Modal Sound Synthesis for Interactive Virtual Environments

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Dipl.-Inform. Dominik Rausch
aus Düsseldorf

Berichter: Univ.-Prof. Dr. rer. nat. Torsten W. Kuhlen
Univ.-Prof. Dr. rer. nat. Michael Vorländer

Tag der mündlichen Prüfung: 14. Dezember 2017

Statement

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig im Rahmen der an der RWTH Aachen üblichen Betreuung angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I guarantee herewith that this thesis has been done independently, with support of the Virtual Reality Group at the RWTH Aachen University and no other than the referenced sources were used.

Aachen, March 18, 2018 ....

Abstract

This thesis will present methods for sound synthesis for real-time application. In an initial study, the applicability and usability of synthesized vibration sounds will be examined for a virtual drilling task. The study shows that for the chosen scenario, realistic drilling sound can support interaction in a similar way to haptic vibrations, and can be utilized to compensate for a lack of haptic feedback.

Modal Synthesis is a promising approach for an automatic synthesis of physically-based contact sounds from the geometry and material properties of scene objects. However, some limitations still restrict the applicability of modal synthesis, which will be addressed in this thesis.

Synthesizing sounds in real-time can be a challenging problem. For this, Modal Synthesis is a promising approach that allows generating the contact of objects based on their physical properties. Modal Synthesis requires a Modal Analysis must be performed. This is a computationally expensive task and usually performed in a pre-processing step. In this thesis, approaches for the computation of a Modal Analysis at run-time will be proposed, which enable the use of Modal Synthesis for objects that cannot be analyzed upfront, e.g. because they are created interactively or are modified by the user. For this, the run-time requirements will be evaluated, and appropriate Levels-of-Detail approximations will be presented.

When a Modal Analysis has been computed, the resulting modal data can be used to compute the vibration sound produced by an object. These vibrations are excited by forces acting on the object. At run-time, the Modal Synthesis has to evaluate the modal vibrations and apply the force excitation. While these computations can be performed on a CPU, this strongly limits the number and complexity of sounding objects and the forces acting on them. This thesis will present specialized algorithms to compute the Modal Synthesis with active forces on a graphics card, allowing for a high number of sounding objects.

ZUSAMMENFASSUNG

Diese Arbeit behandelt Methoden zur interaktiven Soundsynthese für Echtzeitanwendungen. Zunächst wird in einer Studie die Anwendbarkeit und der Nutzen von synthetisierten Vibrations-Sounds anhand einer Interaktionsaufgabe untersucht. Für das in der Studie verwendete Szenario – der Benutzung eines virtuellen Bohrers – liefert das akustische Feedback ähnliche Ergebnisse wie das haptische Vibrationsfeedback, sodass die Haptik durch synthetisierte Sounds ersetzt werden kann.

Modale Synthese ist ein vielversprechender Ansatz für die automatische, physikalisch-basierte Synthese von Kontakt-Klängen. Die Modalsynthese analysiert die Geometrie und Materialeigenschaften eines Objektes, um charakteristische Vibrationsmoden zu bestimmen. Allerdings bestehen noch Probleme für die Anwendbarkeit der Modalsynthese für Echtzeit-Anwendungen, die in dieser Arbeit behandelt werden.

Um Modale Sound-Synthese nutzen zu können, werden Objekte mit einer Modalanalyse analysiert, um ihre Moden zu bestimmen. Die Modalanalyse ist zeitaufwändig, und wird in der Regel vorab berechnet. Für Objekte, die nicht vorverarbeitet werden können, muss die Modalanalyse zur Laufzeit durchgeführt werden. Diese Arbeit stellt Methoden vor, um in interaktiven Anwendungen die Modalanalyse zur Laufzeit zu ermöglichen. Hierfür werden die Laufzeitanforderungen analysiert und passende Approximationsmethoden vorgestellt.

Mit der Modalsynthese werden zur Laufzeit Moden angeregt und ihre Vibration verwendet, um Klänge zu generieren. Die Moden-Anregung und -Auswertung erfordert viele Berechnungsschritte, die die Modenanzahl - und damit die Anzahl und Komplexität von klingenden Objekten – begrenzt. Um eine hohe Zahl an Moden zu ermöglichen, werden spezielle Methoden, insbesondere zur Nutzung von Grafikkarten, vorgestellt, um die Laufzeit-Synthese zu beschleunigen.

CONTENTS

CHAPTER 1

INTRODUCTION

One of the major challenges of Virtual Reality (VR) is the creation of vivid, realistic virtual scenes. To achieve a high degree of realism in a Virtual Environment (VE), both the simulation of the scene itself as well as its reproduction must have a high fidelity. Although the visual aspect of a VE is often the main focus, other senses, especially the auditory one, should be addressed, too.

A high degree of realism in the simulation helps to make a virtual scenery more believable. For example, objects in the scene should behave as in the real world, according to its physical laws. In the real world, a ball that is thrown at a wall bounces off at a certain angle and with a certain velocity, a dropped vase shatters and its fragments fly around, colliding with the scenery and each other. In addition to visual collision responses, such events also produce collision sounds. We have come to expect such behavior, and it can be surprising if the simulation of virtual objects does not correspond to these expectations. Thus, a physical simulation is commonly used to calculate the objects' behavior. However, many aspects of a VE are hard to simulate, especially given the real-time constraint of VR. Instead of a fully accurate simulation, it often suffices to achieve *plausibility*, where simulated objects do not behave exactly as in the real world, but similar enough to be indiscernible by humans [Barzel et al. 1996]. For example, rigid-body physics engines use several simplifications to allow a real-time simulation but still deliver a plausible behavior.

In addition to the simulation, the reproduction aspect of a VE is important, too. The biggest focus is usually put on visual rendering, and the quality of real-time computer graphics has advanced significantly over the last years. In combination with high-fidelity displays like a CAVE [Cruz-Neira et al. 1992], the visual aspect can already achieve a high degree of realism for virtual reproduction.

Audio is another modality that is frequently used in VR and represents an important aspect of VEs. Auditory interfaces allow rendering of spatialized sound sources, allowing users in the VE to locate the origin a the sound source. This can be achieved using different approaches, for example using Vector Based Amplitude Panning [Pulkki 1997], Ambisonics [Gerzon 1985], binaural rendering [Lentz et al. 2007], or Wave Field Synthesis [Berkhout et al. 1993]. The simulation of room acoustics is also advancing, so that the reproduced sound conveys characteristics of the environment, like reflection, diffraction, and reverberation [Vorländer 2008; Schröder 2011]. Auditory feedback can be of great importance in a VE. For example, sounds can improve the feeling of presence [Larsson et al. 2001; Larsson et al. 2002; Chueng et al. 2002; Bormann 2005] as well as the emotional reaction [Västfjäll 2003] of users in VR.

Additionally, sound can provide users with feedback, and thus can help with interaction tasks. The extent of the helpfulness depends on different aspects, like the task, the used interaction method, and the type of supporting sound. While some studies have already been performed to examine the usability of sound, results are still too ambiguous to draw a clear conclusion. Therefore, one goal of this thesis is to examine this aspect in more detail. Specifically, the usefulness of sound to substitute for haptic feedback will be studied. Haptic feedback is another important modality, but it is difficult to reproduce in VR due to the need for specialized feedback hardware. However, because haptic and sound perception share some common characteristics, it can be possible to use auditory feedback to replace certain types of haptic feedback. This will be examined by a user study where users perform a virtual drilling task, with different conditions providing combinations of haptic and auditory feedback.

Even when not used to support interaction tasks, sound is important for creating believable VEs. Especially for interactive environments with physically simulated objects, it would feel unrealistic if sounds were missing. For example, when throwing a stone at a wall or dropping a bottle on the floor, one expects to hear corresponding sounds. If these sounds are not reproduced by the VE, the immersion is reduced. However, generation of sounds for interactive applications is challenging. The common approach is to record or design *sound samples* for all auditory events upfront. This approach, also called Pulse Code Modulation (PCM), is well suited when a limited set of auditory events can occur, and if these are known beforehand. Especially for movies, but also for interactive applications with a limited amount of interactivity, pre-recorded samples can produce a realistic sound environment. Many VEs, however, are highly interactive, so that a high number of different sounds may be required. Especially when there are physically simulated objects in the scene, many diverse sounds can be produced from the contact between objects with different geometries, materials, and contact forces. Recording all potentially occurring sounds would require a high effort, and often impractical.

Another problem of pre-recorded sounds is that a single or very few sounds are recorded for one type of auditory event, like a stone falling on the floor or a wine glass being struck. In reality, however, the sound emitted by an object is interaction-dependent. It depends on aspects like impact positions, force distribution, and potentially prior collisions that already induced sounds. For example, when a wine glass is hit at the center of the bowl, it sounds different than when it is hit at the top or at the stem. Likewise, striking the center of a table produces a different sound than striking the border. If the same sample would be played all the time, it would sound repetitive and artificial.

To overcome these problems, one can use *interactive sound synthesis* to generate new sounds at run-time based on the interaction. There are different approaches for interactive sound synthesis. Among these, a particularly promising approach is *physically-based sound synthesis*, which uses a physical description of the sounding objects to calculate the resulting sounds. Contact sounds are mostly produced by vibrations of the object surfaces, which excite the surrounding air. These vibrations can be modeled by simulating the internal dynamics of the objects. Traditional physical simulation methods explicitly compute the system state at each time step. However, due to the high temporal resolution that is required for accurate sound synthesis, these approaches cannot be used for interactive applications. An approach that is suitable for real-time sound synthesis is *Modal Synthesis (MS)*. Here, an object is first analyzed by computing a *Modal Analysis (MA)* which determines the *vibration modes* of the object. These modes describe the characteristic frequencies and decay rates of the object vibrations. During run-time, these modes can then be excited to interactively generate sounds. For each vertex on the surface of an object, an acting force excites different modes with different amplitudes, and the contributions of all modes are accumulated to compute the resulting sound. Since the main computations are performed during the MA, which only has to performed once per object and is typically computed as a pre-processing step, the synthesis at run-time can be solved efficiently. MS has been used for interactive sound synthesis for some time now [van den Doel et al. 2001; O'Brien et al. 2002; Raghuvanshi et al. 2006; Ren et al. 2010; Chadwick et al. 2009]. However, several limitations remain. This thesis will present methods to further increase the applicability of MS for interactive VEs.

A MA is a compute-intensive process and is thus usually performed in a pre-processing step. This requires that all sound-producing objects are known beforehand. However, objects not always known beforehand due to the interactive nature of VR applications. Thus, objects with previously unknown geometry or material may occur at run-time. For example, objects that undergo geometric changes – e.g. by physically-based deformations or explicit interactions by the user, such as sculpting – would require interactive updates to the modal data. Similarly, some objects may be generated during run-time, like procedurally generated fragments of a shattering window or interactively modeled geometries, and thus cannot be analyzed in a pre-processing step. They require a new MA to be computed, which is challenging due to the high performance requirements and low available time. To allow a MA at run-time, this thesis presents different Levels-of-Detail (LoDs) that reduce the complexity of the object to allow a faster solution, at the cost of accuracy. Using a distributed setup, multiple LoDs can be computed, so that initial, coarse results are quickly available, and are successively replaced by more accurate ones. While losing some quality early-on, this allows using a MA at run-time, so that not all objects have to be pre-processed.

Another challenge of interactive MS is the run-time synthesis of the resulting sounds. For this, an object's modes are first excited by interaction forces. The state of the excited modes is then evaluated and accumulated for each sound sample. While the state of each mode is simple to compute, the high sound sampling rate (typically 44.1 kHz) as well as the potentially high number of modes require a high number of such evaluations. Thus, existing approaches can typically handle a modal sound synthesis for only a few sounding objects. The run-time synthesis is made even more challenging when forces are acting on an object. These forces excite the modal vibrations in an object. To achieve a high accuracy, these force excitations should be applied at a high temporal resolution. While one commonly applies the combined effect of forces at a single sample time – e.g. at the beginning of a sound block – this can lead to aliasing artifacts. Therefore, methods to handle modal sound synthesis with force application at a high sampling rate will be presented, also utilizing the high parallel processing capabilities of modern graphics processors.

This thesis is structured as follows. Chapter 2 provides fundamentals important for this work, including related work on sound synthesis, as well as physical and mathematical foundations of the MA process. In Chapter 3, a user study is presented that examines the applicability of synthesized sounds for substitution of haptic feedback for multi-modal interaction with VEs, using a task where participants remove material using a drill. Chapter 4 examines the computation of a MA at run-time to handle dynamically added or modified objects while adhering to real-time constraints. By benchmarking the run-time requirements, complexity thresholds are deducted, and several Levels-of-Detail (LoDs) are proposed to approximate objects to meet these thresholds while maintaining an acceptable quality. For the interactive computation of sound from modes, Chapter 5 presents and evaluates different approaches for the run-time synthesis, both with and without active forces. Final remarks and a conclusion are given in Chapter 6.

1.1 Contribution to Publications

Part of the research of this thesis has already been published in peer-reviewed articles. In the following, these publications will be listed. A short outline of the differences between the publications and the work in this thesis will be given. Additionally, the contribution of the thesis author and the co-authors for each publication will be explained.

The comparative user study in Chapter 3 has been published in [Rausch et al. 2012]. For this thesis, its statistical evaluation and interpretation have been extended, and further details about technical aspects are provided. Part of the work – most notably the design of the synthesized sounds, the execution of the user study, and parts of the statistical evaluation – were performed by Lukas Aspöck in the context of a term paper (Studienarbeit) [Aspöck 2012]. The term paper was supervised by the thesis author, who also developed the main study application, the haptic feedback, and performed parts of the statistical evaluation. The algorithm for static virtual coupling was provided by Thomas Knott. Sönke Pelzer, Michael Vorländer, and Torsten W. Kuhlen helped with the supervision of the term paper and provided guidance.

parts of the methods for a run-time MA described in Chapter 4 was published in [Rausch et al. 2015]. This thesis provides an extended performance analysis of the hull construction and the distributed architecture. Additionally, the construction algorithms for the *k*-Hull and ShrinkHull have been enhanced. The quality evaluation has been updated and extended. The algorithm development, implementation, and evaluation were performed by the author of this thesis. Bernd Hentschel and Torsten W. Kuhlen have provided helpful guidance in writing the publication.

Chapter 5 presents methods for a fast synthesis at runtime, which was published in [Rausch et al. 2014]. The development and evaluation of the methods włre performed by the thesis author. The co-authors provided helpful advice on writing the publication.

CHAPTER 2
FUNDAMENTALS

This chapter will present fundamentals and related work in the field of sound synthesis and specifically Modal Synthesis (MS) and Modal Analysis (MA).

2.1 Sound Synthesis

In computer-generated virtual environments, sound synthesis describes the process of computing a sound from an underlying model, in order to produce audio output that corresponds to an auditory event. Whenever an auditory event – for example, a collision between scene objects, footsteps of the user, etc. – occurs, a matching sound should be played. The most common approach to synthesize sounds in VEs is the use of sample playback, also referred to as PCM. Here, sounds are pre-recorded and stored, and are replayed when a linked auditory event occurs. This approach can produce very realistic sounds and is extensively used in cinema, where specialists called Foley artists produce the desired sounds using recordings and sample processing. However, this approach requires a recorded sample for each sound that can occur. For non-interactive scenarios, like movies or fully scripted sequences in VR or computer games, this poses few problems because all auditory events are known upfront. For interactive applications, however, the number of possible auditory events – and thus the number of sounds that would require recording – can be very high. Especially when objects in the scene are physically simulated, there can be a variety of different contacts that can occur – like collisions, sliding, or rolling – between all simulated objects. Thus, it would require a high amount of memory to store the pre-recorded sounds, as well as a significant recording effort. This may be acceptable for larger projects; however, designers of smaller applications may be deterred by the high effort and thus use no or only a few sounds.

Instead of pre-recording all sounds, one can use interactive sound synthesis to create the desired sounds at run-time. There exist a variety of synthesis approaches, each with different focus, advantages, and disadvantages. Thus, one should choose a sound synthesis method that suits the specific requirements. For example, speech is easy to use with pre-recorded samples, or using text-to-speech, but would be extremely difficult to achieve using physically-based synthesis, which is better-suited for simple physical objects. The most common sound synthesis methods will shortly be discussed here. For a more comprehensive overview, refer to [Tolonen et al. 1998].

Sample-based sound synthesis processes pre-recorded samples to generate new variations. Examples are pitch shifting [Lent 1989], where the frequency is adjusted, or grain-based re-sampling [Truax 1988], where segments of recorded sounds are re-arranged to create different, similar sounds that do not sound too repetitive. This can add variations to prerecorded samples and thus reduce the disadvantage of repeating sounds, but it still requires a recording step for each type of auditory event.

Parametric sound synthesis uses a pre-designed mathematical model to generate sounds from a set of input parameters. For example, the sound of a race car can be simulated based on the current gear, number of revolutions, etc. Parametric sound synthesis can be used to model interaction-dependent sounds with high quality, but are usually very specialized and have to be specifically designed for each use-case. Many specialized algorithms exist for a wide variety of different use-cases. A popular example is speech synthesis [Van Santen et al. 2013], which creates sounds from an input text. Additionally, specialized algorithms exist for a wide variety of musical instruments like string instruments [Karjalainen et al. 1993], cymbals [Bilbao 2008], or trombones [Smyth et al. 2011], which often combine aspects of parametric, sample-based, and physically-based sound synthesis.

Physically-based sound synthesis uses a description of the physical system to simulate its dynamic movement and deformation of objects and to calculate the pressure transferred to the surrounding medium, i.e. the produces sound waves. Physically-based synthesis is able to synthesize sound produced by interactive forces based on a physical description of the object. This way, little manual work has to be performed in order to allow an object to generate sound, and the results differ for different geometries and interactions. If the physical simulation is accurate, the results are also very realistic. Especially collision and contact sounds can be well synthesized by physical simulations. All in all, physical sound synthesis is a promising approach to synthesize sounds, especially for highly dynamic scenes with many different interacting objects.

For rigid objects, contact sounds are mostly produced by surface vibrations, like a chiming glass or a rock dropping on the floor. Such vibrations can be modeled by analyzing the internal body dynamics of the object that lead to deformations of the surface. The common approach to such a synthesis is to subdivide the domain and to solve it numerically, for example using Finite Element Methods [O'Brien et al. 2001], Finite Difference Methods [Bilbao 2009], or Digital Waveguide Meshes [Van Duyne et al. 1993]. However, accurately simulating sound is costly, because the high audible frequency ($20\,\text{Hz} - 22\,\text{kHz}$) and small wavelength ($15\,\text{mm} - 17\,\text{m}$ in air) require a very high temporal and spatial resolution of the simulation meshes. This makes the numerical simulation very time-consuming so that a real-time numerical analysis is only possible for simple cases like 2D membranes [Rabenstein et al. 2001; Hsu et al. 2013]. To achieve interactive physically-based synthesis of

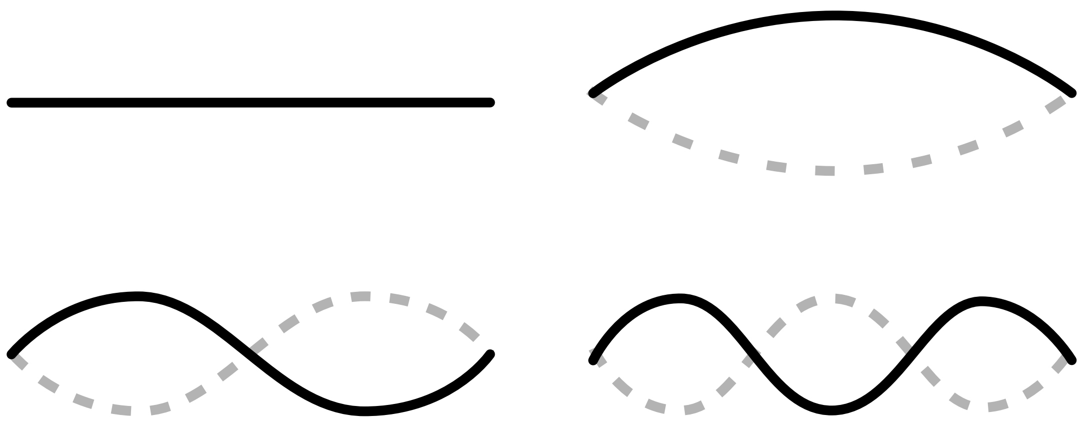


Figure 2.1: Different mode shapes of a guitar string.

vibration sounds, one can use *Modal Synthesis*, which uses the characteristic *vibration modes* of an object to model small-scale deformations that produce sound. MS will be discussed in detail in Section 2.2.

While MS can model the long-lasting, harmonic ringing component of collision sounds that are produced by vibrations, other effects occur, too. The most prominent one is acceleration noise [Richards et al. 1979], which is caused by air being compressed when an object accelerates, and typically produces short, transient and non-harmonic sounds. Especially for small objects, like the clicking sound of marbles, it may even be the dominant effect, because vibrations mainly produce high-frequency sounds for small, hard objects. Chadwick et al. presented a system to allows adding acceleration noise to collisions by pre-computing the produced acceleration noise for different short impulse changes [Chadwick et al. 2012a; Chadwick et al. 2012b]. However, the proposed algorithm is only able to interactively synthesize sounds for simple scenes with few objects and requires a long pre-processing time.

Apart from collisions, there are other sources of sounds that can be synthesized physically. For some of these, specialized synthesis methods have been developed. For example, approaches exist to synthesize the sound of fire [Chadwick et al. 2011], liquids and water bubbles [van den Doel 2005; Moss et al. 2010], aerodynamic effects [Dobashi et al. 2003], or fracturing of rigid bodies [Zheng et al. 2010].

2.2 Modal Sound Synthesis

One approach to model the sound produced by small-scale deformations of rigid physical shapes is *Modal Synthesis* [Adrien 1991]. To describe the deformation, it utilized *modes*, which are characteristic harmonic vibrations of an object. Each solid object has a set of characteristic modes, which are characteristic vibrations with a specific frequency,

To allow a synthesis of sound from modes, these modes have to be determined first. This process is called Modal Analysis and can be performed in various ways. A MA determines the *modes* of an object, each with its own vibration *frequency*, *decay*, and *mode shape*, which

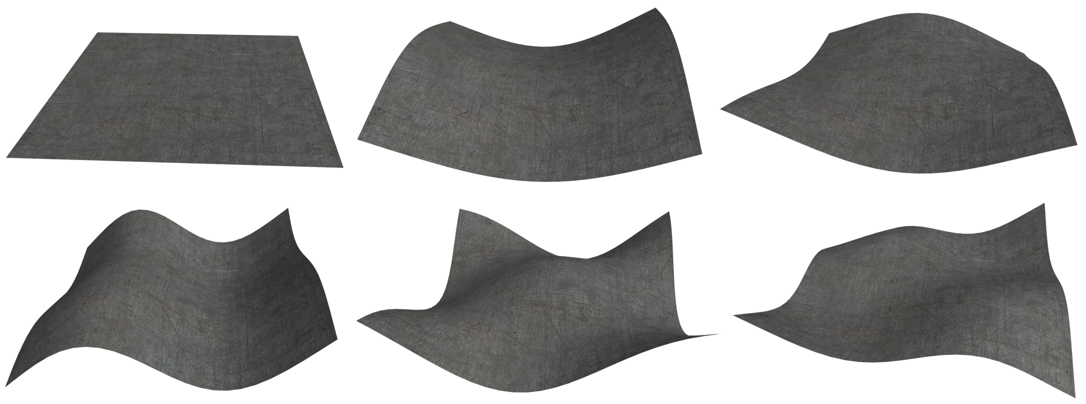


Figure 2.2: The first modes of a thin rectangular plate.

is the deformation induced by this mode. For example, a guitar string with fixed ends has a base mode with certain base frequency, depending on the material and thickness of the string as well as the tensile force acting on the ends, as well as modes representing the overtones. Some of the mode shapes of a guitar string are shown in Figure 2.1. Another example shows different modes of a thin rectangular plate (see Figure 2.2).

When assuming that the deformations of an object induced by the modal vibration are small, the modes of an object can be considered as independent. This allows an efficient use of physically-based modeling of vibrations. Forces acting on an object excite its modes, which are then simulated individually. The deformation of the object can then be computed as the *superposition* of the deformation of its different modes.

Given the physical description of an object, a MA can be performed once to determine the modes, which are then used during run-time to generate sounds. This way, it is possible to separate the MA from the evaluation of the modal vibrations at run-time. This makes the approach well-suited for sound synthesis: the computationally expensive MA can be computed once up-front so that at run-time, the MS can efficiently compute the vibration sound at audio rates.

The process of determining the modes of an object is called *Modal Analysis*. In structural engineering, MA has a long history and is used for the analysis of the stability of buildings, machinery or components. Initially, a modal analysis was typically performed by examining an existing object by applying forces to the object and measuring the response [Hearn et al. 1991; Allemang et al. 1993; Ewins 1995]. With increasing computation capacities of computers, computer simulations became fast enough to allow computing a modal analysis from a numeric description (see Section 2.2.1).

Apart from structural engineering, MA has also been used in other areas. Due to the linear independence of modes, it can be used efficiently for interactive applications, and thus have been utilized to model deformations in interactive computer graphics [Pentland et al. 1989; James et al. 2002; Barbič et al. 2005], or to compute the internal stress to dynamically fracture objects [Glondu et al. 2013]. Recently, MA has been used for interactive sound synthesis and gained a lot of attention due to its capability to model deformations at high sample rates [van den Doel et al. 2001; O'Brien et al. 2002; James et al. 2006; Raghuvanshi et al. 2006; Maxwell et al. 2007; Bonneel et al. 2008; Chadwick et al. 2009; Ren et al. 2010; Picard et al. 2010; Zheng et al. 2011].

There are different approaches to determine the set of characteristic modes of an object: experimental, analytic and numerical Modal Analyses. For an experimental analysis, one applies test forces to an existing object and measures its response [Allemang et al. 1993; van den Doel et al. 1998; Pai et al. 2001; Karjalainen et al. 2002; Lloyd et al. 2011]. For sound synthesis, however, this has several drawbacks. First of all, suitable measurement facilities must be available, and the objects have to actually exist, which often is not the case for objects used in VEs. Additionally, a MA by example creates a high measurement effort, especially because the vibration should be measured for different excitation points. Thus, the effort of an experimental MA often exceeds that of pre-recording fitting audio samples, negating one of its main advantages.

A better alternative is to compute the modes from the physical description of an object. In some cases, modes can be computed analytically [Rossing et al. 1995; van den Doel et al. 1998]. However, this is only possible for few simple geometries, like plates, bars, or membranes. With the advent of computer-aided physical modeling, it has become possible to compute the dynamic behavior of objects with arbitrary geometry and material using numerical mesh-based approaches. O'Brien et al. were first to use a numerical MA for sound synthesis [O'Brien et al. 2002], and the approach was used and extended thereafter [Raghuvanshi et al. 2006; Chadwick et al. 2009; Zheng et al. 2011; Ren et al. 2010; Ren et al. 2013a; Ren et al. 2013b; Picard et al. 2010]. In this thesis, the focus will be on sound synthesis using modes computed from a numerical MA.

2.2.1 Linear Modal Analysis

When determining the modes of a physical system to enable a MS, one typically uses a *Linear Modal Analysis* [Shabana 1996], i.e. a MA of a linear physical system. Linear systems are commonly used for interactive deformable simulation, e.g. using Spring-Mass Systems of Finite Element Methods. These are typically used for iterative simulations but can be utilized to perform a linear MA to calculate the modes. In this work, the focus is on sound synthesis using linear MA. While there are experimental and analytical approaches to performing a MA, for simplicity the term *Modal Analysis* will hereafter be used to describe the linear Modal Analysis from a physical mesh, unless otherwise specified.

A linear MA computes modes from a linear elastodynamics system describing the physical reaction of solid objects, based on the geometry – i.e. a physical mesh – and material properties of the objects. For a mesh-based MA, an object is represented by a physical mesh with n vertices, whose positions are encoded in the vertex vector $\mathbf{p} = \left(p_{1,x}, p_{1,y}, p_{1,z}, \ldots, p_{n,x}, p_{n,y}, p_{n,z}\right)^T \in \mathrm{R}^{3n}$. To model the simulation mesh, one usually regards the *displacement vector* $\mathbf{d}(t) = \mathbf{p}(t) - \mathbf{p}(t_0)$ describing the deformation of the mesh by the vertices' displacement relative to their initial, undeformed position. The dynamics system is modeled as a function of $\mathbf{d}$ and its time derivatives $\dot{\mathbf{d}}$ and $\ddot{\mathbf{d}}$.

$$k(\mathbf{d}) + d\left(\dot{\mathbf{d}}, \ddot{\mathbf{d}}\right) + m\left(\ddot{\mathbf{d}}\right) = \mathbf{f}_{ext} \tag{2.1}$$

Here, $\mathbf{f}_{ext}$ is the vector of external forces, and k, d, and m are functions depending on the displacement and its derivatives. k models the internal stiffness, m describes mass and inertia, and d models the internal damping. In general, the three components k, m, and d can have an arbitrary form, and the dynamics system forms a set of non-linear differential equations that could be solved numerically. This, however, would be computationally expensive.

Instead, one commonly uses a *linear* variant of this system. For this, it is assumed that there are only small deformations, i.e. $\vec{d}$ is small compared to the geometry features. This allows reformulating Equation 2.1 using matrices instead of functions:

$$\mathbf{K}\cdot\mathbf{d}+\mathbf{C}\cdot\dot{\mathbf{d}}+\mathbf{M}\cdot\ddot{\mathbf{d}}=\mathbf{f}_{ext} \tag{2.2}$$

Here, $\mathbf{K}$ is the *stiffness matrix*, $\mathbf{C}$ is the *damping matrix*, and $\mathbf{M}$ is the *mass matrix*, each of size $3n \times 3n$. $\mathbf{d}$ is the vector of vertex displacements, and $\mathbf{f}_{ext}$ is the vector of external forces. There are different ways to derive the system matrices for a geometry [Nealen et al. 2006]. The most common ones are the Spring-Mass System [Teschner et al. 2004] and the Finite Element Method [Bathe 2006].

Equation 2.2 is frequently used in interactive computer-based physics simulation and is commonly solved iteratively at run-time. Such iterative, explicitly time-stepped solutions can be computed for sound synthesis, too [O'Brien et al. 2001], but due to the high temporal resolution of sound, the cost is prohibitive for interactive applications. Instead, the linear MA was proposed for sound synthesis [O'Brien et al. 2002] and has been gaining interest in recent years. The goal of the linear MA is to *decouple* the rows of the linear dynamics system by diagonalizing it so that each row only depends on a single entry of each matrix. If this is achieved, each row can be interpreted as an independent, one-dimensional differential equation, which represents one of the system's modes.

One step towards diagonalizing the dynamics system is the use of *Rayleigh damping* [Strutt et al. 1945] to model the damping matrix as a combination of the mass and stiffness matrices:

$$\mathbf{C}=\alpha\cdot\mathbf{K}+\beta\cdot\mathbf{M} \tag{2.3}$$

Here, α is the *stiffness damping* parameter, and β is the *mass damping* parameter. This representation of damping is not physically correct, and thus α and β do not correspond to actual physical parameters. However, Rayleigh damping is a frequently used approximation, and it provides plausible results. Ren et al. [Ren et al. 2013b] showed that Rayleigh damping is well suited to generate modal sound and that material parameters designed for one geometry can be transferred to other geometries while maintaining the audible properties of the material. The advantage of using Rayleigh damping is that the dynamics system now only depends on two matrices, $\mathbf{M}$ and $\mathbf{K}$:

$$\mathbf{K}\cdot\left(\mathbf{d}+\alpha\cdot\dot{\mathbf{d}}\right)+\mathbf{M}\cdot\left(\ddot{\mathbf{d}}+\beta\cdot\dot{\mathbf{d}}\right)=\mathbf{f}_{ext} \tag{2.4}$$

To linearize this dynamics system, the matrices in the system have to be diagonalized. This can be achieved by solving a generalized eigenvalue problem. Alternatively, one can directly use the *Cholesky Decomposition* of the mass matrix, which computes a lower triangular matrix

$\mathbf{L}$ such that $\mathbf{M} = \mathbf{L} \cdot \mathbf{L}^T$. Pre-multiplying Equation 2.4 with $\mathbf{L}^{-1}$ and substituting $\mathbf{q} = \mathbf{L}^T \cdot \mathbf{d}$ yields:

$$\mathbf{L}^{-1} \cdot \mathbf{K} \cdot \mathbf{L}^{-T} \cdot (\mathbf{q} + \alpha \cdot \dot{\mathbf{q}}) + (\ddot{\mathbf{q}} + \beta \cdot \dot{\mathbf{q}}) = \mathbf{L}^{-1} \cdot \mathbf{f}_{ext} \tag{2.5}$$

This formulation only depends on a single matrix, $\mathbf{Q} = \mathbf{L}^{-1} \cdot \mathbf{K} \cdot \mathbf{L}^{-T}$. To diagonalize the dynamics system, one now computes the standard eigenvalue problem $\mathbf{Q} \cdot \mathbf{\Lambda} = \mathbf{\Lambda} \cdot \mathbf{V}$, where $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues of $\mathbf{Q}$, and $\mathbf{V}$ is a matrix whose columns are the corresponding eigenvectors. Since $\mathbf{Q}$ is a real symmetric matrix, $\mathbf{V}$ is orthogonal and can therefore be inverted by transposition: $\mathbf{V}^{-1} = \mathbf{V}^T$. Using $\mathbf{Q} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^T$, and substituting $\mathbf{r} = \mathbf{V}^T \cdot \mathbf{q} = \mathbf{V}^T \cdot \mathbf{L}^T \cdot \mathbf{d}$ and $\mathbf{g} = \mathbf{V}^T \cdot \mathbf{L}^{-1} \cdot \mathbf{f}_{ext}$ allows formulation the dynamics system as follows:

$$\mathbf{\Lambda} \cdot (\mathbf{r} + \alpha \cdot \dot{\mathbf{r}}) + (\ddot{\mathbf{r}} + \beta \cdot \dot{\mathbf{r}}) = \mathbf{g} \tag{2.6}$$

Equation 2.6 has now been diagonalized because the only occurring matrix $\mathbf{\Lambda}$ is a diagonal matrix. Thus, the dynamics system can be interpreted as $3n$ individual ordinary differential equations:

$$\lambda_i \cdot r_i + (\lambda_i \cdot \alpha + \beta) \cdot \dot{r}_i + \ddot{r}_i = g_i \tag{2.7}$$

Each of these ordinary differential equations represents a *damped harmonic oscillator* and can be solved analytically (see Equation 2.8). To compute the mode state, the mode parameters $S_i = (\omega_i, \gamma_i, \mathbf{s}_i)$ are used, specifying an angular frequency $\omega = \frac{1}{2}\sqrt{4 \cdot \lambda_i - (\alpha \cdot \lambda_i + \beta)^2}$ and a damping coefficient $\gamma = -\frac{1}{2}(\alpha \cdot \lambda_i + \beta)$. The Eigenvectors $\mathbf{v}_i$ in the matrix $\mathbf{V}$ represent the *mode shapes* $\mathbf{s}_i$, i.e. the deformation of each vertex when a certain mode is excited, and are normalized such that each row of $\mathbf{V}$ is normalized, i.e. $|\mathbf{v}_i| = 1$ for all eigenvectors i.

Using this method, the linear MA computes $3n$ linearly independent modes and their corresponding parameters, angular frequency and decay, as well as their mode shapes. When forces act on an object, different modes are excited. After excitation, each mode harmonically vibrates with its angular frequency and loses energy according to its decay (for more details, see Chapter 5). Each individual mode leads to a deformation, and since all modes are mutually independent, the overall deformation can be computed by superposition of all modes' deformations.

2.2.2 Modal Synthesis

The MA is performed once to compute the modal data and yields a set of modes $S_i(t) = (\omega_i, \gamma_i)$. During run-time, these modes are used to calculate the mode state. At time t, the current mode displacement can be calculated as:

$$r_i(t) = \underline{a}_i \cdot e^{(\gamma_i + i \cdot \omega_i) \cdot t} + \underline{a}_i^* \cdot e^{(\gamma_i - i \cdot \omega_i) \cdot t} \tag{2.8}$$

Here, $\underline{x}^*$ denotes the complex conjugate of a complex value $\underline{x}$. $\underline{a}_i$ is the complex-valued amplitude of the mode, and is determined by the impact forces that act on the object. At run-time, any external force $\mathbf{f}_{ext}$ is first transformed into the mode space $\mathbf{g} = \mathbf{V}^{-T} \cdot \mathbf{L}^{-1} \cdot \mathbf{f}_{ext}$, following the substitution that was performed during the MA process. The matrix $\mathbf{V}^{-T} \cdot \mathbf{L}^{-1}$ used to transform the force to mode space is called the *gain matrix*. Due to this transformation of the external forces into the mode space, forces acting on different vertices of the mesh result in a different excitation of modes, which models the effect that striking an object at different locations can produce different sounds. The transformed force $\mathbf{g}$ is then used to update the mode amplitudes $\underline{a}_i$ [Shabana 1996; O'Brien et al. 2002; Raghuvanshi et al. 2006], leading to an excitation of the system (see Section 5.3 for details).

