** ($v_n \propto \partial_n P^{(f)}$) computed from the fluid-domain solver (pass #1). Approximations are shown for differing numbers of regular quadrupole sources, and degrees of convergence.

and an effective area, Δa_i. The relevant equations for vertex point i are

$$\mathbf{U}^{(f)}(\boldsymbol{x}_i)\,\mathsf{c}_f = -s(\boldsymbol{x}_i), \qquad \text{when } \boldsymbol{x}_i \in \Gamma_a \tag{5.24}$$

$$\partial_{\boldsymbol{n}_i}\mathbf{U}^{(f)}(\boldsymbol{x}_i)\,\mathsf{c}_f = -\partial_{\boldsymbol{n}_i} s(\boldsymbol{x}_i), \qquad \text{when } \boldsymbol{x}_i \in \Gamma_s \tag{5.25}$$

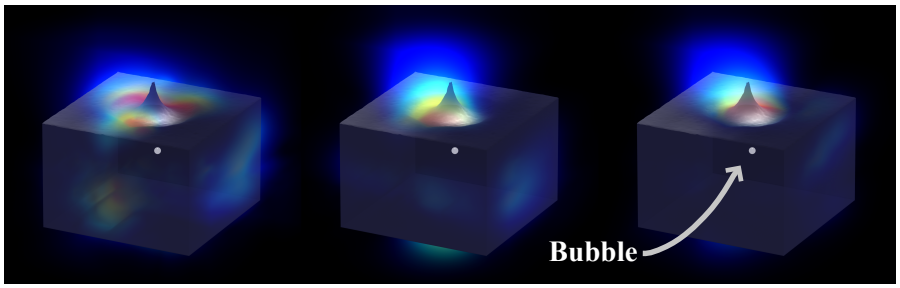
for $i = 1 \ldots N$. Each equation is weighted by $\sqrt{\Delta a_i}$, to assemble the N-by-n_f linear least-squares problem,

$$\boldsymbol{A}\,\mathsf{c}_f = \boldsymbol{b} \quad \Leftrightarrow \quad \begin{bmatrix} \boldsymbol{A}_a \\ \alpha \boldsymbol{A}_s \end{bmatrix} \mathsf{c}_f = \begin{bmatrix} \boldsymbol{b}_a \\ \alpha \boldsymbol{b}_s \end{bmatrix}. \tag{5.26}$$

The relative scaling parameter, α, balances the importance of pressure versus pressure derivative constraints; in our examples (with approximately unit-sized computational domains), we use the ratio of interfacial area, $\alpha = \text{Area}_s/\text{Area}_a$. After robust construction and least-squares solution of (5.26) for $\mathsf{c}_f \in \mathbb{C}^{n_f}$ (discussed in §5.5.6), we can estimate the harmonic fluid-surface vibrations (see Figure 5.9).

Discussion: The boundary integral equation associated with the least-squares problem (5.26) is related to their normal equations, and can be written as

$$\left[\int_{\Gamma_a} \boldsymbol{U}^H \boldsymbol{U}\, d\Gamma + \alpha^2 \int_{\Gamma_s} \boldsymbol{U}_n^H \boldsymbol{U}_n\, d\Gamma\right] \mathsf{c}_f = -\int_{\Gamma_a} \boldsymbol{U}^H s\, d\Gamma - \alpha^2 \int_{\Gamma_s} \boldsymbol{U}_n^H s_n\, d\Gamma$$



40 quadrupoles 60 quadrupoles 80 quadrupoles

Figure 5.10: **Volume-rendered sound pressure,** $|P|$ estimated using the dual-domain solver. Varying quadrupole source counts for the air-domain solver help illustrate visual convergence of the method.

(where $\boldsymbol{U} = \mathbf{U}^{(f)}$, $\boldsymbol{U}_n = \partial_n \mathbf{U}^{(f)}$, $s_n = \partial_n s$). For reasons of solution efficiency and accuracy, we choose to work with the over-determined least-squares problem (5.26) instead of forming the normal equations associated with the matrix boundary integrals.

5.5.3 Pass #2: Exterior Air-domain Solver

The air-domain solver mirrors the fluid domain solver with a couple exceptions. Once we have $\mathtt{c}_f$, we can evaluate the $\partial_n P^{(f)}$ boundary condition on Γ_a (describing the surface velocity; see Figure 5.9), and then solve to get $P^{(a)}$ in the surrounding air (see Figure 5.10). We approximate the exterior radiation solution $P^{(a)}$ to (5.15) by a set of singular multipole sources, whose coefficients are estimated by fitting pressure derivative (normal velocity) data, which is zero except for fluid surface vibrations, $\partial_n P^{(f)}|_{\Gamma_a}$.

Pressure Expansion: We again introduce a pressure expansion, but now use fundamental solutions of the air Helmholtz equation:

$$P^{(a)}(\boldsymbol{x}; \boldsymbol{x}_b) = \mathbf{U}^{(a)}(\boldsymbol{x})\mathsf{c}_a, \tag{5.27}$$

where $\mathbf{U}^{(a)}$ represents n_a singular multipole basis functions,

$$\mathbf{U}^{(a)}(\boldsymbol{x})\mathsf{c}_a = \left[\psi_1^{(a)}\ \psi_2^{(a)}\ \ldots\ \psi_{n_a}^{(a)}\right]\mathsf{c}_a \tag{5.28}$$

the $\mathsf{c}_a \in \mathbb{C}^{n_a}$ are weights, and each basis function $\psi_j^{(a)}$ satisfies the air Helmholtz equation (and Sommerfeld radiation condition),

$$(\nabla^2 + k_a^2)\,\psi_j^{(a)}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Omega_a. \tag{5.29}$$

Since the $P^{(a)}$ expansion (5.27) satisfies the Green's function PDE in (5.15), it only remains to select $\mathbf{U}^{(a)}$ then find coefficients, c_a, to satisfy the $\partial_n P$ boundary conditions. The appropriate basis functions here are *singular* multipole solutions to the free-space air Helmholtz equation (as in [JBP06]),

$$\psi_j^{(a)}(\boldsymbol{x}; \boldsymbol{x}_j^{(a)}) = h_\ell^{(2)}(k_a R)\ Y_\ell^m(\theta, \phi), \tag{5.30}$$

where the source is positioned at $\boldsymbol{x}_j^{(a)}$, $R = \|\boldsymbol{x} - \boldsymbol{x}_j^{(a)}\|_2$, and where $h_\ell^{(2)}$ are spherical Hankel functions of the 2^{nd} kind; $h_\ell^{(2)}(z) = j_\ell(z) - iy_\ell(z) \in \mathbb{C}$, where j_ℓ and y_ℓ are real-valued spherical Bessel functions of the 1^{st} and 2^{nd} kind [AS64]. Again we use quadrupole-order multipoles at each point, so an n-point multipole expansion will have $n_a = 9n$ unknown coefficients.

Collocated Least-Squares Estimation: We estimate the coefficients c_a by matching the boundary conditions (5.16-5.17) using weighted least squares. The equation for collocation sample i is

$$\partial_{\boldsymbol{n}_i}\mathbf{U}^{(a)}(\boldsymbol{x}_i)\,\mathsf{c}_a = -\partial_{\boldsymbol{n}_i}P^{(f)}(\boldsymbol{x}_i), \qquad \text{when } \boldsymbol{x}_i \in \Gamma_a \tag{5.31}$$

$$\partial_{\boldsymbol{n}_i}\mathbf{U}^{(a)}(\boldsymbol{x}_i)\,\mathsf{c}_a = 0, \qquad \text{when } \boldsymbol{x}_i \in \Gamma_s \tag{5.32}$$

which we then weight by $\sqrt{\Delta a_i}$ to obtain the over-determined N-by-n_a linear least-squares problem,

$$\tilde{\boldsymbol{A}}\,\mathsf{c}_a = \tilde{\boldsymbol{b}} \quad \Leftrightarrow \quad \begin{bmatrix} \tilde{\boldsymbol{A}}_a \\ \tilde{\boldsymbol{A}}_s \end{bmatrix} \mathsf{c}_a = \begin{bmatrix} \tilde{\boldsymbol{b}}_a \\ 0 \end{bmatrix}. \tag{5.33}$$

Note that no relative Neumann-vs-Dirichlet scaling parameter (α) is needed here, since only Neumann $\partial_n P$ constraints exist. Finally, we estimate $\mathsf{c}_a \in \mathbb{C}^{n_a}$ using the robust least-squares solver (§5.5.6).

5.5.4 Source Position Selection

Multipole placement affects the quality of the basis functions used in the solver. Traditional equivalent source methods often optimize source placement to increase accuracy [Och95, JBP06], however temporally incoherent source positions can ruin frame-to-frame coherence and lead to noise in synthesized sounds. Our numerical experiments indicate that a sufficient number of randomly selected point sources can achieve a plausible sound. To avoid discontinuities, we randomly select fluid particles as point-source locations when the bubble is created. To ensure both (i) temporally coherent basis functions ((5.23) and (5.30)) and (ii) source positions (and singularities) that remain inside the complex splashing fluid, we advect source positions after each fluid time step.

5.5.5 Sampling Fluid Geometry

After each fluid time step, we extract an N-vertex triangle mesh of the fluid boundary using marching tetrahedra [CP98]; in our examples, mesh resolutions

match that of the fluid grid. Each mesh vertex is used as fluid boundary sample at which to impose boundary condition constraints; for vertex $i = 1 \dots N$ we evaluate and cache the position $\boldsymbol{x}_i$, normal $\boldsymbol{n}_i$, and effective area Δa_i. Sampling fluid geometry can also introduce temporal artifacts in estimated transfer, but these are addressed by temporal filtering/interpolation during the sound rendering process.

One computational difficulty arises when bubbles (or $\psi^{(a)}$) are very close to the fluid boundary, since this can lead to singularities in (5.19) and (5.30). Note that singularities are intrinsic to the problem formulation, since bubbles will always rise to the water surface. In practice we choose to expand the fluid surface slightly to regularize such singularities. In our examples, the boundary isosurface is expanded by one fluid-voxel width by extrapolating the level-set isosurface using the fast marching algorithm. While the accuracy is sacrificed slightly, it is more robust numerically, and we found the sound changes imperceptible. The latter point is perhaps unsurprising since vibrations often decay significantly by the time bubbles reach the surface.

5.5.6 Temporally Coherent Least-Squares Estimation

The under-determined linear systems (5.26) and (5.33) can be nearly singular, and must be solved using a robust least-squares solver. However, common solvers based on the Truncated Singular Value Decomposition (TSVD) should not be used since they can introduce temporal coherence problems: small changes in rank between two time-steps can lead to large magnitude differences in the solution, $\mathtt{c}$ (since the problem is ill-posed). Instead, we use a

ridge regression technique with a QR solver (see §12.1 of Golub and Van Loan [GVL96b]). For example, given our N-by-m linear system, $\boldsymbol{A}\mathsf{c} = \boldsymbol{b}$, the normal equations solution is $\mathsf{c} = (\boldsymbol{A}^H\boldsymbol{A})^{-1}\boldsymbol{A}\boldsymbol{b}$ but $\boldsymbol{A}^H\boldsymbol{A}$ may be near rank deficient. The ridge-regression solution is obtained as $\mathsf{c} = (\boldsymbol{A}^H\boldsymbol{A} + \varepsilon^2\mathbf{I})^{-1}\boldsymbol{A}\boldsymbol{b}$ for a small $\varepsilon > 0$. Unfortunately explicitly forming $\boldsymbol{A}^H\boldsymbol{A}$ can lead to a loss of accuracy (c.f. §5.5.2 *Discussion*). We instead compute c by solving the related $(N + m)$-by-m least-squares problem,

$$\begin{bmatrix} \boldsymbol{A} \\ \varepsilon\mathbf{I} \end{bmatrix} \mathsf{c} = \begin{bmatrix} \boldsymbol{b} \\ \mathbf{0} \end{bmatrix}, \tag{5.34}$$

using LAPACK's double-precision QR-based least-squares solver (`zgels`). The resulting c values (and thus the acoustic transfer pressure values) are more temporally coherent, provided that the same ε value is used; we always use $\varepsilon = 10^{-8}\|\boldsymbol{A}\|_F$.

Linear-time Cost: Since the least-squares solver has complexity $O(m^2N)$, the total dual-domain multipole solver cost is $O(n_f^2N + n_a^2N)$, which is linear in the number of boundary samples, N. In our examples, $n_f < n_a \ll N$, and the dual-domain solves required only 1–4 sec/bubble.

5.5.7 Optimizations and Extensions

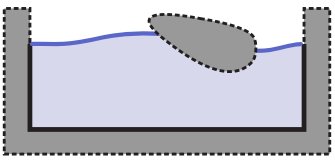
Parallelization: Evaluating independent bubble sound sources is a pleasantly parallel computation. In our fluid preprocess, we implemented the dual-domain multipole radiation solver as a service running on an 80-core Xeon cluster. After each fluid time step, the fluid geometry is updated, the cluster computes every active bubble's transfer function coefficients, c_a, using the radiation solver. Since the simulation of fluids and bubbles are not dependent on radia-

tion calculations, the fluid simulator can advance to the next time step while the acoustic transfer is evaluated.

Adaptive Transfer Evaluation: Evaluating transfer coefficients for each bubble at every time step can be a bottleneck when thousands of bubbles exist. Some simple observations can reduce these bottlenecks without compromising accuracy:

1. *Avoid transfer computations for inaudible bubbles:* Our entrainment-forced acoustic bubble exhibits exponentially decaying vibrations which quickly become inaudible especially in the presence of other bubble entrainment events. In practice, we stop the radiation solve for a bubble after its amplitude decays to 1/1000 of its initial amplitude, e.g., after approximately $T = -\ln 0.001/\beta$.
2. *Temporally adaptive transfer evaluation* avoids computing transfer for bubbles at every timestep. When a bubble's amplitude decays (roughly as $e^{-\beta t}$), we also decrease transfer sampling rates. In our implementation, we use a frequency-dependent sampling rate which roughly gives the sample step size as $\Delta t_{sample} = \Delta t_{fluid}\, e^{\beta t}$, where Δt_{fluid} is the average fluid time-step size. See Figure 5.11.

Triple-Domain Problem: We have considered a dual-domain fluid-air problem where solid objects are abstracted as a thin mathematical interface, Γ_s. However, the sound radiation model could also include nontrivial solid objects, e.g., for splashing objects (see "Splash" example) or a container of finite thickness. In such cases, the interior fluid-domain solve is identical except for the modified



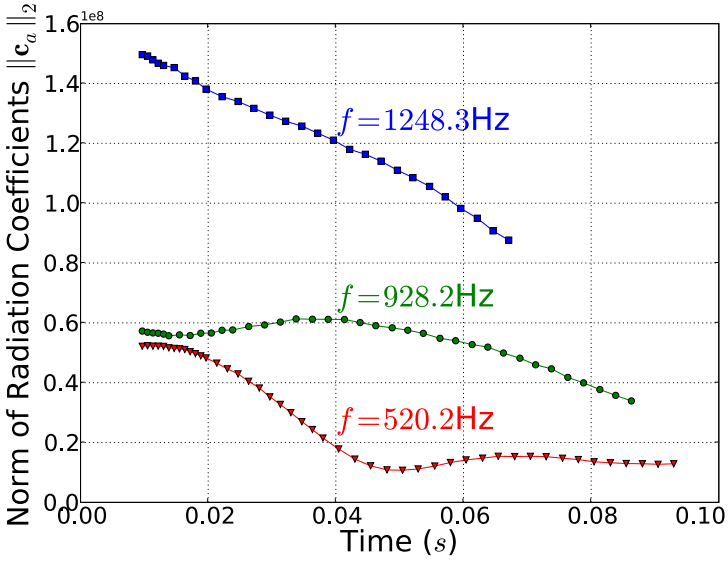


Figure 5.11: **Adaptive transfer evaluation** for three bubbles of different frequency. Bubble lifetimes reflect whether they reached the surface, or became inaudible. We extract a conservative 3× speedup; however, coarser samplings result in greater speedups.

fluid-solid boundary Γ_s. The exterior air-domain problem must be modified to use the larger air-(fluid/solid) interface, and rigid-object scattering can be modeled by adding fixed singular sources $\psi_j^{(a)}$ in the solid region.

5.6 Sound Synthesis Pipeline

We use a two-pass implementation with (i) a fluid and transfer preprocess followed by (ii) a sound synthesis phase.

Fluid Preprocess: Algorithm 4 summarizes the main Harmonic Fluids preprocess. After each fluid timestep (line 4), we advect existing bubbles, $\mathcal{B}$ (line 5) and any multipole-solver source points $\mathcal{P}$ for $\psi^{(f)}$ and $\psi^{(a)}$ (line 6). In line 7 we create new bubbles (updating marker positions, bubble seeds, etc., as described in

Algorithm 4: FluidPreprocess()

```
begin
    while simulating do
        t ← t + Δt;
        timestep_fluid ();
        advect_bubbles (B);
        advect_source_points (P);
        CreateBubbles (B, M, S, t); // (see Appendix B.2)
        create_new_source_points (P);
        record_bubble_φ_values (B);
        if bubbles exist then
            mesh ← mesh_fluid_boundary ();
            (x, n, a) ← pointsNormalsAreas (mesh);
            for bub ∈ bubblesNeedingTransfer(B) do
                eval_transfer (bub, P_bub, (x, n, a));
            end
        end
    end
end
```

Appendix B.2), then (line 8) randomly sample new multipole source points for any new bubbles. Level set values ϕ are recorded (line 9) to model frequency variations (§5.3.4) during sound synthesis.

Parallel transfer computations are then initiated, but only when jobs from the last timestep have completed (line 10). We first mesh the fluid's slightly expanded boundary using marching tetrahedra (§5.5.5), then extract the vector of mesh vertex positions, normals and effective areas, $(\mathbf{x}, \mathbf{n}, \mathbf{a})$. After initializing the remote-procedure-call (RPC) service (line 13), we launch transfer computation jobs on the remote compute nodes using RPC, and send (line 15) each bubble's parameters $(\omega_d, \xi, \ldots)$, multipole-solver source points ($\mathcal{P}_{bub}$), and surface samples $(\mathbf{x}, \mathbf{n}, \mathbf{a})$. Each bubble's transfer job invokes the dual-domain multipole solver (§5.5), first solving for c_f using (5.26), then solving for c_a using (5.33); however, only the small vector c_a of multipole expansion coefficients

Parameter	Value	Description
g	9.8 m/s^2	gravitational acceleration
ρ	1000 kg/m^3	water density
p_0	101.325 kPa	atmospheric pressure
γ	1.4	specific heat ratio of air
σ	0.0726 N/m	surface tension coefficient of water
D_g	2.122e-5 m^2/s	thermal diffusivity of gas
c_f	1497 m/s	sound speed in water
c_a	343 m/s	sound speed in air
μ_f	8.9e-4 $Pa \cdot s$	shear viscosity of water
G_{th}	1.60 $\times 10^6$ s/m	thermal damping constant

Table 5.1: **Physical constants used in our simulations**

are recorded. Adaptive transfer computation (§5.5.7) allows processing only a subset of bubbles (line 14). Finally, once all bubbles have been scheduled for parallel computation, we proceed with the next fluid time step. In our implementation, bubble vibrations and frequency shift (§5.3.4) are not evaluated in the fluid/transfer preprocess.

Sound Synthesis: The sound synthesis stage is much simpler and faster than the fluid preprocess. First, serialized time-series data from the fluid preprocess is loaded, which includes each bubble's trajectory, sampled level-set ϕ values, and multipole expansion coefficients c_a, etc. Given the ear trajectory, the bubble-to-ear transfer functions can be quickly evaluated (in parallel) at the listening position for times when c_a are available. At each audio-rate time step (of size $\delta t = 1/44100$ seconds), the active set of created/deleted bubbles is updated using loaded data, bubble vibrations are time-stepped (including frequency shifts (§5.3.4)), and the ear position determined. Each bubble's sound contribution is accumulated, which involves interpolating/filtering its bubble-to-ear transfer function (to the current time), multiplying by its complex-valued oscillator value $\tilde{q}^{(t)}$ (such that q is the real part of $\tilde{q}$), and applying any head-related trans-

fer function (HRTF) [Vor07]. In our implementation, amplitude filters are used to smoothly blend bubble sound contributions in and out of the sound track since small artifacts can contribute to noise artifacts, especially when thousands of bubbles are present. We synthesize stereo sounds, and use an HRTF model [BD98] (instead of the using the transfer modulus as in (5.11)) to exploit the bubble-to-ear transfer function phase for stereo sound:

$$\mathsf{sound}(t) = \sum_{b \in \mathcal{B}} \mathrm{HRTF}(P_b^{(t)}\, \tilde{q}_b^{(t)};\ \boldsymbol{x}_b^{(t)} - \boldsymbol{x}_{ear}^{(t)},\ \omega_b^{(t)}) \tag{5.35}$$

where the bubble position and frequency parameterize the HRTF.

5.7 Results

We describe results for four different water sounds: (i) falling water drops, (ii) water pouring from a faucet, (iii) water splashing from a falling rigid object, and (iv) a babbling water step. Please see our accompanying video for all animation and sound results. Statistics are in Table 5.2, timings in Table 5.3, and constants in Table 5.1.

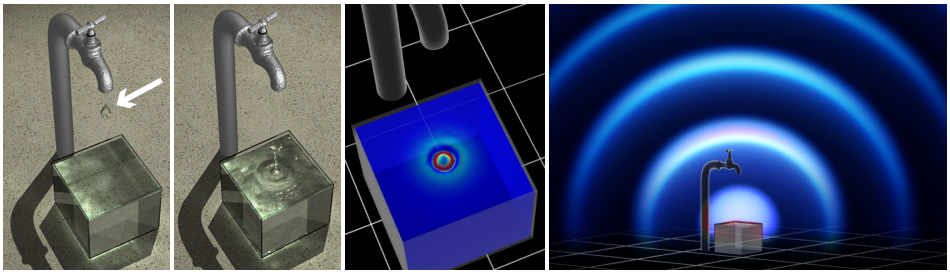
Parallel Implementation: For all our examples, fluid and bubble simulations run on a 16-core 2.4 GHz Xeon node using C++ code. The sound radiation code is compiled into an independent RPC service, and is run on eight 8-core 2.66 GHz Xeon and one 16-core 2.4 GHz Xeon Linux machine. These two parts run in a parallel producer-consumer mode. The fluid simulation generates bubbles and samples surface boundaries as it advances, and launches parallel dual-domain radiation solves using RPC. In our examples, parallel radiation solves complete in less time than each fluid time step, so that parallel sound synthesis adds no additional wall-clock time to fluid simulation. As shown in Table 5.3,

Example	Fluid & Bubble Simulation					Dual-domain Radiation Solve					
	time	Scale (cm)	Voxels	# of Fluid Particles	# of Bubbles	# of Solves	Frequency range (Hz)	min–max #sources		<Fit Error>	max kL
								Fluid	Air	Fluid / Air	Fluid / Air
Droplet	6.4s	14×18×14	70×90×70	1965886	14	2280	500–4K	30–60	80–120	0.06 / 0.18	0.5 / 2.1
Splash	1.5s	45×50×45	90×100×90	3717120	127	25472	300–6K	30–60	80–120	0.08 / 0.24	1.9 / 8.4
Pouring	5.0s	25×40×25	50×80×50	668640	7896	363457	300–6K	30–60	50–80	0.10 / 0.32	1.2 / 5.3
Water Step	8.6s	120×36×72	100×30×60	393376	26657	616846	300–5K	25–60	40–80	0.08 / 0.22	2.0 / 8.7

Table 5.2: **Example Statistics** including temporal duration, grid dimensions, voxel resolutions, the number of FLIP fluid particles and bubbles. Ironically "Water Step" has the fewest fluid particles but the longest fluid simulation time (see Table 5.3); note that particles are "recycled" at the inlet when they exit the computational cell. "Pouring" and "Water Step" have the most bubbles and transfer solves. Frequencies range from about 300 Hz to 6000 Hz. The highest frequency radiation problems are harder to approximate, since for the same domain lengthscale, L, they span more wavelengths per domain, i.e., have higher kL values. We use roughly twice as many quadrupole sources for the highest frequency than the lowest (and linearly interpolate the rest). Similarly, the air-domain problem's smaller wavelengths make it harder to approximate than the fluid-domain problem, i.e., $k_a L \approx 4.4 k_f L$, and therefore we use more sources for the air domain than the fluid domain. Nevertheless, fitting errors for the least-squares problem (average relative residual error, $\|\boldsymbol{A}\mathsf{c} - \boldsymbol{b}\|_2 / \|\boldsymbol{b}\|_2$) were always larger in the air domain. Maximum kL values quantify the difficulty of the highest-frequency Helmholtz approximation problems.

Example	Computation Time (in hours)			
	Fluid	ϕ Update	Radiation	Synthesis
Droplet	0.53 (32%)	1.08 (65%)	0.05 (3%)	0.004 (0.2%)
Splash	0.91 (26%)	2.38 (68%)	0.12 (6%)	0.009 (0.3%)
Pouring	2.57 (29%)	4.34 (49%)	1.86 (21%)	0.044 (0.5%)
Water Step	2.85 (21%)	6.38 (47%)	4.21 (31%)	0.054 (0.4%)

Table 5.3: **Performance Timings:** The parallelized fluid solver (Fluid) and non-parallelized level-set update (ϕ Update) are always the bottleneck in our implementation. Parallelized dual-domain radiation solves (Radiation) are less expensive. Sound synthesis is relatively trivial, and (Synthesis) timings consist primarily of nonoptimized gigabyte file I/O. Overall, transient few-bubble sounds ("Droplet" and "Splash") are significantly less expensive than continuous many-bubble sounds ("Pouring" and "Water Step").



Drop Splash $|v_n|^2$ Sound pressure

Figure 5.12: **Falling water droplet** splashing and entraining bubbles. The estimated surface normal velocity ($|v_n|^2$) is shown at the time of impact. Resulting pressure waves are volume rendered for illustrative purposes only.

even for simulations with tens of thousands of bubbles, the bottleneck is our fluid simulation.

EXAMPLE (Falling Water Drops): We simulated three large droplets falling from a faucet into a small tank of water (see Figure 5.12). As in all our examples, transfer is computed for an isolated fluid source; here we ignore surrounding faucet and floor geometry. Since only 14 bubbles were generated, computing costs are dominated by fluid simulation (see Table 5.3). For convenience, we

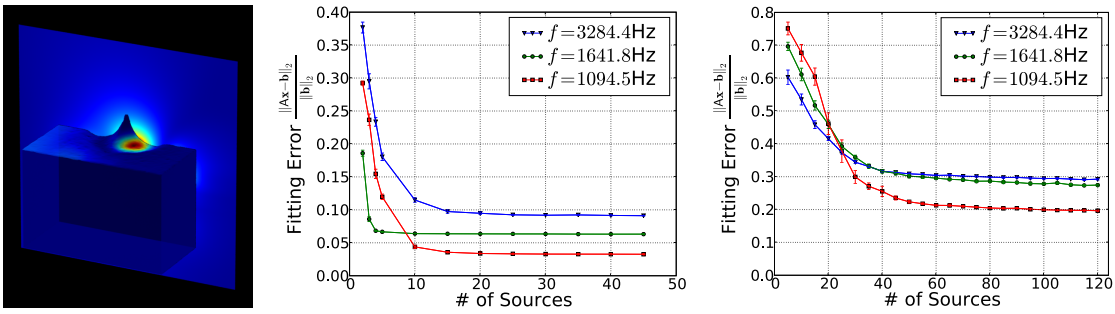


Figure 5.13: **Dual-domain approximation results** for (Left) a single bubble inside a fluid volume deformed after droplet impact: (Middle) fluid-domain and (Right) air-domain convergence rates (with error bars for 95% confidence interval) for randomly distributed quadrupole sources, but fixed geometry and $\boldsymbol{x}_b$. Both curves indicate quick decay to a nominal accuracy suitable for plausible sound rendering.

also provide a "wet" sound using a simple reverberation filter. Recordings of individual bubble sounds were used to originally tune our bubble entrainment model's parameters. See Figure 5.7 for qualitatively similar spectrograms of a recorded droplet sound and our digital mockup. A convergence analysis is provided in Figure 5.13 for the fluid-domain and air-domain solvers.

EXAMPLE (Pouring Water): This example (see Figure 5.2) is geometrically similar to "water drops," but generated 7896 bubbles (564× more) and required 363,457 transfer solves. Characteristic bubble "chirps" can be heard here and in "water drops."

EXAMPLE (Splashing Water): We simulated a small rigid sphere splashing into a water tank (see Figure 5.14) using a technique similar to [CMT04]. This example is an instance of the "Triple-Domain Problem" (§5.5.7), and we place a quadrupole sound source inside the rigid sphere in the air-domain radiation solver. The radiation computation was relatively cheap for this short transient

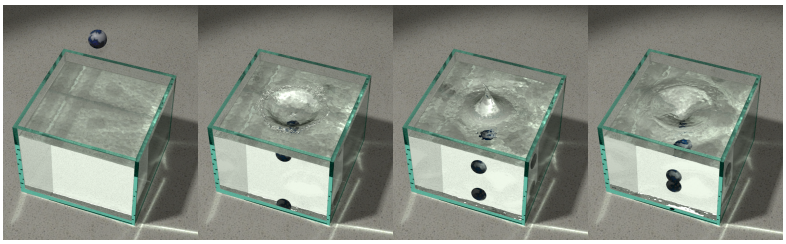


Figure 5.14: **Splash example**

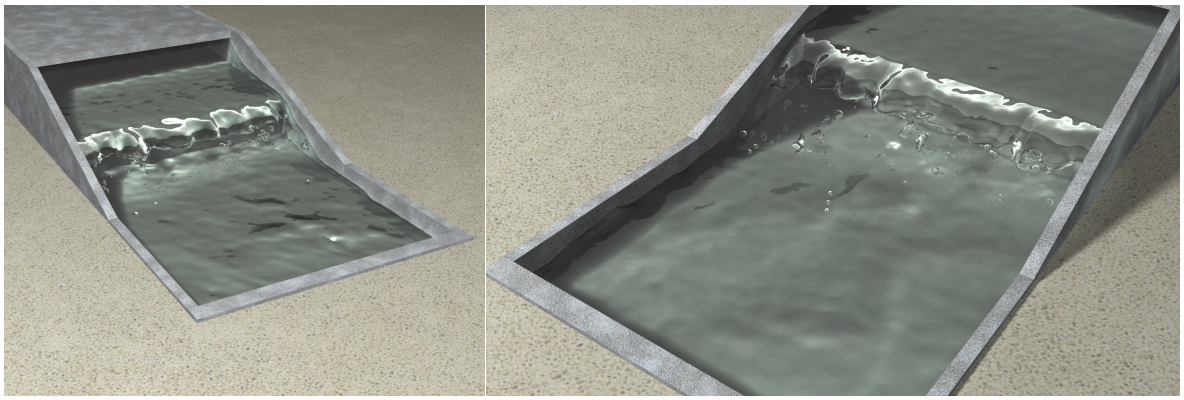


Figure 5.15: **Water "babbles" as it flows over a small step**

sound.

EXAMPLE (Babbling Water Step): Our most computationally intensive example is water flowing over an horizontal surface with a small downward step (see Figure 5.15). The example produces characteristic babbling and chirping sounds.

Fixed sources: Unlike other examples where multipole sources are advected, in this example we fix sources within the water domain to avoid them entering/leaving the domain. Bubbles that reach the interface (or otherwise exit) have their transfer function value frozen at the last computed value.

COMPARISON (constant vs. changing bubble frequency): We synthesized pouring and "water step" sounds with and without bubble frequency changes

(§5.3.4) to demonstrate their subtle but perceptually important effect. Transfer functions were unchanged. The constant-frequency sounds tend to sound more like computer-generated noise, whereas the nonconstant-frequency sounds have richer variations and exhibit more chirping and babbling sounds.

COMPARISON (to real-world splashing): To compare against an actual splashing sound with constant visual stimulus, we replaced the sound track with recordings of real-world splash mock-ups. We provide a single comparison, with mono-phonic sound. Although the sounds are qualitatively similar, the real sound has more complex tonal variations during the latter splashing phase.

COMPARISON (different radiation solver errors): A strength of our radiation solver is that it can exploit the relatively low boundary-condition accuracy (recall Figure 5.13) needed to produce plausible fluid sounds in the listener's far-field location. To evaluate the impact of larger radiation solver errors, the video compares "water step" animations with different boundary-condition errors in the fluid/air domain radiation solvers (18%/40%, 12%/35%, 8%/22%). Although the sounds are qualitatively similar, the low-accuracy radiation coefficients tend to exhibit greater temporal variations (likely due to the ill-posedness of the least-squares approximation) resulting in greater noise in the synthesized sound. Some listeners also perceived localization errors in low-frequency bubbles, possibly due to left/right-ear transfer errors in phase and/or amplitude. We recommend using higher-accuracy approximations when possible to minimize artifacts.

5.8 Limitations and Future Work

Physics-based fluid sound synthesis is a new area, and significant challenges remain. Our proposed model enables sound rendering for harmonic fluid phenomena, however its physical simplifications and limitations provide many avenues for future work.

The mono-frequency acoustic bubble provides a good starting point for modeling sound radiation, but is rather simplistic. It neglects higher-order linear vibration modes, which is often justified by the fact that higher-order linear modes radiate less well than monopoles. However more complex nonlinear vibration modes also exist, and can contribute to far-field radiation [Lei94]. Both linear and nonlinear bubble vibrations can also lead to significant inter-bubble coupling effects; dense bubble concentrations, such as in foam or plumes, pose particular nonlinear challenges, especially for radiation modeling [Dea97]. Very large bubbles can be important, and demand special attention given the complexities of nonlinear vibrations and acoustic radiation. Bubbles approaching the interface can lead to singularities in our boundary integral solver, and a better model of nonspherical acoustic bubbles at the interface is needed. Bubble popping and merging are missing interfacial phenomena, as are boiling and fizzing.

Bubble forcing could be improved. We only considered an initial entrainment-related pressure impulse, but later pressure forces can be important, especially for larger bubbles [Lei94], e.g., consider large bubbles rising from a scuba diver. Unfortunately audio-rate pressure forcing can be expensive to evaluate accurately.

Our bubble entrainment model is stochastic, but actual entrainment statistics are more complex [Lei94]. Our model also lacks dependence on pressure, which can be important for impact and splashing, especially at high velocities [Fra59]. Future models should reduce parameter tuning needed to achieve realistic bubble distributions and spectra.

Our dual-domain multipole solver can be a good approximation for compact sound sources (with modest kL values), however it is less well-suited to larger sources, such as a swimming pool. Similarly, we have not considered underwater listeners, which could avoid air-domain solves but would be complicated by large fluid domains. We have modeled harmonic fluid sound sources, but it still remains to integrate these sound models into larger acoustic environments. Including scattering effects of surrounding geometry, especially for larger sound sources, remains a challenge. Low-error approximations may necessitate more sophisticated frequency-domain solvers [GD05]. However, reviewers pointed out that analytical solutions for simplified planar fluid-interface geometry may suffice for some applications.

Splashing sounds produced by an impacting elastic object can also include significant elastic object sound contributions [Fra59]. In general, fluid-solid-air coupling methods are needed to capture the effects of vibrating solid objects, e.g., when pouring water into a plastic cup or metal sink.

Opportunities exist for accelerating sound synthesis, and real-time Harmonic Fluid sound sources appears feasible. The frequency-domain radiation preprocess is pleasantly parallel, but numerous bubble sound sources may become a bottleneck. Opportunities clearly exist for perceptually based sound rendering by using degraded sound quality and exploiting perceptual mask-

ing, etc. Time-domain solvers for the wave equation may also be a viable methods for integrating the contributions of many bubbles. Finally, physically based sound rendering might be combined with data-driven and stochastic methods to exploit complementary advantages for more complex and noise-like phenomena, e.g., Niagara Falls.

CHAPTER 6

RELATED WORK

In computer graphics, there have been previous work on physics-based sound synthesis for a few simple sound phenomena. On the other hand, much of previous research on animating detailed motions—from rigid and deformable simulation to fluid and bubble simulation—has been addressed. These techniques provide useful physical information about the dynamics for sound synthesis. This chapter first summarizes the previous work on physics-based sound rendering followed by an outline of the previous work on related physically based simulation methods.

6.1 Solid Object Sound

Rigid Modal Sound: Sound synthesis from physically based animations has a long history in graphics [TH92, Gav93]. And linear modal vibration has been used in animation [PW89] and in physically informed sonic modeling of complex multi-impact, percussive, and stochastic sound sources [Coo97]. Modal sound synthesis methods generate plausible rigid body contact sounds efficiently, and were popularized by van den Doel and Pai [vdDP96] who proposed using analytical models of linear modal vibration to produce point-contact sounds in interactive virtual environments. Modal sound models can be conveniently estimated from recorded sounds and measurements [vdDKP01], or estimated using numerical techniques for linear modal analysis [Sha91, OSG02]. For animation, simply running dynamics simulations at graphics rates can greatly limit the range of sounds achievable. Van den Doel et al. [vdDKP01] realized the perceptual importance of resolving micro-collision forces sampled

at (near) audio rates, and explicitly distinguished them from the so-called "dynamics force" sampled at graphics rate, but the range of resolvable contact phenomena was limited by a point like contact model. O'Brien et al. [OSG02] used tetrahedral finite element simulations to synthesize modal sounds for rigid bodies excited by contact forces from graphics simulations; however the rigid model cannot capture deformable contact events.

Numerous simplified (modal) sound models have been proposed for rigid body contact. Hahn et al. [HFGL98] introduced *timbre trees* to model parameterized sound models. Other developments in modal contact sound synthesis include speed-accuracy trade-offs for sound in interactive applications [DKP04, RL06, BDT+08, RYL10], and acoustic transfer models to improve realism of spatialized sounds [JBP06], in which acoustic transfer functions are precomputed and efficiently represented using multi-point multipole expansion [Och95]. Chadwick et al. [CAJ09] considered subspace integration techniques to resolve nonlinear mode-coupling effects in thin shells, but treated objects as rigid for collision processing.

Contact-dependent damping for sound synthesis is not well addressed. Previous works often used a static parametric model for vibration damping [OSG02], and so does our work in [ZJ10], or apply some additional *ad hoc* damping when objects are in contact (e.g., with the ground in [CAJ09]). Our work in [ZJ10] also introduced additional contact damping for objects at rest to mask noise introduced by the iterative contact solver. Our work [ZJ11], in contrast, resolves frictional contact, and proposes a viscous model to approximate spatially and temporally dependent contact damping.

Non-Modal Sound: Non-modal sound synthesis for computer animation was explored in [OCE01] using an explicitly time-stepped large-deformation finite element model to simulate the surface response of an object due to external forces, potentially being able to resolve more detailed contact events. However, explicitly time-stepped detailed models at audio rates is expensive, and the stiff materials (e.g., steel) can result in onerous timestep restrictions. In contrast, in our work [ZJ11] we resolve small modal deformations in multibody contact, but only simulate a subset of vibration modes by using an efficient adaptive, asynchronous contact algorithm.

Fracture Sound: To the best of our knowledge, no prior work has addressed physically based sound synthesis directly from 3D brittle fracture animations. Most fracture sounds in games or computer-animated movies used data-driven approaches based on pre-recorded fracture sound effects (e.g., see [PO09]), with fracture events being natural candidates for event-based sound rendering [TH92]. Unfortunately, such approaches have the usual limitations of lacking synchronization, physical correctness, variability and/or requiring large sound file databases. To improve the effectiveness of recorded sounds, Picard et al. [PTF09] recorded, analyzed, and resynthesized impact and breaking sounds using granular synthesis techniques. Plausible contact sound clips were generated for a rigid-body simulator, however no visual or acoustic simulations of 3D fracture processes were considered. The physical characteristics of breaking sounds are discussed briefly by Rath and Fontana [RF03]. In early psychophysical experiments, Warren and Verbrugge [WV84] explored bouncing and breaking sounds of a glass bottle, and the ability of listeners to correctly identify between the two stimuli. The also manually synthesized plausible sound clips by identifying and editing impact/fracture sound events. Fracture sound record-

ings can exploit the inability of listeners to tell which fracture sound came from which fracture event, but lack variability and synchronization for more complex scenarios.

6.2 Fluid Sound

Fluid sounds can arise via numerous mechanisms [Bla86] including harmonic bubble-based fluid sounds [Lei94], vortex based sounds (e.g., whistling) [How02], shock waves (e.g., from explosions), and through fluid-solid coupling with vibrating solids [How98]. Perhaps the least familiar but most common, harmonic bubble-based fluid sounds come with almost all kinds of fluid movement: splashing or pouring water [Fra59], rain drops [LH90], babbling brooks [Min33], etc. Bubbles have received enormous attention due to vibration-based sound radiation and other exotic behaviors, such as cavitation (which can pit propellors) and even their ability to give off light via sonoluminescence! Bubble-related sounds have been studied for about a century, and people understand, albeit not entirely, the basic mechanisms for sound emission. It was realized nearly a century ago that it is hard for water to make any sound by itself [Bra20], and Minnaert [Min33] described the important role of harmonic acoustic bubbles. Continued work has revealed that most of the sound arises not from the initial impact of fluid but from small bubbles entrained from the resulting surface cavity [PCB89, LH90, OP90]. The acoustics community has studied acoustic bubbles extensively because of their wide importance, e.g., in computational ocean acoustics [Jen94], in estimating rainfall rates for climate models [Uri75], and understanding sounds from complex bubble plumes in breaking waves and surf [Dea97]. However, we still lack practical

algorithms for synthesizing harmonic fluid sounds.

6.3 Sound Auralization

Realistic sound rendering in computer graphics has addressed auralization of sound sources in virtual environments [Beg94, KDS93, Vor07] especially for interactive virtual environment acoustics [FMC99, TFNC01, TGD04, RSM+10], however, less work has addressed the physically based modeling of realistic sound sources.

6.4 Related Simulation Methods

Rigid and Deformable Simulation: Resolving frictional contact with rigid bodies has long been recognized as a challenging problem in graphics and engineering, in part due to the inherent difficulty of contact discontinuities, nonlinear Coulomb friction, and solution non-uniqueness [Bro99, Ste00]. For flexible multibody simulations, e.g., where modal vibrations are included, additional difficulties arise, especially for high-frequency vibrations [WN03]. Early methods for visual simulation of rigid and deformable objects explored penalty-based methods [Hah88, MW88]. While penalty contact methods can generate low-noise contact impulses for rigid body dynamics ([JBP06]), these methods can suffer from stability issues that necessitate extremely small timesteps, and challenges exist for practical modeling of frictional contact and modal chattering. More accurate models of rigid body contact resolution have traditionally motivated Linear Complementarity Programming (LCP) formulations. Unfortunately solving LCP contact problems can be hard computationally, and

for certain problems solutions may not exist or be unique when friction is considered [L̈82]. Developments in graphics tackled rigid body contact using acceleration-level LCP formulations [Bar90, Bar91], however solution existence could not always be guaranteed [Bar93]. Velocity-level LCP formulations were later proposed which can always find a solution [ST96, AP97]. More recently the iterative velocity-level LCP methods have seen greater use in practice due to increased performance in exchange for degraded accuracy and stability [GBF03, Erl07]. Alternate quadratic program (QP) formulations of the LCP are also possible [Mor66] (but are often non-convex and not easy to solve), and have been exploited by some works in graphics [MS01, KSJP08]. Unfortunately, rigid body and flexible multibody contact methods devised for computer graphics and engineering were never designed for physically based sound synthesis in mind, and their straightforward application leads to problems with noise, accuracy, and efficiency (discussed further in §4.2). While low-noise contact simulation has been demonstrated in certain scenarios, e.g., for continuous single-point contact simulation involving smooth surfaces [vdDKP01, KP03], we address the need for sound-aware contact resolution methods for practical sound synthesis involving rigid and flexible multibody systems. Our method leverages prior work on the Staggered Projections (SP) method [KSJP08]; however, simply running SP at audio rates—despite being prohibitively expensive—is unsatisfactory for sound synthesis since the method suffers from accuracy issues which introduce noise in the final sound.

Asynchronous and Adaptive Simulation: Only a few methods for simulating rigid and/or deformable objects provide temporal adaptivity or asynchronous integration. Mirtich [Mir00] introduced Jefferson's timewarp algorithm [Jef85] into graphics, and enabled asynchronous rigid body simulation whose timestep

size is adapted based on accuracy requirements. In comparison, our method can asynchronously evolve both rigid and deformable objects, and can adaptively switch between rigid and deformable simulations. Mode-culling techniques avoid synthesizing sounds for inaudible modes, but are different from adaptive modal contact resolution. Kim and James [KJ09] adapt the modes of a nonlinear subspace deformation model but do not consider contact. For non-modal deformable simulation, hierarchical multi-resolution methods accelerate simulations using spatial [GKS02, CGC+02] and temporal adaptation [DDCB01], whereas we exploit the transient nature of high-frequency modes to provide temporal adaptivity. Harmon et al. [HVS+09] consider the more complex case of asynchronous contact mechanics wherein numerous deformable elements are integrated asynchronously with timesteps dependent on element type, proximity, and contact attributes. In contrast, in our work [ZJ11], we only consider body-level asynchrony wherein each deformable model is synchronized with its contact group, and the timestep size is determined by its highest frequency mode.

Fracture Simulation: The visual simulation of fracture has a long history in computer animation [TF88, NTB+91]. O'Brien and Hodgins [OH99] animated brittle fracture, avoiding mesh aliasing artifacts with tetrahedral remeshing of crack propagation through an explicitly integrated finite element model. To avoid time-step restrictions associated with fast acoustic waves in stiff brittle materials, Smith et al.[SWB00] proposed a constraint-based fracture approach, but lacked sub-element fracture. To address remeshing challenges and time-step restrictions due to sliver elements, the virtual node algorithm [MBF04] and meshless methods have been proposed [PKA+05]. None of these approaches address brittle fracture sounds, however most of these time-domain methods

could, in principle, be modified to apply contact and fracture impulses to our rigid-body sound models.