To calculate the sound sample of a sounding modal object at time step t, the contributions of all modes are computed independently and then summed up. Since the MA only has to be computed once and is often performed as a pre-processing step, the run-time synthesis consists of two simple steps: mode excitation from external forces, and evaluation of the mode contribution for each sound sample. However, sound synthesis requires a high sampling rate, typically 44100 Hz. Thus, with a straightforward implementation, only a few thousand modes can be calculated in real-time on a CPU [Raghuvanshi et al. 2006]. Since even a single sounding object can have thousands of modes, a naive implementation could exceed the computation capabilities. Chapter 5 will approach this problem and present methods to speed up the run-time synthesis, especially when forces are acting on an object.

2.2.3 Acoustic Transfer

Section 2.2.2 explained how to compute the mode displacement at different times. However, the mode state only represents the displacement of the surface vertices when the object is vibrating and does not directly translate to the emitted sound. For this, *acoustic transfer* [Cremer et al. 1985] is required to model the coupling between surface movement and the surrounding medium, which results in sound waves. A simple approach to approximate the acoustic transfer is to simply examine the displacement volume velocity of a mode during each time step [O'Brien et al. 2002; Raghuvanshi et al. 2006], and to sum up the volume displacement of all mode contributions. This way, the whole object is simulated as a single point sound source – i.e. a monopole – with uniform sound radiation. With this approximation, however, directivity characteristics and near-field effects due to the interference of sound waves are lost.

To handle these effects when calculating the sound radiation from the object, a numeric simulation of the surface and the surrounding medium, e.g. using Boundary Element Methods [Ciskowski et al. 1991], can be used. However, in addition to long computation times to compute the transfer functions, the run-time is computationally expensive, too, because the influences of all boundary patches have to be evaluated for each mode. This makes a use for real-time sound rendering difficult.

James et al. [James et al. 2006] developed an approximation using pre-computed acoustic transfer. First, a numeric solution is computed using Boundary Element Methods for each mode. Then, several dipole sources are placed to approximate the sound field. While this introduces an approximation error, it allows a high-quality reproduction of acoustic transfer in real-time.

In this thesis, the simpler approach of approximating of the object as a spherical radiator will be used, neglecting near-field effects. The sounding object it is regarded as a monopole source, greatly simplifying computation and run-time computations [Cremer et al. 1985]. Using this method, the acoustic transfer can be easily computed from the vibration frequency and the mode shapes. While this approach does not provide any near-field effects, it is a decent approximation for distance sources. Its ease-of-use makes it a common choice for modal sound synthesis [O'Brien et al. 2002; James et al. 2006].

2.2.4 Contact Modeling

For VR application, the physical behavior of objects in the scenery is often calculated by rigid body dynamics engines, which solve contacts using interactive constraints and usually run at around 100 Hz. To detect collision between objects, these physics engines usually use coarse collision geometries and few isolated contact points to achieve real-time performance. This simplification allows a plausible simulation of the visual aspect of the physical behavior and can handle many objects simultaneously.

For sound synthesis, however, the collision information from rigid body engines is often too coarse. While transient collisions can be sufficiently displayed, lasting contacts are more problematic. These contact types include one object resting on another, as well as rolling, sliding or scraping motion, where many small-amplitude forces act on the object in quick succession. When contact forces would only be calculated with the visual simulation rate (typically $50 - 100$ Hz, this would lead to a loss of detail from interpolation of the contact forces. Furthermore, many contact effects – especially those depending on the roughness of the surfaces – are usually not represented by the collision or visual geometry.

Some approaches have been developed to create contact forces with higher spatial and temporal resolution suitable for modal sound synthesis. One solution is to simply calculate the dynamics and collision resolution at audio rates [Zheng et al. 2011]. However, this is currently too computationally expensive for interactive applications. If only small regions are simulated with a simplified model, it is possible to gain higher precision, although this still loses details [Avanzini et al. 2001; Zambon 2012] and requires extensive computations. A simpler, faster approach is used by FoleyAutomatic [van den Doel et al. 2001], which generates procedural noise based on the surface roughness to simulate forces due to micro-collisions. In addition to this random noise, additional geometry information from bump maps [Ren et al. 2010] or specially created textures [Picard et al. 2008] can be used to create an intermediate level of resolution.

2.3 Applicability and Limitations of Modal Synthesis

Modal Synthesis (MS) is a promising approach for real-time sound synthesis, because it uses a one-time computation-heavy MA to compute the modal data, which can then be used to efficiently synthesize sounds at run-time in response to external forces. However, this approach still suffers from some problems, which will be discussed here.

2.3.1 Linear Approximation

To compute the modal data, a *linear* MA is performed (see Section 2.2.1). For this, the physics system is formulated in a linear manner, which is a commonly used approximation for physical computation, e.g. using Finite Element Method (FEM). A linear formulation is only an approximation since the matrices are built based on the undeformed geometry. Thus, such a formulation is only valid if the deformation of an object is very small.

This approximation is necessary to allow a diagonalization of the dynamics system, and thus the computation of independent modes. Errors occur when the modal vibrations are so strong that they lead to a significant deformation of the object. This is often the case for thin, hard geometries, such as metal plates. Since this type of shell is often used for musical instruments, this limits the applicability of MS for these objects. When the deformation is not negligible, non-linear effects occur. These include shifts of the mode frequencies (pitch glides), as well as cross-modal effects where the deformation from one mode excites other modes. Such non-linear effects are especially prevalent in some percussion instruments, like cymbals and gongs [Chaigne et al. 2005].

Since these effects are lost when using a linear MS, the sound of such instruments cannot be reproduced accurately or requires specialized handling of these non-linear effects. To address the cross-modal coupling of modes, Chadwick et al. evaluate and apply the inter-modal excitations at run-time [Chadwick et al. 2009]. This improves the realism of the modal sound reproduction for thin objects. However, this approach adds a significant cost to the run-time synthesis and makes it impractical for real-time applications.

2.3.2 Rigid Bodies

Another implication of the linear nature of the MA is that it restricts the possible objects to rigid bodies. The assumption of a linear MA is that the analyzed object only undergoes slight deformations. However, this is only true for objects with a high-enough stiffness. When soft objects, like cloth or rubber, are affected by contact forces, they can deform significantly. Since this leads to strong changes in the physical mesh, the modes calculated for the original mesh no longer match. Likewise, plastic information, like bending of a steel bar, would change the geometry and thus require a new modal analysis.

Another problem is the interaction between the object and the surrounding fluid medium – typically air. Some of these interactions, like acoustic transfer to surrounding air (see Section 2.2.3), or sound due to air compression (see Section 2.3.3) can be modeled using specialized methods. However, some objects, especially many musical instruments, utilize masses of air as resonant cavities that can be very characteristic for their sounds. Since the MA only regards the solid components of an object, such characteristics are missing in the reproduced sound.

2.3.3 Vibrations

The modes of an object represent its characteristic vibrations and are thus well-suited to model the sound produced by such vibrations – which are sometimes called *ringing sounds*.

However, even for contact sounds, there are other effects that contribute to the sound. Most notably, aerodynamic effects can play an important role and occur when objects move or accelerate, or wind is moving along cavities. This includes effects such as clapping hands, the characteristic sound of a spinning coin, or wind instruments such as flutes. Such aerodynamic and acceleration sounds cannot be handled by modal sound synthesis, but may be handled using specialized algorithms [Dobashi et al. 2003; Chadwick et al. 2012b].

Another type of sounds that are non-modal are those that occur during plastic deformations or tearing/fracturing of objects. For such cases, specialized algorithms can be used, et al. to model fracturing of rigid bodies [Zheng et al. 2010].

One often-used approach is to use a residual sound sample extracted from recordings of sounds produced by real objects with the desired material. When a new, virtual object with this material is analyzed, the modal sound it produces is enriched by playing adding the residual sound that contains non-modal contributions [Ren et al. 2013b].

2.3.4 Memory Consumption

One current limitation that poses a problem for interactive applications is the high memory consumption of the modal data required for run-time synthesis. The gain matrix has to be stored for run-time usage, which maps forces acting on surface vertices of the physical mesh to excitation of modes. This matrix has a size of $3n \cdot m$ for an object with n surface vertices and m modes. Without mode truncation (see Section 5.2.2), the number of modes equals the degrees of freedom in the mesh, i.e. $3v$ for geometries with v physical mesh vertices. With truncation, the number of modes can be reduced, but the result may still contain several hundreds or thousands of modes.

Depending on the complexity of the mesh, this is a significant amount of memory. For example, a mesh with 4000 surface vertices and 2000 modes would require 144 MiB (assuming 4 bytes per float). Thus, a high number of modal objects in a scene can exceed the memory resources of a computer.

The easiest way to reduce the memory requirement is the reduction of the number of modes, by pruning inaudible modes and optionally by merging similar modes (see Chapter 5). Reducing the geometric resolution also reduces the memory requirements, but also decreases the accuracy of the synthesized sounds.

A different approach for memory reduction has been proposed by Langlois et al. [Langlois et al. 2014]. By using approximations using Moving Least Squares, they compress the gain matrix. By estimating the psycho-acoustical importance of modes, less important once are approximated with a higher error tolerance. By furthermore utilizing symmetry in the mode shapes, they can reduce the memory footprint even further. Using these approaches, they can reduce the memory requirements by at least an order of magnitude, and up to a factor of 1000 for certain objects.

2.3.5 Damping Model

When using modal sound synthesis, it is most common to use the Rayleigh damping model [Strutt et al. 1945]. While this damping model is used frequently, it has some problems.

Firstly, the damping parameters, stiffness damping α, and mass damping β are no physically measurable values.

Due to the stiffness damping, a mode's decay depends linearly on its Eigenvalue (see Section 2.2.1), and thus rises for higher frequencies. While this is most often also true for real objects, there are some exceptions, especially when objects have inhomogeneous or anisotropic materials.

Another problem of the damping model comes from the decoupling of the modes, so that they are linearly independent. After this step, each mode is modeled as an exponentially damped harmonic oscillator, $r_i(t) = \underline{a}_i \cdot e^{\gamma_i \cdot t} \cdot sin(\omega_i \cdot t)$. Thus, their amplitude always decays exponentially, while real vibration modes often show more complex decay behavior. However, while it is possible to extract such a complex decay when extraction modes from sound recordings [Lloyd et al. 2011], it is an inherent element of the linear MA, and cannot be reasonably circumvented when computing a MA from a physical description of an object.

While there are some problems with the used damping method, the results are still plausible. A recent study has shown that the material type can be determined from synthesized sounds with similar precision as from recorded sounds, and that the damping parameters that have been matched to a specific object can be transferred to other objects of the same material [Ren et al. 2013a]. Therefore, the Rayleigh model appears to be suitable, although more complex damping matrices may be examined in the future.

2.3.6 Geometric Approximation

To compute a MA, a description of the dynamics system is required. These are typically modeled using geometric approaches, e.g. FEM. However, the design of the geometry – especially for FEM – typically has to be tweaked to match the requirements of the simulation. Apart from the shape and material properties, factors like sampling resolution or cell type can have a significant impact on the results.

For many application, like computational fluid dynamics, the design of the mesh is a dedicated work step performed by specialists. However, in the context of VR and other highly interactive environments, one major advantage of using modal sound synthesis is the reduced effort of adding sounds to objects. Therefore, the geometry preparation should be as easy as possible, and work mostly automatic. Since this conflicts with the dependency on mesh quality, non-optimal sounds may be the result. To reduce this effect, especially for geometries that have been reduced very strongly, Chapter 4 will analyze this effect and present an approach to compensate it.

2.3.7 Modal Analysis Performance

For interactive applications, MS is a well-suited approach for sound synthesis, especially because once the modal data is available, the run-time synthesis can be computed efficiently. The modal data is typically computed using a MA, which is a computationally expensive operation. Therefore, it is usually performed as a pre-processing step.

While the MA step allows the efficient run-time synthesis, its long time requirement can be a problem. While most objects can typically be pre-processed, there are some objects in a VE whose geometry or material are not known upfront. These can include procedurally or manually created objects, or modifications of existing objects performed by the user. Since the modal data of these objects cannot be computed up-front, an interactive MA would be necessary to allow modal sounds. Methods to allow such a run-time MA are presented in Chapter 4.

CHAPTER 3

HAPTIC FEEDBACK SUBSTITUTION USING SYNTHESIZED SOUND

One inherent aspect of VR is multi-modal interaction, where in- and output utilizes different modalities, with a focus on using different senses. The visual sense is often the dominant and most important one, and with current advances in computer graphics and display systems, a high-quality visual reproduction is commonly the basis of VR applications. However, other senses should not be neglected, as different modalities can enhance the quality of feedback and the overall VR experience, as well as its believability.

Apart from visual feedback, auditory and haptic feedback are other important modalities that can be utilized in VR. While other senses – like olfactory, gustatory, or the perception of temperature, etc. – can also be helpful, these are rarely simulated in VR applications because of their very specialized use and the difficulty to reproduce them sufficiently well in a VE.

Sound is commonly utilized in VR and can provide multiple benefits for a VE (see Chapter 2). Haptic feedback is another important modality that can be used in VEs. It provides a sense of touch by simulating forces or surface textures and conveys information about contacts, material properties, etc. For some tasks – especially those where users use their hands to perform precise work, like using surgical or precision tools – haptics is a very important feedback channel. Due to the high precision and speed of the haptic perception, it can even be more important than visual feedback for some tasks (see [Coles et al. 2011] for a survey).

Due to these reasons, the use of haptics in VR has been a topic of research for a long time. It is often used for medical simulators, like robot-assisted surgery [Meijden et al. 2009] or palpation and needle insertion [Ullrich et al. 2011]. However, haptic feedback is rather difficult to reproduce in a VE due to the need to use specialized haptic feedback devices. These must reproduce forces at a high rate – typically around 1 kHz – and with sufficient strength.

Current devices, like the Sensable PHANTOM series, have limited workspace area and displayable forces. Furthermore, due to their bulk, haptic feedback devices are commonly mounted at fixed locations. This makes their use for VR difficult, especially in scenarios where the user can freely move around. Another problem of haptics is the high performance

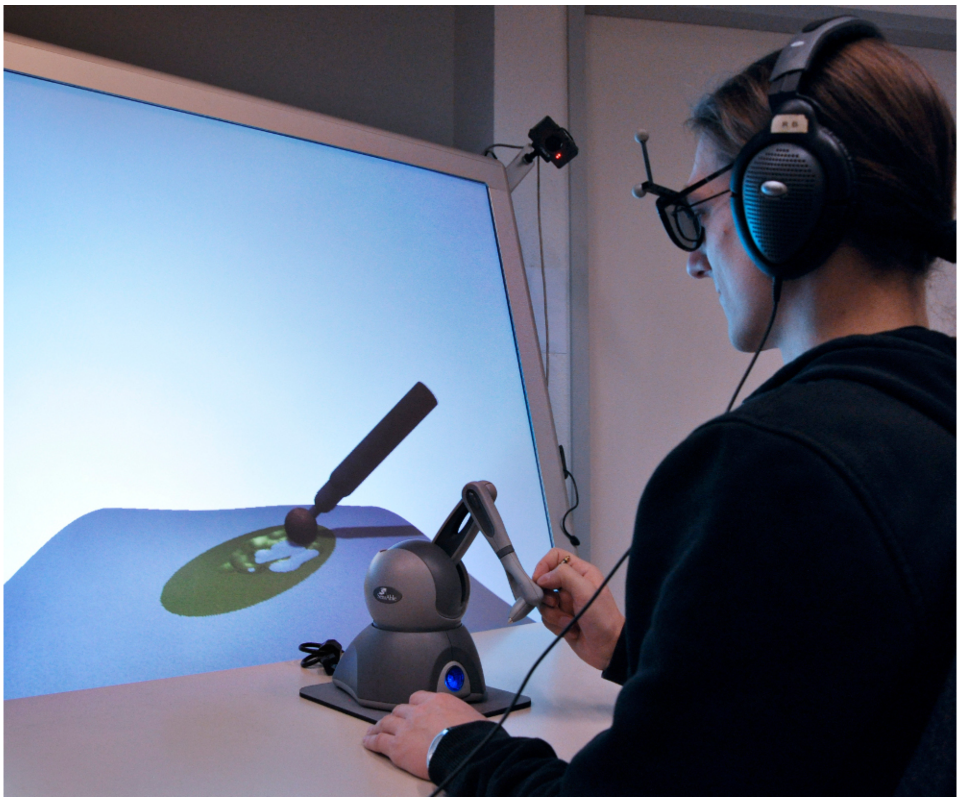


Figure 3.1: Setup of the study. The participant is seated in front of a stereoscopic scree, and uses a Senseable PHANTOM Omni to control a virtual drill in order to remove material from a surface.

requirement for the haptic simulation. For contact forces, this usually requires a collision detection and resolution step at the simulation frequency of around 1 kHz in order to produce forces that are perceived as smooth by the haptic sense. This high simulation frequency poses a difficult challenge for the contact simulation.

For these reasons, haptic feedback is often not available in VR scenarios, which can deteriorate the interaction performance. In these cases, one can try using other modalities to compensate for lacking haptics, by using sensory substitution and cross-modal effects. For example, in one study participants touched a real object hidden behind a screen. When changing the virtual image of the object, they reported feeling different haptic properties even though these properties were only present in the visual representation [Brogni et al. 2011].

However, designing proper substitute feedback using only visuals is often not sufficient. Especially when feedback is used to address the high-frequency perception of the haptic sense, visual-only representations cannot reliably replace this modality. As a more promising alternative, auditory feedback may be used to substitute the haptic feedback by synthesizing corresponding sounds. Sound is potentially better suited for this task, since like haptics it is processed at a high temporal resolution by human senses.

In this chapter, a user study will be presented which evaluates the possibility to substitute certain types of haptic feedback by sound. In the study, the chosen task utilizes a virtual mechanical precision tool, because such a task is commonly simulated in VR and depends heavily on haptic feedback. Specifically, the participants perform a virtual drilling task (see Figure 3.1) using a spherical drill. In addition to the visual feedback, three different feedback types are provided: haptic surface contact forces, haptic vibrations, and vibration sounds. The separation of the two different haptic feedback types allows an examination of which types of haptic feedback may be substituted by sound.

This chapter is structured as follows. First, related work on the topic of sensory substitution – especially regarding haptics and sound – is presented in Section 3.1. Then, the design and procedure of the study will be detailed in Section 3.2. Section 3.3 provides details on the technical aspects of the study and its interaction design. The results are then presented in Section 3.4, followed by a discussion and conclusions in Section 3.5.

3.1 Related Work

The effect of sensory substitution has previously been investigated in multiple studies, with varying goals and results. Some of these studies also compared interaction tasks that utilize both haptic and auditory feedback. However, most the studies use simple tasks like target acquisition, or utilize artificial feedback that does not directly correspond to realistic sound or haptics.

In an early work, Richard et al. performed a study examining the task of grasping and moving a virtual ball [Richard et al. 1994]. The grasping force was displayed using either visual feedback by utilizing multiple LEDs, auditory feedback by modifying the pitch of a sound, or by haptic feedback using a glove with mechanical force actuators. The study's results were inconclusive and depended on the hardness of the virtual ball: for the hard ball, haptic feedback provided the best results, while auditory feedback was best for a soft ball.

A study by Kim et al. examined pointing at 3D grid points, and found that using artificial acoustic or haptic signals as depth cues enhanced the pointing precision, especially along the depth direction [Kim et al. 2007]. The results showed a similar improvement when using haptic feedback, auditory feedback, or both.

Menelas et al. examined the use of abstract, artificial haptic and sound feedback for target acquisition [Ménélas et al. 2010]. In this study, haptic feedback provided much better results than sound, and combining both auditory and haptic feedback led to worse results than using only haptic feedback.

In a study performed by Lécuyer et al., a ball had to be guided through different holes, with different kinds of abstract feedback provided by alarm sounds, vibrations, and supporting forces [Lécuyer et al. 2002]. However, they found no effect of haptic or auditory feedback on the error rate, while the task completion time was best without any feedback.

Diaz et al. examined the users' performance of guiding a ball through a simple maze [Díaz et al. 2006]. When collisions were reported by haptic forces, they found that penetrations were lower than if feedback was provided visually or by playing contact sounds.

In another experiment, it was shown that either visual or auditory feedback can partially substitute one of the feedback axes of a 3-DOF torque haptic wrist device [Martín et al. 2008]. When combining haptics and audio, it was shown that auditory feedback can enhance haptic realism, e.g. by making a contact appear stiffer [Avanzini et al. 2006]. A study of the virtual assembly of clockworks showed that haptic feedback was more beneficial than audio or visuals [Petzold et al. 2004]. Edwards et al. also used an assembly task, but found that audio had little influence and that depending on gender, haptic feedback could have no impact (females) or be detrimental (males) [Edwards et al. 2004]. Another mechanical assembly task was used in a study by Zhang et. al., showing that both visual and auditory feedback improves task performance, and combining both further increases the result [Zhang et al. 2006].

While several studies have already been performed, the results vary significantly and partially contradict each others' results. Furthermore, most prior studies rely on either artificial auditory feedback or simple, short collision sounds to support the interaction tasks. The variation in the results can be attributed to different factors. like the utilized feedback modality, the specific task that was examined, as well as the quality of the feedback which depends on its design, implementation, and the used hardware.

In contrast, the study presented here aims at examining a more realistic task with prolonged feedback, where the quality of the feedback is important. Furthermore, another focus is the comparison of auditory feedback to two distinct haptic feedback types, where the sound has a comparable information content as the vibration haptics, while surface haptics provides a different type of information.

3.2 Study Design

The goal of the presented study is to determine if auditory feedback can be used to substitute haptic feedback – or at least reduce the impact of missing haptics – for certain types of interaction tasks. The chosen task includes vibrations, which are an important type of feedback that can be perceived both haptically and auditorily. Promising applications are tasks that produce vibrations in the range of $50 - 500\,\mathrm{Hz}$, which can be perceived both by auditory and tactile senses, and allow a good discrimination of frequency changes. Such vibrations are commonly produced by certain mechanical tools, e.g. motorized saws, drills, or grinders.

After examining different possible tasks, a suitable scenario for this study was found. It is inspired by dental drilling/grinding but modified to not draw a direct comparison to dental applications. Participants use a haptic device to control a drill with a spherical, rotating head (see Figure 3.1). On a hard surface, a lump of softer material is located. The task is to remove the soft material while avoiding damage to the underlying surface. Apart from the visual representation of the surface, material, and drill, combinations of three additional feedback types are provided. When the drill comes into contact with the surface, a haptic representation of the contact force provides information about the contact pressure. Additionally, the drill's rotation speed changes depending on the amount and hardness of material that is being removed. Drilling softer material will produce a different rotation frequency than drilling harder material, and a higher contact pressure leads to a higher abrasion and thus a lower frequency. This rotation frequency can be used as feedback using either a haptic representation of the vibration or a matching sound.

This scenario was chosen because drilling is a task for which visual, haptic, and auditory modalities provide important feedback, and all these feedback types can be reproduced with high quality in a virtual environment. Furthermore, it includes two different kinds of haptic feedback: vibration of the drill, as well as surface contact forces.

While the task is inspired by dental drilling, a neutral scenario was chosen where the surface is mostly flat instead of tooth-shaped, and the drill has a larger diameter (32 mm). This distinction has multiple reasons. For one, a direct association with dental drilling should be avoided, since it is often considered unpleasant and could lead to discomfort in the participants. Also, the larger drill allows a visual representation on displays with standard resolution, since a small-scale dental drill would require a very high resolution to still be discernible. Additionally, larger drills typically produce lower-frequency vibrations, which are more suitable for a haptic and auditory reproduction. Furthermore, the larger scale makes the task easier, so that participants without experience in using precision drills can perform the task, without requiring long training up-front. Lastly, the used haptic device, a PHANTOM Omni, comes with a stylus whose size corresponds to a drill of the chosen dimensions.

3.2.1 Setup

In the user study, participants are seated in front of a large stereoscopic screen showing a visual representation of the scenario (see Figure 3.1). They use a Sensable PHANTOM Omni haptic device (see Section 3.3.1) to control a virtual drill, which they use to remove a lump of a softer material of dark yellow color from a flat, gray surface.

In addition to the visual representation, three different types of feedback are provided: auditory vibration feedback, haptic vibration feedback, and force feedback representing surface contacts. In reality, drills rotate with a frequency that changes due to friction when the drill comes into contact with a material. This rotation frequency can be perceived both by the sound the tool produces, and by the vibrations that are transferred from the tool to the user's hand. Therefore, this scenario allows an examination of the quality of the same type of feedback – i.e. tool vibration – using two different modalities, either haptics or sound. Since both modalities provide the same type of information, they might allow for a good sensory substitution. In addition to the two vibrational feedback channels, another type of haptic feedback provides information about contact forces between the drill and the material. This is useful to compare the benefit of auditory feedback to a haptic feedback type that does provide a different type of information.

In the study, the task of the participants is to remove a patch of target material from a surface using a spherical drill. A drill with a spherical head was chosen so that users do not have to consider its exact orientation, and it allows a visual estimation of its penetration into the surface. The drill's length is 220 mm, and its drill head has a diameter of 32 mm. As target material, a patch of dark yellow material is placed on a gray, flat surface. The target material always has an ellipsoid shape. The ellipsoid's planar surface cross-section has an area of 17500 mm^2, but its shape changes between trials to reduce training effects. The ratio between its two axes varies from mostly round (1 : 1.05) to elongated (1 : 1.4), and its orientation on the surface is different for each trial. The in-plane diameters of the ellipsoid vary between 93 mm and 182 mm. The thickness of the material is chosen such that its total volume is 75000 mm^3,

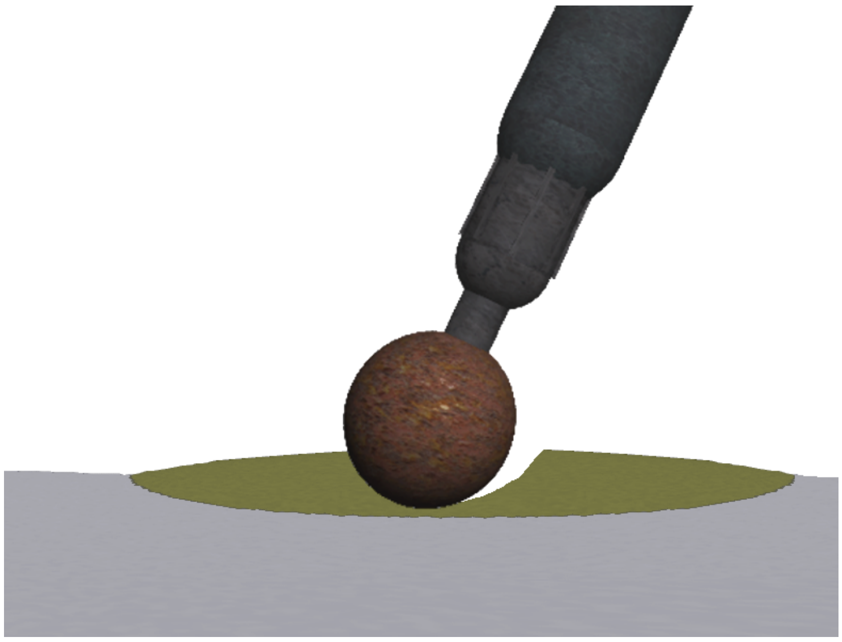


Figure 3.2: Cross section of the scenario, showing the original surface (gray), the target material that should be removed (yellow), and the drill.

which results in a maximum thickness of 8.5 mm in the center, which is close to half the radius of the drill (see Figure 3.2). Since the drill is larger than the target material's thickness, careful drill movements are required over the whole target surface in order to avoid errors, which is well-suited for testing a task that requires a high precision.

The goal of the study is an evaluation of haptic and auditory feedback, for which different feedback types are presented. To allow a distinguishable difference in the vibration feedback between the material types, they are modeled to have a different softness. While the underlying surface is hard, the target material is softer. This has two consequences for the feedback. Firstly, the harder material is removed slower by the drill, so that equal pressure application leads to faster drill movement through the soft material than through hard material. Secondly, the harder material leads to a lower rotation frequency of the drill head, and thus to a change in the vibration. The task of the user is to remove as much of the soft material as possible in a fixed time span, while at the same time as little as possible of the underlying surface material should be removed.

Under all conditions, a visual representation of the drill as well as the surface and target materials are always shown. To allow a better perception of the drill's position relative to the surface, its shadow is rendered. However, no direct visual representation of the amount of removed material – e.g. by drawing dust – is provided, and the drill rotates too fast to allow a visual discrimination of its rotation speed. Since the drill's head also occludes the surface contact location, it is difficult to perceive the current material abrasion from only the visual feedback. However, the visualization is well suited for the general guidance of the drill towards the surface, and to detect patches of remaining target material. In addition to the visual feedback, three feedback types can be provided:

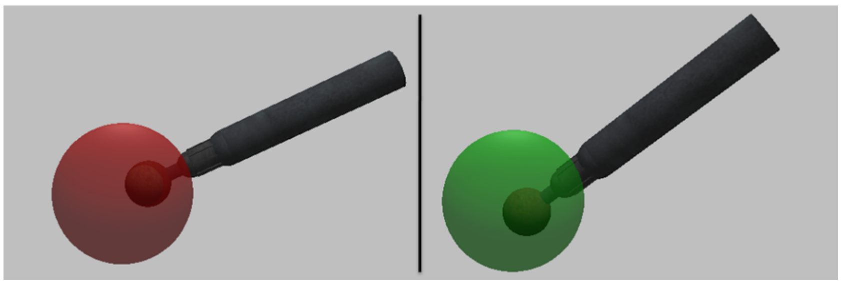


Figure 3.3: To start a task, the user has to move the drill head into a target sphere, and hold it there for 3 seconds.

- *Surface force feedback* (S) provides haptic force feedback of the contact forces that occur when the drill is pressed onto the surface.
- *Vibration haptic feedback* (V) reproduces the vibration of the drill using a haptic feedback device.
- *Auditory feedback* (A) uses sound to reproduce the vibration of the drill.

Details on the different feedback types and their technical realization are given in Section 3.3.

3.2.2 Procedure

In this study, a within-subject design is used, examining the three independent variables *auditory feedback* (A), *vibration feedback* (V), and *surface force feedback* (S). Each participant tests all combinations of feedback types, which results in eight conditions: Ø (only visual feedback), A, V, VA, S, SA, SV, and SVA. The study consists of multiple trials, each testing one of the eight conditions with a different material layout. At the start of each trial, the surface is invisible and cannot be interacted with. Only a drill and a target sphere are visible, and the user has to move the drill's head into the target sphere – which then turns from red to green – and keep it there for 3 seconds in order to start the drill (see Figure 3.3). This ensures that the drill's starting position is the same for all trials. Once the trial starts, the surface and the target material is shown, and the user can begin removing material using the drill. After a fixed time span, the trial ends by stopping the drill interaction and fading to black. As quantitative results, the removed target material (RTM) and error (ERR) – i.e. the volume of material that was erroneously removed from the underlying surface – are recorded.

During the tasks, participants are seated at a table in front of the stereoscopic projection screen (see Figure 3.1). The haptic device is placed in the center of the table, in front of the user. Also, the user wears stereoscopic glasses with tracking markers, which allow using a stereoscopic display and provides head-tracking data. Furthermore, the participants wore headphones during all trials. Since the haptic device can produce significant noise when displaying vibrations, closed headphones were chosen to reduce this sound, and they were worn even when no acoustic feedback was active. Additionally, a constant sound – corresponding to the sound of the drill when rotating without surface contact – was played for non-auditory conditions, to mask out the haptic device's sound. This was necessary because the sound produced by the vibrating haptic device was very similar to the sound from the auditory feedback. If the haptic device's sound would have been audible, it could have influenced the results for conditions without sounds, but with vibration feedback.

The available time for each trial is 25 seconds. This value was determined in pre-studies, and was chosen such that it is possible to remove a large fraction of the target material while being too short to remove all of it. This was done to prevent high errors, since removing small remaining fragments of target materials would often cause high amounts of abrasion of the underlying surface.

The study consists of different phases. First, written instructions (see Appendix C.1) are given to the user, explaining the tasks and the procedure of the study, but not giving information about different feedback types or the goals of the study. In the instructions, the participants are explicitly advised to remove as much of the target material as possible, while taking care to remove none of the gray surface material. This was formulated such that the focus is put on avoiding erroneous removal of surface material, while still putting time pressure on the participants.

Next, the participants perform a block of training trials. First, an initial trial is performed during which all feedback types are active. This training trial allows the users to get accustomed to the task, the device, and the feedback types. It lasts for 60 seconds and is longer than normal trials because pre-studies showed that users not familiar with the hardware and interaction method require a longer period to familiarize themselves with the device and the interaction. After the initial training task, five additional training trials follow, each lasting as long as the actual trials – i.e. 25 seconds – and having different feedback combinations. The five normal-length training trials were, in order, no feedback (Ø), only auditory feedback (A), only haptic vibrations (V), only surface force feedback (S), and all feedback types (SVA). These training trials allow the participants to gain a feeling for the available time, the handling of the drill, how fast material can be removed and how to avoid errors. They also allow the users to familiarize themselves with the different feedback types. The goal of the training was to reduce the impact of training effects, since pre-studies have shown that there is a significant increase in performance during the first few trials, but a low change afterwards.

After the training, the actual trials are performed. These consist of three blocks, where each block contains eight tasks – one for each condition. The order of the conditions within each block is counter-balanced using Latin squares to further reduce the impact of learning effects. All in all, 24 trials are performed, i.e. three for each of the eight condition. This number of repetitions was chosen to provide enough data for a statistical analysis while keeping the overall duration of the study below 30 minutes to avoid fatiguing the users.

After the training phase and between the three blocks of trials, pauses of 30 seconds are added to allow the user to rest his arm, in order to reduce effects of fatigue. During a pause, the screen fades to black, and a message is shown that asks the participant to put down the haptic device. Additionally, participants were told that they could rest between trials before moving the drill into the starting sphere, although no one took advantage of this.

After the final trial, the screen fades to black, and an ending message is displayed. Afterwards, the participant is asked to fill out a questionnaire (see Appendix C.2). The questionnaire records general information (age, sex, relevant task experience), followed by questions regarding the subjective quality of the feedback types, which should be answered using a 5-point Likert scale ranging from *disagree* to *agree*.

The whole study procedure lasts about 25 to 30 minutes per participant, of which 17 minutes were spent on the actual trials.

3.2.3 Hypotheses

The goal of the presented user study is to determine whether or not acoustic feedback is able to substitute for certain types of haptic feedback. For this, the following hypotheses are postulated:

H1 *Adding feedback enhances user performance.* All feedback types – auditory feedback, haptic vibration feedback, and surface force feedback – provide additional information over the visual feedback. Auditory and vibration feedback informs the user about the type of material he is currently removing, and surface forces indicate the contact and the currently exerted force. Thus, it is expected that each individual feedback type helps the user in performing the task better, leading to higher amounts of removed material (RTM), or lower error rates (ERR).

H2 *The influence of auditory and vibration feedback on user performance is similar.* Both auditory and haptic vibration feedback convey the same information about the current interaction state – i.e. the frequency of drill rotation, and thus the type and amount of material that is currently being removed. Since they provide the same information and are modeled based on the same property - drill rotation speed – any difference between them should originate from either the quality of the feedback, or on differences in the perceptional processing. Since both haptics and sound are perceived by humans with a high speed, the expected impact of the latter is expected to be similar. The haptic and auditory vibration feedback was intentionally designed to be of very similar quality so that their effect on task performance should be very similar.

H3 *Combining auditory and vibration feedback does not further increase the performance.* Since both haptic vibration and auditory feedback provide the same type of information to the users, and use channels that have similar perception thresholds, it is expected that a combination of both techniques does not lead to a further increase of user performance, because all the available feedback information is already available when using just one of the feedback channels.

H4 *Surface force feedback improves interaction performance regardless of other active feedback types.* While auditory and haptic vibration feedback provide the same information, surface force feedback provides a different kind, namely the drill's contact with the surface, as well as the contact pressure. Therefore, it is expected that surface force feedback improves the users' performance even if other feedback types are active simultaneously.