Fluid Simulation: On the other hand, the computer graphics community has developed a sophisticated array of computational methods for simulating fluids in computer animation [FM96, Sta99, EMF02, OF03]. Because of their visual importance, numerous methods for animating bubbles and foam have appeared [FM96, GH04, SSK05, ZYP06, CPPK07, TSSMM07, KLL+07, KC07, HLYK08]. However, no methods have addressed fluid sound generation.

CHAPTER 7

FAST SELF-COLLISION DETECTION

In this chapter, we switch gear from sound synthesis and present a fast self-collision detection method. We consider the problem of accelerating self-collision tests for arbitrarily deforming triangle meshes. This method can improve the performance of general deformable simulations, such as cloth, thin shell, and elastic volumes, where self-collision detection can be computationally expensive, and lead to a performance bottleneck. Self-collisions often produce force impulses and generate sounds. Therefore, fast self-collision detection can also significantly improve the performance of physics-based sound rendering. This is because high-performance self-collision tests can largely improve the simulations which must run at small time-steps to resolve audible collision events.

7.1 Introduction

Self-collision detection (SCD) methods are widely used in computer graphics and engineering to enable realistic simulation of self contact for highly deformable objects. Various methods have been devised to accelerate the numerous triangle-triangle overlap tests, however few methods exist that can entirely avoid SCD tests over large mesh regions. Recently Subspace Self-Collision Culling (SSCC) was proposed [BJ10], wherein precomputed certificates are used to cull SCD tests for large mesh regions within bounding volume (BV) nodes– sometimes even the entire model. Unfortunately, the use of SSCC is restricted to a very special class of low-dimensional subspace deformations, such as modal

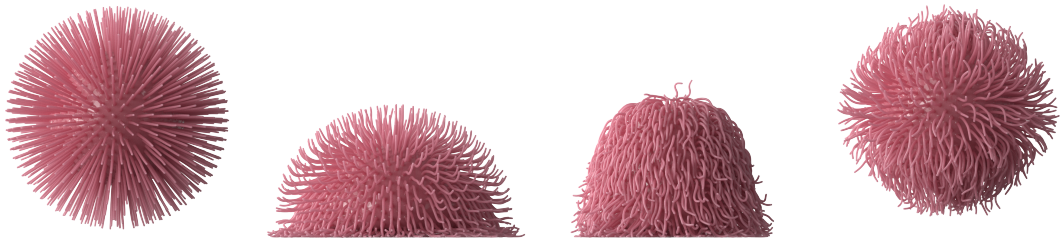


Figure 7.1: **A squishy ball with 820 tentacles** and over 1 million triangles, squishes and bounces on the ground, inducing numerous small interpenetrations. Our Energy-based Self-Collision Culling (ESCC) method accelerates self-collision detection (SCD) for arbitrarily deforming triangle meshes, such as this mesh animated using Oriented Particles [Müller and Chentanez 2011]. We observe an **11.5× speedup** over an optimized AABB-Tree SCD implementation on this challenging example.

deformations, which prevents its use with the majority of deformable models in use.

In the paper [ZJ12], we propose Energy-based Self-Collision Culling (ESCC), a generalization of certificate-based self-collision culling to arbitrary mesh deformations. We use a deformation energy $E(\boldsymbol{u})$ to measure "how much" a mesh patch deforms due to vertex displacements $\boldsymbol{u}$. We precompute the minimum energy $\mathcal{E}$ required for that mesh patch to self collide, then use this self-collision *certificate* to cull SCD tests at runtime: after a mesh deforms we compute $E(\boldsymbol{u})$, then if $E(\boldsymbol{u}) < \mathcal{E}$ we are guaranteed that no self collision can occur. Our ESCC certificates works seamlessly with the traditional BVH-based self-collision detection methods and can accelerate them significantly. We first precompute certificates of surface patches contained in BV nodes. Given a node at runtime, before the standard BVH traversal to test for self collisions, we compute the node's deformation energy to evaluate its certificate. If the certificate is valid, the subsequent BVH traversal can be completely culled. Consequently, ESCC can rule out regions with little deformation, resulting in faster self-collision processing.

While many deformation energy models are possible, we propose an affine-invariant Laplacian energy model that measures nonsmooth deformation, and has several speed advantages. Our major contributions of this method are the algorithms for fast runtime energy evaluation, and fast certificate precomputation for BV nodes. Leveraging the BVH structure, runtime energy computation can be done hierarchically in $O(N)$ flops, and much faster than traditional SCD. As a result, significant speedups in SCD can be achieved (see Figure 7.1 for a preview of our results). To enable a very fast triangle-triangle certificate preprocess, we propose several techniques (detailed in our supplemental appendices) that enable the underlying QCQP optimization problems to be solved exactly, and at low cost.

7.2 Self-Collision Detection Using Certificates

The most common approach to detect self-collisions of a triangle mesh is to build a bounding volume hierarchy (BVH) for the mesh and query the BVH against itself (see Algorithm 5 without the blue-colored code blocks). Each BV node contains a subset of the triangle mesh. To find collisions between mesh regions contained by two BV nodes, the algorithm first checks if their bounding boxes are intersected. If they are separated, it is guaranteed to have no collisions between the two nodes, and is sufficient to go no deeper on the BVH for collision detection. If the two BV nodes overlap, then the algorithm has to check collisions over all pairs of their children. Fast collision detection algorithm relies on culling expensive triangle-triangle intersection tests at high levels of BVH traversal. However, this is usually hopeless for self-collision detection, because, even for undeformed meshes, the algorithm always has to test triangle-triangle

Algorithm 5: Energy-based self-collision detection

```
procedure: BVH_Self_Traversal (r)
input    : BVH root node r
begin
    stack.put (r, r);
    while not stack.empty () do
        n,n' ← stack.pop ();
        if n = n' then
            if has_certificate (n) and
              deformation_energy (n) <certificate (n) then
                continue;
            if is_leaf(n) then
                triangle-triangle intersection tests for all non-neighboring triangle pairs
                of n;
            else
L1              foreach distinct children pairs c, c' of n do
L2                  stack.put(c, c');
                end
L3              foreach child c of n do stack.put(c, c);
            end
        else
            if has_certificate (n ∪ n') and
              deformation_energy (n ∪ n') <
              certificate (n ∪ n')                    then continue;
            if not is_BV_intersected(n, n') then continue;
            if is_leaf(n) and is_leaf(n') then
                triangle-triangle intersection tests for all non-neighboring triangle pairs
                from n and n';
            else
                t1 ←higher non-leaf node in n and n' ;
                t2 ←lower or leaf node in n and n' ;
                foreach child c of t1 do stack.put(c, t2);
            end
        end
    end
end
```

intersections for non-adjacent but geometrically close triangle pairs, which are contained in either the same BV nodes or overlapping BV nodes.

The ESCC algorithm integrates certificate validation into the BVH traversal (see Algorithm 5) to achieve more efficient self-collision culling. It works for certificates defined by any surface deformation energy model, $E(\boldsymbol{u})$. Given an energy model, we precompute certificate values $\mathcal{E}$ for surface patches of individual BV nodes as well as the combined patches of some pairs of connected

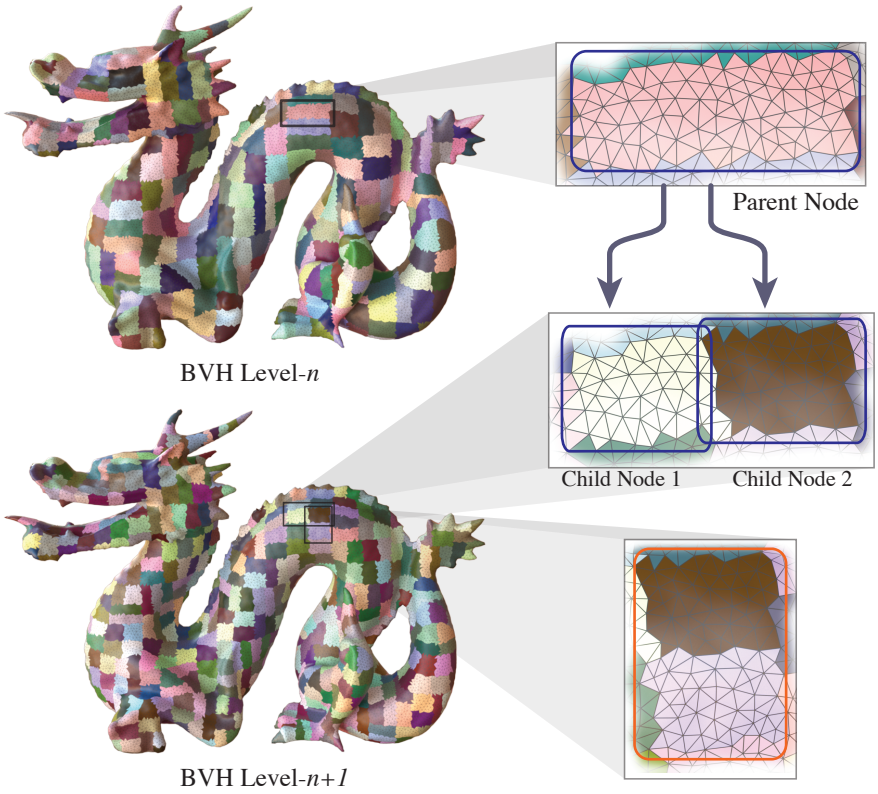


Figure 7.2: **Intra-node and inter-node certificates:** (Top) We compute intra-node certificates for the submesh associated with a BV node, here shown as a level-n node in a binary AABB-Tree. (Bottom) We also compute inter-node certificates between sibling nodes on level $n+1$, as well as (and unlike [Barbič and James 2010]) all same-level nodes with adjacent submeshes.

BV nodes (see Figure 7.2). We call these *intra-node* and *inter-node* certificates, respectively. Certificate computation is described in §7.5. At runtime, to detect self-collisions of a node n, we first compute its deformation energy E_n. If E_n is less than the intra-node certificate $\mathcal{E}_n$, there is no need to traverse its subtree because we are guaranteed that it is self-collision free. Similarly, to detect collisions between node i and j, if the inter-node certificate $\mathcal{E}_{ij}$ has been precomputed, we evaluate the current deformation energy E_{ij} of the joint surface patch of nodes i and j, and compare it to $\mathcal{E}_{ij}$. If $E_{ij} < \mathcal{E}_{ij}$, the traversal of the two subtrees from i and j can be culled, otherwise subtree traversal is needed. In practice, we ob-

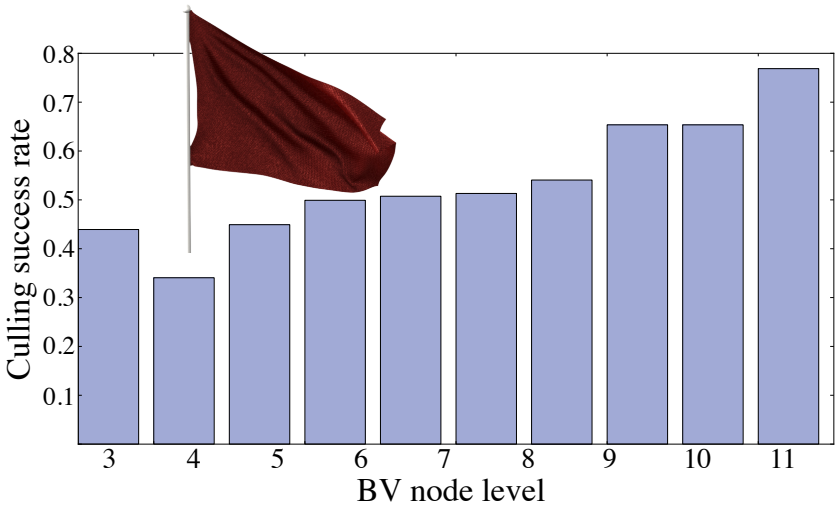


Figure 7.3: **Culling performance increases at finer scales:** Certificates tend to become stronger for smaller nodes as more deformation energy (per triangle) is required to deform smaller submeshes to self collide. (Data for `flag` example.)

serve that certificates have increase culling performance as one traverses deeper on the BVH, so that even for a mesh undergoing large deformation, many BV nodes at fine geometric scales can still be culled (see Figure 7.3). Also, the use of inter-node *and* intra-node certificates ensures that for SCD on a nearly undeformed model, ESCC can cull essentially all triangle-triangle overlap tests.

Selective use of certificates on BV nodes: Computing all possible certificates on a BVH requires quite expensive precomputation effort, and storing them for runtime validation consumes a large amount of memory since there are $O(N^2)$ possible inter-node certificates for N BV nodes. For inter-node self-collision detection, if the sub-meshes of two nodes are not directly connected, we can rely on the traditional BV-intersection tests to cull collision-free nodes because these tests can often offer efficient culling between two distinct tree nodes. However, if two nodes are geometrically connected on the mesh, their BVs always intersect, and we can use inter-node certificates for collision culling. In practice, we only compute inter-node certificates for connected nodes if they are on the

Algorithm 6: **Precompute certificates for BV nodes**

```
procedure: precompute_intranode_certificate (n)
begin
    if number_of_triangles (n) < T_c and is_connected(n) then
        return intranode_certificate (n)                    // see §7.5
    else return null
end
_____________________________________________________

procedure: precompute_internode_certificate (n, n')
begin
    if number_of_triangles(n)+
       number_of_triangles(n') < T_c and
       is_connected(n) and is_connected(n') and
       is_connected(n, n') and
       BVH_level(n) = BVH_level(n') then
        return internode_certificate (n, n')                // see §7.5
    else return null
end
```

same BVH level and their joint sub-mesh are connected and have less than T_c triangles (see Algorithm 6). Consequently, both the computation and storage of certificates require $O(N)$ complexity. For the nodes without certificates, the algorithm simply returns to the traditional BVH traversal scheme.

The order of BV node traversal: While our certificate-accelerated self-collision detection simply adds certificate comparison before subtree traversal and requires no change on the other parts, further performance improvement can be achieved by adjusting the order of node traversal. Notice that the satisfaction of inter-node certificate $\mathcal{E}_{ij}$ guarantees the absence of self-collisions on the *entire* joint sub-mesh of nodes i and j. In other words, the certificates $\mathcal{E}_i$ and $\mathcal{E}_j$ are both guaranteed to be satisfied as well. Therefore, when pushing node pairs into stack to traverse, we push the cross-node test on first (see line L1 and L2 of Algorithm 5) and the self-node tests on last (see line L3 of Algorithm 5). If the inter-node certificate $\mathcal{E}_{ij}$ is satisfied, the subsequent self-node traversals on both node i and j can be immediately skipped. In our implementation, we notice about 2% to 6% performance improvement over the BVH traversal without this

optimization.

7.3 Geometrically Based Deformation Energy

Laplacian Energy: Our energy-based certificate framework can be used with arbitrary surface deformation energy models. However, for practical reasons, we propose a parameter-free geometrically based energy model which is fast for both certificate precomputation and runtime energy evaluation. We use a Laplacian-based mesh deformation energy based on the simple quadratic form,

$$E = \|\mathsf{L}\boldsymbol{u}\|_2^2 = \boldsymbol{u}^T\mathsf{L}^T\mathsf{L}\boldsymbol{u} = \boldsymbol{u}^T\mathsf{K}\boldsymbol{u} \tag{7.1}$$

where L is the discrete *Laplace-Beltrami* operator on a surface mesh patch [MDSB02], $\boldsymbol{u}$ are its vertex displacements, and $\mathsf{K} = \mathsf{L}^T\mathsf{L}$ is the effective stiffness matrix of this potential energy functional. This E will essentially minimize the squared mean curvature of the displacement field (See Figure 7.4).

An important property of K is its particular 3×3 block matrix structure: each (i, j) block is a scalar k_{ij} times a 3×3 identity matrix, such that $E = \boldsymbol{u}^T\mathsf{K}\boldsymbol{u} = \sum_{ij} k_{ij}\boldsymbol{u}_i^T\boldsymbol{u}_j$. We will exploit this structure to devise a faster certificate preprocess, and to accelerate runtime energy computation, by exploiting various properties, e.g., L commutes with affine transforms.

Affine Pull-Back: The Laplace-Beltrami operator L is rank-3 deficient, leading to a translation invariant energy measure (7.1). Unfortunately, this measure is not rotation invariant, and so mere rigid-body displacements will add fictitious deformation energies that reduce culling. It turns out that if we approximate the vertex displacement field with a smooth spatial deformer $\phi : X \to x$ we can use ϕ^{-1} to "pull back" the deformed geometry to a less deformed state while still

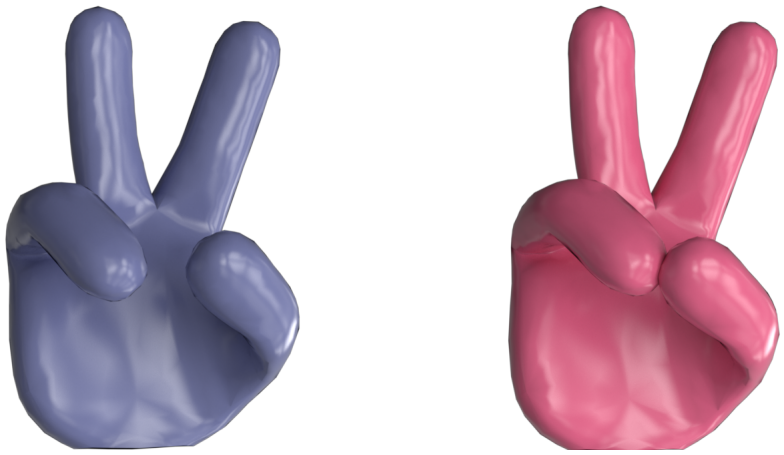


Figure 7.4: **Minimum-energy deformation for self contact:** (Left) A hand model is globally and smoothly deformed (Right) to bring two fingertips into contact as per the minimum displacement Laplacian energy defined in (7.1). The smoothness of minimum-energy deformation enables meaningful certificates on BV nodes.

preserving any intersections that exist. For affine deformers, the pulled-back triangle geometry will also preserve triangle planarity, thereby leading to efficient algorithms. Motivated by these observations, we estimate displacements in a local affine frame via

$$\boldsymbol{x} = \mathsf{F}(\boldsymbol{X} + \boldsymbol{u}) + \boldsymbol{t} \quad \Leftrightarrow \quad \boldsymbol{u} = \mathsf{F}^{-1}(\boldsymbol{x} - \boldsymbol{t}) - \boldsymbol{X}, \tag{7.2}$$

where $\boldsymbol{x}$ are current vertex positions and $\boldsymbol{X}$ are their rest positions, F and $\boldsymbol{t}$ denote linear and translation components, and $\boldsymbol{u}$ is the displacement field deforming the sub-mesh. Note that while affine deformation cannot induce self-collisions, how we estimate F will affect $E(\boldsymbol{u})$. Estimating an F which minimizes E is preferable since it will improve culling, but it is not necessary. In practice, we estimate $(\mathsf{F}^{-1}, \boldsymbol{t})$ using an hierarchical least-squares computation (§7.4). Using this affine pull-back we obtain an affine-invariant measure, and can significantly reduce deformation energy (see Figure 7.5).

Sub-mesh Energy: Given a BV node, we detach its sub-mesh from the rest of the entire mesh and compute the L matrix based on the isolated sub-mesh. Similarly, for a pair of connected BV nodes, we combine their sub-meshes as a single triangle patch isolated from the rest of the original mesh, and compute the L matrix. All these L matrices can be precomputed and stored with the BVH together. Later, the related certificates and energy values are evaluated based on their corresponding sub-meshes detached from the original mesh.

7.4 Hierarchical Energy Computation

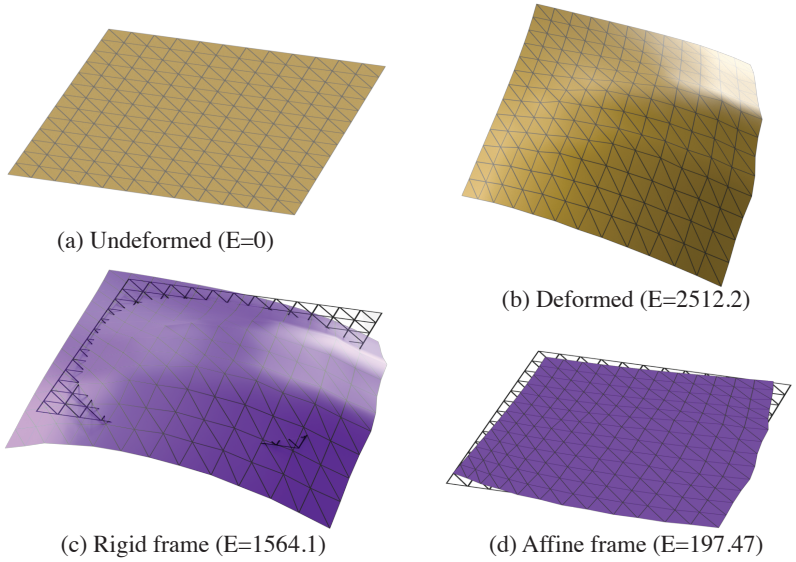
We now present a fast hierarchical method for runtime evaluation of deformation energy on BVH sub-meshes. This method consists of two steps: (i) estimate the affine transform; and (ii) compute the quadratic energy using (7.1).

7.4.1 Hierarchical estimation of sub-mesh transforms

We use least squares to estimate sub-mesh affine transforms similar to "shape matching" [MHTG05], and exploit the hierarchical and overlapping nature of sub-meshes for fast summation of intermediate quantities analogous to [RJ07, SOG08]. Given a set of BV-node vertices deforming from the rest positions $\boldsymbol{X}_i$ to the current positions $\boldsymbol{x}_i$, we estimate the affine matrix A and rigid translation vector $\boldsymbol{t}$ to minimize

$$\sum_i m_i \|\mathsf{A}(\boldsymbol{x}_i - \boldsymbol{t}) - (\boldsymbol{X}_i - \boldsymbol{t}_0)\|^2 \tag{7.3}$$

where m_i is the vertex weights, and $\boldsymbol{t}_0$ is the center of mass of the initial shape. Setting its derivative with respect to A and $\boldsymbol{t}$ to zero yields the solution: $\boldsymbol{t}$ is



(a) Undeformed (E=0)

(b) Deformed (E=2512.2)

(c) Rigid frame (E=1564.1)

(d) Affine frame (E=197.47)

Figure 7.5: **Reducing affine deformation:** Smaller deformation energies can be obtained by using a suitable affine transformation to reduce mesh deformation. Shown are (a) the undeformed mesh, and (b) the deformed mesh. While deformation energies in a tracked rigidbody frame (c) are smaller, they cannot undo stretching. In contrast, pulling back to an affine frame (d) can further reduce deformation energy while still preserving intersection properties. Affine frames are also cheaper to estimate than rigidbody frames.

the center of mass of the deformed shape, i.e., $\boldsymbol{t} = \sum_i m_i \boldsymbol{x}_i / \sum_i m_i$, and A can be computed by

$$\mathsf{A} = \left(\sum_i m_i (\boldsymbol{X}_i - \boldsymbol{t}_0)(\boldsymbol{x}_i - \boldsymbol{t})^T \right) \left(\sum_i m_i (\boldsymbol{x}_i - \boldsymbol{t})(\boldsymbol{x}_i - \boldsymbol{t})^T \right)^{-1} \equiv \mathsf{A}_1 \mathsf{A}_2^{-1}$$

Both A_1 and A_2 can be computed quickly. Computing A_1 of a BV node n requires two summations,

$$\boldsymbol{s}_x = \sum_{\text{vertex } i \in \mathsf{n}} m_i \boldsymbol{x}_i \quad \text{and} \quad \boldsymbol{S}_{\mathsf{A}_1} = \sum_{\text{vertex } i \in \mathsf{n}} m_i \boldsymbol{X}_i \boldsymbol{x}_i^T, \tag{7.4}$$

and then $\mathsf{A}_1 = \boldsymbol{S}_{\mathsf{A}_1} - \boldsymbol{t}_0 \boldsymbol{s}_x^T$. Computing A_2 requires two more summations,

$$m_c = \sum_{\text{vertex } i \in \mathsf{n}} m_i \quad \text{and} \quad \boldsymbol{S}_{\mathsf{A}_2} = \sum_{\text{vertex } i \in \mathsf{n}} m_i \boldsymbol{x}_i \boldsymbol{x}_i^T, \tag{7.5}$$

and $\mathsf{A}_2 = \boldsymbol{S}_{\mathsf{A}_2} - \boldsymbol{s}_x \boldsymbol{s}_x^T / m_c$. These four summations are computed hierarchically on the BVH. For example, to compute $\boldsymbol{S}_{\mathsf{A}_1}$ of node n, we sum up the $\boldsymbol{S}_{\mathsf{A}_1}$ matrices of all its children, and remove the repeated counting on the children's shared boundary vertices. Proceeding recursively, the summations on node n and all its descendants can be computed by going through the subtree of n from bottom up. Shape matching of joint sub-meshes of node i and j can be computed similarly by computing the summations on i and j individually and adding them together without repeating boundary vertices. Furthermore, on multi-core processors, the summations on different subtrees can be computed in parallel, and hence are even faster. In practice, we implement the recursive parallel computation using Intel's *Thread Building Block* [Rei07]. Note that by using affine instead of rigid transforms, we avoid the need for polar decompositions to estimate rotations, achieving both cheaper computation and better culling efficiency (see Figure 7.5).

7.4.2 Hierarchical evaluation of deformation energy

Given a sub-mesh's $\boldsymbol{x}$ and A at runtime, its energy is given by

$$\begin{aligned} E &= \|\mathsf{L}\boldsymbol{u}\|_2^2 = \|\mathsf{L}(\mathsf{A}(\boldsymbol{x} - \boldsymbol{t}) - (\boldsymbol{X} - \boldsymbol{t}_0))\|_2^2 \\ &= \|\mathsf{LA}\boldsymbol{x} - \mathsf{LA}\boldsymbol{t} - \mathsf{L}\boldsymbol{X} + \mathsf{L}\boldsymbol{t}_0\|_2^2 \\ &= \|\mathsf{LA}\boldsymbol{x} - \mathsf{L}\boldsymbol{X}\|_2^2 = \|\mathsf{AL}\boldsymbol{x} - \mathsf{L}\boldsymbol{X}\|_2^2 \end{aligned} \tag{7.6}$$

where $\mathsf{L}\boldsymbol{X}$ is the mean curvature normal vector of the vertices on the undeformed mesh (which is precomputed), $\mathsf{L}\boldsymbol{x}$ evaluates the mean curvature normals of the vertices at the current configuration, and $\mathsf{AL}\boldsymbol{x}$ are node-transformed versions. Here we have used the fact that A and L commute. Note that the

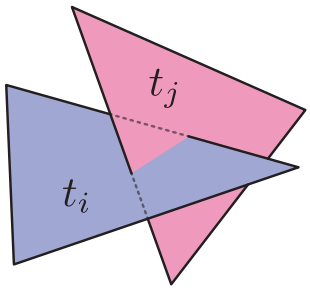
energy is independent of the rigid-body translation, $\boldsymbol{t}$. The mean-curvature normal $(\mathsf{L}\boldsymbol{x})_i$ of vertex v_i is completely determined by the positions of v_i and its direct neighbors, as seen from the sparsity structure of L, which enables us to reuse $\mathsf{L}\boldsymbol{x}$ hierarchically. In particular, consider a vertex v_i contained in a BV node n and its child node n_c. If v_i is not on the boundary of the sub-mesh contained by n_c, then v_i has the same local connectivity on both n and n_c. Therefore, if the mean-curvature normal of v_i on node n has been computed, there is no need to re-compute the normal of v_i on n_c. This observation is particularly helpful for BVH traversal. If the certificate checking fails on node n, we need to traverse to its child nodes to evaluate the deformation energy. For most of the vertices on the children, we can reuse the normals $\mathsf{L}\boldsymbol{x}$ computed at node n and only need to recompute the normals for boundary vertices. Finally, normals are then multiplied by A to construct $\mathsf{AL}\boldsymbol{x}$ for a specific BV node.

7.5 Certificate Precomputation

We now show how to efficiently precompute certificates for a triangle sub-mesh $\mathcal{T}$, such as one associated with a BV node. The precomputation for intra-node and inter-node certificates is identical, since both minimize the energy in (7.1) for a self-colliding sub-mesh $\mathcal{T}$ detached from the original mesh. For an intra-node certificate, $\mathcal{T}$ is the sub-mesh contained by the BV node. For an inter-node certificate, $\mathcal{T}$ is the joint sub-mesh of both nodes.

7.5.1 Certificate for triangle-triangle collision

Here we show how to compute the minimum energy required to intersect two nonadjacent triangles, t^i and t^j. Let $\boldsymbol{X}_k^i \in \mathbb{R}^3, k = 1,2,3$ denote the rest positions of the three vertices of t^i, and, similarly, use $\boldsymbol{X}_k^j, k = 1,2,3$ for t^j. Let $\boldsymbol{u}_k^i, \boldsymbol{u}_k^j \in \mathbb{R}^3, k = 1,2,3$ indicate the vertex displacements of t^i and t^j, respectively. If a point on t^i is in contact with a point on t^j, denote the barycentric coordinates of the two points by α_k and $\beta_k, k = 1,2,3$, respectively. Then the minimum energy E_{ij} required to intersect t^i with t^j is given by the following non-convex quadratically constrained quadratic program (QCQP),



$$
\begin{aligned}
\text{minimize} \quad & \boldsymbol{u}^T \mathsf{K} \boldsymbol{u} \\
\text{subject to} \quad & \alpha_1 + \alpha_2 + \alpha_3 = 1,\ \alpha_i \geq 0,\ i = 1,2,3 \\
& \beta_1 + \beta_2 + \beta_3 = 1,\ \beta_i \geq 0,\ i = 1,2,3 \\
& \alpha_1(\boldsymbol{u}_1^i + \boldsymbol{X}_1^i) + \alpha_2(\boldsymbol{u}_2^i + \boldsymbol{X}_2^i) + \alpha_3(\boldsymbol{u}_3^i + \boldsymbol{X}_3^i) = \\
& \beta_1(\boldsymbol{u}_1^j + \boldsymbol{X}_1^j) + \beta_2(\boldsymbol{u}_2^j + \boldsymbol{X}_2^j) + \alpha_3(\boldsymbol{u}_3^j + \boldsymbol{X}_3^j),
\end{aligned}
\tag{7.7}
$$

where $\mathsf{K} = \mathsf{L}^T\mathsf{L}$ is a positive semidefinite matrix. While non-convex QCQP is considered to be NP-hard in general, we notice that this problem can be solved exactly.

Reduction to Rayleigh quotient form: If we assume the two points in collision (α and β) are known a priori, then the QCQP (7.7) simplifies to a linearly constrained quadratic program (LCQP),

$$
\begin{aligned}
\text{minimize} \quad & \boldsymbol{u}^T \mathsf{K} \boldsymbol{u} \\
\text{subject to} \quad & \alpha_1(\boldsymbol{u}_1^i + \boldsymbol{X}_1^i) + \alpha_2(\boldsymbol{u}_2^i + \boldsymbol{X}_2^i) + \alpha_3(\boldsymbol{u}_3^i + \boldsymbol{X}_3^i) = \\
& \beta_1(\boldsymbol{u}_1^j + \boldsymbol{X}_1^j) + \beta_2(\boldsymbol{u}_2^j + \boldsymbol{X}_2^j) + \alpha_3(\boldsymbol{u}_3^j + \boldsymbol{X}_3^j).
\end{aligned}
\tag{7.8}
$$

This problem can be easily solved analytically using Lagrange multipliers. We defer the details of the LCQP solver to Appendix C.1, and only present the result

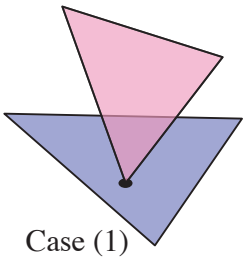
here. In particular, the achieved minimum energy value $\boldsymbol{u}^T \mathsf{K} \boldsymbol{u}$ can be expressed as a generalized Rayleigh quotient,

$$\hat{E}_{ij} = \frac{\boldsymbol{a}^T \mathsf{M} \boldsymbol{a}}{\boldsymbol{a}^T \mathcal{G} \boldsymbol{a}}, \tag{7.9}$$

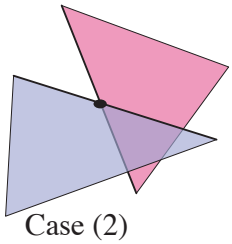
where $\boldsymbol{a}$ is a 6×1 vector, $\boldsymbol{a} = [\,\alpha_1\ \alpha_2\ \alpha_3\ -\beta_1\ -\beta_2\ -\beta_3\,]^T$; $\mathsf{M} = \mathsf{B}^T\mathsf{B} \in \mathbb{R}^{6\times 6}$ is a rank-3 symmetric positive semidefinite matrix, where $\mathsf{B} = [\,\boldsymbol{x}_1^i\ \boldsymbol{x}_2^i\ \boldsymbol{x}_3^i\ \boldsymbol{x}_1^j\ \boldsymbol{x}_2^j\ \boldsymbol{x}_3^j\,]$; and $\mathcal{G} = \tilde{\mathsf{K}}_s^\dagger \in \mathbb{R}^{6\times 6}$ is a sub-matrix of the pseudo-inverse of a related matrix $\tilde{\mathsf{K}}$, where the elements of $\tilde{\mathsf{K}}_s^\dagger$ correspond to the 6 vertices of t^i and t^j. Following the convention of elastostatic mechanics, we call this K pseudo-inverse the *Green's function* matrix. Appendix C.2 details a fast computation of the Green's function matrix that exploits the block structure and null space of K. Now (7.7) is equivalent to the following problem:

$$\begin{aligned} &\text{minimize} && \frac{\boldsymbol{a}^T \mathsf{M} \boldsymbol{a}}{\boldsymbol{a}^T \mathcal{G} \boldsymbol{a}} \\ &\text{subject to} && \alpha_1 + \alpha_2 + \alpha_3 = 1,\ \alpha_i \geq 0,\ i = 1,2,3 \\ & && \beta_1 + \beta_2 + \beta_3 = 1,\ \beta_i \geq 0,\ i = 1,2,3. \end{aligned} \tag{7.10}$$

Its optimum is achieved only at one of the two cases: (i) a vertex of one triangle touches the other triangle; or (ii) an edge of one triangle touches an edge of the other triangle. Intuitively, if a collision does not satisfy either cases, one can relax the vertex displacements to shrink the deformation energy while still remaining the two triangles in collision until either case (i) or (ii) is achieved. In the remainder of this subsection, we consider both cases.



Case (1)



Case (2)

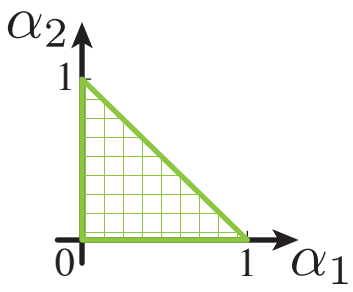
Vertex-triangle collision: There are 6 vertex-triangle collision cases. Without loss of generality, we consider the first vertex $\boldsymbol{X}_1^j$ of t^j in collision with t^i. Then, for problem (7.10), $a_3 = 1 - a_1 - a_2$, $\beta_1 = 1$, $\beta_2 = \beta_3 = 0$, and we

can simplify the problem using reduced parameters. Let $\boldsymbol{a} = \mathsf{D}\tilde{\boldsymbol{a}} + \boldsymbol{b}$, where $\tilde{\boldsymbol{a}} = [\,\alpha_1 \;\; \alpha_2\,]^T$,

$$\mathsf{D} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}^T \text{ and } \boldsymbol{b} = \begin{bmatrix} 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix}^T.$$

Substitution into (7.10), yields the reduced problem,

$$\begin{aligned} &\text{minimize} \quad \frac{\tilde{\boldsymbol{a}}^T \mathsf{D}^T \mathsf{M} \mathsf{D} \tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T \mathsf{M} \mathsf{D} \tilde{\boldsymbol{a}} + \boldsymbol{b}^T \mathsf{M} \boldsymbol{b}}{\tilde{\boldsymbol{a}}^T \mathsf{D}^T \mathcal{G} \mathsf{D} \tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T \mathcal{G} \mathsf{D} \tilde{\boldsymbol{a}} + \boldsymbol{b}^T \mathcal{G} \boldsymbol{b}} \\ &\text{subject to} \quad \alpha_1 \geq 0,\ \alpha_2 \geq 0,\ \alpha_1 + \alpha_2 \leq 1. \end{aligned} \tag{7.11}$$



In the supplemental Appendix C.3, we show that the optimum of this problem is achieved in the interior of the domain only at certain situations which can be verified by solving 3 1-D quadratic equations. If the optimum happens in the interior of the domain, computing the optimum value requires solving two cubic equations, one of which has only a single real root. For most cases in practice, the optimum is on the boundary defined by three segments. A boundary point corresponds to a case where a vertex touches an edge. The optimum on each boundary segment can be computed by solving a 1-D quadratic equation. Please see the detailed derivation of these equations in Appendix C.3. For all 6 vertex-triangle collisions, we need to solve 36 quadratic equations and in very rare cases solve cubic equations.

Edge-edge collision: There are 9 edge-edge collision cases. Again, without loss of generality, we consider the edge connecting $\boldsymbol{X}_1^i$ and $\boldsymbol{X}_2^i$ on t^i in collision with the edge connecting $\boldsymbol{X}_1^j$ and $\boldsymbol{X}_2^j$ on t^j. This corresponds to $a_2 = 1 - a_1$, $a_3 = 0, \beta_2 = 1 - \beta_1$ and $\beta_3 = 0$ in (7.10), and we can construct a reduced problem

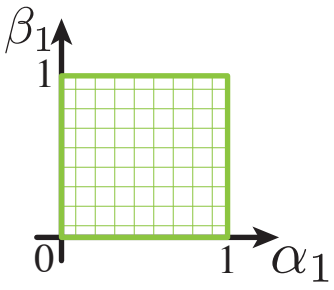
using $\boldsymbol{a} = \mathsf{D}\tilde{\boldsymbol{a}} + \boldsymbol{b}$, where $\tilde{\boldsymbol{a}} = [\,\alpha_1\ \beta_1\,]^T$,

$$\mathsf{D} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}^T \text{ and } \boldsymbol{b} = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 0 \end{bmatrix}^T.$$

Problem (7.10) now becomes

$$\begin{aligned} &\text{minimize} && \frac{\tilde{\boldsymbol{a}}^T \mathsf{D}^T \mathsf{M} \mathsf{D} \tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T \mathsf{M} \mathsf{D} \tilde{\boldsymbol{a}} + \boldsymbol{b}^T \mathsf{M} \boldsymbol{b}}{\tilde{\boldsymbol{a}}^T \mathsf{D}^T \mathcal{G} \mathsf{D} \tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T \mathcal{G} \mathsf{D} \tilde{\boldsymbol{a}} + \boldsymbol{b}^T \mathcal{G} \boldsymbol{b}} \\ &\text{subject to} && 0 \le \alpha_1 \le 1,\ 0 \le \beta_1 \le 1. \end{aligned} \tag{7.12}$$

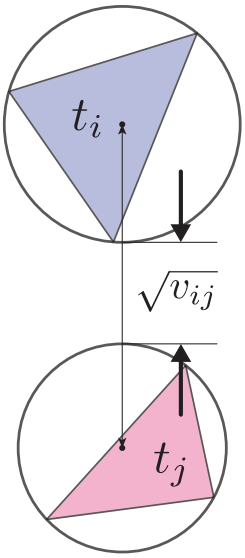
Similar to problem (7.11), the optimum of this problem happens in the interior of the domain very rarely. Checking such situations involves 4 quadratic equation solves. For most cases, the optimum is on the boundary where a vertex touches an edge. Note that all these boundary cases has been considered when we solve the boundary optimum values for vertex-triangle collisions (7.11), therefore we can safely ignore all of them. For all 9 edge-edge collisions, we need to solve 36 1-D quadratic equations and very infrequently solve cubic equations. See the supplemental Appendix C.3 for derivations.



7.5.2 Certificate for an entire sub-mesh

The certificate $\mathcal{E}$ for an entire triangle sub-mesh $\mathcal{T}$ simply takes the minimum of $\mathcal{E}_{ij}$ over all possible pairs of triangles, i.e., $\mathcal{E} = \min_{t^i, t^j \in \mathcal{T}} \mathcal{E}_{ij}$. However, processing through all triangle pairs on a large sub-mesh can be impractical. Instead, we use a two-pass algorithm. At the first pass, we quickly compute a *cheap* lower bound $\tilde{E}_{ij}$ of $\mathcal{E}_{ij}$ for all triangle pairs, i.e., $\tilde{E}_{ij} \le \mathcal{E}_{ij}$. At the second pass,

we process the triangle pairs in order of ascending $\tilde{E}_{ij}$. The computation of $\mathcal{E}_{ij}$ can be immediately skipped if the so-far encountered smallest $\mathcal{E}$ value is less than $\tilde{E}_{ij}$ of the current triangle pair. Similar ideas have been used in [BJ10]. We first compute a lower bound v_{ij} and an upper bound w_{ij} for the numerator and denominator of the generalized Rayleigh quotient (7.9). Then we have $\tilde{E}_{ij} = v_{ij}/w_{ij} \leq \mathcal{E}_{ij}$. Note that the numerator of (7.9) measures the squared Euclidean distance between the two touching points in undeformed mesh configuration. We compute the circumcenter $\boldsymbol{p}_i$ and the circumradius R_i of each triangle. The Euclidean distance of any two points on triangle pair i, j must be no less than $\|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2 - R_i - R_j$. An upper bound of the denominator of (7.9) can be derived by noticing that $\mathcal{G}$ is symmetric positive semidefinite and $\boldsymbol{a}$ satisfies the constraints in (7.10):



$$\boldsymbol{a}^T \mathcal{G} \boldsymbol{a} \leq \max_{k=1..3} \mathcal{G}_{kk} + \max_{k=4..6} \mathcal{G}_{kk} - \sum_{k \neq t, \mathcal{G}_{kt} < 0} \mathcal{G}_{kt}.$$

and so a lower bound for $\mathcal{E}_{ij}$ is

$$\tilde{E}_{ij} = \frac{(\|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2 - R_i - R_j)^2}{\max_{k=1..3} \mathcal{G}_{kk} + \max_{k=4..6} \mathcal{G}_{kk} - \sum_{k \neq t, \mathcal{G}_{kt} < 0} \mathcal{G}_{kt}}. \tag{7.13}$$

Discussion: Note that if $\mathcal{T}$ has disconnected components, one of the mesh components can collide with the other component under rigid translation. Consequently, certificates on such sub-meshes are always zero and provide no culling capability at runtime. Therefore, we ignore disconnected BV nodes or node pairs in Algorithm 6 and leave them to be handled by traditional BVH intersection tests. A pathological case comes with a mesh whose BV nodes are almost all disconnected, e.g., the squishy ball in Figure 7.1, preventing from meaningful certificates at most of its BV nodes. We discuss BVH optimizations to improve connectivity in §7.6.

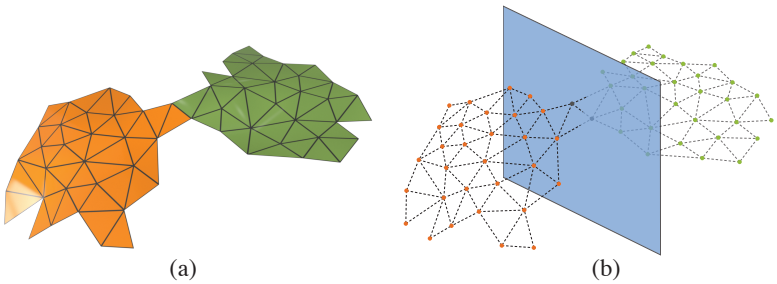


Figure 7.6: **Separating weakly coupled sub-meshes with a separation plane**

Certificates for weakly connected BV nodes: To achieve further performance, we take special care of pairs of weakly connected BV nodes (see Figure 7.6(a)). Due to the weak connectivity, these node pairs can yield pretty low certificate values which can easily fail with small deformations at runtime. However, we note that a weakly connected pair of nodes can often be easily separated by a separation plane $ax+by+cz+1 = 0$ (see Figure 7.6(b)). Formally, let us define the vertex sets of the weakly connected nodes n_1 and n_2 as V_1 and V_2 respectively, and a plane equation $f(\boldsymbol{x}) = ax + by + cz + 1$. Then following the separating plane theorem, the absence of intersections between n_1 and n_2 is guaranteed if the following condition is satisfied (for a suitably oriented plane),

$$\begin{cases} f(\boldsymbol{v}) < 0, & \forall \boldsymbol{v} \in V_1 - V_1 \cap V_2 \\ f(\boldsymbol{v}) > 0, & \forall \boldsymbol{v} \in V_2 - V_1 \cap V_2 \end{cases} \tag{7.14}$$

Finding the optimal separation plane $f(\boldsymbol{x}) = 0$ is a classical problem of finding the *maximum-margin hyperplane* in linear Support Vector Machines [Bur98]. Essentially we are classifying two sets of nodes, $V_1 - V_1 \cap V_2$ and $V_2 - V_1 \cap V_2$, with a separation plane in 3D space. For each pair of weakly connected nodes, we compute a maximum-margin separation plane based on their undeformed positions. At runtime, for each vertex in $V_1 - V_1 \cap V_2$ and $V_2 - V_1 \cap V_2$, we first transform it to the node's initial frame using the already-computed affine shape matching (recall section 7.4.1), and then check whether (7.14) is satisfied if

the corresponding certificates fail. In practice, we only apply separation-plane checks at low-level BV nodes (leaf nodes and their parents), and observe up to 11% speedup over the implementation without separation planes.