3.3 Feedback Realization

In this section, technical details about the realization of the different feedback types as well as the whole study application and hardware are provided. The base application was developed

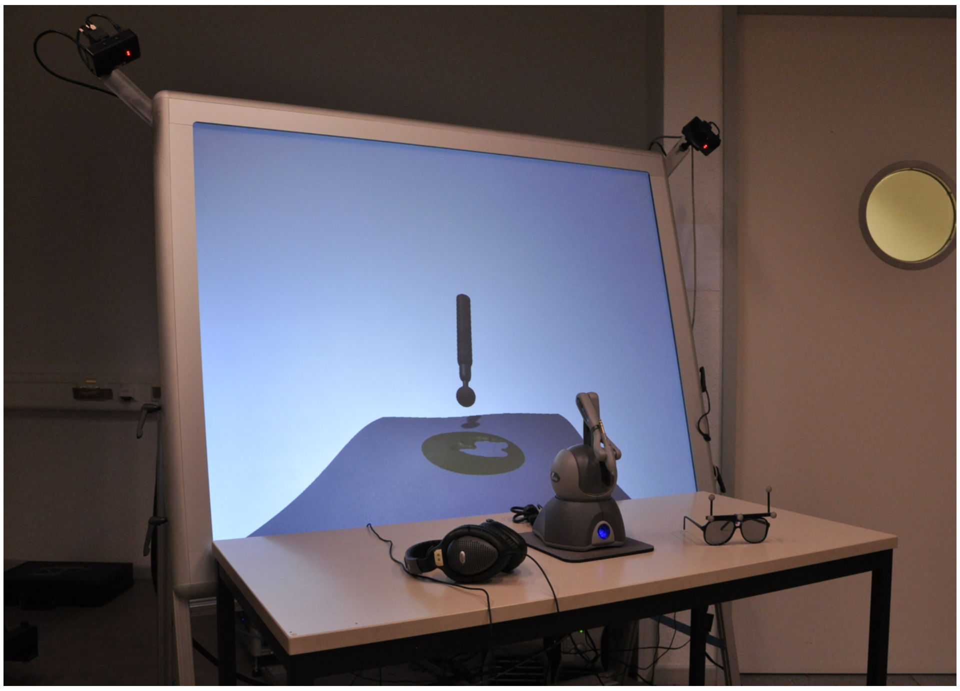


Figure 3.4: Hardware setup used for the user study: imsys flip150 projection system, passive-stereo glasses with head tracking, Sensable PHANTOM Omni haptic device, Beyerdynamics DT 770 headphones.

based on the ViSTA Virtual Reality Toolkit [Assenmacher et al. 2008]. It handles the rendering of the scene, state transitions between trials, and logging of results. The application provides a visual representation of the scene and handles input using the haptic device. Furthermore, it calculates the two haptic feedback types directly and displays them using the attached haptic device. The synthesis of the drill sound is performed by a separate application running on the same PC, which is connected over a network interface.

3.3.1 Hardware

The study was performed on an imsys flip150 system with a 60” back-projection screen (1400x1050 resolution at 60 Hz refresh rate) providing stereoscopic images using passive polarization stereo technology. The 3D glasses were tracked using an ART TrackPad opto-electronic tracking system, in order to provide a user-centered projection and spatialized sound. Participants were seated at a table that was located in front of the screen. At the center of the table, the Senseable PHANTOM Omni haptic device (see Section 3.3.1) was located.

For sound rendering, Beyerdynamics DT 770 headphones were used. These provide a high reproducible frequency range, and furthermore are closed so that the volume of outside sound is reduced.

The study application runs on a Windows 7 PC with an Intel Xeon 5530 (2.4GHz), 4GB RAM, and an Nvidia Quadro FX 5600 graphics card. The haptic device provides 6-DoF input and 3-DoF haptic output. Its position and orientation input is used to control the drill. The

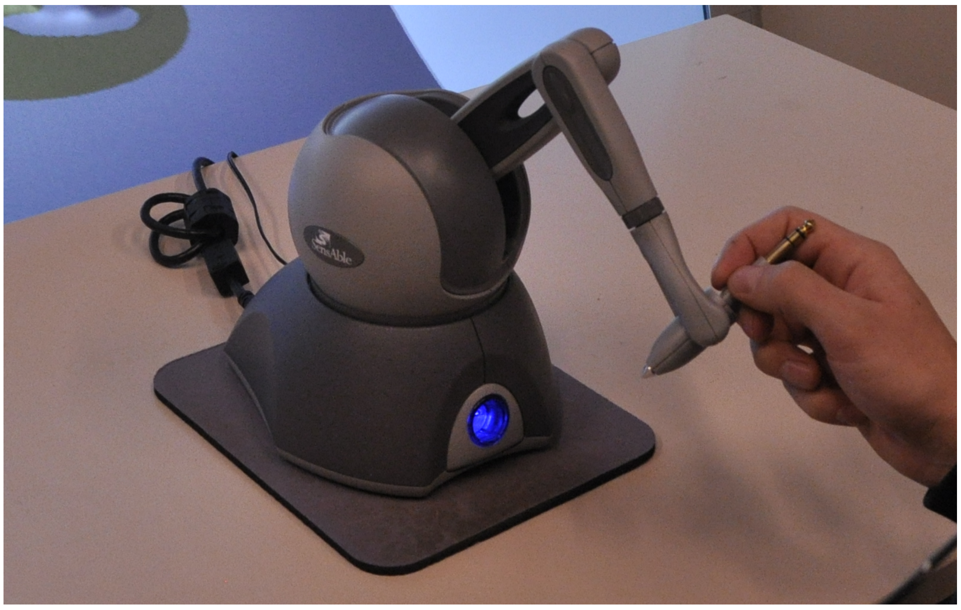


Figure 3.5: The Sensable PHANTOM Omni haptic device used in the study. To improve vibration forces, the plastic stylus cover was removed, and the device is gripped at the metal jack.

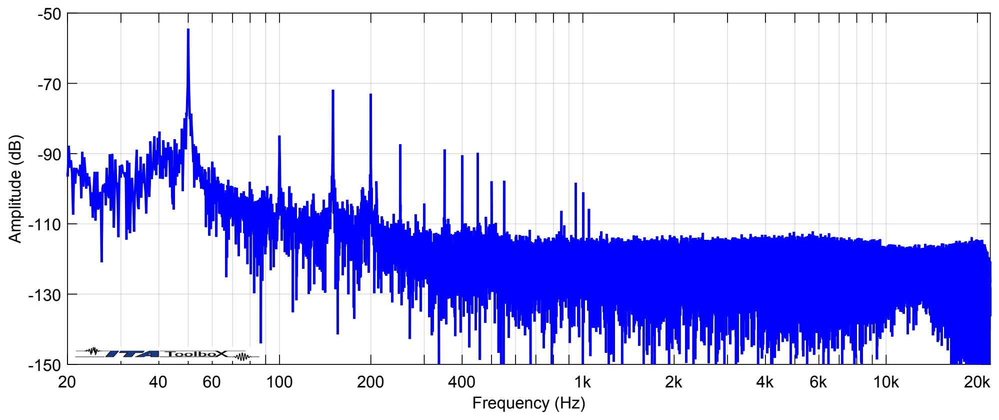
virtual drill is non-collocated with the input device, but offset along the depth direction. This was necessary to be able to display the drill and material on the screen without the haptic device occluding the visual representation.

Haptic Device

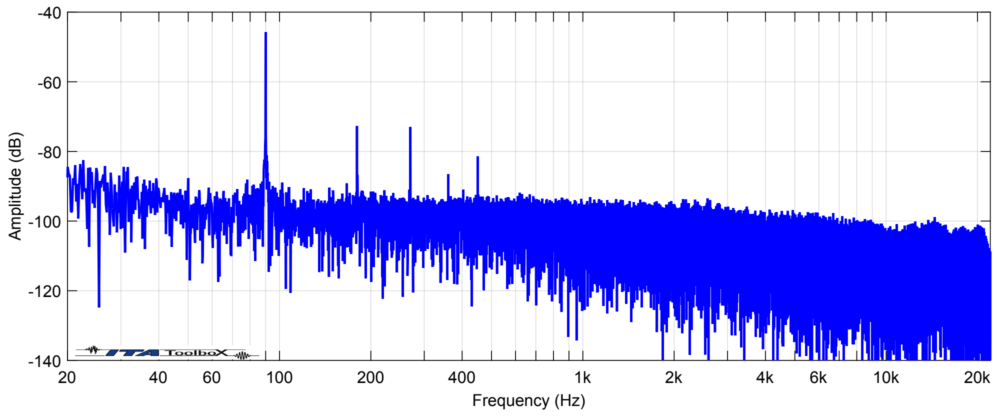
As a haptic device, a Sensable PHANTOM Omni[1] is used (see Figure 3.5). It consists of a stylus attached to a mechanical arm that allows tracking the position and location of the stylus tip. By using mechanical actuators, a 3-DoF haptic force can be exerted on the stylus – thus, only directional forces can be reproduced, but no rotational forces. The interaction space of the device is large enough to incorporate a circle with a diameter of 165 mm, which is enough to include the whole target material and part of the surrounding surface. Using the haptic device, forces of up to 3.3 N can be reproduced with an update rate of 1000 Hz.

Typically, such haptic force feedback devices are designed to reproduce slowly-changing forces, as they occur when simulating surface contacts. For this study, however, vibrations with frequencies in the range of $50 - 100$ Hz should be realized. The manufacturer ensured that frequencies of up to 500 Hz could be reproduced when using a reduced vibration force of at most 1.5 N. However, we wanted to verify that the device can reliably reproduce the desired vibrations with accurate frequency, but without significant overtones. Thus, an examination of its response to vibration forces in the desired frequency range was performed. The device was set up to reproduces continuous sine forces with frequencies between 16 Hz and 100 Hz using a peak force level between 0.4 N and 0.9 N. The motion response of the device was measured

[1]`http://www.geomagic.com/de/products/phantom-omni/overview`



(a) 50 Hz



(b) 90 Hz

Figure 3.6: Vibrometer measurements of the haptic device's response to a sine input force of 50 Hz and 90 Hz with a peak amplitude of 0.4 N.

using a laser vibrometer, recording the vibrations of the tip. These recordings showed that in the tested frequency range, the device is well capable of reproducing the desired vibrations (see Figure 3.6). The results show high peaks at the desired frequencies, with overtones having at least 10 dB lower amplitudes. However, these responses were measured at the tip, but usually, the haptic device is gripped at a plastic stylus covering the handle. Further vibration measurements showed that this plastic stylus absorbed some of the vibration, leading to lower amplitudes and generating more acoustic noise. Therefore, the plastic stylus was removed for the user study, and the device was used by holding the underlying metal adapter, which is formed like an audio jack (see Figure 3.5). This led to a better transmission of the vibration forces.

Another advantage of the removal of the stylus cover was the reduction of sound produced by the haptic device when reproducing vibrations. However, even without the cover, the sound produced by the haptic device is audible and might have interfered with the auditory feedback. Therefore, the haptic device was placed on a foam absorber to dampen vibrations of the device's body and to prevent vibrations from transmitting onto the table. Furthermore, closed headphones were used during the study, which decreases the volume of outside sounds. Lastly, even for non-auditory conditions, a sound was always played to mask any remaining noise.

3.3.2 Material Abrasion

During the study tasks, a virtual drill is used to remove material from a surface. To allow this, the interaction between the drill and the surface must be modeled to simulate material abrasion.

To allow removing surface material, a height field with a cell size of 1 mm is used. This resolution was chosen because it is fine enough to model the surface abrasion – the drill head with a radius of 18 mm covers a large number of cells – and allows for a smooth visual representation of the surface, while it is still possible to simulate and render in real-time. Each cell stores the height of the base surface material as well as any target material on top. When the drill removes material, the entries in the corresponding cells are reduced, leading to a loss of material.

The usage of a height field allows for a fast simulation and rendering, but restricts the possible configurations. Only a single surface height is used per in-plane position, so that certain features like overhangs or cavities cannot be modeled this way. However, for this study, this is not a problem because the drill head is spherical and significantly larger than the height of the target material. Thus, for the drill to produce problematic features, it would have to drill very deep into the underlying surface material. Such deep penetrations would correspond to a very high interaction error, and never occurred during the study.

To model the abrasion, the input point of the haptic device is used to determine the drill position. When the drill is not in contact with any material, a direct mapping from input point to the drill position is possible. However, when the drill comes into contact with the surface, it is possible to still move the haptic device further into the surface, either because the surface force feedback is deactivated for the trial, or because the forces produced by the haptic device are too low to prevent a further penetration. In this case, simply placing the virtual drill head at the input point would lead to the drill being placed inside the surface. This would cause visual clipping, i.e. the visual model of the drill would be drawn partially inside the (non-removed) surface, and may also cause discontinuities in the material abrasion and lead to unrealistic drilling behavior.

To prevent deep penetrations of the virtual drill, the common approach of a distinction between *haptic interaction point (HIP)* and *virtual interaction point (VIP)* is used. While the HIP corresponds to the haptic device's input position, the VIP is a – potentially different – point that is used to calculate interaction with the virtual scenery. Here, the VIP is modeled such that it is still controlled by the HIP and usually stays close to it, but remains on the contact surfaces using a special interaction model in order to prevent surface penetrations. In this study, the VIP is computed using *virtual coupling* [Colgate et al. 1995], which tries to align VIP and HIP while preventing penetration even for surfaces with high contact stiffness. Specifically, *static virtual coupling* [Wan et al. 2003; Barbič et al. 2008] is used, where the HIP and VIP are coupled by a generalized spring so that penetrations of the virtual tool produce penalty contact forces based on Hooke's law.

The VIP is computed at the update rate of the haptic device, which is 1000 Hz. In each haptics frame, a collision detection between the drill head and the surface is performed. For each cell of the surface, it is tested whether or not it intersects the drill, and if it does, the cell's top-most point – i.e. the surface point – is added to the list of contact points. Based on this list of contacts, a linear approximation of the forces and torques acting on the tool is computed, and a linear system is minimized to find an equilibrium between contact and coupling forces, thereby determining the VIP.

After determining the VIP for the next haptics frame, the material abrasion is computed. This, too, is performed within the haptic update loop at 1000 Hz. To compute the abrasion, all cells that intersect the drill head are abraded based on their penetration depth, which is computed as the distance from the contact point to the surface of the drill's spherical head. Based on the material type and the penetration depth – which encodes contact pressure – material is removed from that cell. The harder surface material is removed at a rate of $0.04\,\frac{\text{m}}{\text{s}}$ per meter of penetration depth, and the softer target material at $0.2\,\frac{\text{m}}{\text{s}}$ per meter of penetration. Thus, material is removed faster if the HIP is moved deeper into the surface, and soft material can be removed five times as fast as hard material. Without surface force feedback, it would be possible to create high penetration depth, which can lead to undesirably high abrasion rates. To prevent this, the maximum penetration depth considered for material abrasion is limited to 0.1 m.

3.3.3 Frequency Modeling

Both the haptic vibration feedback and the auditory feedback use the same basis for their feedback, namely the rotation of the drill head. This way, both feedback types provide the same type of information.

Both the vibration frequency and amplitude of the tool are modeled based on the amount and type of material that is currently being removed. The rationale behind the modeling is that contact with material slows down the drill head, but also causes stronger vibration than when it is running without surface contact. The vibration frequency f and amplitude a are computed for each haptic simulation frame, i.e. at the same rate at which surface abrasion is computed, and optional haptic vibration and surface contact forces are updated. They are computed after the material abrasion. As input, the amount of removed soft material Δv_s and removed hard material Δv_h is used.

$$f = \frac{f_n + w_s \cdot \Delta v_s \cdot f_s + w_h \cdot \Delta v_h \cdot f_h}{1 + w_s \cdot \Delta v_s + w_h \cdot \Delta v_h} \tag{3.1}$$

$$a = \frac{a_n + w_s \cdot \Delta v_s \cdot a_s + w_h \cdot \Delta v_h \cdot a_h}{1 + w_s \cdot \Delta v_s + w_h \cdot \Delta v_h} \tag{3.2}$$

Here, $w_s = 100$ and $w_h = 500$ are relative weighting factors for the soft and hard material that determine an interpolation between the three frequencies or amplitudes. The frequencies are given as $f_n = 90\,\text{Hz}$ for rotation without contact, $f_s = 85\,\text{Hz}$ for contact with soft material, and $f_h = 58\,\text{Hz}$ for hard material. The corresponding values for the amplitudes are $a_n = 0.2$, $a_s = 0.5$, and $a_h = 0.8$, respectively.

To determine these values for the frequencies and amplitudes, the vibrations produced by different real drills drilling different materials were analyzed. Since the simulated tool should not directly match any real drill in order to keep the scenario abstract, the chosen values do not correspond to any one real drill. Instead, the characteristics of vibrations from different similar tools like dental drills, grinders, metal drills, etc. were used as a basis to determine the effect of different materials. Further pre-studies were performed with virtual drills using different frequency models. The chosen approach to model the drill frequency provide plausible results for the chosen scenario and also allow a good differentiation of the three material types. Furthermore, the chosen values provide feedback within the frequency and amplitude limits imposed by the haptic device.

3.3.4 Haptic Feedback

The haptic feedback consists of two separate components, surface force feedback, and haptic vibration feedback.

Surface force feedback reproduces contact forces when the drill comes into contact with the surface, allowing to feel the pressure exerted by the surface onto the tool. It is derived from the virtual coupling algorithm that is used to compute the virtual drill's position from the state of the input device (see Section 3.3.2). It produces a force based on the displacement $\vec{d} = \vec{h} - \vec{v}$ between the HIP $\vec{h}$ and the VIP $\vec{v}$. Since the haptic device can only produce a limited amount of force, the maximum force produced by surface contact is 1.2 N. Thus, the surface contact force $\vec{F_S}$ is interpolated between 0 N (no surface contact) and 1.2 N (max penetration depth of 0.1 m.

$$\vec{F_S} = min\left(|\vec{d}|, 0.1\,\mathrm{m}\right) \cdot 1.2\,\mathrm{N} \cdot \frac{\vec{d}}{|\vec{d}|} \tag{3.3}$$

This provides a feeling of the contact pressure, and thus not only indicates a contact, but also allows an estimation of how fast material is removed, since both values depend on the penetration depth. However, the contact force is independent of the type of surface with which the drill is in contact with. Thus, the material type cannot be directly inferred from the surface force feedback, but only indirectly by observing changes in the drill's speed while moving through the material.

Vibrations are reproduced by adding a force vector that rotates around the drill's central axis. From the frequency f and amplitude a computed by the frequency modeling, the vibration force $\vec{F_V}$ is computed. For this, the vibration axis' current angle α, is stored and updated each frame based on f: $\alpha \leftarrow \alpha + 2\pi \cdot \Delta t \cdot f$. The local vibration force $F_{H,l}$ in the drill's coordinate system is then computed as follows:

$$\vec{F}_{H,l} = \begin{pmatrix} sin\alpha \\ 0 \\ cos\alpha \end{pmatrix} \cdot a \cdot 1\,\mathrm{N} \tag{3.4}$$

This global vibration force F_H is then gained by rotating $F_{H,l}$ by the drills current orientation. This produces a vibration force of at most 0.8 N, because $a \leq 0.8$.

The total force reproduced per simulation frame by the haptic device equals the sum of both forces $\vec{F} = \vec{F}_S + \vec{F}_V$ when both haptic feedback types are active. Otherwise, based on the feedback condition, only one of the forces or none at all is output. Because human haptic perception works with a high temporal resolution, it is commonly accepted that haptic forces should be updated with at least 1 kHz, which is also the maximum update rate supported by the haptic device. Therefore, it was taken care that all computation steps that are performed within one haptic frame – i.e. the virtual coupling, material abrasion, and computation of surface contact force and vibration force – can be calculated in less than 1 ms, in order to ensure a smooth haptic feedback.

3.3.5 Auditory Feedback

The goal of the auditory feedback is to produce a sound representing the noise produced by the drill's rotation and should be of comparable quality as the vibration haptic feedback. Therefore, it uses the same input parameters, frequency f, and amplitude a.

For the auditory feedback, it does not suffice to reproduce a sine signal with varying frequency and amplitude, as was used for the haptic vibrations. Such a signal would sound very artificial and might be straining for the participants. While the synthesized sound should not directly correspond to a real drill, it should be plausible for the application. The sound produced by real drills shows significant overtones and higher-frequency noise. Such effects should be modeled to produce a plausible sound, and also correlate to effects observed in the vibration measurement of the haptic device (see Section 3.3.1).

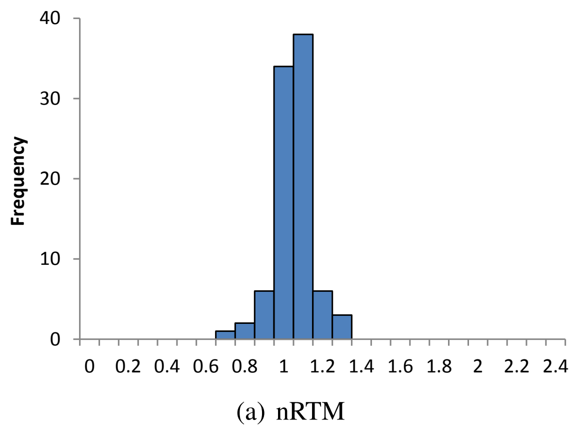
The acoustic feedback was synthesized using a separate application that ran on the same computer as the application handling the main simulation of the study. It connects to the main application using a TCP/IP connection to receive material abrasion data as well as the user's head position and orientation. It was realized using *pureData*[2], a visual programming language for audio synthesis, which allows a graphical design of dataflow-like sound synthesis chains. Sound output is achieved using *ASIO4ALL*[3], producing an audio latency of around $30-40$ ms.

The basis of the synthesized sound is a combination of a sawtooth and a sine signal of the input frequency f. The sawtooth signal was chosen because it more closely resembles the sound of real drills, and includes overtones of the base frequency. However, with just the sawtooth signal, these overtones would have been too strong compared to the base signal. Thus, the sine signal was used to pronounce the base frequency. Furthermore, band-pass filters were used to modulate the amplitude of higher-frequency components of the sounds. This way, the resulting sound is plausible for a drill, and the resulting frequency spectrum resembled the measured vibrations of the haptic device.

After synthesizing the base signal, a short artificial reverberation of 0.3 s was added to emulate reflections in the environment, because replaying a free-field signal without any reverberation effects would appear unnatural. To allow an auditory localization of the drill's head and to increase the realism of the scene, a binaural synthesis was used to spatialize the

[2]puredata.info

[3]www.asio4all.com



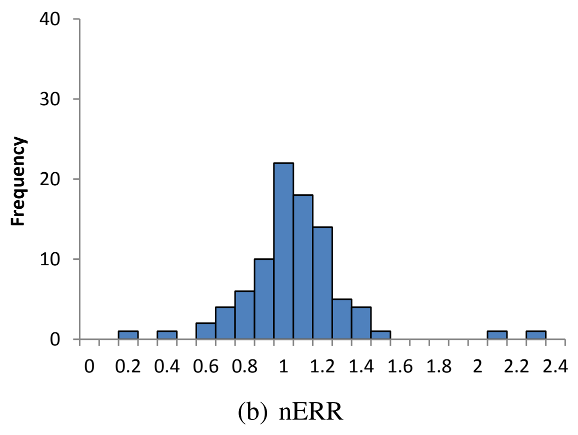


Figure 3.7: Histograms showing the distribution of measurements for the normalized removed target material (a) and the normalized error (b).

signal using a generic HRTF. For this, the center of the drill head is used as the location of the sound source, and the user's head position and orientation are measured by a tracking system. While initial tests have shown that participants usually perform only small head movement, this interactive binaural synthesis still provides a better localization than a static setup, because differences in the user's size and posture can alter the results in addition to small-scale movements.

3.4 Results

A total of 33 persons participated in the user study, with an age of 20 – 33 years (average: 25.5). From these 33 participants, one person had to abort the study due to dizziness, one person was not able to see stereoscopic 3D images, and a third person had prior knowledge of the study and its goals. The results of these three persons were excluded from the evaluation. All of the remaining 30 participants – of which 20 were male and 10 female – had normal hearing and normal or corrected-to-normal vision. Two types of results were recorded - quantitative measurements of the participants' performance during the trial, as well as a post-study questionnaire collecting subjective preferences.

3.4.1 User Performance

During the trials, the main quantitative measurements are the *removed target material* (RTM) and the *erroneously removed material* (ERR), referring to the volume of yellow target material and gray surface material that was removed with the drill. However, these values can vary significantly between participants, making a comparison more difficult. The main problem was that the measurements did not resemble a normal distribution, which is important for a statistical analysis. Since the goal of this study is to show the change of the interaction behavior, the RTM and ERR have been normalized for each participant by his average over all trials, yielding the *normalized removed target material* (nRTM) and the

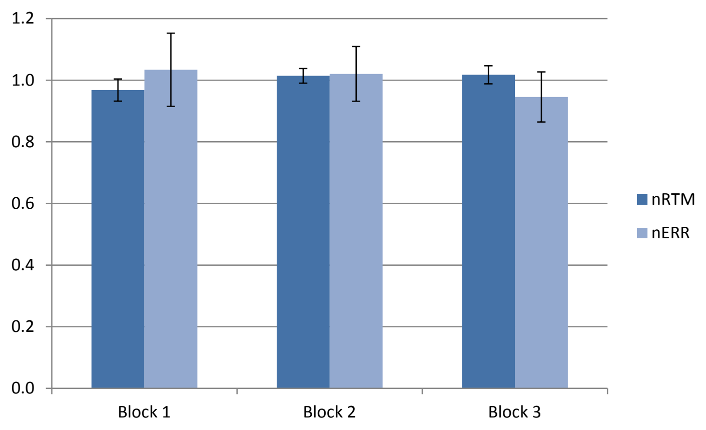


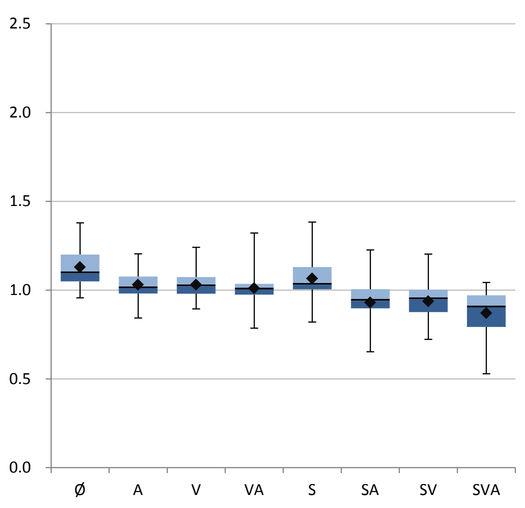
Figure 3.8: Training effect: normalized removed target material (nRTM) and normalized error (nERR) for each of the three trial blocks; bars show average, whiskers show standard deviation.

normalized erroneously removed material (nERR). Plotting the histograms of these normalized measurements (see Figure 3.7) shows that nRTM closely resembles a normal distribution. nERR shows some high-value outliers, but is close enough to a normal distribution to allow a statistical analysis.

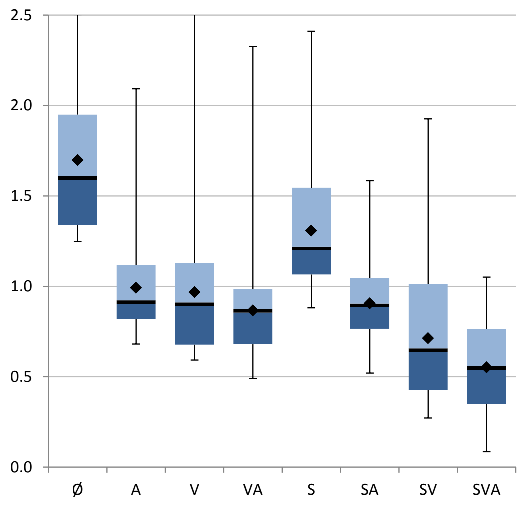
To examine the learning effect during the study, the change of nRTM and nERR over the course of the study was examined. Figure 3.8 shows the average nRTM and nERR for each of the three trial blocks. As can be seen, there is still a slight trend in user performance, but it is rather low. Using an ANOVA, no significant effects could be found for the nRTM ($p = 0.86$) or the nERR ($p = 0.845$) between the three blocks. Furthermore, the different conditions are repeated for each block and their order is counter-balanced using Latin squares, further limiting the influence of training effects. Therefore, learning effect will be neglected in the further analysis.

The most important results are the differences in nRTM and nERR for the eight different feedback conditions (see Figure 3.9). The results show that nRTM, i.e. the amount of target material removed per trial, slightly decreases when feedback is added. The decrease is rather low and shows a low variance. When regarding nERR, adding feedback also leads to a decrease of erroneously removed material. The effect is more pronounced than for nRTM, and also shows a higher variance in the results.

Based on the changes in nRTM and nERR, differences between the different conditions seem to exist. To verify this, a one-way MANOVA was performed to find a statistically significant differences between combinations of conditions. The resulting p-Values are listed in Table 3.1 for both nRTM and nERR, highlighting those pairs of feedback conditions which show a statistically significant difference. When comparing the condition with only visual feedback (Ø) to conditions with feedback, there is always a significant difference for both nRTM and nERR, with the exception of nRTM between Ø and S. Similarly, conditions with surface force feedback and additional feedback types (SA, SV, SVA) show a significant difference to the condition using only surface forces (S). When comparing the three



(a) nRTM



(b) nERR

Figure 3.9: Box plots of the normalized removed target material (a) and the normalized error (b) under different feedback conditions (**A**uditory × **V**ibration × **S**urface force feedback). Shown are the mean (diamond), median (bar), .25- and .75-percentiles (box), and minimum and maximum (whiskers).

conditions that combine auditory feedback, vibration feedback, or both (A, V, VA), no significant effect was found for either nRTM or nERR. When comparing A, V, and VA to the condition using only surface haptics (S), the only statistically significant difference can be found in nERR for S vs. V and S vs. VA. The three conditions A, V, and VA show no significant differences for either nRTM or nERR. Similarly, combining these three conditions with surface force feedback, i.e. SA, SV, and SVA, shows no significant differences between them except for a lower error rate for SVA than for SA. When comparing conditions without surface force feedback to their respective variants including surface force feedback, i.e. A vs. SA, V vs. SV, and VA vs. SVA, a significantly lower nRTM can be found, but nERR shows no significant difference. However, when comparing S to Ø, both nRTM and nERR change significantly.

While the MANOVA can find statistical difference between pairs of conditions, it cannot be used to determine if the effect of two conditions can be considered equivalent. To test for equivalence, a two one-sided *t*-test (TOST) is used, which considers two conditions to be equivalent if the 95% confidence interval of the difference of means is contained within a range $(-\theta, \theta)$ of acceptable difference. The acceptable difference was chosen as 5% of the global average of the values. Using this method, the pairs of conditions that can be considered equivalent have been computed. Condition pairs with equivalent nRTM are A and V, A and VA, V and VA, as well as SA and SV. For nERR, due to the higher standard deviation of the results, only A and V were found to be equivalent. Table 3.1 shows these pairs of conditions that are equivalent.

normalized removed target material

	Ø	A	V	VA	S	SA	SV	SVA
Ø	+	**.012**	**.011**	**.001**	.319	< **.001**	< **.001**	< **.001**
A	**.012**	+	$\boxed{.999}$	$\boxed{.997}$	.914	**.011**	**.023**	< **.001**
V	**.011**	$\boxed{.999}$	+	$\boxed{.997}$	.905	**.012**	**.025**	< **.001**
VA	**.001**	$\boxed{.997}$	$\boxed{.997}$	+	.506	.089	.157	< **.001**
S	.319	.914	.905	.506	+	< **.001**	< **.001**	< **.001**
SA	< **.001**	**.011**	**.012**	.089	< **.001**	+	$\boxed{.999}$	.385
SV	< **.001**	**.023**	**.025**	.157	< **.001**	$\boxed{.999}$	+	.253
SVA	< **.001**	< **.001**	< **.001**	< **.001**	< **.001**	.385	.253	+

normalized error

	Ø	A	V	VA	S	SA	SV	SVA
Ø	+	< **.001**	< **.001**	< **.001**	**.005**	< **.001**	< **.001**	< **.001**
A	< **.001**	+	$\boxed{.999}$	.927	.052	.991	.132	**.001**
V	< **.001**	$\boxed{.999}$	+	.977	**.026**	.999	.221	**.002**
VA	< **.001**	.927	.997	+	**.001**	.999	.821	.055
S	**.005**	.052	**.026**	**.001**	+	**.003**	< **.001**	< **.001**
SA	< **.001**	.991	.999	.999	**.003**	+	.584	**.017**
SV	< **.001**	.132	.221	.821	< **.001**	.584	+	.778
SVA	< **.001**	**.001**	**.002**	.055	< **.001**	**.017**	.778	+

Table 3.1: Statistical significance of the difference between pairs of feedback conditions for the normalized removed target material and the normalized error between feedback conditions. The table lists the p-Values for the difference. Light grey cells indicate significant results ($p < .05$), dark grey highly significant results ($p < .01$). Cells with a frame mark pairs of conditions that were found to be equivalent using a two one-sided t-test with acceptance range of 5%.

3.4.2 Questionnaire

In the post-study questionnaires (see Appendix C.2), different questions about the study and the different conditions were asked. All questions asked the user to rate a statement on a 5-point Likert scale, ranging from *disagree* to *agree*. The results are shown in Figure 3.10.

The first group of questions examined the feeling of involvement felt by the user in general and relating to the different feedback types. The answers indicate a preference for acoustic feedback over visual, vibration, or surface force feedback. The involvement induced by vibration feedback was rated similar to the overall involvement, and both visual and surface feedback are rated slightly lower. In general, the feeling of involvement was rated highly.

When asking about the helpfulness of auditory, vibration or surface force feedback, all three types were rated positively. A and V were rated similar, with a slightly lower score for S. For the direct comparison of feedback types (“X was more helpful than Y”), no clear preference is visible; only V was rated as slightly more useful than A in direct comparison.

The last block of questions asked about the easiness of tasks. For the tasks in general, an average rating was given. Tasks providing neither acoustic nor vibration feedback were rated as difficult. Tasks with either A or V were considered to be of average easiness, with just a slight preference for A. Their combination, VA, was rated to be the easiest. Furthermore, tasks including surface force feedback were rated similarly well as tasks providing VA.

3.5 Discussion

The goal of this study is to find out if auditory feedback can help to reduce the impact of a lack of haptic feedback, at least for some types of feedback. For this, four hypotheses were formulated in Section 3.2.3.

Hypothesis H1 assumes that adding any type of feedback – either auditory, vibration, or surface force feedback – enhances the users’ performance. The results of the qualitative measurements show a significant decrease of both the nRTM and nERR for either A, V, or S when compared to Ø. While the reduction of the error points towards an improvement in user performance, the decrease in the amount of removed target material is a negative development. This reduction in nRTM may be caused by the fact that an enhanced feedback makes users more aware of the errors they produce, leading to a generally more careful behavior and thus a slower removal of material. However, due to the much stronger decrease of nERR, and the task description explicitly stating that users should avoid any damage to the underlying surface, the impact of adding feedback can be interpreted as being an overall improvement. Furthermore, regarding the subjective user evaluation, the answers to the questionnaire show that all feedback types were regarded as helpful and that tasks with additional feedback were rated as easier than tasks without. This also indicates that additional feedback types are beneficial for the user performance. Still, because both nRTM and nERR are reduced, hypothesis H1 cannot be fully confirmed.