7.6 Extensions

Optimizing mesh connectivity of BV nodes: While ESCC certificates can be applied to any BVH, sub-meshes with disconnected or weakly connected triangles will degrade culling performance. We therefore propose a method to construct an AABB-based BVH whose nodes have well-connected sub-meshes. To split a BV node into child nodes, n_1 and n_2, we randomly sample two triangles, and perform region growing using a cost based on the Euclidean distance between the centroids of two triangles [CSAD04]. We compute a cost $C(\mathsf{n}_1, \mathsf{n}_2)$ for the split, and repeat the samples T times and use the one with least cost to create child nodes; we use $T = 60000$ in our examples. Our cost function tries to minimize both the difference in submesh areas, A_1 and A_2, and the bounding-box surface areas, B_1 and B_2:

$$C(\mathsf{n}_1, \mathsf{n}_2) = |A_1 - A_2| + 0.1\,(B_1 + B_2).$$

This metric is similar to the "surface area heuristics" used in previous BV optimization methods [GS87].

Continuous Self-Collision Culling: Our method can also be used in continuous self-collision detection, such as for cloth simulation [BFA02]. For example, consider a piecewise-linear deformation of a sub-mesh from displacement $\boldsymbol{u}^{(0)}$ at $t = 0$ to $\boldsymbol{u}^{(1)}$ at $t = 1$, such that $\boldsymbol{u}^{(t)} = (1 - t)\boldsymbol{u}^{(0)} + t\boldsymbol{u}^{(1)}, t \in [0, 1]$. Now the energy function $E^{(t)} = E(u(t)) = \|\mathsf{L}\boldsymbol{u}^{(t)}\|_2^2$ might not be convex in $\boldsymbol{u}$, therefore we cannot cull

SCD tests by simply validating certificates at $t = 0$ and $t = 1$. The problem comes from a continuously changing affine pull-back, however, we can circumvent it by fixing the affine pull-back. For example, we estimate the affine pull-back based on $\boldsymbol{u}^{(0.5)}$ computed by interpolating $\boldsymbol{u}^{(0)}$ and $\boldsymbol{u}^{(1)}$. We then use the resulting affine transformation to pull back the mesh at both $t = 0$ and $t = 1$, and evaluate corresponding energy values $\bar{E}^{(0)}$ and $\bar{E}^{(1)}$. With this fixed affine pull-back, the deformation energy change from $\bar{E}^{(0)}$ to $\bar{E}^{(1)}$ is convex in t. It follows that $\bar{E}^{(t)} \leq \max(\bar{E}^{(0)}, \bar{E}^{(1)})$, $t \in [0, 1]$. Thus if the certificate indicates a collision-free state at both endpoints, then no collision can occur along the piecewise linear trajectory. If at least one endpoint's certificate fails, then we must recurse and perform additional checks. In this way, the ESCC certificates can be used to augment traditional BVH-based continuous SCD queries as in [BJ10].

7.7 Results

Numerous results and statistics are shown in Table 7.1 for a range of animated mesh examples, including ones made by us and from prior works [BSM+03, JT05, BJ10, SGPO10, CTM08]. Please see the supplemental video in our paper [ZJ12] for animations and detailed performance information. All reported timings were measured on a dual Intel Xeon X5570 (2.93 GHz) processor machine with 8 physical cores. Due diligence has been taken to exploit multi-core parallelization for accelerating both the AABB-Tree updates and queries, and ESCC certificate precomputation and runtime energy evaluation. We also profiled the performance of single-core computation and observed that our parallel ESCC implementation using Intel's TBB achieved 2.1x-3.4x speedups over the single-core computation, depending on the mesh size. Culling effectiveness ver-

Example	Complexity		Precomputation							Runtime (in ms)			Culling difficulty measure
	Tri	Vtx	Intra-node certificates	Inter-node certificates	Cover ratio, r	G cost (s)	Cert cost (s)	sep. plane num	sep. plane cost (s)	AABB	ESCC	Speedup	
squishy ball	1064216	532110	260059	467162	0.92	5469.7	548.7	297002	1064	3100	270	11.5×	0.72
monkeys	108928	54468	30246	61371	0.96	966.8	201.2	40867	144.3	379.4	49.7	7.6×	0.41
flowing cloth	51200	25921	10164	10605	0.98	467.0	183.6	11109	77.6	124.6	6.5	19.1×	0.10
16 bunnies‡	14964	7484	2324	4652	0.99	112.3	37.6	2332	14.7	448.2	29.3	15.3×	0.21
dragon‡	77203	38602	2434	4870	0.78	140.9	36.7	6323	48.3	320.8	12.3	26.1×	0.22
spring wire‡	16000	8000	10163	3820	0.99	129.0	17.4	2002	10.3	33.2	1.2	25.6×	0.10
snake⋆	18354	9179	5275	9433	0.99	230.2	76.0	6667	45.7	44.0	4.9	9.0×	0.71
dance⋆	14118	7061	3896	7174	0.98	223.5	36.6	3207	34.5	33.1	5.8	5.7×	0.55
flag⋆	13436	6906	2524	3300	0.92	136.1	27.7	3324	24.2	41.5	5.0	8.3×	0.29
bunny (low res.)†	13632	6818	1712	3278	0.99	123.7	25.4	1424	10.4	41.7	7.6	5.5×	0.42
bunny (high res.)†	54528	27266	6534	13443	0.99	453.9	256.6	6238	44.5	169.7	41.1	4.2×	0.36
bunny (noisy)†	54528	27271	4068	8526	0.62	403.4	158.5	2865	18.9	185.7	55.9	3.3×	0.78
bunny (spiky)†	54528	27271	1993	4237	0.54	238.4	147.2	1126	8.0	194.9	95.5	2.0×	1.30
cloth sphere†	20000	10201	4080	6251	0.99	216.3	59.6	7867	50.1	54.4	12.0	4.5×	0.40
cat†	41184	20631	4919	9142	0.99	344.6	73.4	3900	28.1	130.1	31.9	4.1×	0.66
collapsing horse†*	25344	12674	2999	5529	0.94	138.7	45.0	2608	17.7	108.7	78.2	1.4×	4.12
cloth-sphere 2◦	91470	46216	9198	16217	0.99	516.8	176.2	10685	62.8	52.4	15.5	3.4×	0.98

Table 7.1: **Example statistics** for triangle/vertex counts, the number of precomputed certificates, intra-node coverage, precomputation timings for Green's function G matrices and certificates, separating planes for weakly connected inter-nodes. Runtime performance timings (per frame) for our optimized AABB SCD test, and the same code with ESCC culling enabled, demonstrate significant ESCC speedups on most examples. The lowest speedups are for examples with significant interpenetration or dynamic close-proximity scenarios as indicated by the *culling difficulty measure* defined as the ratio of the number of inter-node overlap tests between disconnected leaf nodes to the total number of leaf nodes. Examples from prior work are indicated using † for [Schvartzman et al. 2010], ‡ for [Barbič and James 2010], ⋆ for [Briceño et al. 2003; James and Twigg 2005], and ◦ for [Curtis et al. 2008].

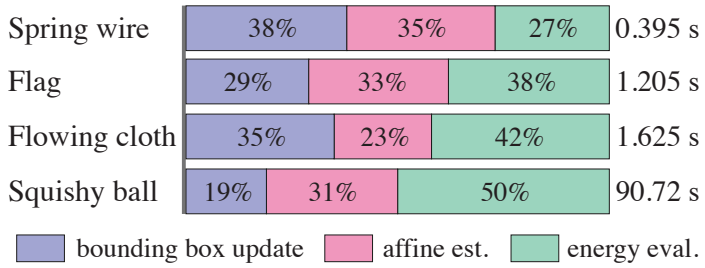


Figure 7.7: **Timing breakdown of post-deformation update**

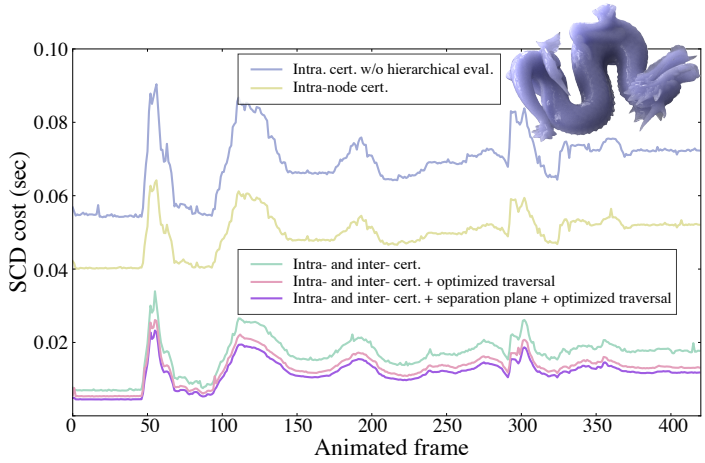
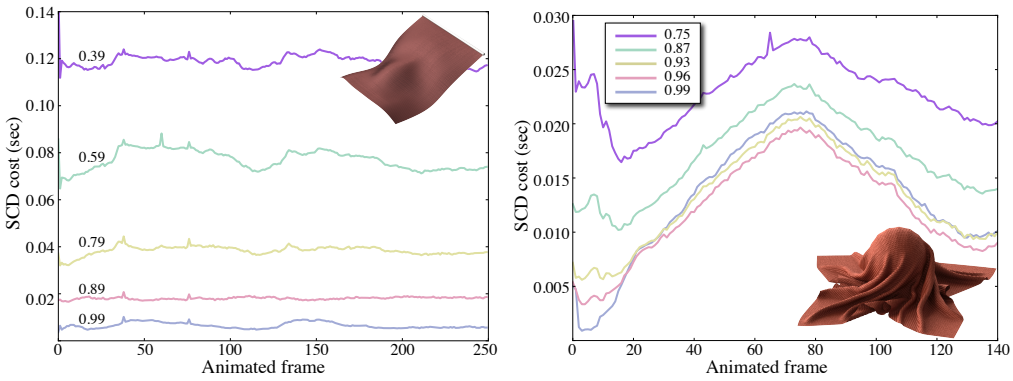


Figure 7.8: **Benefits of ESCC optimizations for runtime SCD**

sus BVH levels is illustrated in Figure 7.3. Detailed runtime timing breakdowns for post-deformation tree updates are shown in Figure 7.7. The impact of various optimizations on runtime SCD are shown in Figure 7.8.

Performance versus certificate coverage: As presented in §7.2 and Algorithm 6, we use certificate-based culling only when the number of triangles on a node is less than a threshold T_c; in our implementation, we use $T_c = 2500$ for all examples, except the giant squishy ball where we only used $T_c = 128$. One natural question is how T_c will affect the culling performance. To indicate the fraction of nodes where certificates are applied, we use a **cover ratio**, r, defined as the ratio of the number of computed intra-node certificates to the



(a) Flowing cloth (b) Cloth sphere

Figure 7.9: **SCD performance vs cover ratio,** r

total number of BV nodes. Table 7.1 shows the r values used for all the examples. Furthermore, Figure 7.9 shows the SCD cost with different cover ratio. For smooth small deformations (e.g., Figure 7.9(a)), the SCD cost decreases as we use more certificates. However, for fairly large deformations (e.g., Figure 7.9(b)), the certificates on high-level BV nodes fail more frequently. Consequently, the payoff of using certificates on high-level BV nodes is smaller than the overhead of evaluating deformation energy, since higher nodes usually have many triangles. Thus too many high-level certificates can hurt the performance slightly (see Figure 7.9(b)).

Conservativeness: We designed a metric to reflect the conservativeness of the certificates. Validating certificates at each node can be a waste of computation if it can not cull SCD tests for collision-free nodes. Therefore, we count the number, A, of succeed certificate validations which cull the SCD tests as expected, as well as the number, B, of false positive leaf-node intersection tests for which there is no self collision, but certificate validations cannot cull the tests. We use the ratio B/A as a conservativeness metric. Figure 7.10 shows how it changes for different examples. This metric varies a lot for different test cases. In the

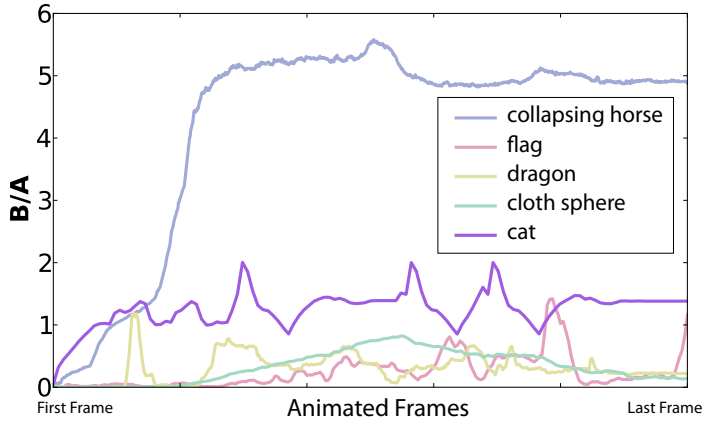


Figure 7.10: **Conservativeness**

examples with lots of self collisions (e.g., the collapsing horse), many triangles that are geodesically far on the mesh are close to each other under the mesh deformation. On the BVH, those triangles are in a single BV node or connected BV nodes only when the nodes are at high levels of the BVH. Certificates on those high-level nodes are less conservative, and are easier to be violated with small deformations. This metric is also generally consistent with our culling difficulty measure in Table 7.1 to reflect the effectiveness of self-collision culling.

Memory overhead: At runtime, we load the precomputed certificates, the sparse Laplace-Beltrami matrices and the coefficients of separating planes into memory. In practice, all the certificates, the Laplace-Beltrami matrices, and separating plane coefficients are stored in single precision (32-bit float number). We counted how much memory required to store these data for the use of ESCC in different examples. They are: squishy ball (354.8M), monkeys (83.2M), flowing cloth (24.9M), dragon (37.5M), dance(8.3M), low-resolution bunny (7.9M), high-resolution bunny (34.8M), and cloth sphere(9.5M),

Difficult Cases: As reported in Table 7.1, there are certain cases where ESCC produces little speedup. Those are the examples involving large deformations

and self-collisions in many parts of the mesh (such as the "collapsing horse" in [SGPO10]). For those examples, our conservative certificate-based culling is ineffective, and the method has to travel on the BVH until the leaf level to detect triangle-triangle collisions. In the last column of Table 7.1, we use a metric defined as the ratio of the number of inter-node overlap tests between disconnected leaf nodes to the total number of leaf nodes to indicate the culling difficulty. For the examples where this ratio is large, ESCC method could not offer good speedups.

The techniques proposed in this paper focuses on fast *self-collision* detection. In practice, inter-object collision detection is another expensive part for resolving collisions. For the simulations where many objects close each other are involved (such as the "flamenco dancer" in [CTM08]), inter-object collision tests can dominate collision processing timings, especially as self-collision tests are reduced greatly. In such cases, the overall speedup would be low. However, our method is complementary to other types of intra-object and inter-object collision detection techniques. One can combine ESCC with other collision detection methods to achieve higher performance.

7.8 Related work

Self-collision detection (SCD) methods for deformable models have a long history in computer graphics, and we optimize the common practice of using a bounding volume hierarchy (BVH) for triangle-based SCD [TKH+05]. Our ESCC certificates represent a geometric property of the sub-mesh inside any BV node (or nodes), and can therefore be used with any flavor of bound-

ing volume, including spheres [Hub95], axis-aligned bounding boxes (AABBs) [vdB97], oriented bounding boxes (OBBs) [GLM96], and discrete oriented polytopes (k-DOPs) [KHM+98]. In our implementation, we use AABB-based binary trees [vdB97]. Our ESCC certificates are complementary to many existing SCD acceleration approaches for BVHs, and can provide yet another way to cull triangle-overlap tests. Our ESCC certificates support continuous SCD needed for simulating thin objects [BFA02].

The most closely related work to ours is subspace self-collision culling (SSCC) [BJ10]. It assumes object-frame subspace deformations of the form $\boldsymbol{u} = \mathsf{U}\boldsymbol{q}$, and used certificates that measure minimal deformation essentially using $\|\boldsymbol{q}\|_2 = \|\boldsymbol{u}\|_2$. The effectiveness of the certificates relied upon the ability of the modal basis U to produce smooth deformations which tend to avoid localized self collisions. For the arbitrary deformations we consider, one cannot use $\|\boldsymbol{u}\|_2$ as a certificate measure since its certificates degenerate into measuring individual vertex displacements. In contrast, we use an energy-based measure $E(\boldsymbol{u})$ of arbitrary patch deformations, and exploit the sparse structure of the arbitrary-deformation problem to derive an exact ESCC certificate preprocess and runtime that is mathematically different and highly efficient.

Barbič and James [BJ10] accelerated BVH-based SCD by reducing both (i) the cost of post-deformation BVH updates, $\mathrm{C}_{\mathsf{Update}}$, and (ii) the cost of recursively testing the BVH against itself to identify overlapping triangle pairs, $\mathrm{C}_{\mathsf{Query}}$. For arbitrary deformations, our method is stuck with inherently $\Omega(N)$ $\mathrm{C}_{\mathsf{Update}}$ and $\mathrm{C}_{\mathsf{Query}}$ costs for an N triangle mesh patch. Our method adds additional $O(N)$ overhead to update patch-specific certificate energies $E(\boldsymbol{u})$, and cannot exploit subspace certificates or BV updates [JP04] to avoid looking at the triangles en-

tirely. Nevertheless, the BVH-based SCD bottleneck for arbitrary deformations remains the C_{Query} cost, and our ESCC certificates measure localized patch deformations, as opposed to global deformation amplitude, and can thus cull localized deformations more effectively—ESCC can even be faster than SSCC on subspace deformations (see §7.7).

For arbitrary deformations, many prior works have attempted to reduce the C_{Query} cost bottleneck. For example, methods based on chromatic decompositions [GKJ+05] and representative triangles [CTM08] can effectively reduce the number of redundant low-level triangle overlap tests. Curvature tests [VMT94, SGPO10] and normal bounds [Pro97, GS01, TCYM09, SGO09] can also cull overlap tests in potentially large regions, however, unlike ESCC, these methods can only cull effectively in regions with smooth geometry and smooth deformations. In contrast, our certificates can efficiently cull self-collision tests even in areas of non-smooth geometry (like [BJ10]), and, furthermore, we can cull patches with nonsmooth deformations. Curvature-based culling also requires a potentially expensive "contour test" for correctness, but recently it was shown how this could be performed efficiently using "Star Contours" [SGPO10]. In comparisons provided earlier (§7.7), we observe larger speedups using ESCC on the same examples. Nevertheless, many of these narrow-phase triangle-triangle optimizations are complementary to patch-based ESSC, and could be used simultaneously.

Our use of ESCC certificates for BVH-based SCD is an instance of a kinetic data structure (KDS) [Gui04] for reasoning about self collisions of dynamic meshes. In contrast to other geometric KDS techniques for SCD [GNRZ02, GGN06], we propose energy-based certificates for triangle meshes.

Our energy-based certificates are complementary to GPU-based methods that use brute force to accelerate triangle-level SCD computations, but less so for rasterization-based methods [HTG04]. In addition to improving the speed of triangle overlap tests, GPU-based methods also try to reduce the number of needed overlap tests [GLM05, SGG+06]. Our current implementation exploits multi-core optimizations, but GPU parallelization is a logical extension.

7.9 Conclusion and Future Work

We have introduced energy-based self-collision culling (ESCC), a new technique for accelerating self-collision detection (SCD) for triangle meshes undergoing arbitrary deformations. Using bounding volume hierarchies augmented with ESCC certificates enables significant speedups, as demonstrated on numerous examples. Our certificate preprocess enables rapid computation of certificates by exploiting numerous algorithmic and mathematical insights.

ESCC certificates are complementary to many existing SCD approaches, and future work should investigate combining ESCC culling with other methods for narrow-phase culling, such as [CTM08, SGPO10]. Our precomputed sub-mesh certificates require a fixed BVH topology, and so one cannot adapt the BVH at runtime, which may reduce speedups for highly deformable models such as cloth. Our ESCC method can be further accelerated if some runtime energy computations can be shared with other code, e.g., a physics simulator which already computes, $\mathsf{L}\boldsymbol{x}$. Future work should investigate the possibility of other energy models. Our affine-invariant Laplacian-based energy model enables fast runtime evaluation, material parameter independence, and a fast certificate pre-

process, however, one could use more sophisticated deformation energy models provided that the runtime and precomputation costs were fast enough. For example, a user simulating cloth with a hyperelastic potential energy model might also use that model for ESCC, thereby benefitting from embedded energy computations. Finally, while we believe that BVHs are ideal for exploiting ESCC certificates, it may be possible to use ESCC certificates with other SCD approaches, e.g., spatial partitionings [TKH+05].

CHAPTER 8

CONCLUSION

8.1 Summary and Conclusions

Computer-simulated realities, although capable of producing realistic and convincing visual effects nowadays, are still inherently silent. The lack of realistic sound rendering counterpart will fundamentally limit our endeavor toward generating highly immersive virtual realities.

This thesis is an attempt to change that. The key contribution is a series of algorithms that capture unprecedented motion details in simulations and render realistic sounds from them. We have demonstrated these algorithms for a number of major classes of sound phenomena in computer animation, including the sound models for brittle fracture (chapter 3), modal contacts (chapter 4) and liquids (chapter 5). Our sound synthesis approaches are based on physical principles, capturing object vibrations for sound sources and simulating sound propagations. We exploit inherent features in sound simulations and build algorithms leveraging reasonable physical simplification, mathematical approximation, possible pre-computation, parallelism, and psychoacoustic insights. Our methods thereby are practical and capable of dealing with general and complex computer graphics scenarios.

In addition, we have introduced a fast self-collision detection methods (chapter 7) for deforming triangle meshes. While this method has been demonstrated for accelerating general deformable simulations, we believe simulations for deformable sound synthesis can enjoy it as well, and expect to use this

method in future deformable sound synthesis work.

8.2 Future Work

Physics-based sound rendering is a new direction, and there are a number of promising directions for future research. Perhaps a direct extension of this thesis is to exploit other sound phenomena common in computer graphics applications, such as deformable sounds and multi-physics interaction sounds. Interactions among different types of physical systems can producing noticeable sounds, and are particularly challenging to simulate. Addressing more complex sound phenomena also requires development of related physics-based simulations which by themselves can be a large area for future work. Fast sound synthesis should incorporate the understanding of human perception of sounds. Psychoacoustic study can provide clues about what sound properties or details are important to capture in sound synthesis, and lead to methods which might be numerically less accurate but perceptually plausible and fast. Therefore how to integrate psychoacoustic considerations into an efficient sound simulation algorithm becomes very interesting for future work. Finally, while physics-based sound rendering promises automatic generation of fully synchronized sounds, it suffers from expensive computation in many situations. On the other hand, it is not the only way for realistic sound synthesis. Traditional methods based on signal-processing or granular synthesis as well as data-driven sound synthesis approaches are less flexible, hard to be synchronized, but are relatively cheap to compute. How to combine the advantages of different approaches for fully synchronized multi-sensory virtual realities can be an interesting research topic.

APPENDIX A

APPENDIX FOR CHAPTER 3

A.1 Sparse least-squares $\mathsf{K}\boldsymbol{u} = \boldsymbol{f}$ solver details

Constructing $\boldsymbol{\mathcal{P}}$: The null space of K is spanned by rigid-body modes, which we project out using $\boldsymbol{\mathcal{P}}$. Given tetrahedral node, i, with position (x_i, y_i, z_i), its translation and linearized rotation are columns of T_i,

$$\mathsf{T}_i = \begin{bmatrix} 1 & 0 & 0 & 0 & z_i & -y_i \\ 0 & 1 & 0 & -z_i & 0 & x_i \\ 0 & 0 & 1 & y_i & -x_i & 0 \end{bmatrix} \quad \Rightarrow \quad \mathsf{T} = \begin{bmatrix} T_1 \\ \vdots \\ T_N \end{bmatrix}, \tag{A.1}$$

so that the rank-6 matrix T spans $\mathsf{null}(\mathsf{K})$. We compute the QR factorization, $\mathsf{T} = \mathsf{QR}$, and cache the $3N$-by-6 orthogonal matrix, Q. At runtime, we implement force/displacement projections $\boldsymbol{\mathcal{P}}\mathbf{v}$ using $\boldsymbol{\mathcal{P}}\mathbf{v} = \mathbf{v} - \mathsf{QQ}^T\mathbf{v}$ at $O(N)$ cost.

Constructing V: We interpret V as a vertex displacement basis with translation and linearized rotation eliminated in a particularly convenient way; V has $3N-6$ columns corresponding to deformation degrees of freedom. While any orthogonal matrix V with size $3N \times (3N - 6)$ which eliminates the rigid-body modes is applicable to (3.2), in practice, sparser V can lead to sparser factorizations of $\mathsf{V}^T\mathsf{KV}$. So that $\mathsf{V}^T\mathsf{KV}$ construction has $O(\mathsf{nnz}(\mathsf{K}))$ cost, our V matrix, with only $3N$

nonzero elements, has the following form:

$$V = \left[\begin{array}{cccccccc} 1 & & & & & & & \\ & \ddots & & & & \mathbf{0} & & \\ & & 1 & & & & & \\ & & & 0 & & & & \\ & & & 0 & & & & \\ & & & 0 & & & & \\ & & & a_1 & & & & \\ & & & a_2 & & & & \\ & & & a_3 & & & & \\ & & & & a_1 & b_1 & & \\ & & & & a_2 & b_2 & & \\ & & & & a_3 & b_3 & & \\ & \mathbf{0} & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{array}\right] \begin{array}{l} \\ \\ \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{vertex } t_1 \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{vertex } t_2 \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{vertex } t_3 \\ \\ \\ \\ \end{array}$$

where t_1, t_2 and t_3 are any three non-collinear tetrahedral vertices upon which V applies the following six constraints: (1-3) we fix t_1 by imposing zero translation; then (4) to eliminate the rotation (c.f. [BHTF07]) we constrain t_2 to move along a fixed direction, $\boldsymbol{a} = (a_1, a_2, a_3) = \mathsf{normalize}(\boldsymbol{x}_{t_2} - \boldsymbol{x}_{t_1})$; then (5-6) we constrain t_3 to lie on an perpendicular plane by letting $(b_1, b_2, b_3) = \mathsf{normalize}([\boldsymbol{a} \times (\boldsymbol{x}_{t_3} - \boldsymbol{x}_{t_1})] \times \boldsymbol{a})$. All other rows match the identity matrix. By construction V is orthogonal.

A.2 Proof of least-residual solution to (3.2)

Given the solution $\boldsymbol{r}$ of (3.2), we show that $\boldsymbol{u} = \mathsf{V}\boldsymbol{r}$ is the least-residual solution of the compatible system $\mathsf{K}\boldsymbol{u} = \boldsymbol{\mathcal{P}f}$. Since K is a $3N \times 3N$ symmetric and rank-deficient matrix, we write it as $\mathsf{K} = \mathsf{USU}^T$ using thin SVD, where U is the $3N \times (3N-6)$ orthogonal basis matrix spanning $\mathsf{range}(\mathsf{K})$, and S is an invertible $(3N-6) \times (3N-6)$ diagonal matrix of singular values. By compatibility, we can write

$\mathcal{P}\boldsymbol{f} = \mathsf{U}\tilde{\boldsymbol{f}}$. Substituting these expressions into (3.2), we see that $\boldsymbol{r}$ satisfies

$$\mathsf{V}^T\mathsf{U}\mathsf{S}\mathsf{U}^T\mathsf{V}\boldsymbol{r} = \mathsf{V}^T\mathsf{U}\tilde{\boldsymbol{f}}. \tag{A.2}$$

Since $\mathsf{V}^T\mathsf{U}$ is nonsingular (both U and V are orthogonal), we have $\mathsf{U}^T\mathsf{V}\boldsymbol{r} = \mathsf{S}^{-1}\tilde{\boldsymbol{f}}$. Therefore $\mathsf{V}\boldsymbol{r}$ is a least-residual solution:

$$\mathsf{K}\mathsf{V}\boldsymbol{r} = (\mathsf{U}\mathsf{S}\mathsf{U}^T)\mathsf{V}\boldsymbol{r} = \mathsf{U}\mathsf{S}(\mathsf{U}^T\mathsf{V}\boldsymbol{r}) = \mathsf{U}\mathsf{S}\mathsf{S}^{-1}\tilde{\boldsymbol{f}} = \mathcal{P}\boldsymbol{f}. \tag{A.3}$$

A.3 Derivation of Proxy Scaling Relations

The scalings (3.8) follow from $\boldsymbol{x} \to \gamma\boldsymbol{x}$ as follows. Since the object's volume and mass scale as γ^3, the mass matrix scales as

$$\mathsf{M} \to \gamma^3\mathsf{M}.$$

Since the modes are mass orthogonal, $\mathsf{U}^T\mathsf{M}\mathsf{U} = \mathbf{I}$, it follows that

$$\mathsf{U} \to \gamma^{-3/2}\mathsf{U}.$$

Stiffness matrix scaling,

$$\mathsf{K} \to \gamma\mathsf{K},$$

follows from the fact that matrix elements are integrals of energy Hessians (with γ^{-2} scaling) over volumes (with γ^3 scaling). It follows that the eigenvalues scale as

$$\omega^2 \to \omega^2 \frac{\text{scaling}(\mathsf{K})}{\text{scaling}(\mathsf{M})} = \omega^2/\gamma^2,$$

so that $\omega \to \omega/\gamma$, $k \to k/\gamma$, and $k\boldsymbol{x} \to k\boldsymbol{x}$. Multipole M_n^m scaling follows from (2.21), where it suffices to consider the term,

$$ik\,R_n^{-m}(\boldsymbol{y} - \boldsymbol{x}_0)\,\frac{\partial p}{\partial \boldsymbol{n}}(\boldsymbol{y})d\Gamma_{\boldsymbol{y}}:$$

k gives a γ^{-1} factor; R_n^{-m} is scale invariant since it depends on kr; the $d\Gamma$ area introduces a γ^2 factor; $\partial_{\boldsymbol{n}} p$ is the Neumann BC, which scales with $\omega^2 u_n$ as $\gamma^{-2}\gamma^{-3/2} = \gamma^{-7/2}$. Multiplying $\gamma^{-1} \cdot \gamma^2 \cdot \gamma^{-7/2}$ we obtain

$$M_n^m \rightarrow \gamma^{-5/2} M_n^m.$$

A.4 Estimation of Proxy Contact Point

Given a force $\boldsymbol{f}_p$ applied to the elliptical sound proxy, we apply it to the surface position where the surface normal direction $\boldsymbol{n}$ is opposite to the force direction, $\boldsymbol{n} = -\boldsymbol{f}_p / \|\boldsymbol{f}_p\|$. Let the implicit surface for the ellipsoid be

$$g(x, y, z) = \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - 1, \tag{A.4}$$

then the surface normal direction is coincident with its gradient

$$\nabla g = (\frac{2x}{a^2}, \frac{2y}{b^2}, \frac{2z}{c^2})^T.$$

The surface position with matching normal direction $\boldsymbol{n}$ satisfies the equation

$$\frac{\nabla g}{\|\nabla g\|} = \boldsymbol{n} \equiv (n_x, n_y, n_z)^T.$$

Letting $X = x^2/a^2$, $Y = y^2/b^2$ and $Z = z^2/c^2$, we obtain a 3×3 linear equation,

$$\begin{bmatrix} (n_x^2 - 1)/a^2 & n_x^2/b^2 & n_x^2/c^2 \\ n_y^2/a^2 & (n_y^2 - 1)/b^2 & n_y^2/c^2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{A.5}$$

whose solution yields the proxy's surface contact point,

$$\boldsymbol{p}_{scp} = \left(\mathrm{sgn}(n_x)\sqrt{a^2 X}, \mathrm{sgn}(n_y)\sqrt{b^2 Y}, \mathrm{sgn}(n_z)\sqrt{c^2 Z}\right). \tag{A.6}$$

APPENDIX B

APPENDIX FOR CHAPTER 5

B.1 Acoustic Bubble Formulae

The bubble's undamped natural frequency is [Lei94]

$$\omega_0 = \sqrt{3\gamma p_0 - 2\sigma/r_0}/\left(r_0\sqrt{\rho}\right), \tag{B.1}$$

and its damping rate is given by

$$\beta = \omega_0\delta/\sqrt{\delta^2+4} \tag{B.2}$$

where $\delta = \delta(\omega_0, r_0) = \delta_{rad} + \delta_{vis} + \delta_{th}$ is a dimensionless damping value describing damping due to wave radiation (rad), fluid viscosity (vis), and thermal conductivity (th):

$$\delta_{rad} = \frac{\omega_0 r_0}{c_f}, \quad \delta_{vis} = \frac{4\mu_f}{\rho\omega_0 r_0^2}, \quad \delta_{th} = 2\frac{\sqrt{\psi-3} - \frac{3\gamma-1}{3(\gamma-1)}}{\psi-4}, \tag{B.3}$$

with $\psi = \frac{16}{9(\gamma-1)^2}\frac{G_{thg}}{\omega_0}$. The numerous parameters are as follows (values given in Table 5.1): c_f is the fluid's speed of sound; p_0 is the hydrostatic pressure of the liquid (which we always approximate as 1 atm in our simulations); γ is the gas's heat capacity ratio (or adiabatic index); μ_f is the liquid's shear viscosity; σ is the fluid surface tension coefficient; $G_{th} = \frac{3\gamma p_0}{4\pi\rho D_g}$ is the thermal damping constant at resonance; and D_g is the gas's thermal diffusivity.

Our *ad hoc* entrainment-related blending function is:

$$q_{blend}(t) = \begin{cases} q(t)e^{-\frac{(e^{-\beta t}-0.85)^2}{0.0028125}}, & e^{-\beta t} \geq 0.85 \\ q(t), & e^{-\beta t} < 0.85 \end{cases} \tag{B.4}$$

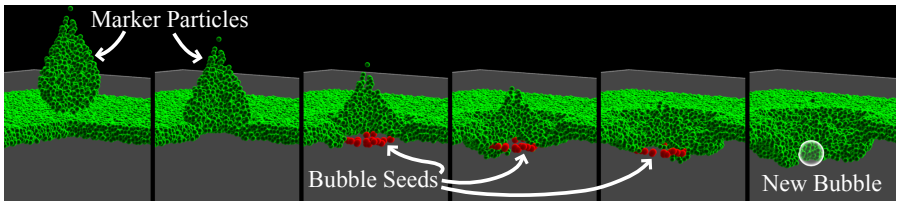


Figure B.1: **Bubble entrainment by a falling water drop** (cut-away view)

B.2 A Stochastic Model of Bubble Entrainment

Complex multi-scale interfacial mixing processes are responsible for bubble formation, but we desire a simplified computational model. We track mixing via the rapid movement of interfacial fluid material into the fluid volume by monitoring rapid changes in ϕ values of fluid material from a value near zero, to a value revealing it is now deep in the fluid. We place *markers* on a layer of particles near the surface: fluid particle i gets a marker if $\phi_\varepsilon < \phi_i < 0$, where ϕ_ε is a constant specifying the thickness of the marker layer (we use $\phi_\varepsilon = -2h$). At each time step, we track each marker's isosurface value. Dramatic decreases in marker ϕ_i values indicate the potential for bubble creation at the marker's position. When a sufficient ϕ_i decrease is detected, we call that marker a *bubble seed*—we use these in our bubble creation process. Markers and bubble seeds are illustrated in Figure B.1.

Unfortunately, the reconstructed isosurface field can be noisy, so that simply detecting rapid decreases in ϕ_i values is not robust. Therefore, we use linear regression to estimate the slope, $\frac{d\phi_i}{dt}$, by maintaining a sliding window (of width between $0.006sec$ and $0.01sec$) for each marker's ϕ_i values. The moment the slope exceeds a threshold (between $-0.9m/s$ and $-2.2m/s$), the marker becomes a bubble seed.

Bubble seed TTL and strength: Each bubble seed has (a) a creation time, t_0, (b) a time-to-live (TTL) value, T_{ttl}, after which the seed dies, and (c) a "bubble creation strength" value, $w_s(t)$, which is 1 initially and decays thereafter. Given a seed created at time t_0, we model the bubble seed's strength by the cubic spline:

$$w_s(t) = \begin{cases} 1 - 4\tau^3, & 0 \le \tau \le 0.5 \\ 4(1-\tau)^3, & 0.5 < \tau \le 1 \end{cases} \quad \text{where} \quad \tau = \frac{t - t_0}{T_{ttl}}.$$

This distribution of strength-weighted bubble seeds provides clues for creating bubbles at seed positions. We model the number of bubble creation attempts (per time step) as proportional to the sum total of seed strengths:

$$N_{bub} = \kappa h^2 \Delta t \sum_{\text{s is seed}} w_s(t), \tag{B.5}$$

where κ is a parameter controlling the bubbliness of the flow; and to try to make the bubble generation rate independent of spatial and temporal discretizations we scale by the interface fluid-grid resolution, h^2, and the time step size, Δt.

Bubble radius and spectra: The radii of created bubbles strongly affects the spectrum of the generated sound. In order to approximate the spectra of real fluid sounds, we use a probability distribution function to randomly sample bubble radii. In principle, by selecting a proper bubble radius distribution, we can match the sound spectrum to real sound cases—although not a sufficient condition for realistic sounds. Similar to [GH04], we use a *Gaussian* distribution: mean and deviation were calibrated by matching the characteristic pitch to typical recorded sounds.

Radius rejection sampling: For each bubble created at a time step, we randomly select a seed as the bubble's initial position. This provides a density-based sampling, so that well-seeded regions are more likely to create bubbles.

Algorithm 7: CreateBubbles($\mathcal{B}$, $\mathcal{M}$, $\mathcal{S}$, t)

Data: The set of current bubbles $\mathcal{B}$, seeds $\mathcal{S}$, current markers $\mathcal{M}$, and current time t

```
begin
    update_markers(M);
    sample_isosurface_value(M);
    create_seeds(M, S);
    update_seed_strengths(S);
    N_bub ← num_bubble_creation_attempts(S);
    for i = 1 ... N_bub do
        seed ← random_select_seed(S);
        r ← random_select_radius;
        if not reject_bubble(seed, r, S) then
            create_bubble(seed.pos, r);
            remove_seeds(seed, r, S);
        end
    end
end
```

Given a randomly sampled bubble radius and position, to avoid placing unrealistically large bubbles in small regions, our check to determine if enough local seeds s are inside the bubble is:

$$r_{par}^2 \sum_{\boldsymbol{x}_s \in Bubble} w_s(t) \quad < \quad \tau_{rej}\, r_0^2 \tag{B.6}$$

where r_{par} is the fluid particle radius ($0.22h$ in our simulations), and τ_{rej} controls bubble sizes (our examples use τ_{ref} values between 0.9 and 2). Otherwise we create a bubble, and the seeds inside a sphere of radius $1.5r_0$ are removed.

Algorithm: Our bubble creation method is summarized in Algorithm 7. In reality, bubbles are generated at very high rates, so that sounds from splashing or pouring appear continuous. To avoid discretization artifacts here, bubble creation times are uniformly distributed during the time step. The bubbles' initial positions and velocities are interpolated from bubble seeds.

	Drop	Splash	Pour	WStep	Description	Eqn
κ	3.2	1.3	1.0	1.8	bubbliness	(B.5)
τ_{rej}	2.0	0.9	1.4	1.5	radius limiter	(B.6)

Table B.1: **User-specified entrainment parameters** are roughly of unit size.

Parameter Tuning: Model parameters can be tuned manually for best results. We first adjust the bubbly flow to get a plausible number of bubbles by tuning κ in (B.5) and τ_{rej} in (B.6), with unit values being good initial guesses (see Table B.1). In the second pass, we can adjust the (Gaussian) distribution for the bubbles' radius (and thus frequency), e.g., to approximate spectra of recorded fluid sounds.

B.3 Derivation of Source Strength, S_b

We estimate the delta-function source strength, S_b, of a point-like bubble from its "divergence sourcing" strength (c.f. [KLL+07]). First, we take the divergence of the relationship between harmonic acoustic pressure $p(\boldsymbol{x})$ and acoustic velocity $\boldsymbol{v}(\boldsymbol{x})$,

$$\nabla p = -i\omega\rho\boldsymbol{v} \quad \Rightarrow \quad \nabla^2 p = -i\omega\rho\left(\nabla\cdot\boldsymbol{v}\right). \tag{B.7}$$

Given our divergence singularity of the form, $\nabla^2 p = S_b\delta(\boldsymbol{x}-\boldsymbol{x}_b)$, we can estimate S_b by integrating over a small domain Ω_b containing the tiny bubble (so that $\int_{\Omega_b}\delta(\boldsymbol{x}-\boldsymbol{x}_b)d\Omega = 1$):

$$S_b = -i\omega\rho\int_{\Omega_b}\left(\nabla\cdot\boldsymbol{v}\right)d\Omega. \tag{B.8}$$

The divergence theorem, and the rate of fluid expulsion from the volume Ω_b due to ε-amplitude pulsations, $r = r_0 + \varepsilon e^{+i\omega t}$, yields

$$\int_{\Omega_b}\left(\nabla\cdot\boldsymbol{v}e^{+i\omega t}\right)d\Omega = -\frac{d\,V_b}{dt} = -4\pi i\omega r^2\varepsilon e^{+i\omega t}. \tag{B.9}$$

It follows that S_b is given by (5.10).

APPENDIX C

APPENDIX FOR CHAPTER 7

C.1 Analytical Solution of LCQP Problem (7.8)

Here we present the analytical solution to the LCQP problem (7.8) which evaluates the optimal deformation energy for specified contact locations given by the barycentric coordinates α and β. Suppose there are n vertices on the mesh, then the displacement vector $\boldsymbol{u}$ is of length $3n$, and K is a $3n \times 3n$ matrix. First, we introduce some notations: let $\boldsymbol{a} = [\,\alpha_1\ \alpha_2\ \alpha_3\ -\beta_1\ -\beta_2\ -\beta_3\,]^T$, $\mathsf{B} = [\,\boldsymbol{x}_1^i\ \boldsymbol{x}_2^i\ \boldsymbol{x}_3^i\ \boldsymbol{x}_1^j\ \boldsymbol{x}_2^j\ \boldsymbol{x}_3^j\,]^T$, and

$$\mathsf{A} = [\,0\ \ldots\ \alpha_1\boldsymbol{I}\ \alpha_2\boldsymbol{I}\ \alpha_3\boldsymbol{I}\ \ldots\ 0\ \ldots\ -\beta_1\boldsymbol{I}\ -\beta_2\boldsymbol{I}\ -\beta_3\boldsymbol{I}\ \ldots\ 0\,],$$

where $\boldsymbol{I}$ is a 3×3 identity matrix, A is a $3 \times 3n$ sparse matrix, and the positions of the 3×3 block matrices $\alpha_t\boldsymbol{I}, -\beta_t\boldsymbol{I}, t = 1, 2, 3$ in A correspond to the positions of vertices $\boldsymbol{X}_t^i, \boldsymbol{X}_t^j, t = 1, 2, 3$ in the vector $\boldsymbol{u}$. With these notations, the equality constraint of (7.8) becomes

$$\mathsf{A}\boldsymbol{u} + \mathsf{B}^T\boldsymbol{a} = 0. \tag{C.1}$$

Using Lagrange multipliers, the displacement $\boldsymbol{u}$ should satisfy the following equation at the optimum,

$$\mathsf{K}\boldsymbol{u} = \mathsf{A}^T\boldsymbol{\lambda}, \tag{C.2}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^3$ are Lagrange multipliers. Note that although K is rank-3 deficient (recall section 7.3), equation (C.2) can be exactly satisfied with infinite number of $\boldsymbol{u}$. This is because the null space of K spans the displacements of rigid translation, and the matrix A, whose sum of each row is always zero, has vanished projection on the null space of K. All the solutions of (C.2) yield the

same energy value $\boldsymbol{u}^T \mathsf{K} \boldsymbol{u}$, and can be expressed as $\boldsymbol{u} = \mathsf{K}^\dagger \mathsf{A}^T \boldsymbol{\lambda}$, where $\mathsf{K}^\dagger$ is the Moore-Penrose pseudo-inverse of K. Then $\boldsymbol{\lambda}$ can be determined using the constraint (C.1). Namely,

$$\mathsf{A}\boldsymbol{u} = \mathsf{A}\mathsf{K}^\dagger \mathsf{A}^T \boldsymbol{\lambda} = -\mathsf{B}^T \boldsymbol{a}. \tag{C.3}$$

Note that the K matrix can be seen as an $n \times n$ block matrix, in which each block is a scaled 3×3 identity matrix, $a\boldsymbol{I}_{3\times 3}$, and similarly A can be seen as a $1 \times n$ block matrix. We first condense K into a $n \times n$ matrix $\tilde{\mathsf{K}}$, where each element $\tilde{\mathsf{K}}_{ij}$ is the scalar of the corresponding 3×3 block in K. It can be shown that $\tilde{\mathsf{K}}$ is a rank-1 deficient matrix, and its pseudo-inverse $\tilde{\mathsf{K}}^\dagger$ is the condensed version of $\mathsf{K}^\dagger$. Moreover, A is sparse, having only non-zero blocks corresponding to the involved 6 vertices. The left-hand side of (C.3) can be written as a 6×6 quadratic form,

$$\boldsymbol{\lambda} = \frac{-\mathsf{B}^T \boldsymbol{a}}{\boldsymbol{a}^T \tilde{\mathsf{K}}_s^\dagger \boldsymbol{a}}.$$

And hence the optimum energy value is

$$\hat{E}_{ij} = \boldsymbol{u}^T \mathsf{K} \boldsymbol{u} = \boldsymbol{u}^T \mathsf{A}^T \boldsymbol{\lambda} = -(\mathsf{B}\boldsymbol{a})^T \boldsymbol{\lambda} = \frac{\boldsymbol{a}^T \mathsf{B}\mathsf{B}^T \boldsymbol{a}}{\boldsymbol{a}^T \tilde{\mathsf{K}}_s^\dagger \boldsymbol{a}}. \tag{C.4}$$

C.2 Fast Precomputation of Green's function

As shown in appendix C.1, the certificate precomputation involves computing the Green's function $\mathsf{G} = \tilde{\mathsf{K}}^\dagger$ of the $n \times n$ matrix $\tilde{\mathsf{K}}$ condensed from K. Direct computation requires the SVD or eigen-decomposition of $\tilde{\mathsf{K}}$ (recall that $\tilde{\mathsf{K}}$ is a rank-1 deficient symmetric positive semi-definite matrix). Let the thin SVD of $\tilde{\mathsf{K}}$ is $\tilde{\mathsf{K}} = \mathsf{V}\mathsf{S}\mathsf{V}^T$, where V is $n \times (n-1)$ orthonormal matrix, and S is $(n-1) \times (n-1)$ diagonal matrix, then the pseudo-inverse is $\tilde{\mathsf{K}}^\dagger = \mathsf{V}\mathsf{S}^{-1}\mathsf{V}^T$. This computation requires $O(n^3)$ work for SVD or eigen-decomposition.