Since the vibration feedback and the auditory feedback provide the same type of feedback – i.e. the drill rotation speed, giving information about the type and amount of material that is currently being removed – Hypothesis H2 states that the influence of auditory feedback is

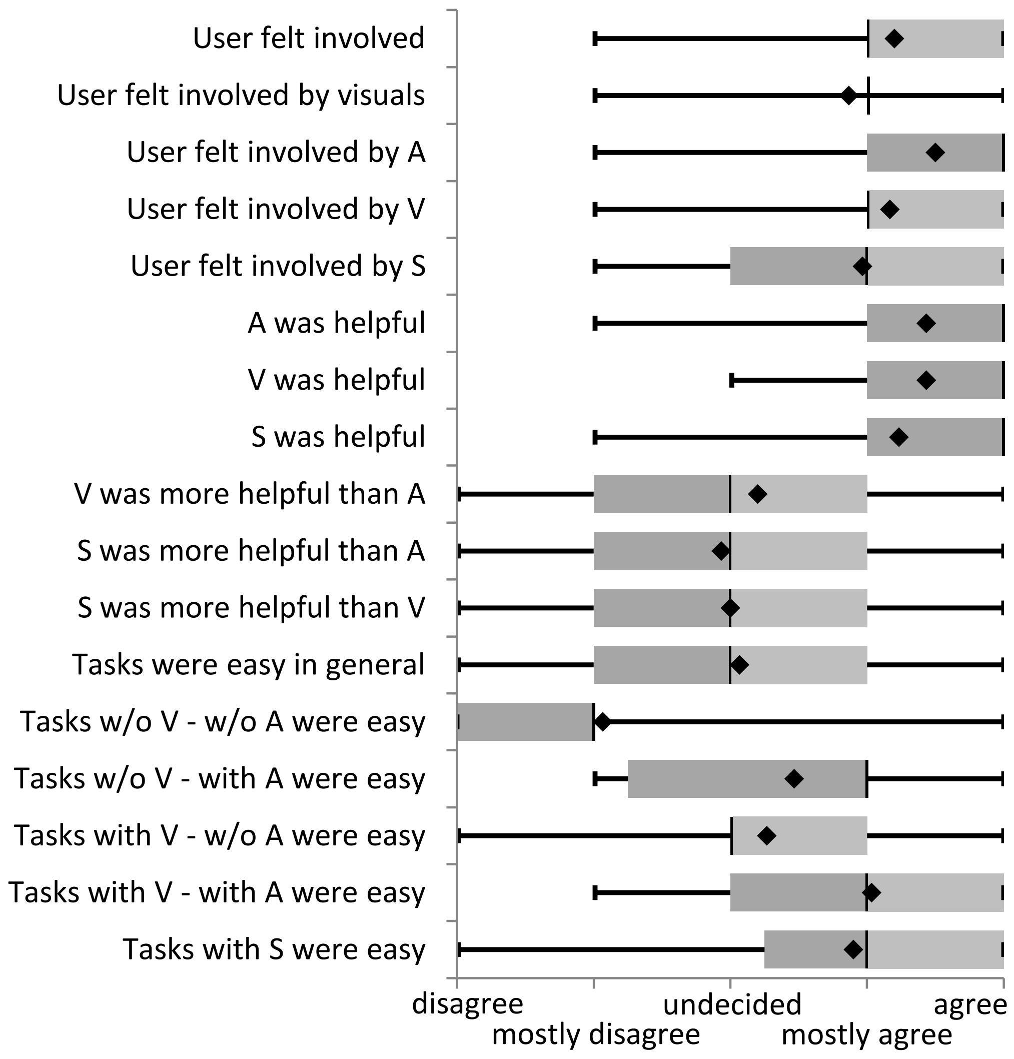


Figure 3.10: Results of the post-study questionnaire, showing for each question the mean (diamond), median, .25- and .75-percentiles (box), and minimum and maximum (whiskers).

similar to that of vibration feedback. When examining the user performance measurements, the conditions A and V result in very similar values for nRTM and nERR and show no significant difference. The same is true when comparing SA and SV, although here a slight yet not statistically significant trend towards a lower error for SV can be observed. Thus, the quantitative evaluation supports this hypothesis. This is also reflected in the results of the equivalence tests, which found A and V as well as SA and SV to be equivalent. Regarding the results of the questionnaire, the helpfulness is rated slightly higher for V than for A, but tasks with A are perceived as slightly easier than tasks with V. This also indicates a very similar quality of the influence of the two feedback types. Thus, hypothesis H2 has been confirmed.

The fact that auditory and vibration feedback provide the same type of information is also the basis of hypothesis H3, which states that combining both feedback types does not provide any additional information over using just auditory or just vibration feedback, and thus should not effect further changes in user performance. To evaluate this hypothesis, the conditions A and V can be compared to the condition VA. However, while the nRTM for these three conditions has been found to be equivalent, the nERR of VA is not equivalent to that of A or V. Thus, hypothesis H3 cannot be confirmed conclusively. However, the results show a strong similarity, and the fact that no equivalence could be determined can be attributed to the high variance of the nERR measurements. Thus, future studies could focus on this aspect in order to fully confirm or reject this hypothesis.

Lastly, hypothesis H4 postulates that because surface force feedback provides a different type of information than auditory or vibration feedback, its impact on the results should be independent of other conditions. When regarding the results, one can observe that the conditions SA, SV, and SVA all show a significant difference to the condition S for both nRTM and nERR. Furthermore, nRTM also changes significantly when comparing A with SA, V with SV, and VA with SVA. For nERR, no such difference can be found, but this can be attributed to the stronger impact of vibration and auditory feedback. Due to the differences in the results, H4 has been confirmed.

In addition to the hypothesis, additional interesting observations can be made. The results show an effect when combining both surface force feedback and auditory feedback. While the conditions A and V were found to be equivalent, the conditions SA and SV show a slightly lower nERR for SV. While the difference is not significant, the trend is interesting to observe because it may indicate a cross-modal effect. Since the condition SA requires the users to use two separate senses, the split focus of attention may lead to a decrease in performance. Verifying and further examining this effect would be an interesting topic of a further, dedicated user study.

All in all, the results have shown that the influence of haptic vibration and auditory feedback on the user performance are very similar. Both the nRTM and nERR are equivalent for both individual conditions, and additionally, a combination of both does not produce a significant performance change. Thus, feedback from haptic vibrations as used in this study's scenario can be substituted by similarly modeled auditory feedback without noticeable degradation of performance. Furthermore, users rated the auditory feedback as more important for the feeling of involvement, indicating that it is more important for achieving a high immersion.

In the study design, great care was taken to design the auditory and vibration feedback similarly, by ensuring that both convey the same type of information to the user and that both are modeled based on the same simulation parameters. Based on the similar results from feedback of similar quality, it can be concluded that humans perceive such vibrations similarly well with both their auditory and haptic senses.

While auditory and vibration feedback were modeled to be similar, surface force feedback provides a different kind of information. It was included in the study to evaluate the usability of substituting haptics with sound for different haptic feedback types with varying degree of similarity. Since the impact of surface force feedback is mostly independent from the auditory condition, it cannot be directly substituted. In this study, the user benefited more from the vibration and auditory feedback types than from surface force feedback, and thus such a scenario is well suited for a substitution of haptic feedback by auditory feedback. However, other scenarios – for example, simulating surgical cutting, object stacking, or grabbing – would benefit more from feedback indicating surface contact strength. For such scenarios, haptic feedback is more beneficial, as is also shown by some of the related work (see Section 3.1). Such lasting surface contacts often cannot be properly conveyed using realistic sound. While an initial collision can produce a characteristic sound, there are many scenarios that use long-lasting contacts with varying contact point and strength, which often do not produce any sound. For such applications, substituting the haptic feedback with auditory feedback would be difficult, especially if the auditory feedback should remain realistic. While one could use artificial auditory feedback – e.g. by representing contact force by changing pitch – this would feel abstract and unrealistic, and would probably reduce the feeling of presence.

In this study, a drilling task was used as a scenario. The results of the study should apply to similar tasks where a comparable kind of vibrational feedback is provided – for example when using grinders, saws, or other kinds of drills. Other scenarios using feedback based on vibration, like driving or flight simulators where the revolution speed of engines is conveyed both haptically and by sound, are less comparable. Future studies could investigate such different scenarios to further examine the suitability of sound to substitute haptic feedback.

CHAPTER 4

RUN-TIME MODAL ANALYSIS

In the previous chapter, the usefulness of sound synthesis for interactive applications was demonstrated. To synthesize sound in real-time, different approaches have been discussed in Chapter 2, with a focus on Modal Synthesis (MS), which uses the vibration modes of an object to generate contact sounds. For this, a Modal Analysis (MA) must be computed to determine these modes.

As described in Section 2.2.1, MA performs a diagonalization of the dynamics system that models the physical response of a solid object to external forces. By doing this, the individual Degrees-of-Freedom (DoFs) are decoupled, and corresponding modes are extracted. However, the computation of modal data of an object for sound synthesis using a MA is often very time-consuming. Thus, a MA is typically performed in a pre-processing step, and the results are stored for a usage at run-time. However, pre-processing is only possible for objects whose geometry and material properties are known beforehand. In VR and other interactive applications, this is not always the case. For example, it may be possible to interactively change the material properties of objects, e.g. to change the material of walls and floors. Similarly, the user may modify the geometry of an existing object, for example by scaling it or by using interactive sculpting methods. Geometry modifications may also be generated by simulations, like physically-simulated plastic deformations. Furthermore, new objects can be generated at run-time, whose material or geometry may not be known upfront. For example, a user may interactively sculpt a new object from virtual clay, or a procedurally cracking glass plate produces dynamically generated shards (see Figure 4.1). In these cases, the MA cannot be pre-computed. To still enable these objects to produce sound using MS, one must calculate the corresponding modal data.

It is not always necessary to compute a new MA when an object is modified. If only the density, Young modulus, or damping parameters are changed, the results of a prior MA can be adjusted to update the results. However, if the Poisson ratio or the actual geometry is changed, a new MA has to be computed. When a pre-processing is not possible, the MA has to be computed at run-time, which poses a significant challenge due to its high computational cost.

A typical MA requires minutes or hours to compute, which is unsuited for real-time applications. When computing a MA at run-time for a new or modified object, it is important to adhere to the constraints of real-time applications. In order to produce sound, the modal

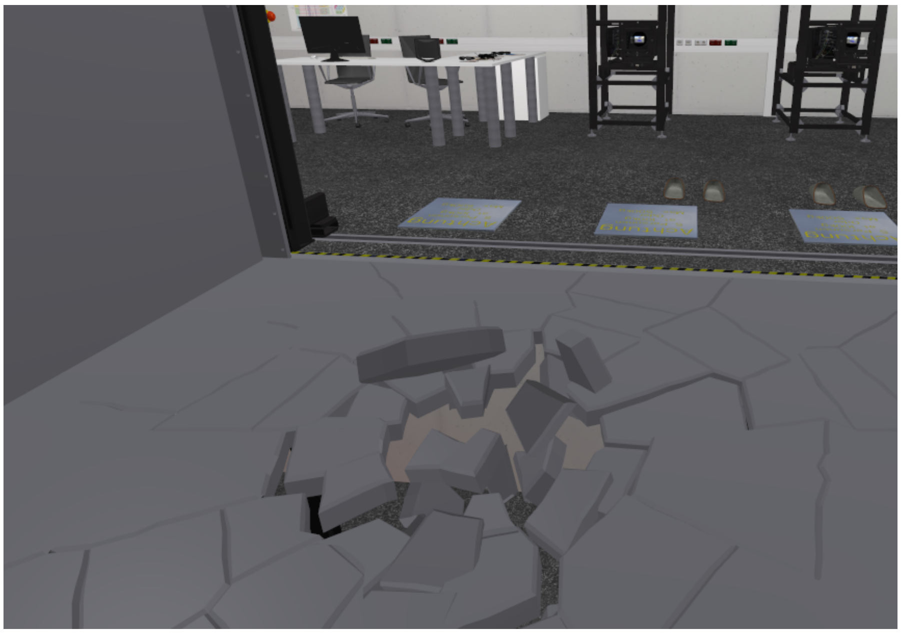


Figure 4.1: A dynamic fracture of a glass floor produces multiple new objects that are procedurally generated. Since their geometry is determined at run-time, a MA cannot be computed upfront.

result has to be available as soon as an auditory event, like a collision of the object with the scenery, occurs. Depending on the current scene and interactive behavior of the objects as well as the user's actions, such a sonic event can occur after a longer time span, or almost immediately. For example, when an object fractures into several smaller objects that are physically simulated, some of these collide almost instantly. To handle such a case, the modal results of multiple objects have to be computed very quickly. While one could delay the synthesis of sounds generated in order to finish computing the modal analysis, this would introduce additional audio latency. Since this can lead to audio-visual asynchronicity, the delay should be very small. This sets a tight time constraint for the modal analysis.

This chapter presents novel methods to allow the computation of a MA at run-time, so that dynamically created or modified objects can produce modal sounds. This requires that the modal results are available quickly enough, which is challenging due to the high run-time cost of a MA. To achieve this, different Levels-of-Detail (LoDs) will be used to reduce the complexity of the MA, and the usage of multiple workers allows an efficient distribution of tasks to handle multiple LoDs and objects simultaneously.

This chapter is structured as follows. Section 4.1 presents related work on the area that examines approaches to speed up MA computations, as well as alternate methods to generate or modify modal results interactively. The time constraints for a run-time MA are discussed in Section 4.2, followed by details about the general procedure and individual steps of the MA computation (see Section 4.3). To determine the time requirements of a MA for geometries of different complexity, its performance is benchmarked (see Section 4.4). In Section 4.5, different geometric LoDs are presented, which allow trading computation speed for accuracy.

To reduce the effect of low-resolution meshes on the result, a correction factor is deduced in Section 4.6. The quality loss caused by the different LoDs is examined in Section 4.7 using several example geometries. A setup for computing multiple incremental LoDs at run-time will be presented in Section 4.8. Section 4.9 concludes with a discussion of the presented approaches.

4.1 Related Work

Considering the run-time generation or modification of modal results, little research has been performed yet.

To speed up the computation of a modal analysis – although without a focus on real-time applications – Picard et al. propose a multiscale FEM approach. Multiple resolution levels are constructed by using a hierarchical voxel volume, where lower resolutions are assembled from the mass and stiffness matrices of higher resolutions [Picard et al. 2009a; Picard et al. 2009b]. They found that a very low grid resolution of $5 \times 5 \times 5$ may be sufficient for some objects, for which the total analysis requires $\sim$3.5 s. This is reasonably fast, yet does not fully meet the time constraints for run-time analyses even for such a low resolution. Additionally, it only works well if objects can be approximated using uniformly sized voxels, which can be problematic for many objects, especially those with thin shells.

Using a different approach to speed up the analysis, Maxwell utilizes rotational symmetry of bells to reduce the problem size [Maxwell 2008]. While this significantly speeds up the process while maintaining a high quality, it is only applicable to a specialized class of objects, and still requires several seconds for the analysis.

Maxwell also examined the possibilities of adjusting the modal result of an object when it undergoes deformation [Maxwell 2008; Bruyns-Maxwell 2007; Maxwell et al. 2007]. By modifying the prior Eigendecomposition of the undeformed object, the frequencies of the modes can be adjusted accordingly without having to recompute the expensive Eigendecomposition The authors report a decent approximation quality as long as the scale of the deformation is moderate. However, due to the limit to previously known objects with moderate deformations, this is not applicable to most scenarios where a run-time MA is required.

Another approach to gain approximate modal results at run-time is the retrieval of pre-calculated results from a database. For this, Glondu et al. propose a database of pre-computed proxy shapes, which stores information used for physical animation, including vibration modes [Glondu et al. 2011]. By extracting a mesh description for new objects, the best-matching entry in the database can be retrieved and is scaled and rotated to match the initial model. Since this approach uses the vibration modes for visual animation, it suffices to only store the lowest vibration modes. For sound synthesis, however, a higher number of modes should be used, increasing the memory required for storage. Also, since modes depend heavily on the shape and scale of an object, a large number of pre-computed shapes would have to be stored in order to produce high-quality results. Such a database would require a very high amount of memory and pre-computation time. Thus, for MS such an approach is mostly reasonable if a limited number of object variations are to be expected, so that the database can be specialized.

4.2 Run-Time Time Constraints

Since none of the approaches from related work can reliably provide matching modal results during run-time, this chapter will present approaches for a run-time MA. When requesting a run-time MA of a new or modified object, it should be finished before the object first produces sound. Sounds are caused by an auditory event, which for a MS is an external force acting on the new object, which usually originates from contacts. To determine the time available for a MA at run-time, two aspects are important: when can an auditory event first occur, and how much delay can be tolerated between the auditory event and the produced sound. Regarding the occurrence of auditory events, few assumptions can be made without specific knowledge of the scene, and thus one has to expect that new objects may start producing modal sounds almost immediately. For example, when a window fractures into dynamically computed shards, these shards are almost immediately colliding with one another and should produce corresponding sounds.

Contacts forces between objects in VEs are usually computed by a physics engine that simulates the movement and collision responses of objects in the scene. Thus, a first collision of a newly requested modal object can occur after the next physics simulation step. Thus, the physics time step is a good estimate for the time available to compute a first modal result. While the rate of the physical simulation may vary, a commonly used update rate is $60-100\,\mathrm{Hz}$, corresponding to time spans of 10 to $16\,\mathrm{ms}$.

When a run-time MA lasts too long so that an auditory event occurs before a first modal result is available, the sound output must be delayed until the MA finishes. This leads to a delay of the sound output, and thus a higher latency of sound in relation to the visual feedback. While few fixed guidelines exist, the delay between visual and audio reproduction of an event – e.g. a collision – should be very low. The audio rendering pipeline, including auralization, spatialization etc., is typically set up to utilize the available latency tolerance, and thus an additional delay introduced by the modal sound synthesis – including the analysis – should be kept minimal.

For a run-time MA as presented in this thesis, the goal is to avoid an addition of significant audio latency. Thus, the primary aim is to maintain a limit of $16\,\mathrm{ms}$ – i.e. approximately one physics simulation frame – to compute the first modal results. This allows synthesizing modal sounds of new objects almost immediately but puts strict limits on the possible complexity of the analyzed objects.

4.3 Modal Analysis Procedure

The Modal Analysis (MA) – whether it is performed at run-time or as a pre-processing step – computes the modal data required for the modal sound synthesis. The physical foundations and mathematical methods used to perform a MA are described in detail in Section 2.2. To compute modal data suited for interactive modal sound synthesis, however, more steps have to be performed, both to compute the required input data for the mathematical analysis, and to process the results and allow an efficient run-time synthesis. These steps will be detailed here.

4.3.1 Modal Analysis Input

One of the major advantages of MA is that it only requires a physical description to compute the sounds of arbitrarily shaped rigid objects with different material properties. Thus, as input to the modal analysis process, it suffices to provide the material parameters and geometry of the object.

The material parameters that are required depend on the method used to assemble the physics matrices. For this thesis, all matrices are assembled using a standard Finite Element formulation and thus require material parameters for the *Young Modulus E* and *Poisson Ratio* ν. Additionally, to model the decay of the resulting modes, material parameters for the *stiffness* and *mass damping*, α and β, are required.

For the input geometry, the requirements again depend on the method used to construct the physics matrices. For the tetrahedral finite element method used in this work, a tetrahedral volume mesh is required. However, such a mesh is usually not available for most objects used in VEs. Instead, a visual surface mesh is commonly used, and the designer specifies the type of volume this mesh should represent, as well as the desired resolution. The physics mesh is then created by processing the input geometry.

4.3.2 Geometry Processing

The input geometry is usually a surface mesh, while the construction of physics matrices requires tetrahedral volume meshes. To allow an easy utilization of modal sound synthesis for arbitrary objects, a volume mesh is automatically extracted from the input triangle meshes representing the visual geometry. For this, two different methods are used in this work: thin-shell meshes and solid volume meshes.

Thin-shell meshes are constructed from the surface mesh by extruding all triangles in normal direction. For each vertex, its normal is computed, and a second vertex is created, which is moved along the normal by the desired shell thickness. For each triangle, four tetrahedra are inserted to fill the volume between the original and the extruded surfaces. Optionally, this process can be repeated to create multiple shell layers. Thin shells can work on arbitrary meshes, including open and non-two-manifold meshes. The only requirements are that the normal directions are uniformly oriented and that the shell thickness is small compared to feature sizes. The latter requirement is important to prevent tetrahedron inversion that can occur when the extruded mesh intersects itself.

Solid volumes have a closed outer surface that is completely filled. Such a mesh can be computed from a surface mesh using tetrahedral meshing – for this work, tetgen[1] and CGAL's 3D Mesh Generation package[2] are used. However, tetrahedralization requires closed, two-manifold surface meshes as input, thus only some objects can be meshed without further

[1]`http://wias-berlin.de/software/tetgen`

[2]`http://doc.cgal.org/latest/Mesh\_3/index.html\#Chapter\_3D\_Mesh\_Generation`

processing. Therefore, the mesh is cleaned and any holes are closed before computing the solid tetrahedral mesh.

In both cases, additional steps are performed to meet geometric resolution requirements. Models designed for rendering are optimized for visual quality, usually creating the visual impression using a combination of polygons and textures. Thus, the actual geometry is often non-uniformly meshed, with triangles of varying size and shape. The physics mesh for the MA has different requirements, however. Especially important is a decent resolution and uniformity of the triangles (in both shape and size). For example, large planar areas are commonly modeled using few, large polygons for visual models. However, a suitable physics mesh would require a reasonably fine sub-sampling of this area to properly model deformations and vibrations in this area. Therefore, the resolution and mesh quality of the final mesh has to be improved using mesh refinement and optimization, as detailed in Section 4.5.1.

4.3.3 Physics Matrix Assembly

From the tetrahedral mesh, the mass matrix $\mathbf{M}$ and stiffness matrix $\mathbf{K}$ for the MA are assembled. For this work, a classical tetrahedral finite element formulation is used for the stiffness matrix. The matrices are assembled using standard finite element methods [Bathe 2006], detailed in Appendix A.

The mass matrix can be assembled using an exact formulation based on the element mass matrices of each tetrahedron. Alternatively, it is possible to use a lumped-mass formulation. The latter introduces a small error, but produces a diagonal mass matrices, where all non-diagonal entries are zero. This allows an easier computation of the Cholesky factorization of the mass matrix (see Section 2.2.1), which accelerates the Eigendecomposition process. The choice between lumped and full mass matrix can be made for each object individually. Since the error introduced by the lumped mass formulation is rather low, it is better suited for a run-time MA where performance is critical. Thus, it will be utilized for all examples in this chapter.

4.3.4 Eigendecomposition

The Eigendecomposition is an important step of the MA, as it computes the decoupled ordinary differential equations representing the modal vibrations. It takes the physics matrices $\mathbf{M}$ and $\mathbf{K}$ of size $3n \times 3n$ as input, and computes the eigenvalues λ_i and eigenvectors $\mathbf{V} = [\ldots \mathbf{v}_i \ldots]$, as described in Section 2.2.1.

4.3.5 Mode Processing

After the eigenvalues and -vectors have been computed, the characteristic properties of the modes can be derived. Each eigenvalue λ_i corresponds to one mode, with an angular frequency $\omega_i = \frac{1}{2}\sqrt{(\alpha \cdot \lambda_i + \beta)^2 - 4 \cdot \lambda_i}$ and a decay value $\gamma_i = -\frac{1}{2}(\alpha \cdot \lambda_i + \beta)$. Also, the mode shape $\mathbf{s}_i$, which describes the deformation induced by the mode, is derived from the corresponding eigenvector $\mathbf{v}_i$. The gain matrix $\mathbf{V}^T \cdot \mathbf{L}^{-1}$ is computed to determine of external forces (see Section 2.2.1).

This information is sufficient to simulate the modal vibrations, since it allows computing the excitation from external forces, as well as the damping, frequency, and mode shape that describe the dampened oscillation of modes after an excitation. However, for a run-time synthesis, additional optimizations can be performed to increase performance.

First of all, the number of modes can be reduced. A first step is to discard modes with a frequency of zero. For objects without boundary conditions, there are six modes with an eigenvalue of zero, which correspond to rigid-body motion, i.e. translation and rotation. Since these modes do not represent vibrations, they can be discarded. Additionally, humans only hear sound with frequencies between 20 Hz and 22 kHz, and therefore all modes outside this range can be omitted.

Another possible optimization is to reduce the gain matrix. The gain matrix is used to determine the excitation produced by external forces (see Section 5.3). The matrix is of size $m \times 3n$ for an object with m modes and n physical mesh vertices. Since external forces can only act on surface vertices, one can discard all columns corresponding to internal vertices, which can significantly reduce the size of the gain matrix. The remaining gain matrix is of size $m \times 3s$, containing three columns for each of the s surface vertices, corresponding to its three DoFs.

Each row of the Eigenvalue matrix determines the distribution of external forces onto the m modes for a force acting on the vertex along the corresponding direction. Usually, only a limited amount of modes receive a notable excitation from any given vertex. If the gain of a mode is significantly smaller than the highest gains, it can be omitted for the synthesis. Storing the gains separately for each vertex instead of as a matrix, these negligible entries can be dropped. This further reduces memory requirements and also reduces the number of modes affected by external forces, which can speed up the sound synthesis (see Section 5.3).

The pre-processing of the gain matrix can be enhanced by examining the acoustic transfer coefficient (see Section 4.3.6) of the respective modes. Often, a single mode's vibration leads to mostly longitudinal or mostly transverse vibrations (see Figure 4.2). However, sound is only produced by transversal surface movement, and thus modes that produce mostly longitudinal deformations typically have very small acoustic transfer factors. By considering the transfer factor in the truncation of values from the gain matrix, more entries can be discarded.

Optionally, it is possible to reduce the mode count by removing high-frequency modes. For modal synthesis, the damping is computed as a combination of mass and stiffness damping, $\gamma_i = -\frac{1}{2}(\alpha \cdot \lambda_i + \beta)$. Thus, modes with a higher frequency – corresponding to a higher eigenvalue λ – are more strongly damped than modes with a lower eigenvalue, and thus become inaudible more quickly. However, high-frequency modes can have audible contributions to the resulting sound directly after an excitation, or may even be continuously excited – e.g. by scraping or sliding interaction. Thus, it is advisable to utilize all modes in the audible frequency range if the performance allows it. Therefore, no high-frequency modes will be omitted for examples and benchmarks in this thesis.

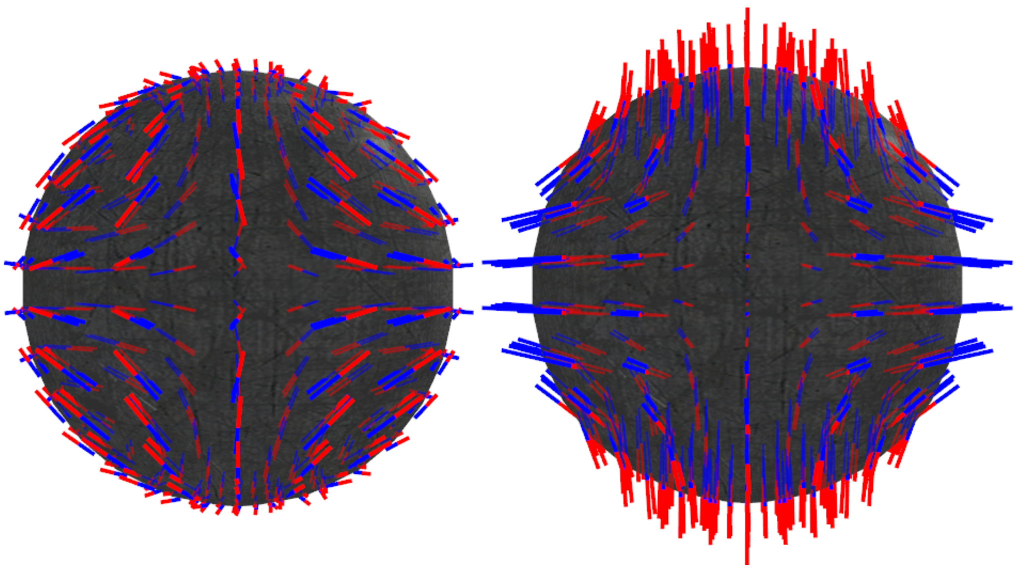


Figure 4.2: Modes inducing mainly (a) longitudinal or (b) transverse deformations in a sphere. The red/blue lines indicate the vertex displacement induced by the mode.

4.3.6 Acoustic Transfer Computation

As mentioned before, different modes produce different deformations of the object surface. These surface vibrations induce pressure changes in the surrounding air, thereby producing sound. The surface deformation is different for each mode. To model this, the acoustic transfer for each mode has to be computed. There are different methods to achieve this, which are discussed in Section 2.2.3. To properly model the near-field effects of sound radiation, a costly simulation of the acoustic transfer is required, e.g. using the Boundary Element Method (BEM). The sound field can then be approximated using multiple dipole sources in order to allow an interactive synthesis at run-time [James et al. 2006]. When the MA is computed in a pre-processing step, the added computation cost does not pose a big problem. For a run-time MA, however, it is currently too slow.

For the presented case, a simplified variant is used, which is based on a far-field approximation of the sound radiation of a sphere [Cremer et al. 1985; James et al. 2006]. This models a uniform sound radiation, i.e. as an acoustic monopole. While this neglects sound directivity and near-field effects, which occur especially for low-frequency modes and close distances, it has a high performance and is a suitable approach for MS for interactive applications. We follow the approach of O'Brien et al. [O'Brien et al. 2002], where the acoustic transfer factor q_i for mode i is computed as follows:

$$q = \omega \cdot \sum_{f \in F} \mathbf{n_f} \cdot \mathbf{A}_F \tag{4.1}$$

Here, F is the set of all surface triangles, $\mathbf{n_f}$ is a triangle's normal, and $\mathbf{A}_F$ is the volume between the triangles rest state, and its state after being deformed by the ith mode. This approach computes an acoustic transfer factor q for each mode, which is multiplied with the mode's relative amplitude to gain the corresponding sound contribution (see Section 5.2).

4.3.7 Collision Data Structures

When external forces act on an object, these forces excite modes so that a corresponding sound is generated. For this, the gain matrix is used, which stores the mode excitation by forces acting on vertices of the mesh. Thus, to excite the modes, it is necessary to determine the vertices of the physics mesh that are affected by these forces. Since the physics mesh is specifically created for the MA, it is not possible to directly use results from other collision data structures of the object – e.g. the one used by the rigid body physics engine.

Thus, a collision data structure is required to compute the triangle of the physical mesh on which the force acts. In this thesis, AABB trees are used as triangle collision data structure. Then, barycentric coordinates are used to distribute the force on the triangle's vertices. By providing neighborhood information, it is furthermore possible to distribute the force over adjacent triangles, e.g. to simulate extended collision areas.

4.4 Modal Analysis Performance

To compute the modal data required for MS, all the aforementioned steps have to be performed. When calculating a MA at run-time, the complete computation has to be performed fast enough so that the result is available when a first auditory event occurs. As discussed in Section 4.2, this puts strict time constraints on the computation – if possible, a modal result should be computed in less than 16 ms. To gauge the number of vertices that an object can have to still allows an analysis in a given time, different benchmarks are performed. First, the performance of different toolkits and methods for the Eigendecomposition are examined for their performance and suitability for MA computations. Afterwards, the complete MA process is benchmarked.

4.4.1 Eigendecomposition Benchmark

While a MA consists of different steps, the most time-consuming one is the Eigendecomposition of the physics matrices. As described in Section 2.2.1, the Eigendecomposition is a central element of the MA process and allows the decoupling into mutually independent vibration modes. Typically, the complexity of an Eigendecomposition is $O(n^3)$ for a matrix of size $n \times n$, and thus the computation time quickly rises with increasing geometric complexity. Eigendecomposition algorithms with a lower asymptotic complexity of approximately $O(n^{2.37}$ exist [Demmel et al. 2007]. However, these come with larger constant factors and thus are only beneficial for very large problem sizes.

To determine the maximum geometric complexity that still allows for a run-time analysis, a comparative benchmark of different math packages and their routines for the symmetric real Eigendecomposition was performed. The examined packages are LAPACK[3], the optimized

[3]version 3.4.2, default BLAS, `http://www.netlib.org/lapack/`

LAPACK routines of the Intel Math Kernel Library MKL[4], Eigen[5], and Meschach[6]. Since the decomposition matrix is inherently sparse, it is also possible to use sparse matrix routines, for which MKL's `feast` routines as well as ARPACK[7] have been examined. Furthermore, the high computation power of GPUs can be utilized for the Eigendecomposition. For this, the benchmark includes the Eigendecomposition routines of CULA[8], a toolkit that implements BLAS/LAPACK routines on the GPU.

LAPACK, MKL, and CULA provide multiple Eigendecomposition routines, which follow the LAPACK naming convention: `ssyev` and `ssyevx` using a traditional QR decomposition [Wilkinson et al. 1965], `ssyevd` uses a divide-and-conquer approach [Cuppen 1980], and `ssyevr` utilizes the relatively robust representation method [Parlett et al. 2000].

Although all tested decomposition routines also provide double precision variants, tests have shown that computation using single precision floating point numbers produces sufficiently accurate results. Thus, only results for single-precision computations are presented here. Double-precision computations require 1.3 to 2 times longer, depending on the problem size and used method.

When using a modal analysis for applications like vibration analysis in mechanical engineering or deformation modeling in computer animation, it is common to only compute the lowest vibration modes, since these are the ones that produce the strongest deformation in an object. For modal sound synthesis, however, all modes in the audible ranges are important. While higher-frequency modes produce smaller deformations, their higher frequency leads to similar sound pressure as for vibrations of lower-frequency modes. While high-frequency modes typically have a stronger damping and thus decay faster than lower-frequency modes, they still play an important role in the produces sound, and thus computing only a small number of modes would degrade the performance.

Thus, for modal sound synthesis, one should either compute all eigenvalues and -vectors, or those that correspond to modal vibrations in the audible frequency range of $20\,\text{Hz} - 22\,\text{kHz}$. Since the eigenvalues correspond to the undampened characteristic vibration frequency (see Section 2.2.1), one can limit the computation of eigenvalues to a suitable range. This can speed up the computation and reduce memory requirements. However, it introduces a small error in the results, since the eigenvectors are normalized. Omitting entries leads to a different scaling of the gain matrix that is computed from the eigenvectors, and thus an external force may have too high an influence because it is distributed to fewer modes. Most notable, the six rigid body modes typically receive a high fraction of the acting forces. Thus, when omitting modes outside the audible frequency range, the volume of synthesized sounds would be overestimated. Still, since a limited computation range can speed up the computation, benchmark results are also provided for Eigendecomposition variants computing only a limited range of eigenvalues.

When computing a limited frequency range, the number of modes that lie in the audible range depends on the object's geometry and material. This fraction varies significantly,

[4]version 11.0, `https://software.intel.com/en-us/intel-mkl`

[5]version 3.2.0, `http://eigen.tuxfamily.org`

[6]version 1.2, `http://homepage.math.uiowa.edu/~dstewart/meschach`

[7]version 1.1, using ARPACK++ v1.2 and SuperLU v4.3, `www.caam.rice.edu/software/ARPACK/`

[8]version R17, `http://http://www.culatools.com`

			num physical vertices						
package	routine		32	64	128	256	512	1024	2048
Eigen	SelfAdj.		**0.001**	0.006	0.043	0.338	2.671	27.636	255.814
Meschach	symmeig		0.108	0.672	5.440	63.088	520.611	6758.420	—
LAPACK	ssyev		0.003	0.022	0.157	1.172	9.390	77.511	652.003
	ssyevd		0.022	0.022	0.140	1.000	7.673	—	—
	ssyevx		0.003	0.022	0.157	1.175	9.405	77.952	656.040
	ssyevx	L	0.003	0.019	0.129	0.941	7.440	57.528	475.687
	ssyevr		0.003	0.019	0.122	0.868	6.815	51.921	416.693
	ssyevr	L	0.003	0.019	0.128	0.935	7.481	58.010	477.478
MKL	ssyev		0.002	0.018	0.132	0.943	5.544	50.330	325.325
	ssyevd		0.001	**0.004**	**0.023**	**0.140**	**1.032**	7.744	67.442
	ssyevx		0.002	0.018	0.131	0.943	5.631	50.625	328.255
	ssyevx	L	0.002	0.007	0.035	0.210	1.476	11.544	120.444
	ssyevr		0.001	0.006	0.033	0.189	1.314	9.021	72.401
	ssyevr	L	0.002	0.007	0.035	0.210	1.502	11.824	119.475
	feast	L	0.118	0.516	2.473	12.702	64.761	323.583	—
ARPACK	regular	L	0.003	0.016	0.108	0.786	5.790	60.742	554.824
	shift-invert	L	0.004	0.022	0.140	0.860	5.613	57.248	504.761
CULA	ssyev		0.023	0.054	0.132	0.409	1.480	**6.127**	26.735
	ssyevx		0.024	0.055	0.134	0.422	1.489	6.147	**26.637**
	ssyevx	L	0.011	0.029	0.080	0.293	1.500	11.002	102.801

Table 4.1: Average computation times for different math packages and their Eigendecomposition routines for different problem sizes. Times are given in seconds, for Eigendecompositions of matrices of size $3n \times 3n$ for physical meshes with n vertices. If an L is added to a method name, only eigenvalues in the audible frequency range are computed. Bold numbers represent the fastest result for the problem size.

however. Large or soft objects with low-resolution meshes may result in very few inaudible modes. On the other hand, high-resolution models with a high-frequency base mode can have more than 90% of inaudible modes. An increase in geometric resolution mostly adds high-frequency modes so that a higher percentage would be discarded. On the other hand, low-resolution objects typically produce fewer modes and thus have very few inaudible modes. For a run-time MA, where the geometric resolution has to be kept small, the percentage of inaudible modes is typically rather low. For this benchmark, the material properties of the tested objects were chosen such that 50% of the computed modes were in the audible range.