In this section, instead of computing $\tilde{\mathsf{K}}^\dagger$ directly, we present a fast computation of a matrix G equivalent to $\tilde{\mathsf{K}}^\dagger$ in the sense that

$$\boldsymbol{a}^T \mathsf{G} \boldsymbol{a} = \boldsymbol{a}^T \tilde{\mathsf{K}}^\dagger \boldsymbol{a} \tag{C.5}$$

for all possible $\boldsymbol{a}$, where $\boldsymbol{a}$ corresponds to barycentric coordinates of two points on the triangle, and has the form $\boldsymbol{a} = [\,0 \ldots \alpha_1\ \alpha_2\ \alpha_3 \ldots 0 \ldots -\beta_1\ -\beta_2\ -\beta_3 \ldots 0\,]$, with $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\beta_1 + \beta_2 + \beta_3 = 1$. The equivalence (C.5) is sufficient for the certificate computation using (C.4).

First, note that the null space of $\tilde{\mathsf{K}}^\dagger$ is spanned by the vector $\boldsymbol{v} = [\,1\ 1 \ldots 1\,]^T_{1\times n}$, and $\boldsymbol{v}^T \boldsymbol{a} = 0$. Then the vector $\boldsymbol{a}$ can be written using the eigen-matrix of $\tilde{\mathsf{K}}$ as $\boldsymbol{a} = \mathsf{V}\boldsymbol{k}$. Now given an $n \times (n-1)$ matrix U such that $\mathsf{V}^T\mathsf{U}$ is a $(n-1)\times(n-1)$ full-rank matrix, then $\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U}$ is invertible, and $\mathsf{G} = \mathsf{U}(\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})^{-1}\mathsf{U}^T$ is equivalent to $\tilde{\mathsf{K}}^\dagger$ in the sense of (C.5). This is because

$$\begin{aligned}
\boldsymbol{a}^T \mathsf{U}(\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})^{-1}\mathsf{U}^T\boldsymbol{a} &= \boldsymbol{k}^T\mathsf{V}^T\mathsf{U}(\mathsf{U}^T\mathsf{V}\mathsf{S}\mathsf{V}^T\mathsf{U})^{-1}\mathsf{U}^T\mathsf{V}\boldsymbol{k} \\
&= \boldsymbol{k}^T\mathsf{V}^T\mathsf{U}(\mathsf{V}^T\mathsf{U})^{-1}\mathsf{S}^{-1}(\mathsf{U}^T\mathsf{V})^{-1}\mathsf{U}^T\mathsf{V}\boldsymbol{k} \\
&= \boldsymbol{k}^T\mathsf{S}^{-1}\boldsymbol{k} = \mathsf{k}^T\mathsf{V}^T\mathsf{V}\mathsf{S}^{-1}\mathsf{V}^T\mathsf{V}\boldsymbol{k} \\
&= \boldsymbol{a}^T\tilde{\mathsf{K}}^\dagger\boldsymbol{a}.
\end{aligned}$$

In practice, we use a sparse matrix $\mathsf{U} = \begin{bmatrix} \boldsymbol{I}_{n\times n} & \boldsymbol{0}_{n\times 1} \end{bmatrix}^T$, and then $\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U}$ is just the $(n-1)\times(n-1)$ upper-left sub-matrix of $\tilde{\mathsf{K}}$. We compute $\tilde{\mathsf{G}} = (\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})^{-1}$ using the sparse Cholesky factorization of $\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U}$ (with complexity $O(n^{\frac{3}{2}})$) and a solve $(\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})\tilde{\mathsf{G}} = \boldsymbol{I}_{(n-1)\times(n-1)}$ (with complexity $O(n^2 \lg n)$). Both the factorization and back-substitution for the solve can be efficiently performed in parallel using PARDISO direct solver [SG04]. Finally, the $n \times n$ G matrix is a simple expansion of $\tilde{\mathsf{G}}$,

$$\mathsf{G} = \mathsf{U}\tilde{\mathsf{G}}\mathsf{U}^T = \begin{bmatrix} \tilde{\mathsf{G}} & \boldsymbol{0} \\ \boldsymbol{0} & 0 \end{bmatrix}.$$

For complexity estimates, we used the fact that a 2D grid Laplacian on n vertices suggests an $O(n^{\frac{3}{2}})$ cost for sparse Cholesky factorization, and $O(n^2 \lg n)$ for back-substitution (or less if done in parallel) to get G assuming the Cholesky factors require $O(n \lg n)$ space [Dem97].

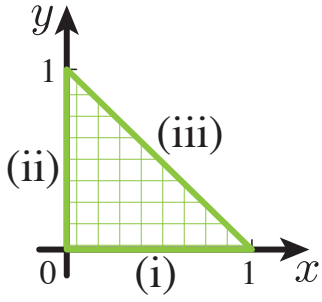
C.3 Exact Evaluation of Tri-Tri Certificates

In this section, we present the details of computing the triangle-triangle certificate by solving optimization problems (7.11) and (7.12). Both problems are instances of the more general problem,

$$\begin{aligned} \text{minimize} \quad & \frac{\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} + \boldsymbol{a}^T \boldsymbol{x} + c_u}{\boldsymbol{x}^T \mathsf{B} \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c_d} \\ \text{subject to} \quad & \boldsymbol{x} \in C, \end{aligned} \tag{C.6}$$

where both A and B are symmetric positive-definite matrices, $\boldsymbol{x} = [\,x\ y\,]^T$ is a 2D vector, and C is a convex set on 2D space. Recall that C is a triangle for problem (7.11), and a square for problem (7.12).

C.3.1 Optimum value on the boundary



First we consider the optimum value on the boundary of C. As presented in section 7.5.1, a boundary point of problem (7.11) corresponds to a case where a vertex touches an edge, and so does a boundary point of problem (7.12). Therefore, we only need to compute the boundary optimum for (7.11). There are 3 segments on the boundary: (i)

$\boldsymbol{x} = [\,a\ 0\,]^T, a \in [0,1]$, (ii) $\boldsymbol{x} = [\,0\ a\,]^T, a \in [0,1]$, and (iii) $\boldsymbol{x} = [\,a\ 1{-}a\,]^T, a \in [0,1]$. Substituting them respectively into (C.6), we obtain a 1D objective function,

$$f(a) = \frac{A_u a^2 + B_u a + C_u}{A_d a^2 + B_d a + C_d}.$$

For the segment (i),

$$A_u = \mathsf{A}_{11},\ B_u = \boldsymbol{a}_1,\ C_u = c_u$$

$$A_d = \mathsf{B}_{11},\ B_d = \boldsymbol{b}_1,\ C_d = c_d.$$

For the segment (ii),

$$A_u = \mathsf{A}_{22},\ B_u = \boldsymbol{a}_2,\ C_u = c_u$$

$$A_d = \mathsf{B}_{22},\ B_d = \boldsymbol{b}_2,\ C_d = c_d$$

For the segment (iii),

$$A_u = \mathsf{A}_{11} - 2\mathsf{A}_{12} + \mathsf{A}_{22},$$

$$B_u = 2(\mathsf{A}_{12} - \mathsf{A}_{22}) + \boldsymbol{a}_1 - \boldsymbol{a}_2,$$

$$C_u = \mathsf{A}_{22} + \boldsymbol{a}_2 + c_u$$

$$A_d = \mathsf{B}_{11} - 2\mathsf{B}_{12} + \mathsf{B}_{22},$$

$$B_d = 2(\mathsf{B}_{12} - \mathsf{B}_{22}) + \boldsymbol{b}_1 - \boldsymbol{b}_2,$$

$$C_d = \mathsf{B}_{22} + \boldsymbol{b}_2 + c_d.$$

To compute the optimum on a segment, taking $f'(a) = 0$, we get a cubic equation. Fortunately, we observe that for all cases the 3rd-order coefficient always vanishes, and therefore we only need to solve the quadratic equation $A_q a^2 + B_q a + C_q = 0$, where $A_q = A_u B_d - B_u A_d$, $B_q = 2(C_d A_u - C_u A_d)$, and $C_q = C_d B_u - C_u B_d$. If the solution of these quadratic equations are in $[0,1]$, we compute its corresponding optimum value $f(a)$. Otherwise, the optimum occurs at the segment end-points, i.e., $f(0)$ or $f(1)$.

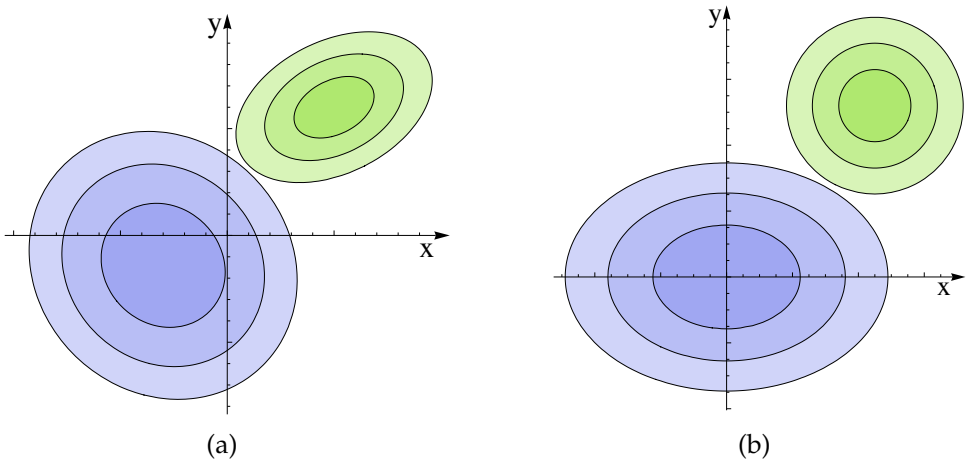


Figure C.1: **Simplification of the objective function:** (a) The numerator and denominator of the objective function in (C.6) are plotted as two elliptical contours. (b) We apply an affine transformation to regularize the objective function into a simpler form (C.7), in which the numerator has a circular contour and the denominator has an axis-aligned elliptical contour centered at the origin.

C.3.2 Optimum value in the interior of domain

Next we check if an optimum appears in the interior of C. For each interior optimum, we compute the optimum value, compare it with the boundary optimum values, and take the minimum. This case is more involved. To ease the derivation, we first regularize the problem (C.6) into a simpler form. Both the numerator and denominator of (C.6) are 2-D quadratic forms, representing two sets of ellipse contours on the 2-D plane (see Figure C.1(a)). Using an affine transformation $\psi : \boldsymbol{x} \mapsto \mathsf{F}_a\boldsymbol{x} + \boldsymbol{t}_a$, we simplify (C.6) into the following form (see Figure C.1(b)),

$$\begin{aligned} \text{minimize} \quad & \frac{(u - c_x)^2 + (v - c_y)^2 + \tilde{c}_u}{au^2 + bv^2 + \tilde{c}_d} \\ \text{subject to} \quad & [\, u \;\; v \,]^T \in \psi[C]. \end{aligned} \tag{C.7}$$

In practice, this simplification is performed in three steps: (i) we compute the eigen-decomposition, $\mathsf{A} = \mathsf{V}\Sigma\mathsf{V}^T$, to simplify the numerator of (C.6), and using

$\boldsymbol{x} = \mathsf{V}\Sigma^{-1/2}\mathsf{V}^T\boldsymbol{y}$, the objective function of (C.6) becomes into

$$\frac{\boldsymbol{y}^T\boldsymbol{y} + \boldsymbol{a}^T\mathsf{V}\Sigma^{-\frac{1}{2}}\boldsymbol{y} + c_u}{\boldsymbol{y}^T\Sigma^{-\frac{1}{2}}\mathsf{V}^T\mathsf{B}\mathsf{V}\Sigma^{-\frac{1}{2}}\boldsymbol{y} + \boldsymbol{b}^T\mathsf{V}\Sigma^{-\frac{1}{2}}\boldsymbol{y} + c_d} \equiv \frac{\boldsymbol{y}^T\boldsymbol{y} + \tilde{\boldsymbol{a}}^T\boldsymbol{y} + c_u}{\boldsymbol{y}^T\tilde{\mathsf{B}}\boldsymbol{y} + \tilde{\boldsymbol{b}}^T\boldsymbol{y} + c_d},$$

(ii) next we compute the eigen-decomposition, $\tilde{\mathsf{B}} = \mathsf{U}\mathsf{D}\mathsf{U}^T$, to diagonalize the denominator: using $\boldsymbol{z} = \mathsf{U}\boldsymbol{y}$ further transforms the above objective function into

$$\frac{\boldsymbol{z}^T\boldsymbol{z} + \tilde{\boldsymbol{a}}^T\mathsf{U}\boldsymbol{z} + c_u}{\boldsymbol{z}^T\mathsf{D}\boldsymbol{z} + \tilde{\boldsymbol{b}}^T\mathsf{U}\boldsymbol{z} + c_d} \equiv \frac{\boldsymbol{z}^T\boldsymbol{z} + \hat{\boldsymbol{a}}^T\boldsymbol{z} + c_u}{\boldsymbol{z}^T\mathsf{D}\boldsymbol{z} + \hat{\boldsymbol{b}}^T\boldsymbol{z} + c_d},$$

and (iii) we take the translation $\boldsymbol{z} = \boldsymbol{v} - \frac{1}{2}\mathsf{D}^{-1}\hat{\boldsymbol{b}}$, and finally simplify the objective function into

$$\frac{\boldsymbol{v}^T\boldsymbol{v} + (\hat{\boldsymbol{a}} - \mathsf{D}^{-1}\hat{\boldsymbol{b}})^T\boldsymbol{v} + c_u + \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-2}\hat{\boldsymbol{b}} - \frac{1}{2}\hat{\boldsymbol{a}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}}}{\boldsymbol{v}^T\mathsf{D}\boldsymbol{v} + c_d - \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}}}.$$

This form of objective function agrees with (C.7): let $\boldsymbol{p} \equiv \hat{\boldsymbol{a}} - \mathsf{D}^{-1}\hat{\boldsymbol{b}}$, then $c_x = -\boldsymbol{p}_1/2$, $c_y = -\boldsymbol{p}_2/2$, $a = \mathsf{D}_{11}$, $b = \mathsf{D}_{22}$,

$$\tilde{c}_u = c_u + \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-2}\hat{\boldsymbol{b}} - \frac{1}{2}\hat{\boldsymbol{a}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}} - c_x^2 - c_y^2, \text{ and}$$
$$\tilde{c}_d = c_d - \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}}.$$

Affine transformation preserves convexity of a domain, therefore $\psi[C]$ is still a convex set. Since the simplified problem (C.7) is equivalent to (C.6), from now on, we check if the optimum of (C.7) can be achieved in the interior of $\psi[C]$, and then compute the optimum value if it is inside of $\psi[C]$. Let $n(u, v)$ and $d(u, v)$ denote the numerator and denominator respectively in the objective function of (C.7). Consider the objective value on a single contour $\Omega_C : d(u, v) = C$ (see blue ellipse in Figure C.2). The method of Lagrange multipliers shows that the local optimum of the objective function $\frac{n(u,v)}{d(u,v)}$ on the contour Ω_C occurs when a contour of $n(u, v)$ meets Ω_C tangentially (see green circle in Figure C.2a). Let $\boldsymbol{\tau}$ denote the optimum point on $d(u, v) = C$. If $\boldsymbol{\tau}$ is outside of the $\psi[C]$, then the optimum on the set $\Omega_C \cap \psi[C]$ (see the gray polygon in Figure C.2a) is on the

boundary of $\psi[C]$, in which case the optimum has been computed as presented in Appendix C.3.1. Otherwise, we need to compute the interior-point optimum.

Now put together the optimum points of all contours of $d(u, v)$. We observe that they form a curve C_p (see red curve in Figure C.2a) which is analytically determined by the following function,

$$y = \frac{1}{C_A x + C_B} - \frac{1}{C_B}, \tag{C.8}$$

where $C_A = \frac{-(b-a)^2}{abc_x c_y}$ and $C_B = \frac{b-a}{ac_y}$. Then we determine if the curve C_p intersects with the domain $\psi[C]$ by checking the intersection of C_p and the piecewise boundary segments of $\psi[C]$. In particular, to check the intersection of C_p and a boundary segment defined by its two end points (x_1, y_1) and (x_2, y_2), we solve a quadratic equation $A_b a^2 + B_b a + C_b = 0$, where

$$\begin{aligned}
A_b &= C_A C_B (x_1 - x_2)(y_1 - y_2), \\
B_b &= C_B^2 (y_1 - y_2) + C_A[(x_1 - x_2) + C_B(x_1 y_2 + x_2 y_1 - 2x_2 y_2)], \\
C_b &= C_B^2 y_2 + C_A(x_2 + C_B x_2 y_2).
\end{aligned}$$

The curve C_p intersects with the boundary segment if a root a of this quadratic equation is in $[0, 1]$. For problem (7.11), its domain has 3 boundary segments, hence it requires 3 quadratic solves, and for problem (7.12), it needs 4 quadratic solves to check intersections. If C_p is separated from $\psi[C]$, no further computation is needed, because the optimum will occur on the boundary of $\psi[C]$, and the optimum values have been computed in Appendix C.3.1. Otherwise, we compute the local optimum $\boldsymbol{q}$ of the objective function $\frac{n(u,v)}{d(u,v)}$ on the curve C_p. If $\boldsymbol{q}$ is outside of $\psi[C]$, then again the optimum on the set $C_p \cap \psi[C]$ is on the boundary of $\psi[C]$, and hence the optimum of the entire domain $\psi[C]$ occurs on the boundary. If $\boldsymbol{q}$ is in the interior of $\psi[C]$, we compare its ob-

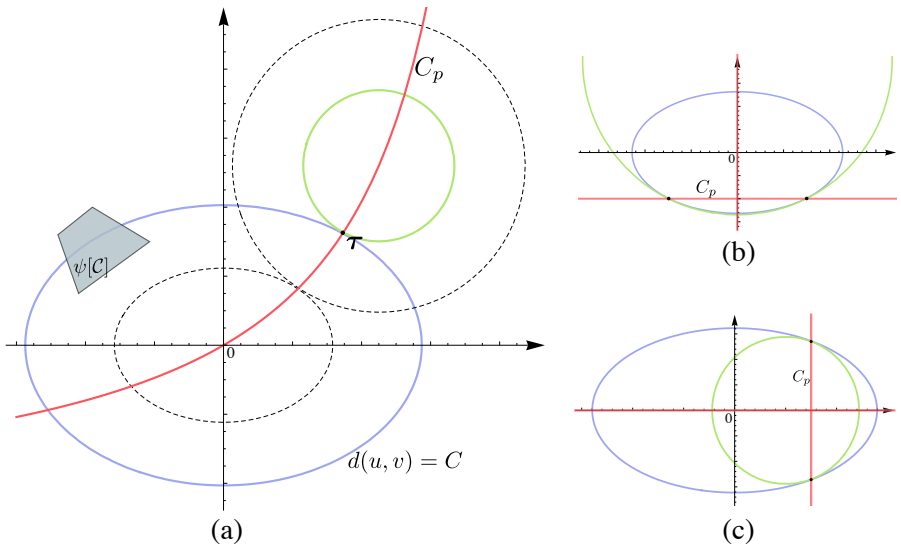


Figure C.2: **Contour optimum:** The optimum of objective function (C.7) on elliptical contours form the red curve which can be analytically determined. (a) shows the general case, and (b) and (c) illustrate the special cases where the center of the circular contours of the numerator is on x- or y- axes.

jective value with the minimum value from the boundary (computed in Appendix C.3.1) and take the minimum. $\boldsymbol{q}$ is computed as follows. Substituting (C.8) into the objective function of (C.7), we get a single variable rational function $r(x)$. The optimum occurs when $r'(x) = 0$. This equation is a 6-order polynomial equation, which fortunately can be factorized into two cubic equations, i.e. $r'(x) = p(x)q(x) = 0$. The first cubic equation $p(x) = P_a x^3 + P_b x^2 + P_c x + P_d$ has the coefficients

$$P_a = (a-b)^3,$$

$$P_b = 3(a-b)^2 b c_x,$$

$$P_c = 3(a-b)b^2 c_x^2, \text{and}$$

$$P_d = b^2 c_x (b c_x^2 + a c_y^2).$$

The discriminant of this cubic equation is always negative, indicating this equation has only one real root. The second cubic equation $q(x) = Q_a x^3 + Q_b x^2 + Q_c x +$

Q_d has the coefficients

$$
\begin{aligned}
Q_a &= a(a-b)c_x, \\
Q_b &= (b-a)(a\tilde{c}_u - \tilde{c}_d) + ac_x^2(2b-a) + abc_y^2, \\
Q_c &= -c_x(\tilde{c}_d(a-2b) + ab(\tilde{c}_u + c_x^2 + c_y^2)), \text{and} \\
Q_d &= -b\tilde{c}_d c_x^2.
\end{aligned}
$$

We solve these cubic equations using the method of Nickalls [Nic93]. In practice, the curve C_p is separated from the domain $\psi[C]$ in most of the cases. In practice, we only need to solve cubic equations for less than 0.2% of the triangle-triangle certificates.

Care needs to be taken for two special cases where C_p produces lines parallel to the X or Y axes: (i) $c_x = 0$ and (ii) $c_y = 0$. Both cases largely simplify the computation. When $c_x = 0$, C_p are the lines $x = 0$ and $y = \frac{a}{a-b}c_y$ (See Figure C.2b); when $c_y = 0$, C_p are the lines $y = 0$ and $x = \frac{b}{b-a}c_x$ (see Figure C.2c). Determining the intersection of these straight lines with the convex domain is trivial. When $c_x = 0$, the optimum on the curve C_p always occurs on Y-axis, and is determined by a quadratic equation $A_y y^2 + B_y y + C_y = 0$ (instead of a cubic equation), where

$$A_y = 2bc_y,\ B_y = 2[\tilde{c}_d - b(\tilde{c}_u + c_y^2)],\ \text{and}\ C_y = -2\tilde{c}_d c_y.$$

Symmetrically, when $c_y = 0$, the optimum on the curve C_p appears on X-axis, and is computed by the quadratic equation $A_x x^2 + B_x x + C_x = 0$, where

$$A_x = 2ac_x,\ B_x = 2[\tilde{c}_d - a(\tilde{c}_u + c_x^2)],\ \text{and}\ C_x = -2\tilde{c}_d c_x.$$

BIBLIOGRAPHY

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[AP97] M. Anitescu and F.A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997.

[APKG07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively Sampled Particle Fluids. In *Proc. ACM SIGGRAPH*, San Diego, August 2007.

[AS64] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 1964.