In the benchmark, the run-times of the different Eigendecomposition routines are measured. Results for CPU-based routines were computed on a PC with two Intel Xeon X5650 CPUs (each with 6 cores at 2.67 GHz) and 24 GB RAM. The test program was compiled with gcc 4.8. The CULA benchmark was performed on a PC with an Nvidia GeForce GTX 480, and

			num phyiscal vertices						
package	routine		32	64	128	256	512	1024	2048
Eigen	SelfAdj.		**0.001**	0.006	0.058	0.530	3.721	36.776	364.075
Meschach	symmeig		0.167	1.148	9.698	116.128	887.559	12345.300	—
LAPACK	ssyev		0.004	0.024	0.174	1.413	11.829	87.278	696.196
	ssyevd		0.004	0.023	0.149	1.175	9.632	—	—
	ssyevx		0.004	0.024	0.175	1.422	11.935	87.428	704.293
	ssyevx	L	0.004	0.023	0.132	0.956	9.256	61.526	519.105
	ssyevr		0.004	0.020	0.127	0.950	8.474	56.067	456.138
	ssyevr	L	0.003	0.020	0.131	0.961	9.055	63.037	520.160
MKL	ssyev		0.004	0.024	0.176	1.141	6.372	67.442	453.372
	ssyevd		0.001	**0.005**	**0.025**	**0.160**	**1.635**	9.405	86.395
	ssyevx		0.004	0.025	0.176	1.075	6.512	72.555	451.400
	ssyevx	L	0.002	0.007	0.036	0.232	2.130	13.899	144.718
	ssyevr		0.002	0.007	0.036	0.209	1.945	11.993	102.893
	ssyevr	L	0.002	0.007	0.039	0.231	2.239	14.488	152.114
	feast	L	0.135	0.575	2.816	14.389	71.699	353.991	—
ARPACK	regular	L	0.004	0.020	0.132	0.908	6.569	76.886	691.382
	shift-invert	L	0.005	0.025	0.159	0.980	6.586	72.419	576.269
CULA	ssyev		0.026	0.059	0.142	0.476	1.657	**7.139**	29.204
	ssyevx		0.026	0.059	0.150	0.500	1.721	7.146	**29.131**
	ssyevx	L	0.012	0.032	0.088	0.341	1.727	13.784	121.693

Table 4.2: Maximum computation time for different math packages and their Eigendecomposition routines for different problem sizes. Times are given in seconds, for Eigendecompositions of matrices of size $3n \times 3n$ for physical meshes with n vertices. If an L is added to a method name, only eigenvalues in the audible frequency range are computed. Bold numbers represent the fastest result for the problem size.

compiled with Visual Studio 2013.

Since the decomposition time can vary depending on the input matrix, multiple example matrices were tested in the benchmark. These correspond to 15 different physical meshes representing different geometries, including geometric primitives, the Standford bunny, Stanford armadillo, and Stanford dragon, a wine glass, a tea glass, and a spoon. To achieve a given geometric complexity, they were reduced to the desired vertex count and extruded to form a thin-shell mesh. The matrices were assembled using a tetrahedral Finite Element formulation, with a Poisson Ratio 0.29 and a density of $7850\,\frac{\text{kg}}{\text{m}^3}$. The Young Modulus was adjusted for each object such that 50% of the resulting modes lay in the audible frequency range.

Since a goal of the benchmark is to determine the allowed geometric complexity that can be reliably handled within a given time span, the maximum computation time is given in addition to the average computation time. In the following, the maximum computation time will be used to compare routines and to deduce geometry limits.

The results for the different libraries and routines are listed in Table 4.1 (average computation time) and Table 4.2 (maximum computation time). Due to exceedingly long run-times, results for the largest mesh were omitted for Meschach and `feast`. LAPACK's `ssyevd` results for 1024 and 2048 vertices are missing because the routines failed to compute correct results.

The results of the benchmark show that the MKL `ssy*` routines are considerably faster than other CPU-based algorithms. From the different MKL routines, the `ssyevd` algorithm has a slightly better performance than `ssyevx` and `ssyevr`, and thus presents a good choice for a MA. It is also notable that `ssyevd` – which does not support the calculation of limited eigenvalue ranges – even outperforms variants calculating a limited range of modes for the tested 50% fraction of audible modes. Only for complex geometries and a high percentage of inaudible modes, limiting the modal range improves the performance.

To compare the performance of the dense matrix routines to sparse matrix routines, the sparse matrix packages `feast` and ARPACK were tested. These methods are mainly developed for the calculation of few eigenvalues of very large matrices. They are less suited for the use-case of a run-time MA, where a high number of eigenvalues is computed from comparatively small matrices. This is reflected in the results, where even the fastest methods are at least four times slower than MKL's `ssyevd`. Thus, the sparse matrix routines are not well-suited for a run-time MA. Still, sparse decomposition routines are important when pre-processing complex geometries, where the high memory requirements of dense matrix storage may exceed memory limits. When analyzing geometries with a very high resolution, the corresponding matrix sizes may be too large for a dense matrix storage, exceeding memory storage capabilities. In these cases, the use of sparse matrix routines is advised, although this is only an option during a pre-processing step due to the very high computation times.

The results in Table 4.1 and Table 4.2 were all computed using a single CPU thread. MKL also supports multi-threaded computation based on OpenMP[9], which can further improve the performance. The scalability of MKL's `ssyevd` was benchmarked for different numbers of CPU threads and different matrix sizes. Figure 4.3 shows the speedup of the computation when using multiple CPU threads. It shows that for larger matrices, the performance can be increased significantly by utilizing multi-threading. For small problem sizes, however, the performance gain from using multiple cores is rather lower.

In addition to multi-threaded parallelization on a single PC, the Eigendecomposition can also be performed on a distributed cluster, further increasing the available computation power. As representatives of cluster-capable Eigendecomposition routines, the ScaLAPACK[10] and SlepC[11] packages were examined, which are both using MPI[12]. Similar to single-threaded sparse matrix routines, these libraries focus on solving the Eigendecomposition of very large matrices and the computation of only a few eigenvalues and -vectors. For a run-time MA with

[9] `http://openmp.org`
[10] `http://www.netlib.org/scalapack`
[11] `http://slepc.upv.es`
[12] `http://www.mpi-forum.org/`

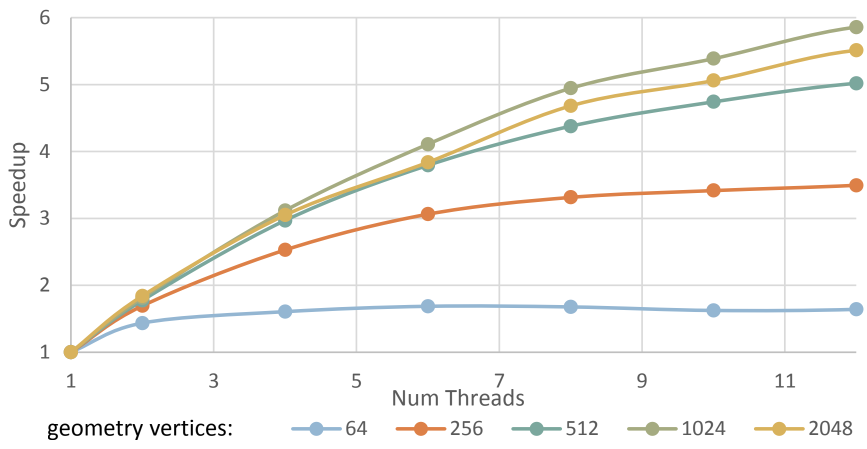


Figure 4.3: Multi-threaded performance of MKL `ssyevd` for different numbers of threads and number of physical mesh vertices.

its smaller problem sizes, however, these methods are not well suited because the distribution overhead is too large. Thus, similar to sparse matrix routines, cluster-based decomposition methods are options for a MA of very complex, high-resolution geometries, but are not useful for the desired run-time MA.

Another approach to speed up an Eigendecomposition is the utilization of GPUs. As a representative, the CULA math library was benchmarked. As the results show, CULA can significantly speed up the Eigendecomposition of large matrices. However, GPU-based algorithms typically come with a constant overhead for starting the GPU computation and also require a high level of parallelism to fully utilize the capabilities of graphics cards. Thus, CULA's `ssyevx` is significantly faster than MKL's `ssyevd` for larger problem sizes. However, when using smaller matrices, CULA is less efficient and becomes slower than the MKL routine. On the used hardware, MKL outperforms CULA when decomposing matrices with less than $\sim$520 vertices, which require around 1.7 s (maximum computation time) using either CULA or MKL.

In consequence, MKL's `ssyevd` proved to be the best method to compute the Eigendecomposition for the run-time MA with its limited matrix sizes, and this method will be used in following MA benchmarks. The GPU-based CULA routines are a good alternative for larger matrices if a powerful GPU is available. By using MKL's `ssyevd`, one can compute an Eigendecomposition of geometries with $\sim$90 physical vertices within the time-frame of 16 ms on the used hardware. While different hardware may lead to higher or lower performance, this does not affect the possible vertex count too much. For the Eigendecomposition, the complexity is $O(n^3)$ for n vertices, so for example doubling the performance would only allow $\sqrt[3]{2} \approx 1.26$ times more vertices. Therefore, the vertex estimates are a good guideline on most current processors, and will only slowly increase in the future.

4.4.2 Complete Modal Analysis

A MA has to perform several steps to compute the modal data that allows a real-time modal synthesis. As detailled in Section 4.3, these steps include the geometry preparation (including construction of collision data structures), physics matrix assembly, Eigendecomposition, and mode processing (including acoustic transfer computation).

In the previous section, a benchmark of different Eigendecomposition routines was presented. The Eigendecomposition is usually the most time-consuming operation of the MA process. However, other components may still have a notable influence on the computation time, especially for smaller resolutions. Thus, another benchmark was performed that includes all of the analysis steps.

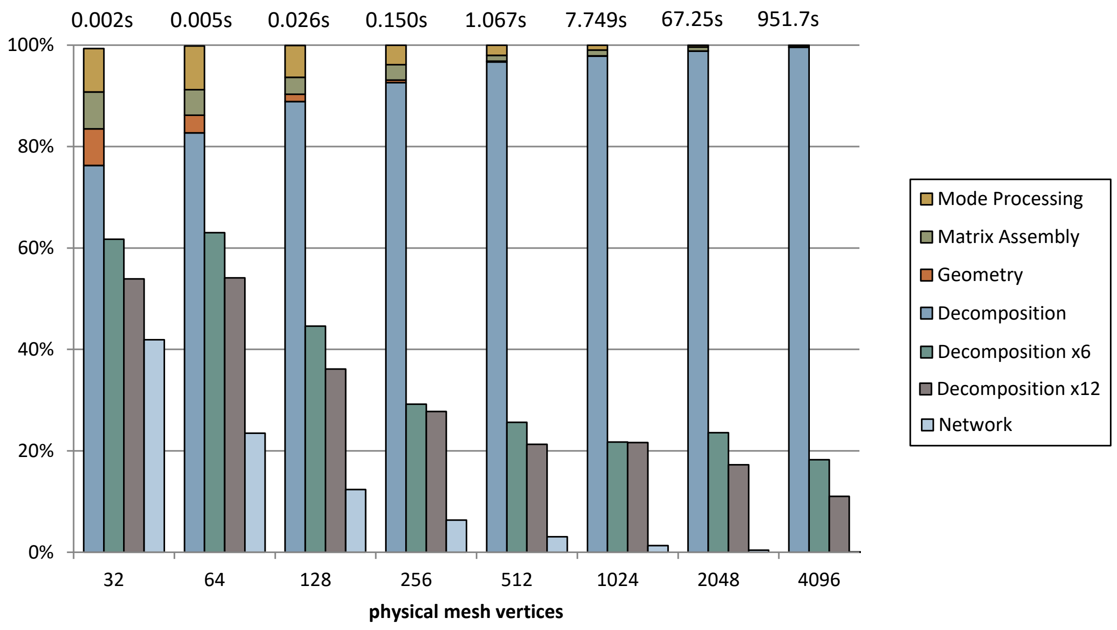
The performance of the complete MA was measured on the same hardware and with the same sample geometries as the Eigendecomposition benchmark (see Section 4.4.1). The Eigendecomposition was computed using MKL `ssyevd` using a single CPU thread, as well as using 6 or 12 threads for comparison.

Figure 4.4 shows the average and maximum computation time for geometries of different complexity and lists the relative computation time of the individual steps. As the results show, the Eigendecomposition is the most time-consuming step even for geometries with few vertices, and the computation time for other steps becomes negligible for more complex geometries. Corresponding to the results of the Eigendecomposition benchmark, using multiple CPU threads accelerates the MA computation of more complex geometries, but only provides small performance gains for smaller geometries.

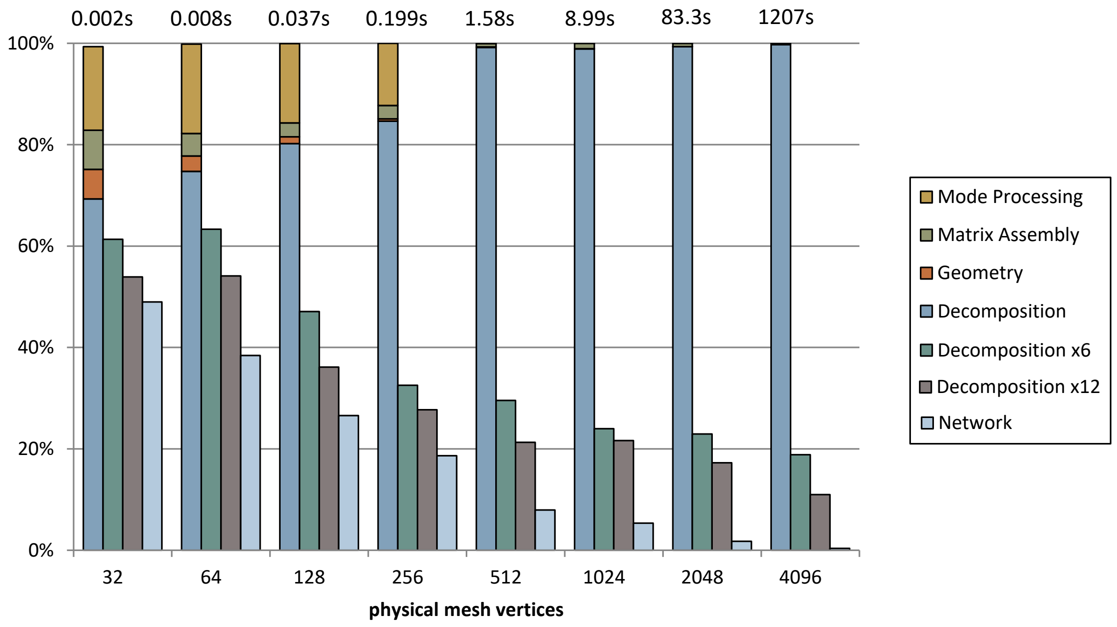
When computing a MA locally on a single CPU thread, it is possible to analyze geometries with approximately 90 physical mesh vertices within the time limit of 16 ms based on the average computation time. When using the maximum computation times as a reference, it is still possible to analyze geometries with around 80 physical vertices. Using a multi-threaded Eigendecomposition for such small geometries would only allow for few additional vertices.

While one can simply compute the run-time MA on the sound server – i.e. the PC where the resulting modal data is used to synthesize the modal sounds – this is not always the best option. Usually, the sound server has to handle other tasks, like the modal synthesis, sound spatialization, and auralization. It may also run other, non-audio-related components of the application, like interaction handling, the physics engine, or visual rendering. To prevent the run-time MA from slowing down these other processes too much, it is possible to perform the MA computation on a remote worker PC (see Section 4.8). By using multiple remote workers, one can increase the available computational resources so that multiple MAs can be performed simultaneously.

To enable an analysis on a remote worker, network communication is necessary. To start a remote analysis, a request is sent to a remote worker. The request contains the geometry data, material properties, and MA-related settings. Once the query is received by the remote worker, it computes the MA and sends back the result, which consists of the mode data and the gain matrix. The request and response can be rather large, because they contain the geometry data and the gain matrix. Especially the latter one has a significant size, since it contains $3s \times m$ entries for geometries with s surface vertices and m modes, where m may be as high as $3n$. This requires up to $\sim$34 MB of memory for objects with 1000 vertices and $\sim$550 MB for 4000



(a) Average computation time



(b) Maximum computation time

Figure 4.4: (a) Average and (b) maximum computation time required by different steps of the modal analysis for meshes with different vertex counts. Values are given as percentage of total analysis time, which is listed on top. The left-most bars of each group display the steps required for a MA. The Eigendecomposition duration when using multi-threaded computation using a different number of cores are provided as a reference. Additionally, the left-most bar per group lists the relative run-time added when optionally network communication to compute the MA on a remote PC.

vertices. Therefore, the impact of a remote analysis on the computation time is also of interest. Figure 4.4 shows the relative network transfer time – as percentage of the time required for a local MA – for geometries of different complexities, measured using a Gigabit Ethernet. As the results show, the network overhead is significant for small analyses, but becomes less relevant for larger geometries. As a consequence, it is advised to compute smaller objects with up to 100 vertices locally. For more complex geometries, however, a MA on a remote worker can provide more resources while not increasing the MA computation time too much.

4.5 Levels-of-Detail for Modal Analysis

As the previous benchmarks showed, the MA is a computationally expensive process, especially for more complex geometries. When modal data is computed in a pre-processing step, the required time is of limited importance. However, when new objects are created at run-time, or existing ones are changed, it is necessary to calculate a new MA at run-time. For such a run-time MA, the time constraints of an interactive application have to be obeyed (see Section 4.2). Thus, an approach is required that allows computing modal results fast enough to produce sound when the first auditory event with the new object occurs.

To achieve a high-quality MA, the involved geometries often consist of thousands of vertices. For such complex geometries, an analysis would take far too long to be suitable for a run-time analysis. Thus, it is necessary to reduce the geometric complexity in order to achieve suitable computation times. This chapter will discuss different approaches to computing multiple Levels-of-Detail (LoDs) that can be used to reduce the geometric complexity and thus the time requirements of an analysis. Since these LoDs reduce the geometry resolution, they introduce an error, thus producing lower-quality results. Therefore, it is best to compute multiple LoDs, where very coarse approximations are computed quickly, followed by successively more refined analyses that produce more accurate results.

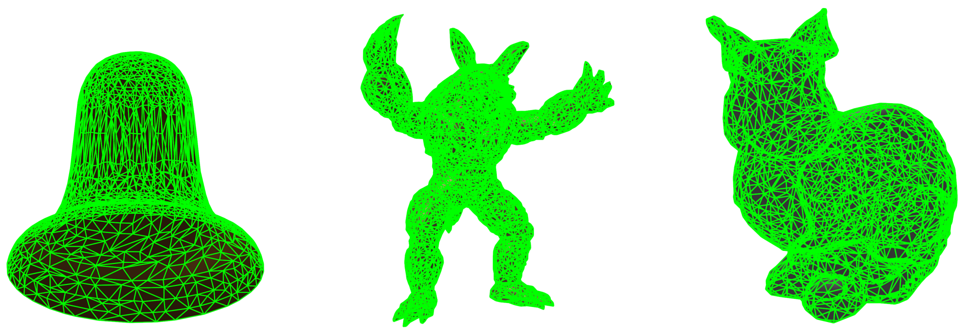
In this section, different approaches to computing LoDs are presented, which offer different levels of approximation to trade quality against computation time. Here, the focus is on their suitability for a run-time MA. Thus, the LoDs should provide a good approximation of the geometry, while also being fast to construct in order to maintain a low overall computation time.

For all computations, the input is assumed to be a surface mesh as it would be used for visual rendering, as well as settings on how to derive the physical mesh, e.g. by specifying a shell width for the thin-shell extrusion. In this work, two types of volumes are examined: thin-shell volumes, which extrude the 2D surface to form 3D elements, as well as closed tetrahedral volume meshes where the entire inside is filled with tetrahedra.

To visualize the different LoDs, example geometries will be used: a bell, the Stanford Armadillo, and the Stanford Bunny (see Figure 4.5).

4.5.1 Full Mesh

The highest quality is achieved with the full-resolution mesh. For the full resolution, it usually does not suffice to use the input geometry directly. Geometries for visual rendering



(a) Bell (b) Stanford Armadillo (c) Stanford Bunny

Figure 4.5: Example geometries for the LoD construction.

are optimized to reduce their complexity, such that, for example, large flat areas are represented with few vertices. For a FEM mesh, however, elements should be equally sized, so that a uniform vertex distribution is desired. Additionally, the size of the elements should be small, potentially requiring a refinement of the initial mesh. Thus, the input geometry is processed to produce a suitable mesh.

When processing a full-sized mesh, different approaches are used depending on whether a thin-shell or a solid geometry is used. For thin-shell meshes, we found that best results are gained by remeshing the surface with a target edge length. The target edge length can be determined using different metrics. A good choice is to use edge length of half the wavelength of a wave with 22 kHz, based on the speed of sound in the material. This yields a high-quality mesh with a good vertex distribution and a high resolution and uniformity of triangles, which benefits the analysis.

For thin-shell meshes, tetrahedra are created by extruding the surface vertices by a given thickness. To create well-shaped tetrahedra, it is possible to adjust the number of layers of the thin-shell, leading to multiple extruded vertices per surface vertex. The number of layers for the full-size mesh should be adjusted to an appropriate level, either manually or by determining a suitable layer count automatically. The latter is achieved by computing an average edge length of the surface mesh and dividing the thin-shell thickness by this average edge length.

For solid geometries, the surface geometry has to represent a closed surface, so that the interior can be filled. Therefore, a first step is to close any holes in the surface. Afterwards, a tetrahedralization is performed with a target edge length similar to the remeshing metric used for thin-shell meshes. This process proceeds similar to the remeshing of the surface for thin-shell geometries, but fills the whole volume with tetrahedra.

When creating the physics mesh from the surface geometry, either by thin-shell extrusion or by solid volume tetrahedralization, internal vertices are created, raising the total vertex count. Thin-shell extrusion doubles the number of surface vertices when using a single layer, and may add even more vertices for additional layers. Tetrahedral volume meshes also add vertices inside the volume to create well-formed tetrahedra, and thus have an increased number of physical mesh vertices. After refining, optimization, and volume construction, the full physical mesh typically consists of thousands of vertices, which may take several minutes or even hours to analyze.

4.5.2 Mesh Simplification LoDs

To lower the computation time of a MA, the number of mesh vertices has to be reduced. For reliable results, the used method should ideally support a specific target vertex count to achieve, so that the resulting MA computation time can be controlled. For this, the common approach is to use mesh decimation algorithms, which are common in computer graphics and reduce the complexity by removing elements from the original mesh [Cignoni et al. 1998]. A well-established method is quadric-based decimation [Garland et al. 1997], which is available in many optimized implementations.

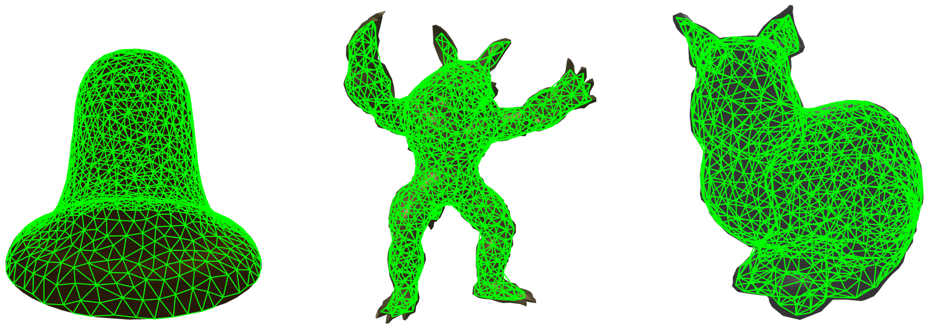
Depending on the used algorithm and settings, the mesh decimation produces close approximations of the input mesh, reduced to the desired vertex count (see Figure 4.6). Thus, the resulting geometries can be good candidates for LoDs for a MA. For the presented work, the `Surface_mesh_simplification` of CGAL[13] as well as `vtkQuadricDecimation` of the Visualization Toolkit[14] are used to compute the simplification results.

Another approach for mesh simplification is remeshing [Alliez et al. 2003], as was discussed for the full mesh. For the presented examples, CGAL's `isotropic_remeshing` is used. Remeshing works by creating a new mesh whose vertices are distributed on the original surface. By using different remeshing criteria, meshes with reduced vertex counts can be constructed (see Figure 4.7). This approach typically produced meshes with a better vertex distribution than when using quadric simplification, which are more suitable for a MA. Furthermore, features are less likely to be lost for approximations with low vertex counts. Additionally, remeshing can add vertices even where no vertices were placed in the input mesh. This way, large flat areas can be evenly subdivided, leading to a better vertex distribution. However, since remeshing typically uses geometric properties – like edge length or face angles – as remeshing criteria, it is more difficult to control the resulting vertex count, which cannot usually be specified exactly, making an estimation of the time requirements of the following MA steps more difficult.

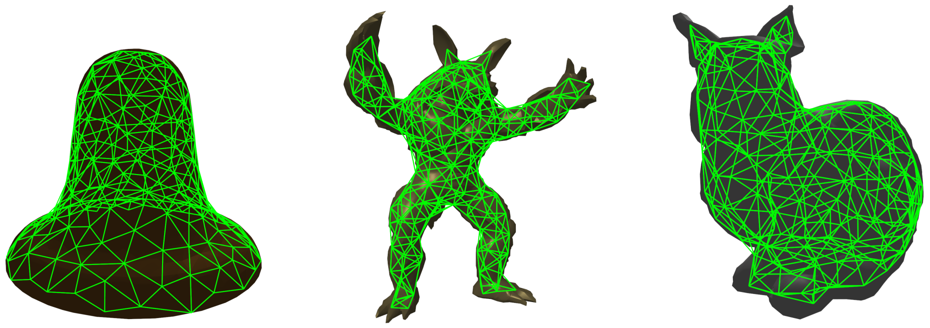
While the resulting meshes from decimation or remeshing are well-suited for a MA, the computation of the simplified meshes itself requires some time, especially when simplifying a high-resolution mesh. For example, when decimating a Stanford bunny mesh from 8000 vertices to 512 vertices, the reduction takes 219 ms using `vtkQuadricDecimation` and 300 ms using `Surface_mesh_simplification` (measured on a Windows PC with an Intel Core 15 6600K, compiled with MSVC13). Remeshing typically requires even longer – 694 ms for the above example. The actual time required for the simplification process depends on the mesh and the initial and target resolution and is hard to specify upfront. However, the potentially high time required to compute the simplified mesh increases the overall computation time. While the computation cost of the mesh simplification has little impact on the analysis of higher-resolution LoDs, it would severely delay the analysis of low-resolution LoDs that should be available quickly.

Another problem of both decimation and remeshing algorithms is that they are typically not tailored towards producing meshes with very low vertex counts while maintaining a shape approximation suiting the requirements of a MA. Thus, low-resolution approximations often

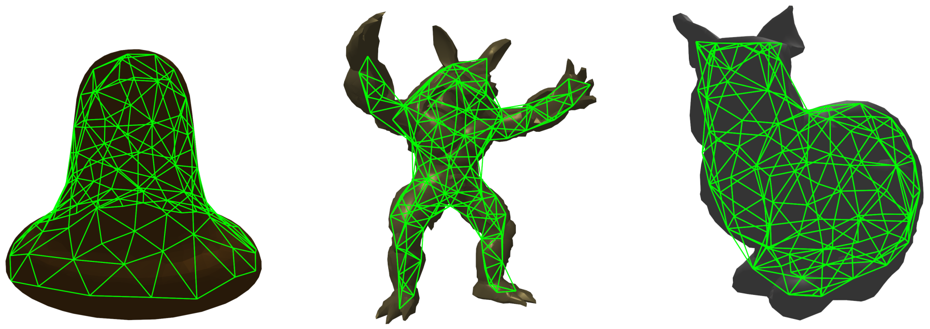
[13]`http://www.cgal.org`, version 4.9
[14]`http://www.vtk.org`, version 6.1.0



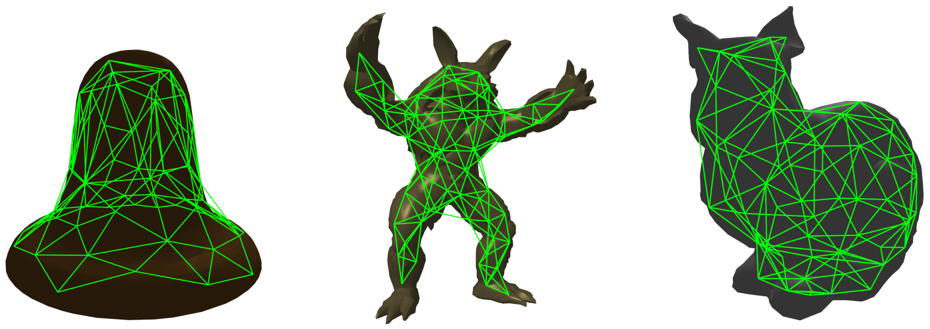
(a) 800 vertices



(b) 200 vertices

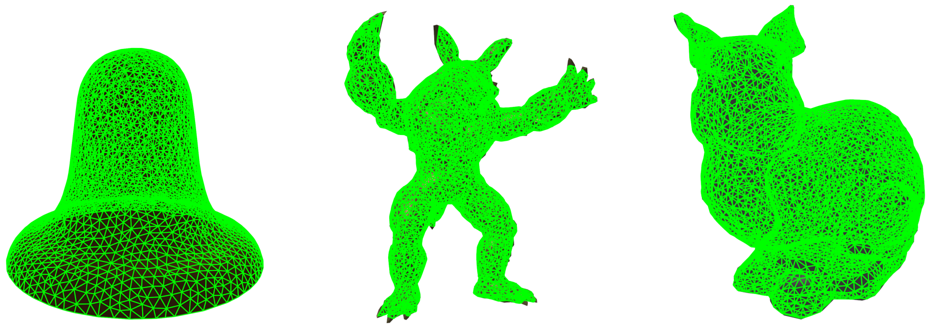


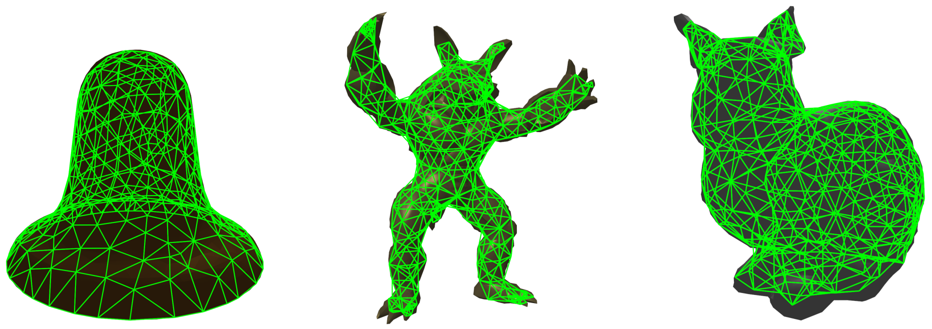
(c) 100 vertices

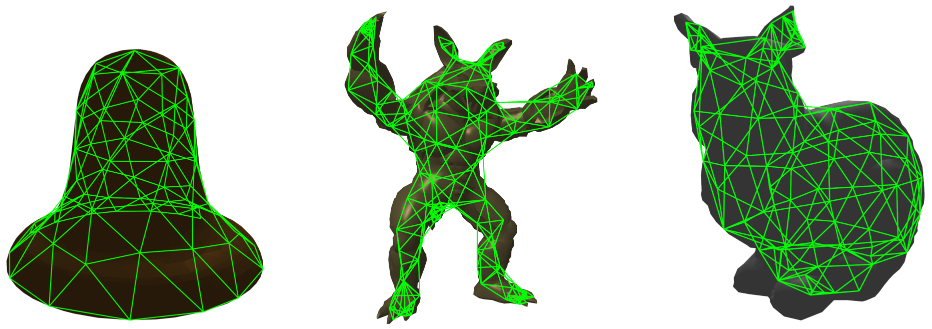


(d) 64 vertices

Figure 4.6: Different simplification LoDs of example geometries, produced by quadric decimation of the input mesh to 48, 100, 200, 400, and 800 surface vertices.


(a) 2% target edge length


(b) 6% target edge length


(c) 12% target edge length

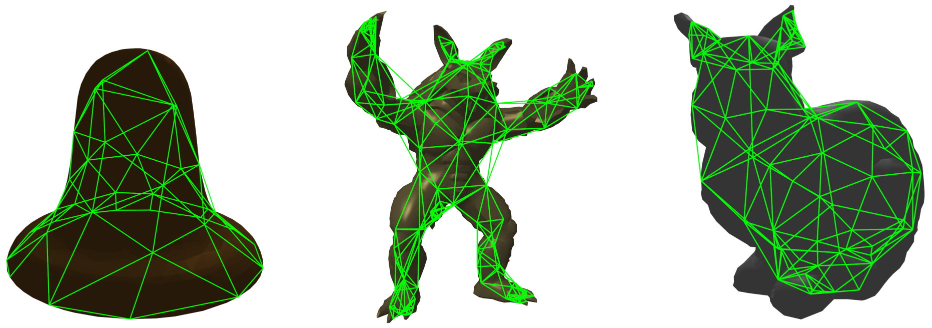

(d) 16% target edge length

Figure 4.7: Different simplification LoDs of example geometries, produced by a remeshing with different edge length limits (2%, 6%, 12%, and 16% of object diameter).

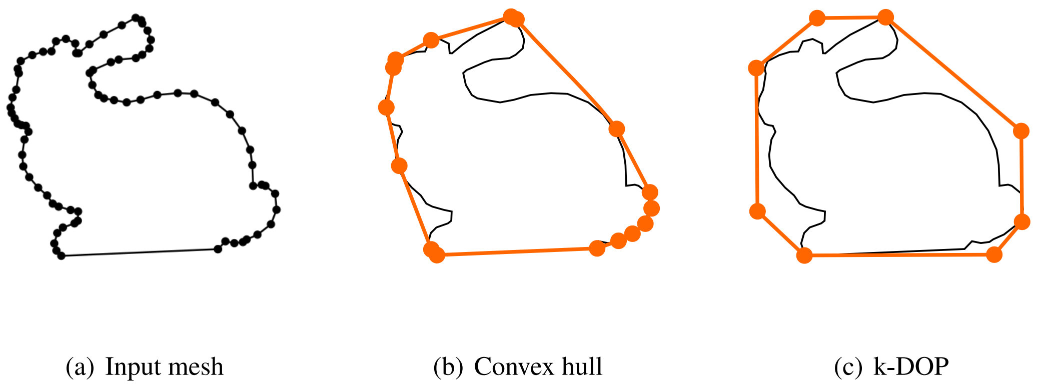

(a) Input mesh (b) Convex hull (c) k-DOP

Figure 4.8: 2D Examples for different hull approximations of a bunny mesh.

produce meshes that are not well-suited for a MA due to low uniformity.

When using volume meshes, the tetrahedralization can be performed using different parameters to adjust the resolution. But as for remeshing, the exact vertex count is hard to control, since the volume mesh is typically based on tetrahedron edge length and angle criteria. Alternatively, one can use a simplified mesh as outer shell, and only coarsely sample the interior while maintaining the vertices on the outer hull. While this allows a better control over the final number of vertices, it produces less suitable meshes because the tetrahedra may be of varying shape and size. For both variants, the computation of the reduced-resolution tetrahedralization requires computation time that is too high for low-resolution LoDs.

Therefore, mesh simplification is well suited for medium to high-resolution LoDs where the impact of the mesh processing is low compared to the overall analysis time, and where time constraints are not as strict so that variances in geometry processing duration and the resulting vertex counts can be accepted. For low-resolution LoDs, however, it is important to finish quickly and within a fixed time frame, so that the constraints of real-time applications are obeyed.

4.5.3 Hull Approximations

The simplification LoDs are well suited for higher-quality results, but are too slow to be able to provide first MA results fast enough to maintain the real-time constraints. For this, an approximation method is required that can be computed very quickly, but still produces geometries with a controllable vertex count and delivers plausible MA results even for low vertex counts. Therefore, the reduced geometry should capture prominent features – at least as far as possible with low vertex counts – and furthermore should provide a uniform distribution of vertices so that the triangles are well-formed. Ideally, the approximation should work with less than 80 physical vertices to allow an analysis within less than 16 ms (see Section 4.4.1). The use of extruded thin-shell meshes doubles the resolution, and a tetrahedral mesh also requires some internal vertices. Thus, the lowest-resolution LoDs should produce a surface mesh with at most 40 vertices.

Inspiration for such approximations can be drawn from approaches for collision detection, where simplified, low-resolution approximations of an object's shape are used to speed up the collision detection process. Commonly used ones are primitive shapes that enclose the object, e.g. using a bounding sphere or box. While these bounding volumes are well-suited for a broad phase of a collision detection algorithm, they are too coarse an approximation for the purpose of a MA.

Another bounding volume is the convex hull, which simplifies a shape by omitting all concave vertices, and is thus formed by all vertices that are extremal along any axis (see Figure 4.8(b)). The convex property of the resulting shape allows for an easier collision detection, but is not well suited for MA because important features may be lost, and the volume can be significantly larger than the original mesh. Additionally, a convex hull may still consist of a large number of vertices. For example, the convex hull of a sphere would consist of the same vertices as the input mesh. Therefore, a convex hull does not allow a controllable vertex count of the result. Furthermore, the distribution of vertices may be very uneven, because straight or concave regions are sampled with lower resolution than round borders or convex features. For example, the convex hull of a bunny mesh in Figure 4.8(b) produces a high number of vertices at the tail, but large un-sampled edges between vertices on the bottom and between ear and back.

A more suitable type of convex bounding volume is a k-DOP [Klosowski et al. 1998], which is an extension of an axis-aligned bounding box that incorporates more bounding faces. It is computed by defining k half-axes, and for each half-axis computes the half-space matching the extent of the object along the half-axis. The intersection of all these half-spaces forms the bounding volume (see Figure 4.8(c)). Choosing the number of axes allows for a controllable complexity, and the k-DOP allows for a fast collision test. However, the extraction of an actual mesh from the k half-planes is cumbersome, and the resulting mesh may have a low uniformity in the vertex distribution and varying vertex counts, similar to the problems of the convex hull.

Another problem of the convex hull, the k-DOP and other bounding volumes is that these are designed to fully enclose an object. This is a useful property for collision detection, but for a MA, it is not necessary to encapsulate the whole mesh. Since a bounding volume over-estimates the size of the object, this often leads to a change in the object's sound spectrum, since larger objects typically produce lower-frequency sounds. Instead, it would be acceptable for an approximate mesh to penetrate parts the original mesh if it benefits the overall approximation.

The bounding volumes used for collision detection do not match the requirements of a MA LoD. In this thesis, specialized hull approximations will be proposed for the lowest resolution LoDs: the *k-Hull* and the *ShrinkHull*.

k-Hull

As a first kind of low-resolution approximation, the k-Hull combines aspects from both convex hulls and k-DOPs and is adjusted to better match the requirements of geometries for a MA. Like k-DOPs, they utilize k half-axes to determine the resulting geometry. However, instead of using the axes to define half-spaces, the k-Hull construction projects the input mesh's vertices on each of the half-axes. Then, for each of the k half-axes, the vertex is chosen whose projection is extremal, i.e. which is the farthest away from the center in

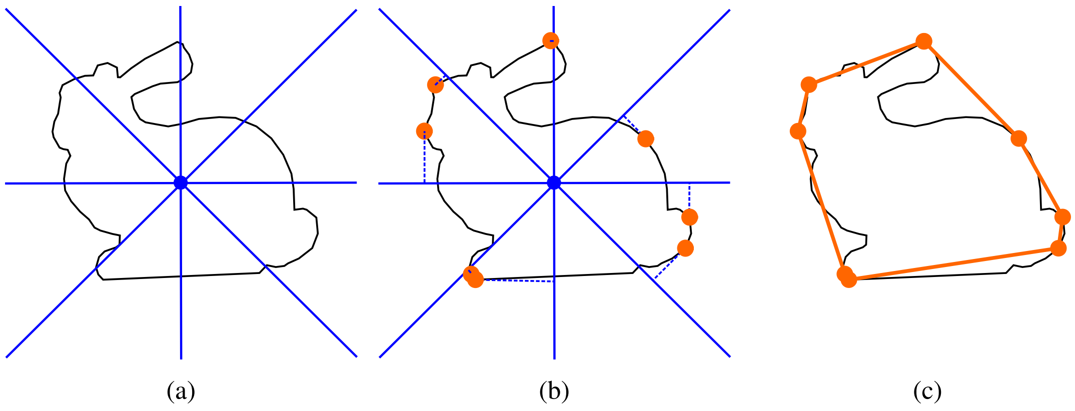


Figure 4.9: 2D Examples for the construction of a k-Hull with 8 directions. (a) Definition of 8 half-axes. (b) Determination of extremal vertex along each half-axes. (c) Finished k-Hull constructed from convex vertices.

Input mesh	1000v	4000v	8000v	1000v	4000v	8000v	1000v	4000v	8000v
	Physical vertices			k-Hull computation			Complete MA		
18-Hull	28	28	28	0.1 ms	0.1 ms	0.2 ms	2.0 ms	2.0 ms	2.1 ms
26-Hull	44	44	44	0.1 ms	0.3 ms	0.5 ms	4.2 ms	4.3 ms	4.5 ms
38-Hull	60	60	60	0.3 ms	0.4 ms	0.6 ms	7.3 ms	7.5 ms	7.4 ms
66-Hull	100	100	100	0.7 ms	0.8 ms	1.5 ms	21.4 ms	22.2 ms	22.7 ms

Table 4.3: Computation times of the k-Hull approximation and the total MA for different hull sizes and input meshes with different vertex counts. k-Hulls produce surfaces meshes with at most k vertices. The physical mesh was created using thin-shell extrusion, doubling the vertex count of the k-Hull surface.

direction of the half-axis. The set of extremal vertices from all k half-axes defines a sub-set of the input geometry's convex vertices (see Figure 4.9). This set of vertices is then used to define the approximate mesh, which can be constructed efficiently due to the convexity of the set of vertices. The number of vertices is at most k, but may be lower if the same vertex is extremal for more than one half-axis. This k-hull approximation manages to find the outer features pointing along each of the k directions.

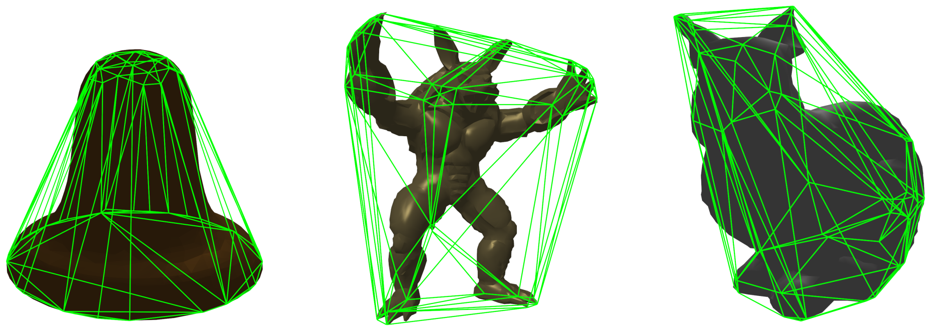
The construction of k-Hulls can be performed efficiently, especially when defining $\frac{k}{2}$ axes instead of k half-axes by choosing pairs of anti-parallel half-axes. To determine the k half-axes for the k-Hull computation, a uniform distribution is desired. As a basis to define the directions, the vertices of geodesic domes are used, which are sphere geometries with uniformly distributed vertices. Since geodesic spheres can only be constructed for certain vertex counts, the variability of k is limited, but sufficiently high. For k-Hulls, the used implementation provides the option of using 18, 26, 38, 42, or 66 half-axes. Some examples of the k-Hull approximation of different target vertex counts applied to different geometries are depicted in Figure 4.10.

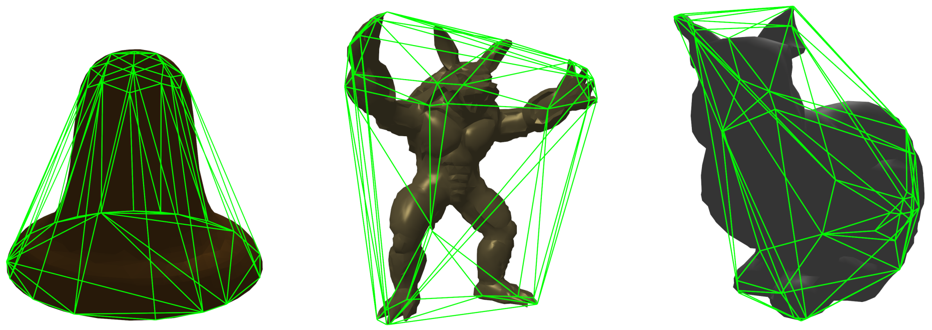
A special case occurs if the input geometry is very flat, i.e. if the smallest axis of its oriented bounding box is significantly shorter than the longest (for the presented framework, less than 0.1%). Depending on the approach chosen to create the physical FEM mesh from the input, such a flat geometry may require special handling. If it represents a closed volume to be meshed, the normal 3D k-Hull algorithm is used to extract a shell, and a standard tetrahedralization is performed, just as for non-flat geometries. Note, however, that this should rarely occur, since geometries representing the shell of a closed volume tend to have a more significant thickness. If the flat geometry represents a thin-shell physics mesh, non-closed geometries are possible, e.g. flat plates modeled without thickness. In such a case, a specialized $2D$ variant of the k-Hull algorithm is used, which does not distribute the axes on a sphere, but on a circle lying in the geometry's plane. Unlike for the $3D$ variant, only $\frac{k}{2}$ half-axes are used to determine the outer vertices. The final mesh is formed by the (at most) $\frac{k}{2}$ outer vertices, a center vertex, as well as an additional inner ring of vertices where each vertex lies midway between the center and the corresponding outer vertex. This way, a flat double-ring mesh is constructed, which has fewer corners, but typically shows a better shape of its triangle than a single-ring mesh.

Computation times for the k-Hull construction and the total analysis are provided in Table 4.3. The measurement was performed using a bell mesh with different initial vertex counts and computed on an Intel Xeon X5650 CPU using a single CPU core. The run-time of the k-Hull construction depends on the number of axes and the complexity of the input mesh. Due to its simplicity, the k-Hull approximation can be computed very quickly – only for hulls with many axes and very large input geometries, the computation time exceeds 1 ms. In general, the k-Hull construction time is small when compared to the overall cost of the MA, and is thus fast enough to provide a first approximate geometry quickly. However, the resulting geometries show some problems.

Using the k-Hull construction produces low-resolution geometries with at most k surface vertices. The number of vertices may still vary, though, because a single vertex may be extremal for multiple directions, so that fewer vertices may be produced.

The vertex distribution can also show a low uniformity. When several vertices are placed


(a) 66-Hull


(b) 38-Hull

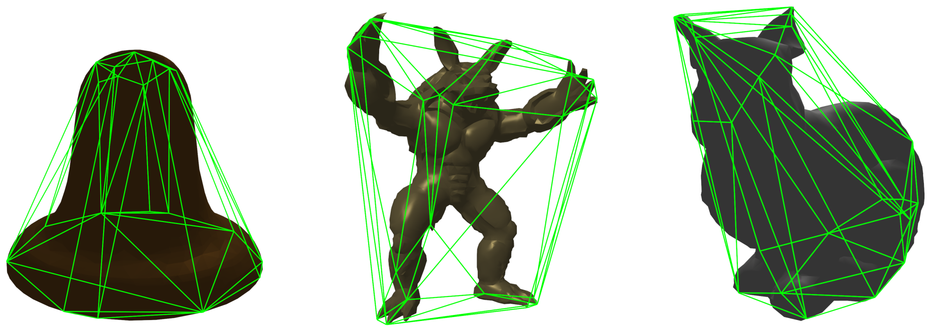

(c) 26-Hull

Figure 4.10: *k*-Hull meshes for different geometries, using 26, 42, and 66 half-axes.

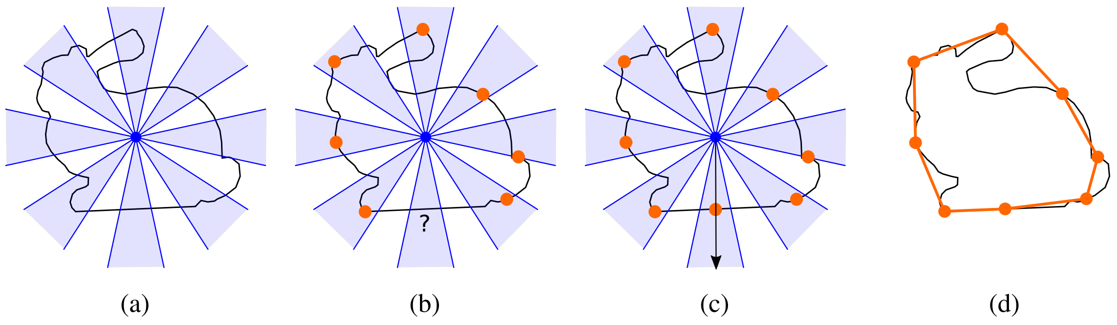


Figure 4.11: 2D Examples for the construction of a ShrinkHull with 8 directions. (a) 8 directions with surrounding cone are defined. (b) In each cone, the vertex with highest distance to the center is chosen. (c) If no vertex is found inside a cone, a ray-cast is used to find a triangle intersection. (d) The resulting ShrinkHull mesh.

on a protruding feature, they may be located close together. On the other hand, flat or concave regions are skipped, so that areas with low sampling and long edges occur. The example in Figure 4.9 shows an example of this: While each two of the eight vertices are placed on the front paw and the tail, the rest of the bottom is approximated with only a single edge.

Another problem of the *k*-Hull construction is that the resulting mesh is always convex. The problem of this can be seen from the examples in Figure 4.10. For the armadillo model, the hull directly connects the arms, ears, and legs, thereby creating a much larger object than the input geometry, filling the volume between the limbs. This also leads to a loss of distinctive features like the arms and legs. Similarly, the *k*-Hulls of the bell shows some problems. While the top and the rim are densely sampled, the sides of the bell are not sampled by vertices, leading to large but slim triangles. Furthermore, the concave inward curve is lost. Lastly, the bottom of the bell is originally open, but since *k*-Hulls only produces closed meshes, the bottom is filled in the approximated meshes.

All in all, the *k*-Hull is able to provide approximate meshes fast enough for low-level LoDs, but the resulting geometries have several problems that reduce the quality of the resulting MA. Thus, it is not fully suited as a low-resolution LoD for run-time MA.

ShrinkHull

To improve the quality of the low-resolution MA approximations, the concept of the *k*-Hull was developed further to overcome some of the problems. Some of the problems originate from the fact that the *k*-Hull construction forms a sub-set of the convex vertices of the input mesh. This way, it is neither possible to properly approximate concave features, nor can vertices be placed in regions where no vertices exist in the input mesh, potentially creating a low uniformity in the vertex distribution.

Thus, an enhanced approximation scheme, named *ShrinkHull*, was devised. Like the *k*-Hulls, it uses *k* uniformly sampled direction and determines the extremal vertex along these directions. However, unlike the *k*-Hull's half axes, the ShrinkHull uses rays that are emitted from the object's center in uniform directions. Around each of these rays, a cone is defined.

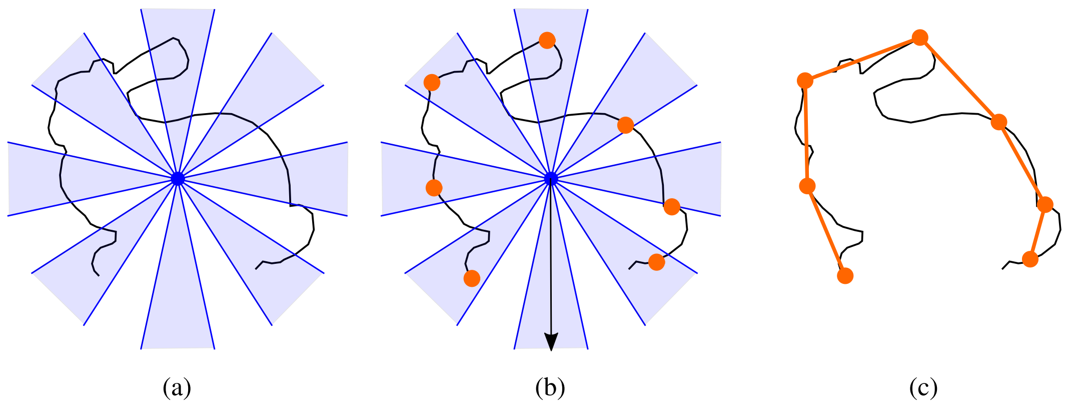


Figure 4.12: 2D Examples for the construction of a ShrinkHull with 8 directions for a mesh with a hole. (a) 8 directions with surrounding cone are defined. (b) Ray cast on bottom does not find intersection. (c) The resulting ShrinkHull, with missing vertex/faces on bottom.

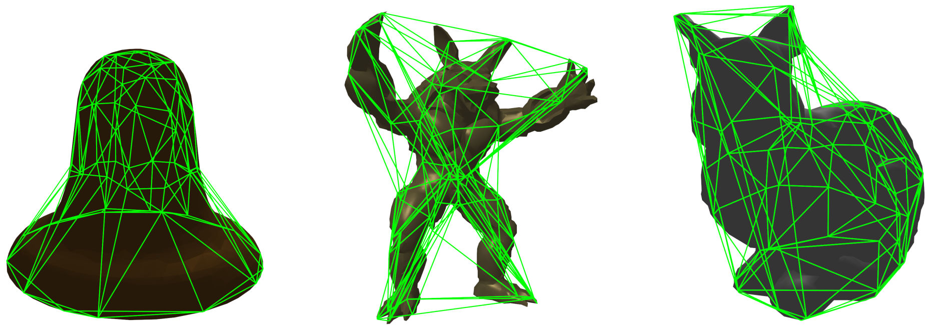
For each of the k rays, all vertices inside the corresponding cone are considered, and the one furthest from the center is chosen, as illustrated in Figure 4.11.

When a geometry contains large triangles, there may be no vertices at all within the cone. If this is the case, an explicit ray cast is used to compute the intersection point between the ray and the input mesh's triangles. The furthest intersection point is then used as the vertex for this direction (see Figure 4.11(c)). This allows placing vertices at new locations that were not occupied by vertices in the input mesh, and can thus help increasing the sampling uniformity.

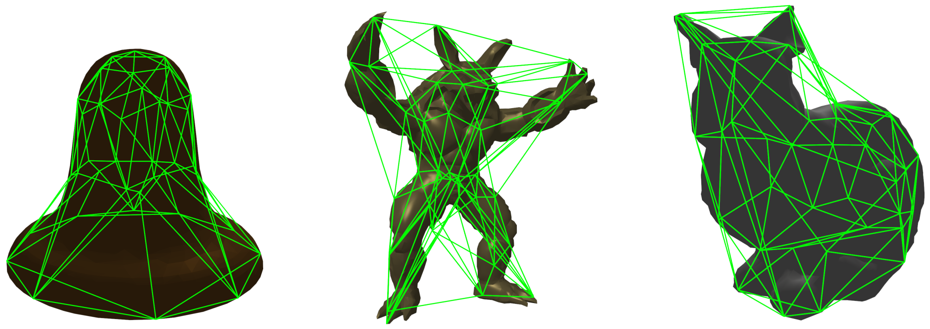
Some objects may also have holes in them. If such a hole is large enough, it is possible that the cone does not intersect with any of the input mesh's geometry. In such a case, neither an extremal vertex is found, nor does the ray cast determine an intersection with the input mesh. If this occurs during the ShrinkHull construction, the corresponding vertex and its adjacent triangles are removed from the ShrinkHull mesh (see Figure 4.12). Thus, the ShrinkHull algorithm is able to model sufficiently large holes in order to better approximate the input mesh.

For good results of the ShrinkHull algorithm, it is important to choose a proper opening angle of the cones around the axes. A larger angle allows detecting smaller protruding features that are far away from an axis, but at the same time may lead to a less uniform vertex distribution and a higher chance to miss holes or concave features. To define the opening angle independent of the geometric resolution, it is based on the angle between the uniformly distributed axes of the ShrinkHull. For the presented application, an opening angle of half the inter-axes angle proved to yield good results and is used for all presented ShrinkHull LoDs.

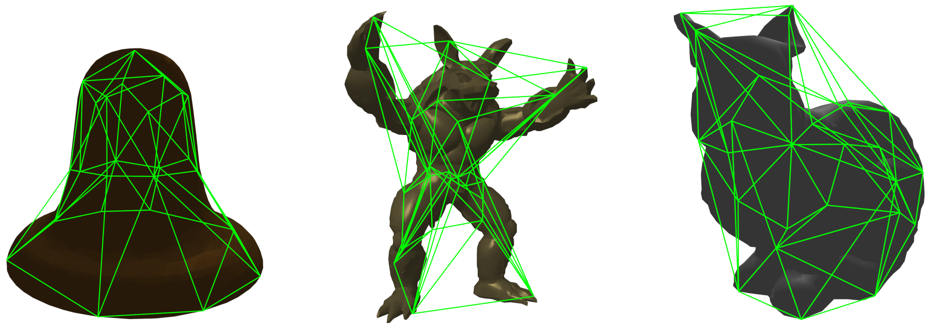
Another factor than can impact the quality of the result is the distribution of rays. Initially, the k sampling directions were distributed uniformly over all directions based on geodesic spheres, as for the k-Hull construction. This works well as long as the input geometry is approximately equally sized along all directions, i.e. similarly shaped to a sphere or a cube. However, when an object is not uniformly sized in all directions, the center of flat areas are sampled more densely than the edges when using uniformly distributed rays. To allow a proper sampling of an object with non-uniform dimensions, it is first scaled such that its bounding



(a) ShrinkHull with 66 vertices



(b) ShrinkHull with 38 vertices



(c) ShrinkHull with 26 vertices

Figure 4.13: ShrinkHull meshes for different geometries, using 26, 42, and 66 sampling directions.

box is of uniform scale. This then produces an even axis distribution over its surface. While it would be possible to simply scale the axis directions, this would produce problems with the cone angles, since the cones would need to be scaled, too, producing distorted cones that no longer have a spherical cross-section.

As for the k-Hull, the ShrinkHull also uses special handling of flat geometries intended for extrusion into a thin-shell physical mesh. $\frac{k}{2}$ cones are evenly distributed in the flat plane, extruding from the center. For each cone, the furthest vertex is selected. If no vertex is found for a cone, a ray-cast is performed to find an alternative vertex. However, unlike the 3D variant, no holes are allowed – if the ray cast fails, a new vertex is created by interpolating the vertices from the neighboring cones. Once the $\frac{k}{2}$ outer vertices are found, the mesh is generated as for k-Hulls by adding a center vertex and intermediate vertices between the center and each outer vertex. This yields a double-ring mesh with $k+1$ vertices.

Figure 4.13 shows some example geometries produced by the ShrinkHull approximation approach. These examples show the benefits of the ShrinkHull over a k-Hull, but also some remaining problems.

For the ShrinkHull LoDs of the bell, the approximation is much better than the results of the k-Hull. The sampling is more uniform, and the concave curve of the shell is captured. Additionally, the hole in the bell's bottom is modeled by the ShrinkHull, but closed for k-Hulls. For the armadillo and the bunny, some of the features are approximated more accurately, and the vertex distribution of the ShrinkHull is better than for the k-Hull.

Because of the limited coverage of the cones, no vertex can be chosen as representative for multiple axes. As a result, there is less variation in the vertex counts of the resulting ShrinkHull mesh. Unless the object has holes, the resulting ShrinkHull geometry contains one vertex for each of the k directions. Also, since the cones limit the deviation of vertices from the sampling directions, a better, more even distribution of the vertices over the surface can be achieved. Furthermore, concave regions can be approximated be the ShrinkHulls – like the inward curve of the bell or the girth of the armadillo. However, such concave features can only be captured if they are sufficiently large, so that the relevant cones do not contain elements of other features that are located further outwards. Furthermore, the features have to be oriented such that they can be sampled by the cones, which requires them to point towards the center. For example, the bunny's ears in Figure 4.11 and Figure 4.13 are features that point away from the head, but the sampling cones originate from its center of origin. Thus, the indentation between ears and back is not captured, and the resulting ShrinkHull incorrectly fills this region.

All in all, the ShrinkHull LoD provides a geometry approximation that is more suitable than k-Hulls for the low vertex counts required for a run-time MA. However, the additional ray casts during the construction require additional computations, which is especially noticeable when the input geometry has a high resolution, but large triangles or holes that require many ray casts. When no vertex is found for a cone, a ray-mesh intersection has to be computed. For meshes with many triangles, the high number of ray-triangle intersections can be too slow to maintain the desired computation time restrictions when using a naive ray-casting implementation. The use of an elaborate broad phase like spatial partitioning or bounding volume hierarchies can help to reduce the computation time required for a ray cast. However, these broad phases require the construction of special data structures, which can

Input mesh	1000v	4000v	8000v	1000v	4000v	8000v	1000v	4000v	8000v
	Physical vertices			ShrinkHull comp.			Complete MA		
18-ShrinkHull	34	34	34	0.3 ms	1.3 ms	2.6 ms	3.0 ms	3.5 ms	5.9 ms
26-ShrinkHull	50	50	50	0.4 ms	1.5 ms	2.9 ms	5.8 ms	6.6 ms	8.1 ms
38-ShrinkHull	72	72	72	0.8 ms	2.8 ms	5.3 ms	13.3 ms	14.9 ms	17.9 ms
66-ShrinkHull	122	122	122	1.1 ms	3.6 ms	7.3 ms	33.8 ms	36.1 ms	42.3 ms

Table 4.4: Computation times of the ShrinkHull approximation and the total MA for different hull sizes and input meshes with different vertex counts. *k*-ShrinkHulls produce surfaces meshes with at most *k* vertices. The physical mesh was created using thin-shell extrusion, doubling the vertex count of the *k*-Hull surface.

require a significant amount of time. Especially since there are typically only few ray casts to be performed when building a ShrinkHull, using an elaborate broad phase is typically slower than using a naive approach.

Instead, the spherical distribution of rays around the sphere is used to efficiently cull a high number of rays per axis. For each triangle, the direction from the center of the object to the center of the triangle is used as base direction, and the angle of a cone containing the triangle is computed as the minimum of the dot product between this direction and the directions from the object center to each of the three vertices. When a ray cast is requested, one can simply check the ray direction against the cone angle, and omit the ray-triangle intersection test if the angle between the ray and the direction towards the triangle center has an angle larger than the sum of the cone's and triangle's opening angles. While this culling also requires some pre-processing of the triangle data structure, it is much faster than constructing a more advanced broad phase collision data structure, and thus helps to maintain a high performance.

Table 4.4 lists computation times of the ShrinkHull and the complete MA for the example of the bell geometry with different resolutions of the input mesh and for different ShrinkHull vertex counts, using the same hardware and input as for the *k*-Hull performance measurements. Note that the bell has a hole in the bottom, so that the final vertex count does not equal twice the axis count, as would be the case for meshes without holes. This also ensures that some ray casts have to be performed to compute the ShrinkHull. Otherwise, if no ray-casts are required, the acceleration data structure is not computed, and thus the ShrinkHull construction time is mostly equal to that of the *k*-Hull. While the results of the ShrinkHull construction for the example geometry is only slightly higher than for the *k*-Hull, the overall MA computation time is often significantly higher for ShrinkHull. This comes from the fact that *k*-Hull algorithm often chooses the same vertex for multiple axes. This results in a lower vertex count, so that the actual MA requires less time. Since ShrinkHulls only have fewer vertices when holes are found, they provide a much more constant resulting vertex count, allowing a better estimate of the overall run-time of the analysis.

As the performance measurement shows, the ShrinkHull construction requires more computation time than *k*-Hulls – at least if ray casts have to be performed – which increase the overall computation time. Still, the overhead is low enough to compute a ShrinkHull with

26 directions in less than 16 ms even for large meshes that also require multiple ray intersection tests. Together with the better geometric approximation, this makes the ShrinkHulls a suitable choice as a low-level LoD for the desired application. If a higher number of axes is used, the computation time may exceed this limit, but provides geometries for slightly slower LoDs of higher accuracy. Due to its performance and the properties of the geometric approximation, the ShrinkHull promises to be a good choice for initial approximate LoDs.

4.5.4 Analytic Solution

The ShrinkHull with 26 directions allows computing one or even multiple low-level LoDs within the allotted time span of 16 ms. However, there are cases where this is still insufficient to ensure a fast enough analysis. In some cases, it may happen that multiple new objects are created simultaneously, for all of which a run-time MA has to be computed. For example, in the before-mentioned scenario of a fracturing glass plate, a high number of shards are created at the same time. For such a scenario, an even faster approximation that can produce modal data in much less than 1 ms would be necessary to handle all objects without additional latency.

Such an approximation cannot be computed using a numerical MA from an explicit geometry mesh, since even for very low-resolution meshes the matrix assembly and Eigendecomposition would be too slow. Instead, one can utilize the fact that for certain primitive geometries, a MA can be computed *analytically* [Morse 1948; Leissa 1969; Rossing et al. 1995; van den Doel et al. 1998; Reddy 2006; Kang et al. 2008; Bilbao 2009]. Instead of using an approximate mesh and numerical methods, the analytic solution is directly computed by solving the wave equation of the system describing the physical setup. This analytic solution can be computed much faster than a numerical solution. However, it is only possible to analytically solve the modal vibrations of few simple geometries, like membranes, thin beams, and plates. Therefore, a strong approximation is required to allow an analytic MA of arbitrary input geometries.

Most objects that can be analyzed analytically are limited to one- and two-dimensional shapes, like strings, beams, membranes, or plates. Also, analytic solutions typically require fixing part of the boundary of the object using boundary conditions, which further limits the applicability. For the presented application as a LoD to provide modal sound data for arbitrary geometries, one can utilize the analytic solution for plates.

To approximate an arbitrary geometry using analytic plates, it is first approximated as an oriented box by performing a *Principal Component Analysis (PCA)* on the vertices. This provides a coordinate system aligned with the directions of maximum and minimum spread of vertices. The same procedure is used for the computation of oriented bounding boxes [Gottschalk et al. 1996], which use the axes from the principal component analysis and the maximal extent of the geometry along these axes to define a bounding volume suitable for collision detection. However, as for the hull approximations, such a bounding volume can be significantly larger than the original geometry. Instead, for the analytic LoDs, the extent of the box is computed from the spread of vertices along the axis, which is captured by the eigenvalues λ_i of the PCA. For the presented use-case, the extent along each axis a_i is computed as $2.2 \cdot \sqrt{\lambda_i}$. This extent proved to be a good approximation on average, although it may still over- or underestimate the size of the object. However, a better estimate of the extents would require a more detailed analysis of the mass distribution, which would require too much time.

Once the approximate box has been computed, it is fitted by six plates, one for each side. Plates with clamped edges are one of the few shapes for which a MA can be computed analytically [Reddy 2006]. A plate with clamped edges is defined by its dimensions $a \times b$, thickness d, and material parameters of density ρ, Young Modulus E and Poisson Ratio ν. The mode S_{ij} is the mode with wave number i along the direction of extent a and wave number j along the direction of extent b. Its natural angular frequency $\tilde{\omega}_{ij}$ is computed as:

$$\tilde{\omega}_{ij} = \sqrt{\frac{\pi^4 \cdot D}{2 \cdot \rho \cdot d}} \cdot \left(\left(\frac{i+1}{a} \right)^2 + \left(\frac{j+1}{b} \right)^2 \right)^2 \tag{4.2}$$

Here, $D = \frac{E \cdot d^3}{12 \cdot (1-\nu^2)}$ is the flexural density. This computation provides the natural frequency $\tilde{\omega}_{ij}$, i.e. the frequency with which the mode would vibrate when no damping is present, and can thus be used equivalently to the results of the numerical MA by computing the damped frequency $\omega_{ij} = \frac{1}{2}\sqrt{4 \cdot \tilde{\omega}_{ij}^2 - \left(\alpha \cdot \tilde{\omega}_{ij}^2 + \beta \right)^2}$ and the damping coefficient $\gamma_{ij} = -\frac{1}{2}\left(\alpha \cdot \tilde{\omega}_{ij}^2 + \beta \right)$.

When a force acts on the plate at the relative position (x, y) in planar coordinates, the respective modes have to be excited. For this, the mode shape s_{ij} is utilized:

$$s_{ij}(x,y) = sin\left(\pi \cdot (i+1) \cdot \frac{x}{a} \right) \cdot sin\left(\pi \cdot (j+1) \cdot \frac{y}{b} \right) \tag{4.3}$$

An external force is distributed among all modes in relation to their mode shape at the location the force acts on. This corresponds to the use of the gain matrix for the numerical approach, which is also based on the mode shapes.

The overall box is constructed from six plates, consisting of three pairs of identical plates. For each of the three plates, 25 modes are computed. When the model represents a thin-shell geometry, the shell thickness is used for each plate. For volumetric geometries, the box thickness is used, which however only provides inaccurate results because the analytic plate modes are only correct for thin plates.

One special case is a mesh that is very flat, i.e. whose box extent along one axis is much smaller than along the others. If this is the case, the object is approximated using only a single plate of appropriate thickness.

The main advantage of the analytic approach is that it can be computed very quickly. The analytic MA, including the PCA and mode parameter computation, can be performed in $<$ 0.2 ms even for very complex geometries with $>$ 10000 vertices. Thus, several dozens of analytic results can be computed within one frame of 16 ms.

However, since every geometry is approximated as a box, and the extents of the boxes are only estimated based on the vertex distribution, the analytic LoD only provides a very rough approximation and should be regarded as a last resort approach (see Section 4.7).

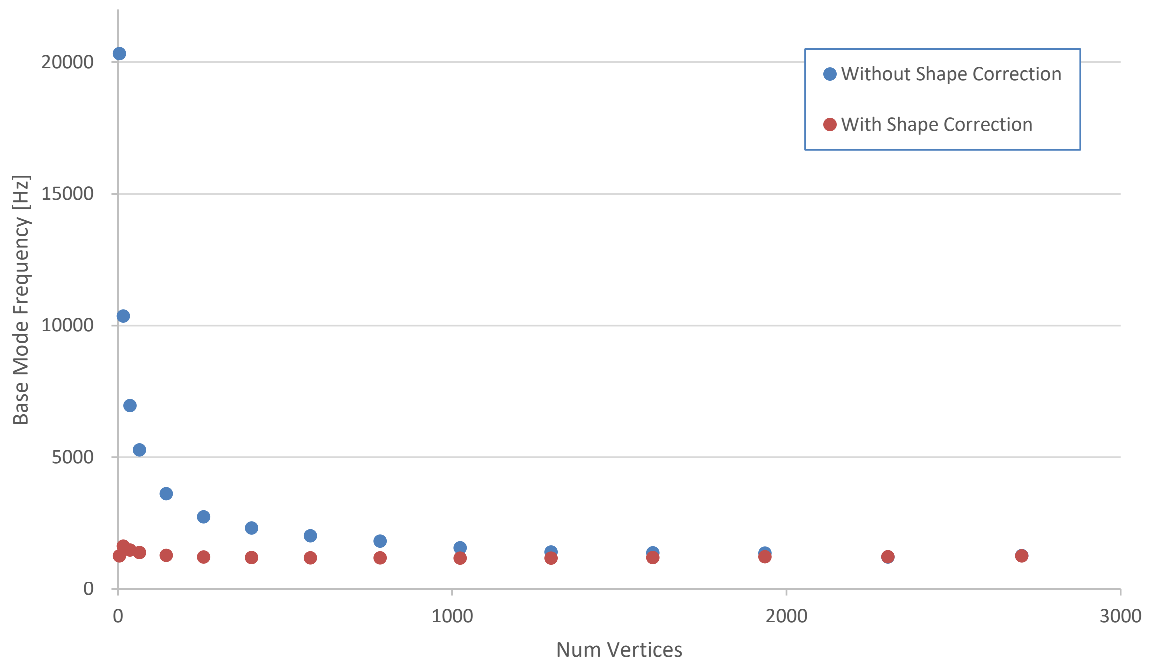


Figure 4.14: Frequency of the base mode of a square plate of dimensions 10 cm × 1 cm × 10 cm using different resolutions, where higher surface vertex counts lead to smaller, better-shaped tetrahedra. The graph plots the frequencies computed using unmodified FEM meshes (blue) and when using a correction factor based on individual tetrahedron shapes (red).

4.6 Correction Factor

The construction of a geometric mesh is an important step of a numerical FEM analysis. When requiring a high-quality approximation, the type of primitive cell types, size and shape of the cells, and the vertex distribution play an important role. Thus, mesh generation is an important aspect when using FEM for applications such as structural engineering, where it is a specialized field.

For the desired application of sound synthesis in VR, however, the goal is to infer the physical meshes automatically from the input geometry. The use of LoDs to reduce the geometric complexity puts severe limits on the quality of the resulting mesh. Especially for thin-shell meshes, which often have a shell thickness of few millimeters or less, this can lead to badly formed tetrahedra that have one or more very short edges. This can pose problems for the resulting analysis if it significantly alters the modal results and thus the synthesized sound. Therefore, the effect of resolution on the results is examined this thesis.

When analyzing the dependency of the MA results on the input mesh, an effect of the mesh resolution and tetrahedron quality can be observed. Already in the first publication promoting the use of numerical MA for sound synthesis, O'Brien et. al. mentioned that “We have found that low mesh resolution tends to shift frequencies higher and may add a “hollow” quality to the sound.” [O'Brien et al. 2002]. As a solution, they propose to simply adjust the material parameters to compensate for the frequency shift.