[Bar90] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, pages 19–28, August 1990.

[Bar91] David Baraff. Coping with friction for non-penetrating rigid body simulation. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, pages 31–40, July 1991.

[Bar93] David Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10(2-4):292–352, 1993.

[BD98] C. Phillip Brown and Richard O. Duda. A Structural Model for Binaural Sound Synthesis. *IEEE Trans. on Speech and Audio Processing*, 6(5), 1998.

[BDT+08] Nicolas Bonneel, George Drettakis, Nicolas Tsingos, Isabelle Viaud-Delmon, and Doug James. Fast modal sounds with scalable frequency-domain synthesis. *ACM Transactions on Graphics*, 27(3):24:1–24:9, August 2008.

[Beg94] D.R. Begault. *3-D sound for virtual reality and multimedia*. Academic Press Professional, Inc. San Diego, CA, USA, 1994.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust Treatment of Collisions, Contact, and Friction for Cloth Animation. *ACM Trans. on Graphics*, 21(3):594–603, 2002.

[BGTG04] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus H. Gross. A state machine for real-time cutting of tetrahedral meshes. *Graphical Models*, 66(6):398–417, 2004.

[BHTF07] Zhaosheng Bao, Jeong-Mo Hong, Joseph Teran, and Ronald Fedkiw. Fracturing rigid materials. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):370–378, March 2007.

[BJ10] Jernej Barbič and Doug L. James. Subspace Self-Collision Culling. *ACM Transactions on Graphics*, 29(4):81:1–81:9, July 2010.

[Bla86] W.K. Blake. *Mechanics of Flow-Induced Sound and Vibration*. Academic Press, 1986.

[Bra20] S.W.H. Bragg. *The World of Sound*. G. Bell and Sons Ltd., London, 1920.

[Bro99] Bernard Brogliato. *Nonsmooth Mechanics*. Springer, second edition, 1999.

[BSM+03] Hector M. Briceño, Pedro V. Sander, Leonard McMillan, Steven Gortler, and Hugues Hoppe. Geometry videos: a new representation for 3d animations. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 136–146, August 2003.

[Bur98] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[But03] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, New York, NY, 2003.

[BW97] J. Bonet and R. D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, 1997.

[CAJ09] Jeffrey N. Chadwick, Steven S. An, and Doug L. James. Harmonic Shells: A practical nonlinear sound model for near-rigid thin shells. *ACM Trans. Graph.*, 28(5):1–10, 2009.

[CGC+02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. A multiresolution framework for dynamic deformations. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 41–48, July 2002.

[CMT04] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid Fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. on Graphics*, 23(3):377–384, August 2004.

[Coo97] P.R. Cook. Physically Informed Sonic Modeling (PhISM): Synthesis of percussive sounds. *Computer Music Journal*, pages 38–49, 1997.

[CP98] SL Chan and EO Purisima. A new tetrahedral tesselation scheme for isosurface generation. *Computers & Graphics*, 22(1):83–90, 1998.

[CPPK07] Paul W. Cleary, Soon Hyoung Pyo, Mahesh Prakash, and Bon Ki Koo. Bubbling and Frothing Liquids. *Proc. ACM SIGGRAPH*, August 2007.

[CPS92] R.W. Cottle, J.S. Pang, and R.E. Stone. *The linear complementarity problem*. Academic Press, 1992.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905–914, August 2004.

[CTM08] Sean Curtis, Rasmus Tamstorf, and Dinesh Manocha. Fast collision detection for deformable models using representative-triangles. In *Proc. ACM Symp. Interactive 3D Graphics and Games*, pages 61–69, 2008.

[DDCB01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 31–36, August 2001.

[Dea97] Grant B. Deane. Sound generation and air entrainment by breaking waves in the surf zone. *The Journal of the Acoustical Society of America*, 102(5):2671–2689, November 1997.

[Dem97] J.W. Demmel. *Applied numerical linear algebra*. SIAM, Philadelphia, PA, 1997.

[DKP04] K. Doel, D. Knott, and D.K. Pai. Interactive simulation of complex audiovisual scenes. *Presence: Teleoperators & Virtual Environments*, 13(1):99–111, 2004.

[Duf01] Dean G. Duffy. *Green's Functions with Applications*. Chapman and Hall, 2001.

[DYN03] Yoshinori Dobashi, Tsuyoshi Yamamoto, and Tomoyuki Nishita. Real-time rendering of aerodynamic sound using sound textures based on computational fluid dynamics. *ACM Trans. on Graphics*, 22(3):732–740, July 2003.

[EMF02] D.P. Enright, S.R. Marschner, and R.P. Fedkiw. Animation and rendering of complex water surface. *ACM Trans. on Graphics*, 22(3):736–744, 2002.

[Erl07] Kenny Erleben. Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics*, 26(2):12:1–12:20, June 2007.

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. *Proc. ACM SIGGRAPH*, pages 23–30, 2001.

[FM96] N. Foster and D. Metaxas. Realistic Animation of Liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[FMC99] Thomas A. Funkhouser, Patrick Min, and Ingrid Carlbom. Real-time acoustic modeling for distributed virtual environments. In *Proc. of SIGGRAPH 99*, pages 365–374, August 1999.

[Fra59] G. J. Franz. Splashes as Sources of Sound in Liquids. *Journal of the Acoustical Society of America*, 31(8):1080–1096, Aug 1959.

[Fun77] Y. Fung. *A First Course in Continuum Mechanics*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[Gav93] W.W. Gaver. Synthesizing auditory icons. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 228–235. ACM, 1993.

[GBF03] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. In *Proc. ACM SIGGRAPH*, pages 871–878, 2003.

[GD04] Nail A. Gumerov and Ramani Duraiswami. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Elsevier Science, first edition, 2004.

[GD05] N.A. Gumerov and R. Duraiswami. *Fast multipole methods for the Helmholtz equation in three dimensions*. Elsevier, 2005.

[GFM+93] S.P. Gross, J. Fineberg, M. Marder, WD McCormick, and H.L. Swinney. Acoustic emissions from rapidly moving cracks. *Physical review letters*, 71(19):3162–3165, 1993.

[GGN06] J. Gao, L.J. Guibas, and A. Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Appl.*, 35(1-2):2–19, 2006.

[GH04] S.T. Greenwood and D.H. House. Better with Bubbles: Enhancing the Visual Realism of Simulated Fluid. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2004.

[GHF+07] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient Simulation of Inextensible Cloth. *SIGGRAPH (ACM Transactions on Graphics)*, 26(3), Jul 2007.

[GKJ+05] Naga Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics*, 24(3):991–999, 2005.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Transactions on Graphics*, 21(3):281–290, July 2002.

[GLM96] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 171–180, August 1996.

[GLM05] N.K. Govindaraju, M.C. Lin, and D. Manocha. Quick-CULLIDE:

Fast inter-and intra-object collision culling using graphics hardware. In *Proc. IEEE Virtual Reality*, pages 59–66, 2005.

[GMW81] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, London, UK, 1981.

[GNRZ02] Leonidas Guibas, An Nguyen, Daniel Russel, and Li Zhang. Collision Detection for Deforming Necklaces. In *Proc. of the ACM Symp. on Computational Geometry*, pages 33–42, 2002.

[GS87] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.

[GS01] E. Grinspun and P. Schröder. Normal bounds for subdivision-surface interference detection. In *IEEE Visualization 2001*, pages 333–340, October 2001.

[GS06] Dietmar Gross and Thomas Seeling. *Fracture mechanics: with an introduction to micromechanics*. Springer, 2006.

[Gui04] L.J. Guibas. Kinetic Data Structures. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

[GVL96a] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.

[GVL96b] G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins University Press, third edition, 1996.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, pages 299–308, August 1988.

[Ham83] R. W. Hamming. *Digital Filters*. Prentice-Hall, 1983.

[Har07] Mark Harris. Optimizing Parallel Reduction in CUDA. *NVIDIA Developer Technology*, 2007.

[HFGL98] J.K. Hahn, H. Fouad, L. Gritz, and J.W. Lee. Integrating sounds and motions in virtual environments. *Presence*, 7(1):67–77, 1998.

[HLYK08] Jeong-Mo Hong, Ho-Young Lee, Jong-Chul Yoon, and Chang-Hun Kim. Bubbles alive. *ACM Trans. on Graphics*, 27(3):48:1–48:4, August 2008.

[How98] M. S. Howe. *Acoustics of Fluid-Structure Interactions*. Cambridge Press, 1998.

[How02] M. S. Howe. *Theory of Vortex Sound*. Cambridge Press, 2002.

[HTG04] B. Heidelberger, M. Teschner, and M. Gross. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG*, 12(3):145–152, 2004.

[Hub95] Philip M. Hubbard. *Collision Detection for Interactive Graphics Applications*. PhD thesis, Department of Comp. Science, Brown University, 1995.

[HVS+09] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. *ACM Transactions on Graphics*, 28(3):87:1–87:12, July 2009.

[IJN09] Takashi Imagire, Henry Johan, and Tomoyuki Nishita. A fast method for simulating destruction and the generated dust and debris. *The Visual Computer*, 25(5–7):719–727, May 2009.

[JBP06] Doug L. James, Jernej Barbic, and Dinesh K. Pai. Precomputed Acoustic Transfer: Output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Transactions on Graphics*, 25(3):987–995, July 2006.

[Jef85] D.R. Jefferson. Virtual time. *ACM Transaction on Programming Languages and Systems*, 7(3):404–425, July 1985.

[Jen94] F.B. Jensen. *Computational Ocean Acoustics*. American Institute of Physics, 1994.

[JP02] Doug L. James and Dinesh K. Pai. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. In *Proc. ACM SIGGRAPH*, pages 582–585, New York, NY, USA, 2002. ACM.

[JP04] Doug L. James and Dinesh K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics*, 23(3):393–398, August 2004.

[JT05] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Transactions on Graphics*, 24(3):399–407, August 2005.

[KC07] Theodore Kim and Mark Carlson. A simple boiling module. In *Proc. of Symp. on Computer Animation (SCA)*, 2007.

[KDS93] M. Kleiner, B.I. Dalenbaeck, and P. Svensson. Auralization-An Overview. *Journal-Audio Engineering Society*, 41:861–861, 1993.

[KHM+98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of *k*-DOPs. *IEEE Trans. Vis. & Comp. Graphics*, 4(1):21–36, 1998.

[KJ09] Theodore Kim and Doug L. James. Skipping steps in deformable simulation with online model reduction. *ACM Transactions on Graphics*, 28(5):123:1–123:9, December 2009.

[KJM10] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Efficient yarn-based cloth with adaptive contact linearization. *ACM Trans. Graph.*, 29(4):105:1–105:10, July 2010.

[KK95] E. Kita and N. Kamiya. Trefftz method: An overview. *Advances in Engineering Software*, 24:89–96, 1995.

[KLL+07] Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. Simulation of bubbles in foam with the volume control method. *ACM Trans. on Graphics*, 26(3):98:1–98:10, July 2007.

[KP03] Paul G. Kry and Dinesh K. Pai. Continuous contact simulation for smooth surfaces. *ACM Trans. Graph.*, 22(1):106–129, 2003.

[KPK00] R.L. Klatzky, D.K. Pai, and E.P. Krotkov. Perception of material from contact sounds. *Presence: Teleoperators & Virtual Environments*, 9(4):399–410, 2000.

[KSJP08] Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K.

Pai. Staggered projections for frictional contact in multibody systems. *ACM Transactions on Graphics*, 27(5):164:1–164:11, December 2008.

[L82] P. Lötstedt. Mechanical systems of rigid bodies subject to unilateral constraints. *SIAM J. of Appl. Math.*, 42(2):281–296, 1982.

[Lei94] T.G. Leighton. *The Acoustic Bubble*. Academic Press, 1994.

[LH90] Michael S. Longuet-Higgins. An analytic model of sound production by raindrops. *J. Fluid Mech.*, 214:395–410, 1990.

[Liu09] Y. J. Liu. *Fast Multipole Boundary Element Method: Theory and Applications in Engineering*. Cambridge University Press, 2009.

[LSY98] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, 1998.

[LT09] Stig Larsson and Vidar Thomee. *Partial Differential Equations With Numerical Methods*. Springer, 2009.

[MBF04] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics*, 23(3):385–392, August 2004.

[MCR04] Stephen McAdams, Antoine Chaigne, and Vincent Roussarie. The psychomechanics of simulated sound sources: Material properties of impacted bars. *The Journal of the Acoustical Society of America*, 115(3):1306–1320, 2004.

[MDM01] M. Müller, J. Dorsey, and L. McMillan. Real-time simulation of deformation and fracture of stiff materials. In *Eurographics Computer Animation and Simulation*, pages 113–124. Springer-Verlag Wien, 2001.

[MDSB02] Mark Meyer, Mathieu Desbrun, Peter Schröeder, and Al H. Barr. Discrete differential geometry operators for triangulated 2-manifolds. In *Proceedings of VisMath*, 2002.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and

Markus Gross. Meshless deformations based on shape matching. *ACM Trans. on Graphics (SIGGRAPH 2005)*, 24(3):471–478, 2005.

[MI86] P.M.C. Morse and K.U. Ingard. *Theoretical Acoustics*. Princeton University Press, 1986.

[Min33] M. Minnaert. On musical air-bubbles and sounds of running water. *Phil Mag*, 16:235–248, 1933.

[Mir00] Brian Mirtich. Timewarp rigid body simulation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 193–200, July 2000.

[Mor66] J.J. Moreau. Quadratic programming in mechanics: One-sided constraints. *J. of SIAM Control*, 4(1):153–158, 1966.

[MS01] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. In *Proc. of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 37–46, August 2001.

[MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, pages 289–298, August 1988.

[Nic93] R.W.D. Nickalls. A new approach to solving the cubic: Cardan's solution revealed. *The Mathematical Gazette*, 77(480):354–359, 1993.

[NTB+91] A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *The Visual Computer*, 7(4):210–219, 1991.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. *ACM Transactions on Graphics*, 21(3):291–294, July 2002.

[OCE01] James F. O'Brien, Perry R. Cook, and Georg Essl. Synthesizing sounds from physically based motion. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 529–536, August 2001.

[Och95] Martin Ochmann. The Source Simulation Technique for Acoustic Radiation Problems. *Acustica*, 81, 1995.

[OF03] S. Osher and R.P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146, August 1999.

[Oha04] R. Ohayon. Reduced models for fluid-structure interaction problems. *Int. J. Numer. Meth. Engng*, 60(1):139–152, 2004.

[OP90] H.N. Oguz and A. Prosperetti. Bubble entrainment by the impact of drops on liquid surfaces. *J. Fluid Mech.*, 219:143–179, 1990.

[OSG02] James F. O'Brien, Chen Shen, and Christine M. Gatchalian. Synthesizing sounds from rigid-body simulations. In *2002 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 175–182, July 2002.

[PCB89] H.C. Pumphery, L.A. Crum, and L Bjørnø. Underwater sound produced by individual drop impacts and rainfall. *J. Acoust. Soc. Am.*, 85:1518–1526, 1989.

[PKA+05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. *ACM Transactions on Graphics*, 24(3):957–964, August 2005.

[PO09] Eric G. Parker and James F. O'Brien. Real-time deformation and fracture in a game environment. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 165–175, New York, NY, USA, 2009. ACM.

[Pro97] X. Provot. Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface*, pages 177–189, 1997.

[PS75] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numerical Analysis*, 12:617–629, 1975.

[PTF09] Cécile Picard, Nicolas Tsingos, and François Faure. Retargetting

example sounds to interactive physics-driven animations. In *AES 35th International Conference on Audio for Games*, feb 2009.

[PTVF07] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes: The art of scientific computing*. Cambridge University Press, 2007.

[PW89] Alex Pentland and John Williams. Good Vibrations: Modal Dynamics for Graphics and Animation. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):215–222, July 1989.

[Rag02] Saty Raghavachary. Fracture generation on polygonal meshes using Voronoi polygons. In *ACM SIGGRAPH 2002 Conference Abstracts and Applications*, pages 187–187, New York, NY, USA, 2002. ACM.

[Rei07] James Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly, first edition, 2007.

[RF03] D. Rocchesso and F. Fontana, editors. *The Sounding Object*, chapter High-level models: bouncing, breaking, rolling, crumpling, pouring (by M. Rath and F. Fontana), pages 186–187. Edizioni di Mondo Estremo, Florence, Italy, 2003.

[RJ07] Alec R. Rivers and Doug L. James. FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. *ACM Trans. on Graphics (SIGGRAPH 2007)*, 26(3):82:1–82:6, 2007.

[RL06] Nikunj Raghuvanshi and Ming C. Lin. Interactive Sound Synthesis for Large Scale Environments. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 101–108, New York, NY, USA, 2006. ACM Press.

[RSM+10] Nikunj Raghuvanshi, John Snyder, Ravish Mehra, Ming Lin, and Naga Govindaraju. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. *ACM Transactions on Graphics*, 29(4):68:1–68:11, July 2010.

[RYL10] Z. Ren, H. Yeh, and M.C. Lin. Synthesizing contact sounds between textured objects. In *IEEE Virtual Reality*, 2010.

[Sch05] K. Schittkowski. QL: A Fortran code for convex quadratic

programming–user's guide, version 2.11. *Research Report, Department of Mathematics, University of Bayreuth*, 2005.

[SG04] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Computer Systems*, 20(3):475–487, 2004.

[SGG+06] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha. Fast Proximity Computation Among Deformable Models using Discrete Voronoi Diagrams. *ACM Trans. on Graphics*, 25(3):1144–1153, 2006.

[SGO09] Sara C. Schvartzman, Jorge Gascón, and Miguel A. Otaduy. Bounded normal trees for reduced deformations of triangulated surfaces. In *Symp. on Computer Animation (SCA)*, pages 75–82, 2009.

[Sha91] A. A. Shabana. *Theory of Vibration, Volume II: Discrete and Continuous Systems*. Springer-Verlag, New York, NY, first edition, 1991.

[SOG08] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. Fast adaptive shape matching deformations. In *2008 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 87–94, July 2008.

[SSF09] Jonathan Su, Craig Schroeder, and Ronald Fedkiw. Energy stability and fracture for frame rate rigid body simulations. In *SCA '09: Proc. 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–164, New York, NY, USA, 2009. ACM.

[SSK05] Oh-Young Song, Hyuncheol Shin, and Hyeong-Seok Ko. Stable but nondissipative water. *ACM Trans. on Graphics*, 24(1):81–97, January 2005.

[ST96] D.E. Stewart and J.C. Trinkle. An implicit time-stepping scheme for rigid-body dynamics with inelastic collisions and coulomb friction. *Inter. J. for Numerical Methods in Engineering*, 39:2673–2691, 1996.

[Sta99] Jos Stam. Stable fluids. In *Proc. ACM SIGGRAPH*, pages 121–128, 1999.

[Ste00] D.E. Stewart. Rigid-body dynamics with friction and impact. *SIAM Review*, 42(1):3–39, 2000.

[Str53] M. Strasberg. The pulsation frequency of non-spherical gas bubbles in liquids. *J. Acoust. Soc. Am.*, 25:536–537, 1953.

[SWB00] Jeffrey Smith, Andrew Witkin, and David Baraff. Fast and controllable simulation of the shattering of brittle objects. In *Graphics Interface*, pages 27–34. Blackwell Publishing, 2000.

[SGPO10] Sara C. Schvartzman, lvaro G. Pérez, and Miguel A. Otaduy. Star-contours for efficient hierarchical self-collision detection. *ACM Transactions on Graphics*, 29(4):80:1–80:8, July 2010.

[TCYM09] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Trans. on Visualization and Computer Graphics*, 15:544–557, 2009.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, pages 269–278, August 1988.

[TFNC01] Nicolas Tsingos, Thomas Funkhouser, Addy Ngan, and Ingrid Carlbom. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *Proc. of ACM SIGGRAPH 2001*, pages 545–552, August 2001.

[TGD04] Nicolas Tsingos, Emmanuel Gallo, and George Drettakis. Perceptual audio rendering of complex virtual environments. *ACM Trans. on Graphics*, 23(3):249–258, August 2004.

[TH92] Tapio Takala and James Hahn. Sound rendering. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, pages 211–220, July 1992.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision Detection for Deformable Objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[TO09] Fernando Trebien and Manuel M. Oliveira. Realistic real-time sound re-synthesis and processing for interactive virtual worlds. *The Visual Computer*, 25(5–7):469–477, May 2009.

[TSSMM07] N. Thuerey, F. Sadlo, S. Schirm, and M. Gross M. Müller. Real-time

simulations of bubbles and foam within a shallow-water framework. In *Proc. of Symp. on Computer Animation (SCA)*, 2007.

[Uri75] R.J. Urick. *Principles of Underwater Sound*. McGraw-Hill, 1975.

[vdB97] Gino van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *J. of Graphics Tools*, 2(4):1–14, 1997.

[vdD05] Kees van den Doel. Physically based models for liquid sounds. *ACM Trans. on Applied Perception*, 2(4):534–546, 2005.

[vdDKP01] Kees van den Doel, Paul G. Kry, and Dinesh K. Pai. FoleyAutomatic: Physically-Based Sound Effects for Interactive Simulation and Animation. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 537–544, August 2001.

[vdDP96] K. van den Doel and D. K. Pai. Synthesis of shape dependent sounds with physical modeling. In *Intl Conf. on Auditory Display*, Xerox PARC, Palo Alto, November 1996.

[VG00] Jean Vroomen and Beatrice de Gelder. Sound enhances visual perception: Cross-modal effects of auditory organization on vision. *Journal of Experimental Psychology: Human Perception and Performance*, 26(5):1583–1590, October 2000.

[VMT94] P. Volino and N. Magnenat-Thalmann. Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity. *Comp. Graphics Forum*, 13(3):155–166, 1994.

[Vor07] M. Vorlander. *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. Springer Verlag, 2007.

[WN03] T.M. Wasfy and A.K. Noor. Computational strategies for flexible multibody systems. *Appl. Mech. Rev.*, 56(6), 2003.

[Wri06] Peter Wriggers. *Computational Contact Mechanics*. Springer Berlin Heidelberg, second edition, 2006.

[WV84] WH Warren and RR Verbrugge. Auditory perception of breaking and bouncing events: A case study in ecological acoustics. *Journal of Experimental Psychology*, 10(5):704–712, 1984.

[ZB05] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. on Graphics*, 24(3):965–972, August 2005.

[ZJ09] Changxi Zheng and Doug L. James. Harmonic fluids. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–12, New York, NY, USA, 2009. ACM.

[ZJ10] Changxi Zheng and Doug L. James. Rigid-body fracture sound with precomputed soundbanks. *ACM Transactions on Graphics*, 29(4):69:1–69:13, July 2010.

[ZJ11] Changxi Zheng and Doug L. James. Toward high-quality modal contact sound. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4), August 2011.

[ZJ12] Changxi Zheng and Doug L. James. Energy-based Self-Collision Culling for Arbitrary Mesh Deformations. *ACM Transactions on Graphics*, 29(4), August 2012.

[ZYP06] Wen Zheng, Jun-Hai Yong, and Jean-Claude Paul. Simulation of bubbles. In *Proc. of Symp. on Computer Animation (SCA)*, pages 325–333, 2006.