For the run-time MA, it is important to maintain a decent approximation even for lower-resolution sounds in order to still produce plausible results. When using a lower-resolution geometric LoD, the loss of certain features of the original mesh can lead to a change in sound. This cannot be circumvented with the limited number of vertices. However, if the lower-resolution LoD represents the same geometry with only fewer vertices – e.g. for a box – or only loses smoothness, but no features – e.g. for a sphere – there should be no significant change in the resulting sound. It turned out, however, that this is the case. When the resolution of such simple objects is changed, the modes provided by the MA can have a significantly higher frequency than for meshes with a high enough resolution to produce only well-shaped tetrahedra. This poses a significant problem for the automatic LoD construction, because a significant frequency shift would produce lower-resolution LoDs to have a different pitch than the full-resolution object, and might no longer be a plausible approximation. Since the LoDs are created automatically, a manual adjustment of material parameters to counter this effect is not viable. Such a manual effort would negate the main benefit of using a MA for sound synthesis, i.e. the reduction of manual effort compared to recording or design specific sounds. Thus, a method to compensate for resolution-dependent effects is required for a usable run-time MA with LoDs.

To examine the effect of lower resolution on the resulting modes in more detail, a detailed analysis has been performed using simple geometries – boxes and thin-shell plates of different dimensions. These were modeled using a regular mesh of varying resolution, and different numbers of thin-shell surface layers. This way, it was possible to create geometries consisting of tetrahedra which are all equally shaped. Depending on the object's resolution, the size and shape of tetrahedra vary form small, well-shaped tetrahedra to larger ones with uneven size. Especially for very low-resolution plates, it is possible to create tetrahedra that are very large in the plate's surface plane, but very thin along its thickness. Such splinters are problematic for FEM meshes.

The analysis confirmed a correlation between the resolution and the pitch of the modes. A lower resolution with larger, less well-shaped tetrahedra led to a higher frequency of the resulting modes. For lower resolution, this effect can become quite significant, which leads to serious changes in the produced sound. Figure 4.14 shows the frequencies of the base mode of a square plate, modeled with different resolutions. As the graph shows, lower-resolution models produce modes with too high a frequency. While the high-resolution meshes yield a base mode with a frequency of 1260 Hz, the lowest-resolution plate, which was modeled using only 5 vertices, produces a base mode with a frequency of 20320 Hz. Thus, the lowest-level approximation produces a mode with approximately 16$\times$ the frequency of the high-resolution model, which is a very noticeable error. For other low-resolution approximation, a similar pitch shift is observed. This leads to intolerably bad results when using LoDs with low vertex counts for a run-time analysis.

In order to achieve acceptable results, it is necessary to compensate for this resolution-dependent frequency shift. Using the data collected from the example geometries with uniform geometry, an analysis of the dependence of the frequency on different tetrahedron parameters has been performed. For each resolution, the lowest-frequency mode was examined in relation to properties of the tetrahedrons in the mesh, like volume, tetrahedron angles, os average, minimum and maximum edge length.

Using the tetrahedron parameters, an analysis as performed to find a dependency between these parameters and the observed pitch shift. The goal is to compute a correction factor that can be applied to the results in order to compensate for the frequency change.

One approach that produced very consistent mode frequencies over all resolutions was scaling the eigenvalues (which are the basis for the mode frequencies) by $f_1 = \frac{1}{\sqrt{vol \cdot ratio}}$, where vol is the volume of the tetrahedra, and $ratio = \frac{edge\_max}{edge\_avg}$ is the ratio of the length of the longest edge to the average edge length of the tetrahedra. This led to a deviation of at most $\sim$25% for the lowest-resolution results from the best result. This is a significant enhancement over the $\sim$1500% deviation without correction.

However, using f_1 as a correction factor poses a significant problem. The volume that is used in the correction factor is an absolute value, and thus scales the frequencies of all modes, regardless of resolution. This introduces a resolution-independent error into the mode frequencies. To avoid this, it would be necessary to normalize the volume by an optimal volume to gain a relative measure. However, there is no meaningful method to determine such an optimal volume, which makes this approach to determining the correction factor impractical. Thus, the correction factor should be computed from relative tetrahedron shape parameters only.

One such parameter is the relative edge ratio, which was already used in the previous approach to computing the correction factor. Further analyses of the data showed that good results can be achieved by using only the edge ratio. The correction factor $f_2 = \frac{1}{\sqrt{edge\_ratio^3}} = edge\_ratio^{-\frac{2}{3}}$ was determined to be the best fit based on the analyzed example objects. It achieves slightly less consistent mode frequencies for the different resolutions, leading to low-resolution geometries producing up to 32% higher frequencies than the full-resolution mesh. However, for high-resolution meshes with well-shaped tetrahedra, it does not significantly alter the modes' frequency, since the edge ratio is close to 1 for these cases.

Using the correction factor f_2 to scale the results provides a significant improvement for low-resolution approximations. The frequencies of the base modes of an example plate are plotted in Figure 4.14, computed with and without the correction factor. As can be seen, even with the correction factor applied, there is still a slight deviation of the frequency for geometries with very low resolution. However, it is significantly less pronounced that without the correction factor.

For the analysis, geometries with uniform tetrahedra were used to allow a general deduction of the resolution dependency. For generic applications, however, tetrahedra within one geometry can be shaped differently. Therefore, the correction factor is applied individually for each tetrahedron during construction of the stiffness matrix. The global stiffness matrix is assembled by accumulating the element stiffness matrices of each tetrahedron. The correction factor is computed individually for each tetrahedron, and the corresponding element stiffness matrix is scaled by f_2 before adding it to the global stiffness matrix. While this requires some additional computation, it only slightly increases the duration of the physics matrix assembly and is already included in the performance benchmark presented in Section 4.4.2.

4.7 Evaluation of Levels-of-Detail

In the previous sections, different LoDs were presented to allow decreasing the MA computation time in exchange for accuracy. This allows modal data to be computed fast enough for interactive applications. Since lower-resolution approximations produce a noticeable frequency shift, a correction factor was introduced to compensate this effect when using automatically generated LoDs.

Since the approximate LoDs can produce different modal data, and thus the corresponding objects' may produce a different sound, it must be evaluated how strong this change is, and what types of geometries create problems. While lower-quality LoDs can produce different sounds, a moderate deviation is acceptable for the application of interactive sound synthesis, especially for VR applications. For these applications, it is not necessary to create fully realistic sounds; instead, it suffices to create plausible sounds that match the expectations of objects. Additionally, strong approximations are only necessary for run-time a MA, which is usually used for rather simple objects that only produce ambient sound. Objects with higher complexity and are rarely created at run-time usually, but are known beforehand so that a MA can be pre-processed using the full resolution.

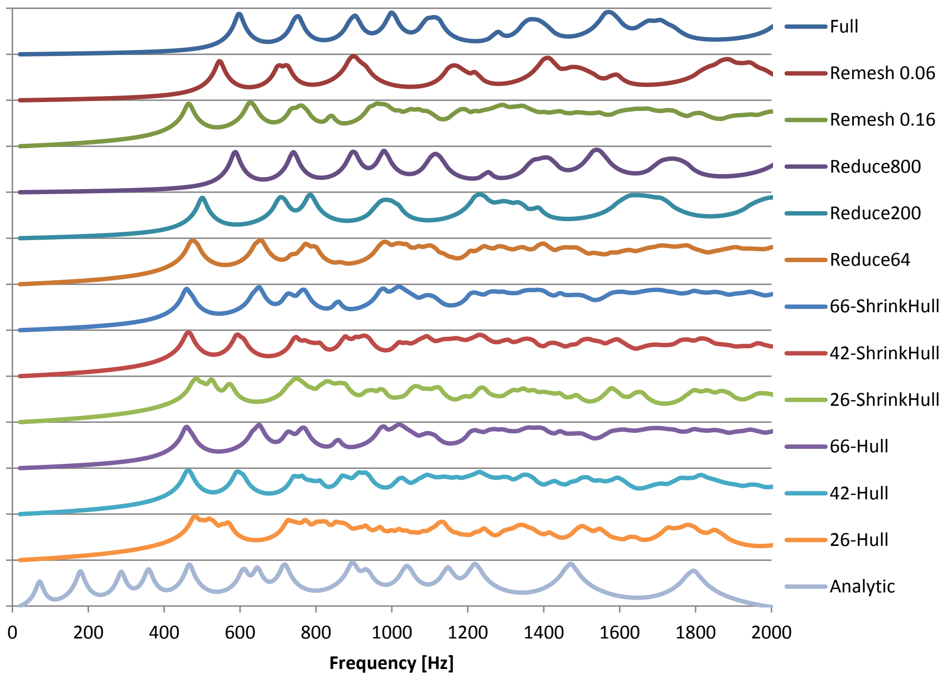
This is supported by related work, which examined humans' recognition of material properties or geometry from sounds. One study examined which aspects of a sound influence the material perception and found that while the pitch has some influence, the decay behavior has a much more significant impact [Klatzky et al. 2000]. Another study also examined the material parameters especially for modal sound synthesis, where the physical material parameters were extracted from recorded sound samples of real geometries [Ren et al. 2013a]. Using these material parameters for a modal synthesis of different geometries showed that the participants of the study had problems distinguishing some materials. However, similar problems were shown to exist when distinguishing materials from real-world sound recordings, further supporting evidence that modal sound data does not have to be fully accurate to be plausible. Bonneel et. al. performed a study where both the visual and the auditory representation of objects was displayed with different quality levels [Bonneel et al. 2010]. The results showed that for good graphical representations, where the material of the objects could be visually determined, it sufficed to use sound with a lower quality to achieve a good multi-sensory perception of materials. This research indicates that a moderate change in pitch or timbre can be accepted for modal sound synthesis.

4.7.1 LoD Examples

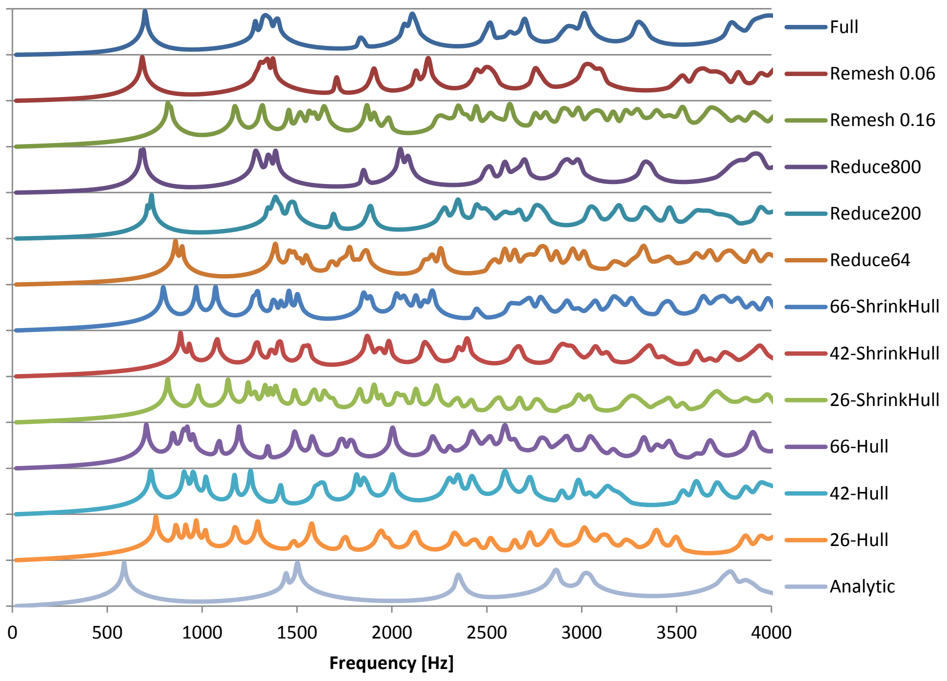
To evaluate the quality of the modal results computed using approximate LoDs, different geometries were examined. Four examples will be discussed here:

- A sphere with a diameter of 1 m, a shell thickness of 10 mm, and material parameters for granite.

- A bell with a base diameter of 637 mm and a height of 485 cm, a shell thickness of 30 mm, and material parameters for brass (see Figure 4.5(a)).

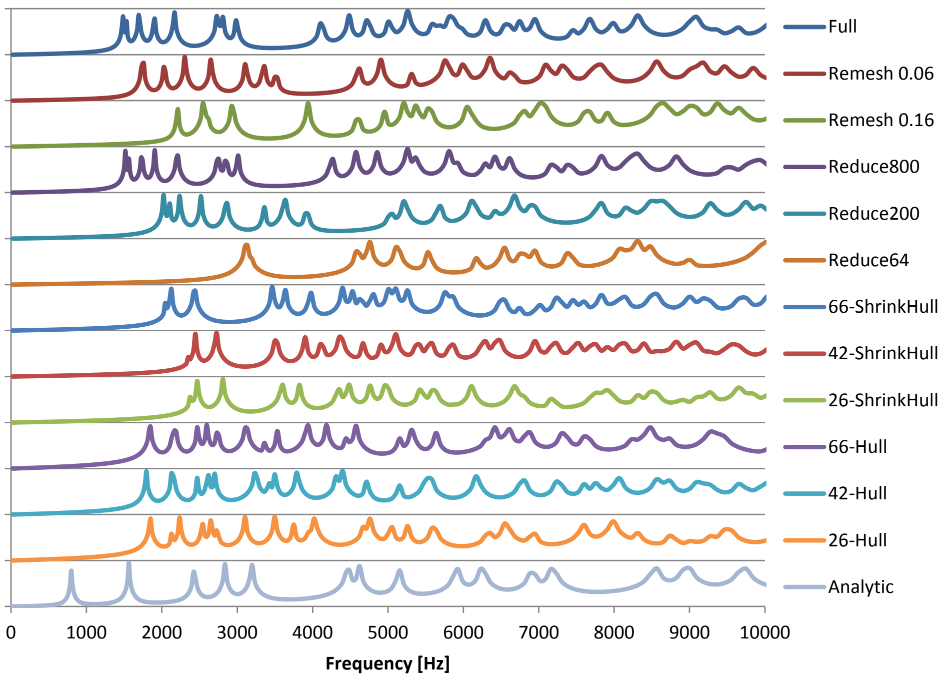


(a) sphere

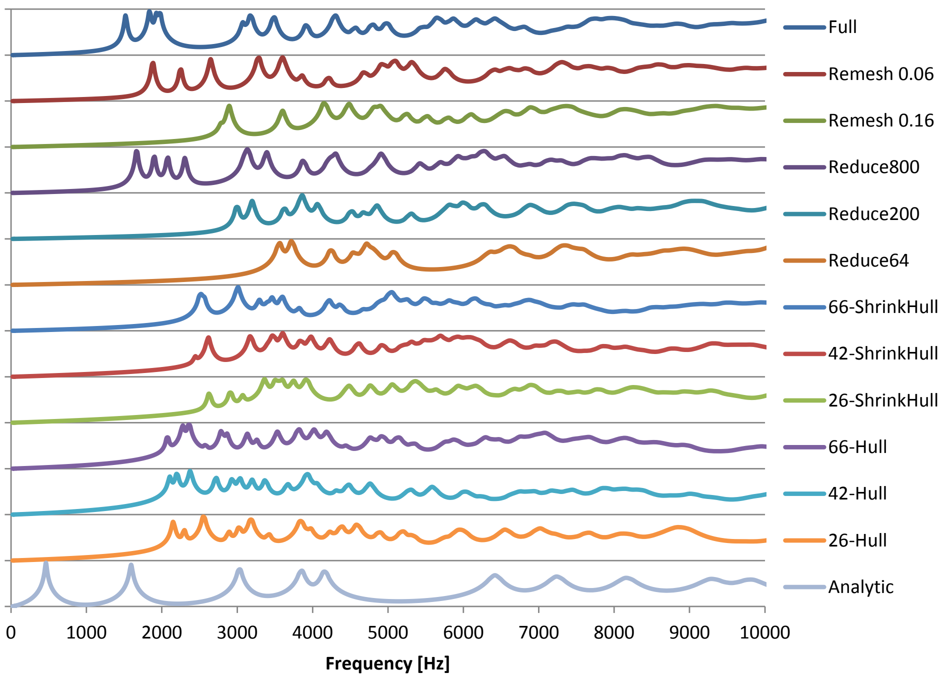


(b) bell

Figure 4.15: Frequency spectrums of the sound produced by different LoDs of (a) a sphere and (b) a bell.



(a) Stanford Armadillo



(b) Stanford Bunny

Figure 4.16: Frequency spectrums of the sound produced by different LoDs of (a) the Stanford Armadillo and(b) the Stanford Bunny geometries.

- The Stanford Armadillo with dimensions of 126 mm × 151 mm × 115 mm, a shell thickness of 2 mm, and material parameters for bronze (see Figure 4.5(b)).
- The Stanford Bunny with dimensions of 155 mm × 153 mm × 118 mm, a shell thickness of 1 mm, and material parameters for ceramic (see Figure 4.5(c)).

The physical parameters for the different materials are listed in Appendix B.

Each of the example geometries was analyzed with full resolution as well as using different approximate LoDs. For the full-resolution LoD, the meshes were optimized using remeshing with a low, object-dependent edge length that resulted in meshes with more than 4000 surface vertices. Intermediate simplification-based LoDs were created using mesh decimation with a target vertex count of 800, 200, and 64 vertices, as well as using remeshing with relative target edge lengths of 6% and 16% of the object's diameter. As low-level LoDs, both the *k*-Hull and the ShrinkHull approximations were computed with 66, 42, and 26 sampling directions. Lastly, an analytic approximation represents a very coarse, but fast to compute LoD.

Sphere

The sphere represents a very simple example that is easy to approximate due to its symmetric, convex shape. Thus, any changes in the results come from the resolution, smoothness, and uniformity of the mesh, not from any missed features in the geometry. Figure 4.15(a) shows the spectrums of modal results computed using the different LoDs of the sphere. The spectrum of the full-resolution mesh shows a base mode at 541 Hz and few clearly separated higher-frequency modes. When reducing the mesh's resolution using either decimation or remeshing, a pitch shift can be observed, resulting in slightly lower frequencies. The base modes for very coarse approximations (decimating to 64 vertices or remeshing with a target edge length of 16% of the diameter, or using hull approximations) lead to a base mode with ∼475 Hz, corresponding to approximately 88% of the full mesh's base mode's frequency. For this example, the introduced correction factor slightly over-compensates the frequency shift observed for non-corrected low-resolution meshes. However, the results are much better than without the correction factor, and the pitch shift is still small enough to produce quite similar sounds.

For lower-resolution meshes, the higher-frequency modes, which are very clearly separated in the full mesh, are often less clear, showing multiple modes with similar frequencies. These can be attributed to the less spherical shape of low-resolution approximations. For the *k*-Hull and ShrinkHull with 26 axes, even the dominant base mode is split into several similar modes, leading to a less distinct sound. However, all higher-level approximations achieve clear peaks for at least the first two modes. Thus, while there is a slight audible frequency shift, the approximations produce very similar results, where only the lowest-level hull approximations have a slightly less clear sound.

Bell

The bell shows similar sound characteristics as the sphere (see Figure 4.15(b)). It has several clearly-defined and separated groups of modes, which creates a very clear, chime-like sound.

As for the sphere, a frequency shift can be observed for lower resolutions. However, unlike for the sphere, the modes shift leads to slightly higher frequencies. The decimation- and remeshing-based LoDs provide good approximations, where only some of the higher-frequency mode groups get less defined.

The Hull approximations also yield a similar base frequency. However, the mode groups are not as well-defined, leading to a less clear sound. This is mostly due to the lower sampling of the bell's rim, which is significant for the lower-frequency modes. Furthermore, the *k*-Hull does not model the hole in the bottom, producing an additional group of low-frequency modes that leads to a different timbre.

Armadillo

The armadillo and the bunny (see Figure 4.16(b)) represent geometries that are more challenging for the approximate MA. Both have protruding features – the bunny's ears, and the armadillo's arms, legs, and ears – which are difficult to model using low-resolution approximations, but produce characteristic sounds.

The armadillo has four limbs – two arms and two legs – that have very characteristic vibration modes with a lower frequency than the modes of the torso (see Figure 4.16(a)). In the resulting sounds, this leads to a group of modes with low frequency that is clearly separated from a group of higher-frequency modes. This property is maintained for medium-quality simplification LoDs. For more strongly simplified geometries (remeshed with 16% edge length or decimated to 64 vertices), the limbs are no longer as well preserved (see Figure 4.6 and Figure 4.7), resulting in changed dimensions and thicker connections to the torso. This results in fewer modes representing the limbs, and an increase in their frequency.

This problem is even more pronounced for the hull approximations. The ShrinkHull approximation still manages to capture the protruding features (see Figure 4.13). However, the ShrinkHull can sample them only coarsely, so that the legs are connected and the arms are attached to the head. As for the low-resolution decimation LoDs, this leads to a higher frequency for the limb mode group, which is also represented with fewer modes. Furthermore, the ShrinkHull over-estimates the object's volume, since some concave features are missed. This leads to a reduced frequency of the torso modes, and thus the separation between the two mode groups is not as strong as for the full-resolution result. Thus, a difference is audible, but the ShrinkHull still produces plausible results.

For the *k*-Hull LoDs, the protruding features are not captured at all, resulting in a convex volume with increased volume, but without modeled limbs (see Figure 4.10). This leads to lower frequencies of the modes, which happen to be similar to the limb modes of the full-resolution mesh. However, there is no longer a separation between the limb and the torso modes. Thus, while the base frequency of the *k*-Hull LoDs is similar to the full-resolution result, the timbre is changed significantly. This illustrates the problem of the *k*-Hull's limitation to convex geometries.

Bunny

The bunny geometry represents a high challenge for the approximation, and leads to the strongest differences in the resulting sound, as shown in Figure 4.16(b). As for the armadillo, the bunny has protruding features – the ears – that produce a group of characteristic, low-frequency modes. However, its ears are more intricate, having a concavity themselves, and are oriented to point along the back.

The shape of the ears poses a problem even for the simplification-based LoDs with medium and low vertex counts. The mesh decimation reduces the ears' features too much, so that the ears lose their special shape and start to shrink for lower-resolution approximations (see Figure 4.6). The remeshing (see Figure 4.7) manages to maintain the size of ears, but does not accurately model their intricate shape. Therefore, the resulting sounds show a similar group of high-frequency modes from the torso, but do not correctly capture the low-frequency mode group representing the ears. This leads to a significantly higher frequency spectrum of the resulting sound for the lower-resolution simplification LoDs.

The hull LoDs have additional problems with approximating the bunny geometry. While the simplification LoDs reduce the fidelity of the ears, the hull approximations merge the ears with the rest of the body, creating a single, larger volume (see Figure 4.10 and Figure 4.13). For the *k*-Hull, a similar problem already occurred for the armadillo. However, while the ShrinkHull manages to coarsely sample the armadillo's features, it fails to do so for the bunny. This is because the ShrinkHull uses sampling directions originating from its center, and thus can only detect concave features pointing towards this center. Since the bunny's ears are pointing backwards from the head, the empty space between the ears and the back is missed by the ShrinkHull construction. Therefore, no separate mode groups are present in the resulting sounds anymore. Since the overall volume is larger, the result has a lower frequency than the full mesh's mode group representing the body, but still higher than the frequency of the ear's modes.

All in all, the bunny requires a decent resolution (around 400 vertices) of the mesh to provide a good quality of modal results produced by the LoD. Otherwise, an audible difference between the sound of the approximation and the full mesh occurs.

4.7.2 Analytic LoDs

While the geometric approximations provide acceptable results for reasonably complex objects, the analytic LoDs' results show large deviations. The analytic result for the bell shape provides a very similar sound to the full mesh and is a better fit than most lower-resolution geometric LoDs. However, this is mostly coincidental, because the base frequency happens to match well, and the analytic MA always provides a very chime-like sound with clearly separated modes. For most objects, however, the analytic result is a much worse fit, as the other three examples show. For the sphere, the frequency is significantly under-estimated, producing a very strong pitch shift. For the bunny and the armadillo, a generally lower pitch can be observed, too. Additionally, the analytic MA generally produces few, widely spaced modes that represent a chime-like sound, which often deviates from the objects' actual timbre. Therefore, the analytic LoD provides too bad an approximation to yield plausible results. As a consequence, geometric LoDs should be preferred whenever possible. The analytic MA was designed as a fallback when the initial geometric approximation cannot be computed fast enough to be available when the object starts sounding, e.g. when several MAs are requested simultaneously. However, due to the analytic results' low quality, it is preferable to instead delay the sound synthesis and wait for the first hull-based LoD to finish. While this can introduce some additional latency, this is better than the high error of the analytic sound.

4.7.3 Discussion

As a conclusion, the use of approximate LoDs for interactive modal sound synthesis is feasible using the presented methods. The quality of the resulting approximate sounds depends on the input object and the approximation type. When comparatively simple objects with few features are analyzed, the approximate LoDs provide good results even for lower-resolution LoDs. Here, the proposed correction factor (see Section 4.6) successfully compensates most of the resolution-dependent frequency shift. For more intricate objects, the lower-resolution LoDs can produce a larger deviation in the resulting sounds, because certain features are missed or volumes are over- or underestimated. In some cases, this can lead to noticeably different results.

For very complex geometries, the geometric approximations can be insufficient. For example, a comb has many individual teeth, all of which represent individual features with corresponding modes. They are very difficult to model using hulls or even medium-resolution meshes, but are dominant for its sound. Such complex objects are problematic for the LoD-based runtime analysis. However, objects with such a high complexity are typically modeled up-front, and can thus be analyzed in a pre-processing step using a full-resolution mesh. Objects that are created or modified at run-time tend to be simpler, so that they are usually less problematic to approximate.

4.8 Run-Time Multi-LoD Analysis

The different presented LoDs allow computing modal data while trading computation time against quality. During run-time, it is important to handle the different LoDs properly, so that low-level approximations are always computed quickly, even when other objects are already being analyzed. This section will present an approach to managing the computation of multiple LoDs of different quality at run-time, using different analysis levels and prioritization schemes. To allow proper prioritization as well as a high utilization of available computational resources, a multi-worker architecture will be presented for the MA computation.

4.8.1 Analysis Levels

The different LoDs are defined by *analysis levels* that specify which geometric approximation to use as a LoD, and how to prioritize its computation.

As the evaluation showed, the Shrink-Hulls are an acceptable first approximation for most objects and can be computed very quickly. They form the initial, highest-priority LoDs to be computed. For the lowest-level LoD, it should be possible to compute multiple objects within a single frame of 16 ms, for which the ShrinkHull with 26 vertices is best suited. Two additional ShrinkHulls with 42 and 66 vertices are used. These ShrinkHull complexities produce geometries with vertex counts increasing by approximately a factor of $\sim$1.5, corresponding to $\sim$3 times the computation cost. This is a suitable increment for low-quality LoDs, because it quickly provides better results without using too many – possibly unnecessary – intermediate steps.

To compute higher-quality results, mesh simplification LoDs are used. Remeshing showed better results than decimation due to a better vertex distribution and proper sampling of large triangles of the input mesh. Even though the remeshing process typically requires more time to compute, this is acceptable for higher-quality LoDs, where the computation time is dominated by the Eigendecomposition. The remeshing method used for this implementation uses a target edge length as the optimization criterion. While this creates well-shaped triangles in the resulting mesh, it makes it harder to control the final vertex count. For low-priority LoDs, a factor of approximately 2 is a good choice for the vertex counts of increasingly complex analysis levels, leading to a decent improvement in the geometry, while increasing the computation time by a factor of around 8. The remeshing criteria can be adjusted based on individual geometries, or the expected amount of geometries that require a run-time analysis. Based on examined geometries, a suitable choice is to use three remeshing analysis levels using 7.5%, 5%, and 3.3% of the object's diameter as target edge length. This typically leads to a good spacing between the remeshing LoDs, but the size of the remeshing LoD is difficult to estimate. The chosen lowest-resolutions remeshing parameters of 7.5% usually result in geometries with 200 to 400 vertices.

It is possible to skip some of the analysis levels when requesting a run-time MA of a new object. For example, one could skip low-resolution LoDs if they are not be needed, e.g. because knowledge of the scene and application ensure that the new object will not start producing sound for a certain time period. Alternatively, one can request only fast, low-quality approximations if the object only exists for a short period or is modified frequently.

When a multi-LoD analysis is requested for a new object, the corresponding approximate MAs are computed for the different analysis levels. When an analysis level finishes, it provides the corresponding modal data. Once a higher-quality result becomes available, the currently available modal data can be replaced with the new one. This can be performed immediately if the object is currently not generating any sound. However, if it has already been excited and is thus producing sound, a smooth transition must be ensured. One could store a history of forces that acted on the lower-quality modal data and apply it to the new data to reproduce a corresponding excitation. However, this can require significant computation time if the forces have been acting over a longer time frame (see Section 5.3). Additionally, since there may be a difference in the sound generated by the two LoDs, a noticeable discontinuity would occur when switching the modal results while the sound is active. Instead, the lower-quality modal result is kept active while it still produces sound, and is only replaced by the newer result once its excitation has decayed.

4.8.2 Priority Classes

When a run-time analysis is requested for a new object, modal data is computed for each analysis level. Thus, multiple LoD analyses are requested simultaneously. The number of simultaneous analysis levels can be even higher if multiple objects request a run-time analysis. It is important to provide an appropriate framework that manages the computation of the different requests. Otherwise, the computation of a long-lasting high-resolution MA could block resources for low-level, high-priority LoDs and delay them too much. Instead, it should be ensured that a first modal result is computed as fast as possible.

For this, a suitable framework is required that prioritizes the different requested LoD analyses. Furthermore, it should be possible to utilize multiple CPU cores as well as multiple remote computers to increase the available computational resources and therefore allow faster analyzes of a higher number of objects.

To achieve a proper prioritization of the different analysis levels, a set of *priority classes* is defined. Each analysis level is assigned to one priority class, although a single priority class can be responsible for multiple analysis levels. When a MA of an analysis level is requested, it is added to the corresponding priority class' queue to await processing. Additionally, each MA request specifies an additional in-class priority that allows sorting the analysis requests within a priority class' queue. This additional priority is assembled from two components. Each analysis level defines a base priority for sorting within its priority class – for example, if both the 42-ShrinkHull and the 66-ShrinkHull analysis levels use the same priority class, the 42-ShrinkHull defines a higher base priority so that it is computed before the 66-ShrinkHull. Additionally, a user-defined priority modifier can be specified, which gives application designers more control over the scheduling. For example, this can be useful to further prioritize objects that are likely to start sounding very soon and thus have to be computed as fast as possible. On the other hand, when a new object is created, but cannot start sounding for some time, a lower priority can be used.

4.8.3 Modal Analysis Workers

Each priority class is assigned a set of *workers* that are dedicated to computing the corresponding LoD analyses. Each worker can compute one MA at a time, and after finishing will request a new analysis from its associated priority class. If no queued analysis is pending, the worker goes to sleep until a new analysis query is queued.

To allow a flexible use of computation resources both on the sound synthesis server and on remote PCs, two different types of workers are supported: local workers and remote workers.

Local Workers

A local worker consists of a thread running on the same PC as the run-time sound synthesis, i.e. on the computer where the resulting modal data will be used to generate modal sounds. This local worker computes the MA inside its thread. While this allows concurrent analyses using multiple worker threads, some aspects have to be considered. If the worker threads would all be working simultaneously, running low-priority analyses could occupy a majority of the resources, thereby delaying the computation of high-priority analysis levels. To prevent this, the thread priority is reduced for workers of lower priority classes, so that the operating system can assign more computation time to high-priority threads.

When using MKL's methods for the Eigendecomposition, it is possible to use a multi-core routines (see Section 4.4.1). However, using these parallelized methods no longer allows prioritizing the usage of computation resources between workers. Therefore, local workers should only use single-threaded Eigendecomposition routines.

Another aspect to consider is that there may be other application components running on the sound synthesis server. In addition to synthesizing the modal sounds after an impact, the same PC might also be used for rendering, physics simulation, interaction computations, etc. It should be avoided to occupy too many resources by the analysis workers, which might otherwise impact the functionality of these other application components. While one could simply limit the number of workers to the number of CPU cores that may be occupied for the modal analysis, this would only allow very few threads and limit the versatility of setups. Instead, thread affinities are used to restrict the worker threads to certain CPU cores, thereby leaving other cores free for other tasks.

Using local workers, small MAs can be computed quickly. However, especially when requesting multi-LoD analyses of several objects, the higher-quality LoDs require significant computation time. Computing all analyses locally can delay the high-resolution results, especially since multi-core Eigendecomposition routines should not be used by local workers.

Remote Workers

To speed up the computation of medium and high quality LoDs, one can use a remote workers that transmit the analysis request to a MA server running on a different PC. The modal result is then computed on the remote PC and the result is sent back. This allows using multi-core Eigendecomposition, and using multiple remote workers running on multiple PCs provides more computation ressources and allows to compute more analysis levels at the same time. However, as discussed in Section 4.4.2, computing an analysis on a remote PC produces an overhead since the query and result must be serialized and transferred over a network connection. Therefore, while remote workers are well-suited for higher-quality analysis levels, time-critical high-priority analysis levels should be computed locally.

4.8.4 Performance

The performance of the multi-LoD analysis will be demonstrated using two different worker configurations. Both configurations compute seven analysis levels in different priority classes: *highest*, *high*, *medium*, and *low*. The highest priority class computes the 26-ShrinkHull approximation, while the high class computes the ShrinkHull levels for 44 and 66 directions. The medium priority class analyzes LoDs using remeshing with 7.5% and 5% target edge length. The low priority class computes MAs for geometries remeshed with 3.3% target edge length, as well as the full-resolution geometry.

For the benchmark, the sound synthesis PC uses an Intel Xeon E5-1620 (4 cores at 3.66 GHz) and 16 GB RAM. The test application was compiled using gcc 4.9. In addition to the MA, the test application also runs interaction threads, as well as one thread performing continuous computations to simulate load.

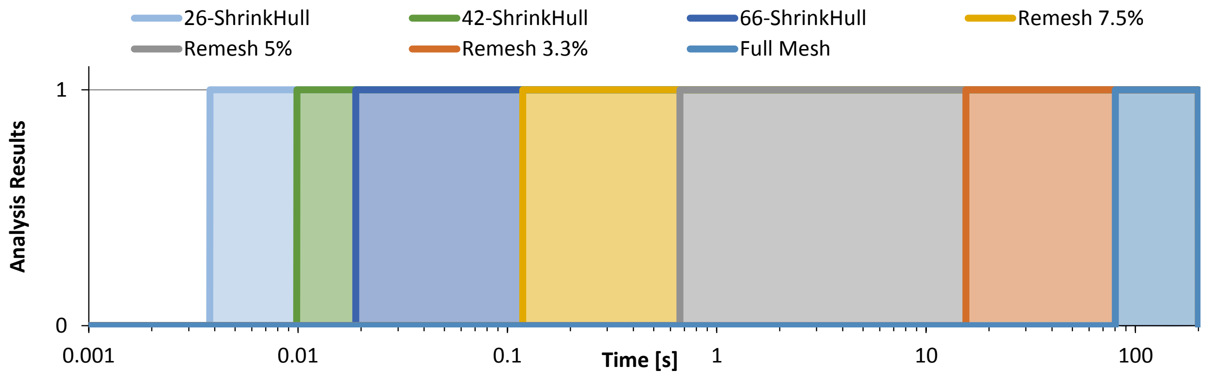
For the benchmark, different amounts of objects are created and a multi-LoD analysis is requested for each. The objects represent procedurally generated glass shards and consist of flat, non-convex shapes with jagged edges. Each shard has 120 to 160 vertices and a diameter of 40 to 50 cm and is modeled as a thin-shell volume with a thickness of 5 cm. For the full mesh, a relatively low resolution is chosen, resulting in ∼2500 physical mesh vertices. This is sufficient for the simple shards analyzed here, and the limited resolution leads to reasonable run-times even for a higher number of objects. Depending on the objects and the scenario, a higher resolution for the full mesh can be chosen for other applications.

The benchmark is performed using two different worker configurations. The first configuration utilizes only a single PC with local workers. Each priority class uses four local workers, resulting in a total of 16 analysis threads. To prevent them from slowing other parts of the application, all local workers are restricted to the same two CPU cores by using thread affinities. Although only two cores are used, four worker threads were chosen per priority class. This proved to provide a slightly better performance than using two or three threads each, due to the CPU's hyper-threading capability.

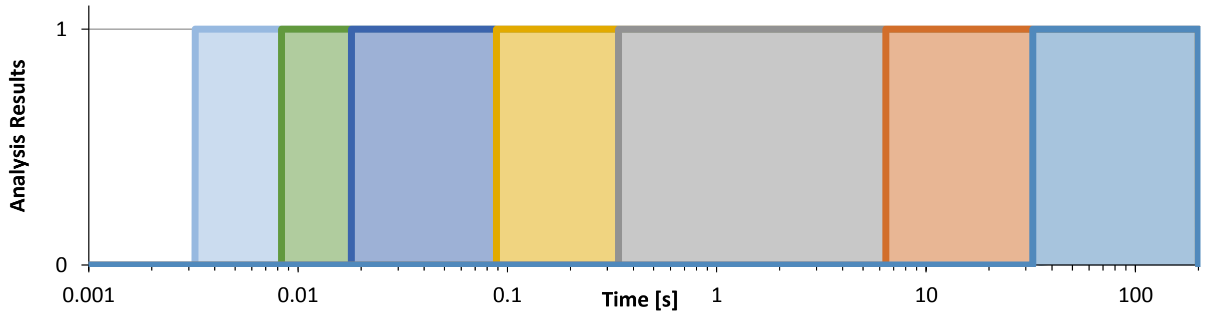
The second configuration uses local workers only for the highest and high priority classes, again using four threads per priority class, restricted to two CPU cores. For the medium and low priority classes, remote workers are used. A total of six remote computers with the same hardware as the sound synthesis PC are used, which are connected by a Gigabit Ethernet. Each remote PC runs one MA server configured to use four local threads for the Eigendecomposition, and both the medium and low priority classes utilize three of the remote workers each.

The first benchmark scenario examines the performance when computing a multi-LoD analysis of a single object using the two configurations (see Figure 4.17). Initially, when computing the ShrinkHull approximations, both setups perform very similarly, providing the first result after 3 to 4 ms, and finishing the 66-ShrinkHull in about 18 ms. However, the remote setup can compute the remeshing LoDs as well as the analysis of the full mesh faster, because it can distribute them onto different hosts and utilizes four cores for each Eigendecomposition. Thus, the final analysis result of the full-resolution mesh is available after 32 s seconds when using the remote setup, while it requires 80 s when using only local threads.

As a more challenging benchmark scenario, 25 shard objects are created simultaneously, corresponding to the fracture of a glass plate creating many procedurally generated shards. This leads to a high number of analysis requests to be created at the same time, saturating the analysis queues. The results in Figure 4.18 show a noticeable delay before a first result is available for all requested objects. 26-ShrinkHull or better results are available for all objects after 49 ms using the local configuration, and after 35 ms for the remote configuration. Because the operating system does not allocate all computation resources to high-priority threads, threads computing a higher-quality LoD use up some computation resources. Since the local configuration has more worker threads active, its highest-priority workers are

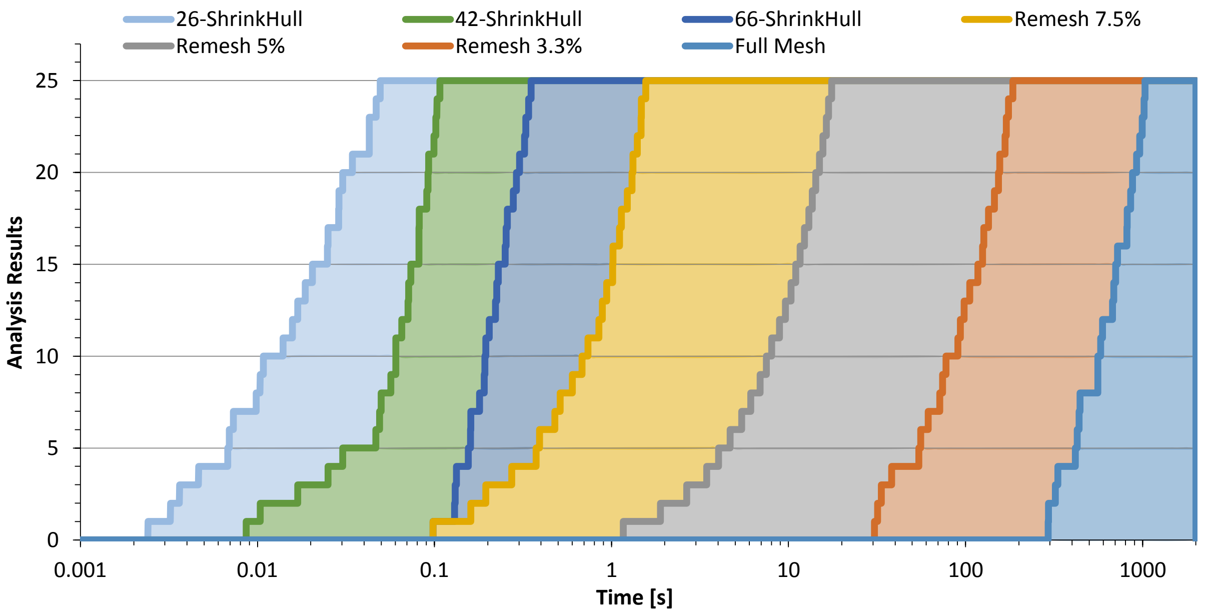


(a) local setup

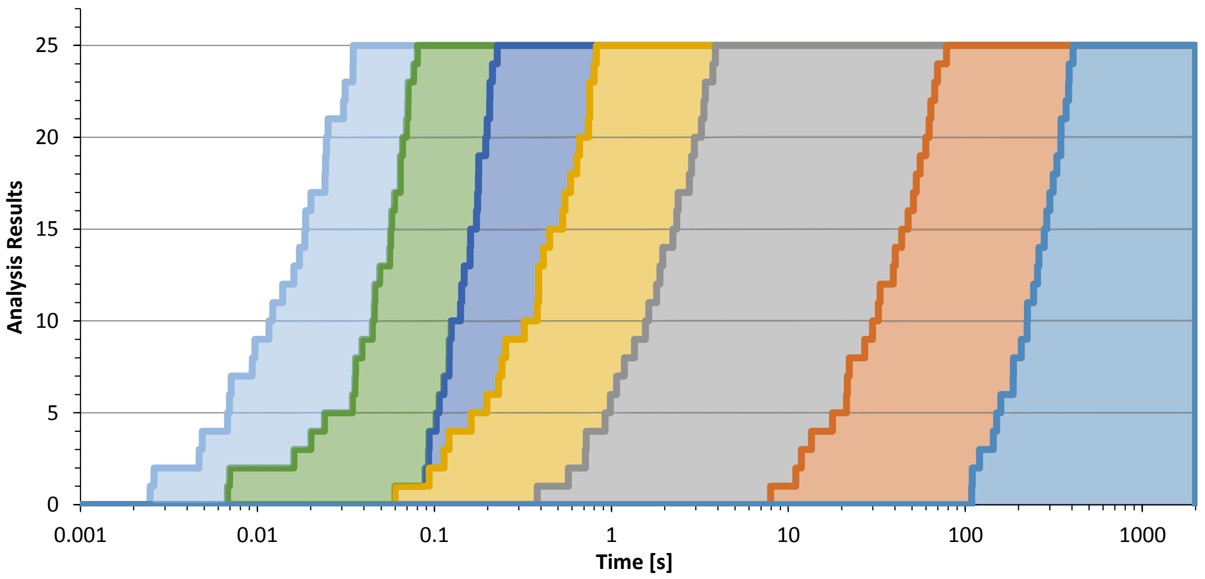


(b) remote setup

Figure 4.17: Time to compute multiple analysis levels when analyzing a single object using (a) a local and (b) a remote computation setup.

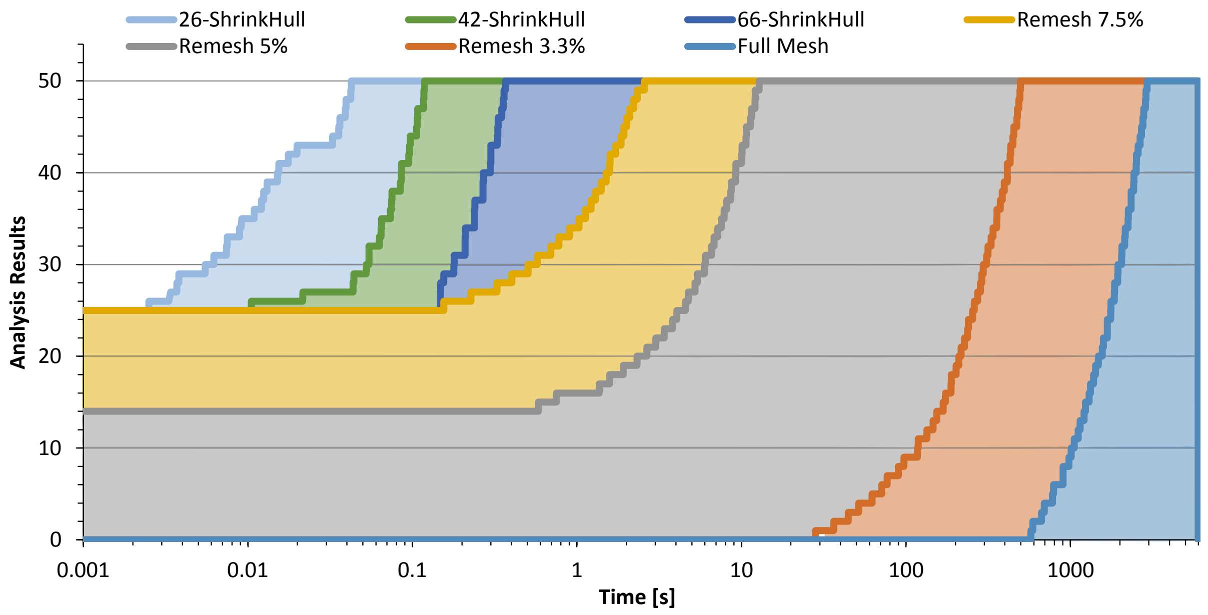


(a) local setup

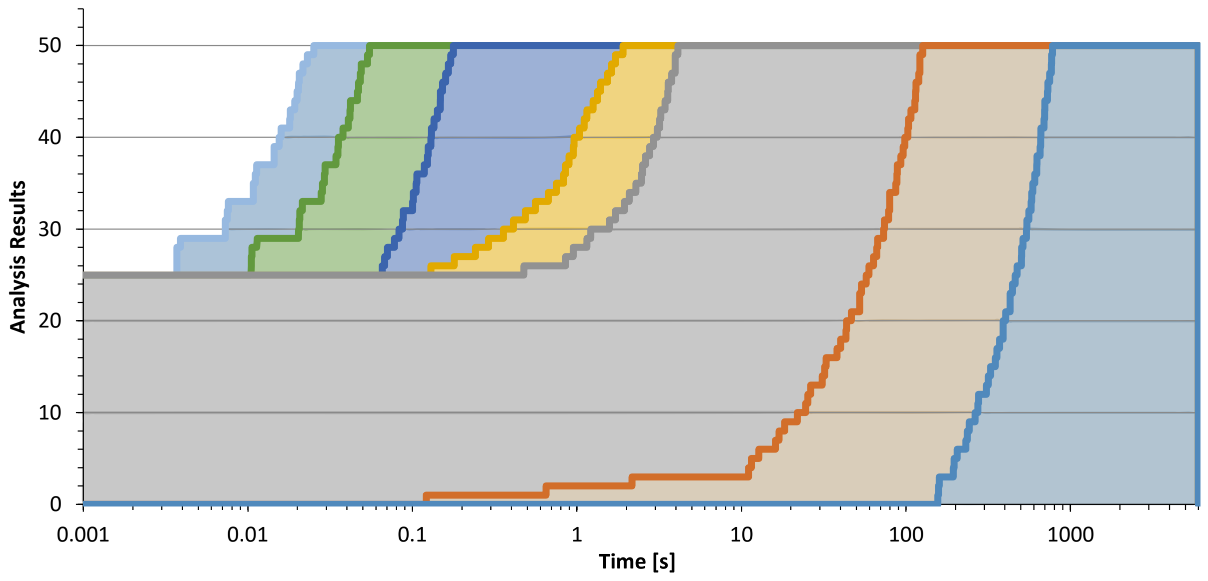


(b) remote setup

Figure 4.18: Time to compute multiple analysis levels when requesting an analysis of 25 objects using (a) a local and (b) a remote computation setup.



(a) local setup



(b) remote setup

Figure 4.19: Finish time of different analysis levels when requesting 25 objects, followed by another 25 object requests 10 seconds later, using (a) a local and (b) a remote computation setup. Finish times are relative to the start time of the second batch of objects.

delayed more than for the remote setup. In should be noted that this effect depends on the scheduling of the operating system – the Linux OS on which the benchmark was performed assigned some computation time to lower-priority threads, while on a Windows OS the prioritization was stricter, so that higher-priority threads received nearly all of the computational resources and thus the highest priority analyzes finish fast even if many low-priority threads are active.

Neither setup can compute initial modal results of all 25 object within the time span of 16 ms. Still, considering the high number of objects, the results are acceptable, leading to a delay of one to two frames over the targeted 16 ms limit. In these ranges, it is still acceptable to increase the latency to wait for results.

Higher-quality results arrive afterwards, successively improving the results. For the hull approximations, the remote setup is slightly faster than the local configuration, finishing all 42-ShrinkHulls in 80 ms and all 66-ShrinkHulls in 226 ms, as opposed to 107 ms and 350 ms for the local setup. For the higher-quality results, the remote setup can utilize its remote resources and thus is several times faster. The three remeshing analysis levels and the full mesh are finished after 0.8 s, 3.9 s, 78 s, and 406 ms using the remote workers, but require 1.8 s, 17.5 s, 184 s, and 1030 s when using the local setup. All in all, the remote setup speeds up the computation especially of the higher-quality results, and allows a higher number of object to be analyzed, or to provide results faster.

Another important requirement for the multi-LoD analysis is that low-priority computations are not delayed if higher-quality analysis levels of previously queued objects are already being processed. To examine such a scenario, a second batch of 25 objects is created 10 s after a first batch of 25 objects. Figure 4.19 shows the finish times of the pending queries after the second batch was started. For both configurations, the results for the high-priority jobs, i.e. the three ShrinkHull analysis levels, are very similar to the second bechmark, which had no pending requests. Thus, the different priority classes work as intended, allowing a prioritization even under high load.

4.9 Discussion

This chapter presented methods to allow run-time computation of a MA for interactively created or modified objects. Using different LoDs, it is possible to compute a first approximate result within the time constraints of a real-time application. Different geometric approximations are proposed, including the ShrinkHull that was specifically designed to provide a low-resolution approximation fitting the requirements for a MA mesh, and can be computed very quickly. Since the MA of lower-resolution meshes can produce a strong frequency shift, a correction factor was derived that compensate this effect, leading to a better match between the results of high- and low-resolution analysis.

An evaluation showed that the approximate results are reasonably similar to the results of a full-resolution analysis as long as the input object has a limited complexity. However, complex objects with distinctive protruding features cannot always be handled properly, leading to an audible change in sound. Still, the changes in the sound are usually limited enough to produce a plausible sound.

To allow computation of different LoDs for multiple objects, a priority-based scheduling was proposed, utilizing local threads and optionally remote PCs. This enables the analysis of several objects within the limit of real-time applications.

In conclusion, the presented approaches make it possible to compute the MA of new objects during run-time A first result is computed fast enough even if multiple objects are created simultaneously and start sounding almost immediately afterwards.

CHAPTER 5

FAST RUN-TIME SOUND SYNTHESIS WITH ACTIVE FORCES

In the previous section, the procedure of a MA with a focus on the run-time analysis of interactively created objects was presented. The MA computes modal data for the object from its geometry and material properties, either at run-time or in a pre-processing step, . This modal data contains the modes corresponding to the objects characteristic vibrations, and the corresponding mode shapes as well as the gain matrix, which defines how forces acting on different parts of an object excite the modes.

Using this data, it is possible to excite the modes of an object through acting forces, and compute the resulting modal vibration. This vibration leads to deformations of the surface, which create periodic changes in the pressure of the surrounding air, thus producing sound.

Since the dynamics system is split into independent damped oscillators during the MA, the modes are regarded as independent. Thus, to compute the sound produced by modal vibrations, one can accumulate the sound contribution of each individual mode (see Section 2.2.2), which makes the synthesis efficient enough to be computed in real-time. The computation of the modal sound contribution of a single mode at a given time is a simple operation. Still, the complete modal sound synthesis can require significant computation time because sound has an audible range from 20 Hz to 22 kHz, and thus the sampling rate at which the sound samples are calculated is typically 44.1 kHz. Furthermore, there are often multiple objects that produce sound simultaneously. Since each object can have a lot of modes – often several hundreds or thousands – a high number of mode contribution evaluations have to be computed to synthesize the final sound.

In the past, several approaches have been proposed to speed up the real-time synthesis from modes, allowing for a high number of active sounding objects (see Section 5.1). These assume objects with one initial excitation followed by a continuously decaying sound – i.e. after a force acted on an object. However, for high-quality sound synthesis, it is important to allow sounding objects with dynamic forces acting over a longer period and with a high temporal resolution. This is an important aspect of the sound synthesis and relevant for a convincing contact sound (see Section 5.3). Especially for longer-lasting, changing contacts – as for example occur when an object slides, rolls or scratches along another one – the dynamic nature of forces is an important factor, and the excitation should be computed at a high temporal resolution to avoid sound artifacts. However, a force excitation with high temporal resolution adds significant complexity to the synthesis process, and poses an additional challenge to the run-time synthesis.

The high number of primitive operations required to compute the synthesized modal sound as well as the excitation by acting forces provides a challenge for the real-time sound synthesis. This chapter examines approaches to allow a high number of simultaneously sounding objects. Here, the cases of synthesis without active forces as well as with multiple active forces will be handled.

Due to the mutual independence of modes, the modal sound synthesis lends itself well to parallelization. Since a very low latency is required when synthesizing sounds, a network-based parallelization using multiple PCs is problematic. However, Graphics Processing Units (GPUs) can provide large computational capabilities for certain parallel tasks, and can be useful to speed up the modal sound synthesis.

In this chapter, approaches for a fast run-time modal synthesis will be examined. First, related work on accelerating the modal sound synthesis process is discussed in Section 5.1. In Section 5.2, approaches for the synthesis without active fores will be presented. In Section 5.3, these methods are extended and enhances for the synthesis while forces are acting on the object, leading to addition excitation of modes during the synthesis. For both cases, different approaches for the computation of the sound samples as well as the force excitation are discussed, and examined for their viability for CPU- and GPU-based synthesis. The results of these approaches are discussed in Section 5.4.

5.1 Related Work

Modal sound synthesis has gained increasing attention over the last decade. As a consequence, improving the performance of the run-time synthesis has been a topic of previous research, which will be discussed here.

One way to reduce the computation time required for the synthesis is to lower the number of modes whose contributions have to be calculated. Here, one can use the fact that depending on the location where external forces are acting on the object, different modes are excited with different magnitudes. Thus, after forces have excited an object, some modes may be inaudible because they received such a small excitation that their contribution to the overall souns is much lower than for other modes. Over time, modes are damped and thus can become inaudible during the synthesis process. These inaudible modes can be omitted from the synthesis process.

Doel et. al. have proposed a synthesis framework that truncates modes using different perceptual heuristics [van den Doel et al. 2004]. At regular intervals, they check the amplitude of each mode to determine if it provides an audible contribution to the overall sound. In addition to an absolute audibility threshold, the authors propose using masking effects, where one loud mode dominates the perception of more silent modes with a similar frequency, which are then dropped from the synthesis. Depending on the sounding objects in a scene and the chosen truncation threshold, this can lead to a significant reduction in the number of modes to be computed per sound sample.

The approach of mode truncation was later adopted by Raghuvanshi and Lin [Raghuvanshi et al. 2006]. Additionally, they proposed to use a pre-processing step that merges similar modes of an object, in order to reduce the total mode count independent of the run-time excitation. The authors merge modes whose frequency difference is below the threshold that allows humans to discriminate two frequencies. This mode compression can significantly reduce the number of modes per object, which leads to a faster run-time synthesis.

However, merging modes with similar frequencies poses some problems. Each mode has a mode shape and a corresponding gain that define the relative force they receive when external forces are acting. When modes are merged, a simple addition of the mode shapes leads to errors due to the superposition of different shapes. Thus, forces may excite merged modes to a different degree than the original modes. Another problem is that the authors propose merging modes when their individual frequencies cannot be distinguished. However, when modes with slightly different frequencies are active simultaneously, they can produce beats, which create a characteristic sound. This effect would be lost when merging such modes.

Another approach for speeding up the summation of a large number of periodic functions is to perform the evaluation in the frequency domain. Instead of evaluating and accumulating the functions for each sound sample – i.e. in the time domain – the computation is performed in the frequency domain [Freed et al. 1992] for a larger block of samples, and an inverse Fast Fourier Transform (FFT) is used to compute the resulting samples. This approach was also adopted for modal sound synthesis [Bonneel et al. 2008]. The authors report a speed-up of factor five to eight for typical scenes when using frequency-domain modal synthesis instead of a time-domain synthesis. However, the contributions of modes are not pure sinusoids, but are also exponentially dampened. Thus, the frequency spectrum of each mode is defined by a Gaussian bell. Since the frequency domain contributions of the modes are uniformly sampled into blocks, this can lead to aliasing artifacts when only few samples are available per mode spectrum. The reported speed-up was gained by using only three to five samples per mode, and no evaluation of the quality of the resulting sound was provided.

Other approaches have examined the use of graphics cards for modal synthesis. An early approach used shaders to evaluate the modes on a GPU, and report a maximum of 32000 modes that can be handled in real-time, although the results omit details like sampling rate, excitation, and data transfer [Zhang et al. 2005].

In a more recent publication, the GPU was used to evaluate and accumulate sines [Savioja et al. 2010]. They can add the contributions of more than a million sines when using larger sample buffer sizes, but only when ignoring phase information. Similarly, Tsai et. al. presented a GPU-based framework to compute instruments modeled using spectral components – i.e. sinusoidal tones – combined with a residual parameter-independent sound [Tsai et al. 2010]. Using their approach, they can handle up to 5000 of instruments with 50 spectral components each. Both of these approaches use sine functions as basis for the sound synthesis, However, for modal analysis, it is also required to handle the damping of the modes, which is typically modeled as an exponential decay. This adds additional computational cost.

Previous research has proposed methods to speed up the modal synthesis after an initial excitation of an object modes. Computing this excitation from acting external forces is also a time-consuming step, which should be included in the synthesis at audio sample rate to achieve high-quality results. To our knowledge, there currently exists no related work that examines this force-based mode excitation at sound sample rates in real-time.

5.2 Sound Synthesis from Active Modes

In this section, the modal sound synthesis after an initial excitation will be discussed. Thus, it is assumed that an object has already been excited by an external force, but does not receive any further excitation while the sound is being generated. Approaches for the application of force excitation at run-time will be described in the following Section 5.3.

5.2.1 Sound Contribution of Modes

A MA computes the modal data of an object, consisting of a set of modes (see Section 2.2.1). Each of these modes is defined by its angular frequency ω_k and damping γ_k, as well as its mode shape $\mathbf{s}_k$. The decay and angular frequency can also be combined into the *complex angular frequency* $\underline{w}_k = \gamma_k + i \cdot \omega_k$.

At run-time, each mode has a complex *amplitude* $\underline{a}_k = a_k + i \cdot b_k$ that represents its current excitation. The real component of the amplitude a_k represents the potential energy of the mode at time $t = 0$, i.e. the amount of the deformation induced by the mode. The imaginary component b_k represents the kinetic energy at $t = 0$. The mode vibrates because energy is periodically tranferred from potential to kinetic energy, leading to surface deformations. Together, these two values specify the amplitude and phase of the modal vibration.

Once a mode has been excited, its mode state $\underline{r}_k$ can be computed at time t:

$$r_k(t) = \underline{a}_k \cdot e^{\underline{w}_k \cdot t} + \underline{a}_k^* \cdot e^{\underline{w}_k^* \cdot t} \tag{5.1}$$

Here $\underline{x}^*$ is the complex conjugate of a complex number $\underline{x}$. Note that although the amplitude $\underline{a}_k$ and angular frequency $\underline{w}_k$ are complex, the mode state is a real number.

$$\begin{aligned} r_k(t) &= \underline{a}_k \cdot e^{\underline{w}_k \cdot t} + \underline{a}_k^* \cdot e^{\underline{w}_k^* \cdot t} && (5.2)\\ &= (a_k + i \cdot b_k) \cdot e^{\omega \cdot t} \cdot e^{i \cdot \gamma_k \cdot t} + (a_k - i \cdot b_k) \cdot e^{\omega \cdot t} \cdot e^{-i \cdot \gamma_k \cdot t} && (5.3)\\ &= e^{\gamma_k \cdot t} \cdot \left(a_k \cdot e^{i \cdot \omega_k \cdot t} + a_k \cdot e^{-i \cdot \omega_k \cdot t} + i \cdot b_k \cdot e^{i \cdot \omega_k \cdot t} - i \cdot b_k \cdot e^{-i \cdot \omega_k \cdot t}\right) && (5.4)\\ &= 2 \cdot e^{\gamma_k \cdot t} \cdot (a_k \cdot cos(\omega_k \cdot t) + b_k \cdot sin(\omega_k \cdot t)) && (5.5) \end{aligned}$$

Given a mode state $r_k(t_s)$ at a sample time t_s, its contribution to the overall sound can be computed. For this, the acoustic transfer factor q_k is used, which is computed for each mode based on the corresponding mode shape, and represents the relative sound energy radiated by the surface vibration induced by this mode (see Section 4.3.6 for details). Using the mode states and transfer factors of all modes, the overall sound $s(t_s)$ produced by an object at sample time t_s can be computed by accumulating the individual mode contributions.

$$s(t_s) = \sum_{\text{Mode } k} q_k \cdot r_k(t_s) \tag{5.6}$$

While the formula to compute the mode state $r_k(t)$ at an arbitrary sample time t_s is fairly simple, it still requires the evaluation of trigonometric functions and natural exponents for each mode and sample. Using this computation method of directly evaluating Equation 5.5 is hereafter called the *explicit* method.

A more efficient approach that was proposed by O'Brien et. al. [O'Brien et al. 2002] is the use of an *incremental* computation. For sound synthesis, audio samples are computed at intervals with a fixed time difference Δt, e.g. $\Delta t = \frac{1}{44100}$ s for a typical audio sampling rate of 44.1 kHz. This fixed time step can be used to simplify the computation. The mode state $\underline{a}_k^{(n)}$ at sample n can be computed based on the amplitude $\underline{a}_k^{(n-1)}$ at the previous sample $n-1$.

$$\underline{a}_k^{(n)} = \underline{a}_k^{(n-1)} \cdot e^{\underline{w}_k \cdot \Delta t} \tag{5.7}$$

Since Δt is constant, the update factor $e^{\underline{w}_k \cdot \Delta t}$ can be computed up-front for each mode. Then, the update only requires a single complex multiplication to update the amplitude from the previous to the current sample. Now, the evaluation of the mode state is simplified by the fact that the relative time is $t = 0$. Thus, the mode state can be easily computed.

$$r_k(0) = \underline{a}_k \cdot e^{\underline{w}_k \cdot 0} + \underline{a}_k^* \cdot e^{\underline{w}_k^* \cdot 0} = \underline{a}_k + \underline{a}_k^* = 2 \cdot a_k \tag{5.8}$$

This way, successive sample contributions can be computed using a simpler, faster update step.

It can be noted that the factor q_k used for computing the k-th mode's sound contribution is constant for each mode. To reduce the performance cost of the sound synthesis, this factor can be directly included in the mode amplitude $\underline{a}_k$ by scaling the corresponding gain values accordingly, without changing the overall results. For the remainder of this chapter, it will be assumed that the acoustic transfer is already included in the amplitudes.

5.2.2 Mode Truncation

As proposed by previous research, it is possible to reduce the computation effort by omitting modes that have no audible contribution to the final sound (see Section 5.1). Due to a possible reduction in sound quality, merging of modes with similar frequency will not be used in the approaches presented in this thesis. Instead, the presented approaches will only utilize run-time truncation of modes based on their relative amplitudes. For this, a measure of the loudness of the mode is required. This can be measured by the absolute value of the complex-valued amplitude $l = |\underline{a}| = |a + i \cdot b| = \sqrt{a^2 + b^2}$. Note that this is only a valid measure for the mode's loudness if the amplitude already includes the acoustic transfer, as discussed in Section 5.2.1. Otherwise, the value still has to be scaled by the mode's acoustic transfer.

Given the loudness of all modes, the set of currently active modes can be determined. If a mode's amplitude l_k is below a certain fraction of the amplitude of the loudest mode $l_{max} = \max_{\text{Mode } k} \{l_k\}$, it is dropped from the list of active modes. In the presented implementations, a truncation threshold of $l_{rel} = 0.001 \cdot l_{max}$ is used, corresponding to a loudness difference of

$-60\,\mathrm{dB}$. Additionally, an absolute threshold of $l_{abs} = 0.0001$, corresponding to $-80\,\mathrm{dB}$ is used, below which a mode is regarded as inaudible regardless of other modes' loudness.

Using this method, it is possible to reduce the number of modes that have to be evaluated for the sound synthesis. The modes' truncation state is computed at regular intervals, and then used for a block of multiple sound samples before re-computing it. After an object has been excited, assuming that no further excitations follow, all modes' amplitude decays over time, leading to fewer and fewer active modes, until finally all modes are inaudible and the object stops sounding. The percentage of modes that can be truncated thus depends on the time elapsed since the excitation, but also on the geometry, resolution and material of the object and the location the excitation forces act on. Therefore, it is difficult to estimate the amount of modes that can be disabled. In the benchmarks presented later on, example geometries will be used to illustrate the efficiency of truncation depending on the number of active modes.

Related work also proposes the use of mode masking, where loud modes mask more silent modes with similar frequency [van den Doel et al. 2004]. While this can lead to more modes being truncated, this approach is not used in the presented method for multiple reasons. Using masking at run-time requires more computation to determine if a mode can be truncated. The masking threshold depends on the amplitude of all other active modes, thus requiring $O(n^2)$ truncation checks for n active modes, instead of $O(n)$ checks when only comparing to the maximum amplitude. Especially for objects with many modes, the time required by the masking check can diminish the performance gained by truncating modes. Additionally, for the scenarios examined in this thesis conservative thresholds were used in order to reduce the perceptibility of truncation. For such thresholds, masking only slightly reduces the number of truncated modes over the threshold-based approach. When using less conservative thresholds, truncated modes that are masked by a nearby louder mode may still have a non-negligible amplitude, and simply deactivating them can lead to discontinuities in the synthesized signal, producing audible artifacts in the sound output. To prevent this, truncated modes would require a fade-out to prevent audible artifacts, thus delaying the effect of the truncation. For these reasons, masking will not be used in the implementations in this chapter. However, the general algorithms presented here can be applied to other truncation approaches, which may add masking or use different, less conservative thresholds.

5.2.3 CPU-based Sample Computation

When synthesizing the sound produced from a modal object using a CPU, the computation can be implemented using either the explicit and the incremental approach.

Explicit Algorithm

The explicit algorithm is listed in Algorithm 5.1. As data, it only requires the mode parameters and amplitude, as well as the current time stamp. It computes and accumulates each mode's sound contribution according to Equation 5.5, and writes it to the corresponding position in the output signal buffer.

While the explicit algorithm is easy to implement, it should be noted that problems with denormalized floating point numbers can occur due to the continuously increasing time stamp.

```
function ResetTime()
  foreach mode m do
    exp_t = exp(m.decay * t)
    sin_t = sin(m.ang_freq * t)
    cos_t = cos(m.ang_freq * t)
    real = m.amplitude[0]
    imag = m.amplitude[1]
    m.amplitude[0] = exp_t * (real * cos_t - imag * sin_t)
    m.amplitude[1] = exp_t * (real * sin_t + imag * cos_t)
  end
  t = 0
end

function compute_buffer(buffer, buffer_size)
  if t > 0.1 then ResetTime() end
  for i from 0 to buffer_size - 1 do
    buffer[i] = 0
    foreach mode m do
      contrib = 2 * exp(m.decay * t)
                * ((m.amplitude[0] * cos(m.ang_freq * t)
                  + m.amplitude[1] * sin(m.ang_freq * t))
      buffer[i] = buffer[i] + contrib
    end
    t = t + 1 / sample_rate
  end
end
```

Algorithm 5.1: Explicit CPU algorithm for modal sound synthesis using (CPU-E).

```
function prepare_modes()
  delta_t = 1 / sample_rate
  foreach mode m do
    m.delta[0] = exp(m.decay * delta_t) * cos(m.ang_freq * delta_t)
    m.delta[1] = exp(m.decay * delta_t) * sin(m.ang_freq * delta_t)
  end
end

function compute_buffer(buffer, buffer_size):
  for i from 0 to buffer_size - 1 do
    buffer[i] = 0
    foreach mode m do
      real = m.amplitude[0]
      imag = m.amplitude[1]
      buffer[i] = buffer[i] + real
      m.amplitude[0] = real * m.delta[0] - imag * m.delta[1]
      m.amplitude[1] = imag * m.delta[0] + real * m.delta[1]
    end
  end
end
```

Algorithm 5.2: Incremental CPU algorithm for modal sound synthesis (CPU-I).

The result of the exponential function $e^{-\gamma \cdot t}$ in Equation 5.5 can become very small for highly damped values and larger values of t. For example, objects made from soft wood often has modes with a damping coefficient of around $\gamma = -400$, and thus denormalized values occur when $t > 0.2\,\text{s}$. Denormalized floating point values can lead to decreased performance or wrong results. To prevent denormalization, the time is reset to zero at regular intervals, by adjusting the mode amplitude accordingly (see Algorithm 5.1).

Implicit Algorithm

The implementation using the incremental variant is shown in Algorithm 5.2. It requires computing an additional complex-valued update factor $e^{\underline{w}_k \cdot \Delta t}$ for each mode, based on the sample rate-dependent time difference between two samples. This update factor can be computed once, and stored for each mode for usage during the run-time synthesis. This factor is used to update the mode's amplitude at each sample according to Equation 5.7. The real part of each mode's amplitude is accumulated into the sample value. This approach leads to a more efficient implementation, using less arithmetic operations than the explicit approach.

Parallelization

The presented algorithms for the explicit and the implicit synthesis both only use a single CPU thread. One could use multiple CPU cores to compute the modal sound synthesis of a single object by distributing its modes onto multiple threads. However, a single sounding object can

```
function prepare_modes()
  active_modes = {}
  delta_t = 1 / sample_rate
  foreach mode m do
    m.delta[0] = exp(m.decay * delta_t) * cos(m.ang_freq * delta_t)
    m.delta[1] = exp(m.decay * delta_t) * sin(m.ang_freq * delta_t)
    active_modes.add(m)
  end
end

function check_mode_truncation()
  max_amp = get_max_mode_amplitude()
  sq_thresold = max(0.001 * max_amp, 0.0001)ˆ2
  for m in active_modes do
    sq_amplitude = m.amplitude[0]ˆ2 + m.amplitude[1]ˆ2
    if sq_amplitude < sq_threshold then
      active_modes.remove(m)
    end
  end
end

function compute_buffer(buffer, buffer_size)
  check_mode_truncation()
  for i from 0 to buffer_size - 1 do
    buffer[i] = 0
    foreach mode m do
      real = m.amplitude[0]
      imag = m.amplitude[1]
      buffer[i] = buffer[i] + real
      m.amplitude[0] = real * m.delta[0] - imag * m.delta[1]
      m.amplitude[1] = imag * m.delta[0] + real * m.delta[1]
    end
  end
end
```

Algorithm 5.3: Incremental CPU algorithm with truncation of modes (CPU-IT).

	#modes	256	1024	4096
CPU-E	time [μs]	4234	16231	64708
	mode limit	350	366	367
CPU-I	time [μs]	173	689	2676
	mode limit	8589	8625	8885
CPU-IT	time [μs]	141 (81%)	505 (73%)	1829 (68%)
	active modes	183 (71%)	665 (65%)	1757 (42%)
	mode limit	10514	11769	13001

Table 5.1: Performance comparison of different CPU-based modal synthesis algorithms. CPU-E uses an explicit algorithm, CPU-I an incremental one. CPU-IT is an algorithm using mode truncation to reduce the number of active modes. The measured time corresponds to the first audio block of 256 samples after an excitation, for an object with 256, 1024, or 4096 modes. For the truncating algorithm, the number and percentage of active modes are listed, as well as the relative run-time compared to the non-truncating variant. The mode limit scales the object's mode count to the maximum number that can be computed within the duration of one audio block (5805 μs).

be computed fast enough on a single CPU threads. Instead, it is easier to distribute multiple active objects onto different cores. Since a parallelization at object level is trivial, it is usually more appropriate than an in-object parallelization.

Truncation

The basic explicit and incremental algorithms can be augmented to allow for truncation of inaudible modes. For this, different approaches can be used. It is possible to store an activation state for each mode, and only evaluate active modes per sample. This, however, still requires checking each mode once per sample, which can slow down the computation especially when a large number of modes is inactive. A better approach is to maintain a list of active modes, which is then traversed during sample evaluation. For the benchmarked C++ implementation, an `std::vector` of pointers to the modes proved to be the most efficient way to store and update the active mode list. While adding and removing modes to the active list is comparatively costly when compared to a linked list, this operation is only performed when updating the truncation state. Since sample evaluation is performed much more often than the activation state update, the increased performance of iterating an `std::vector` outweighs the higher truncation check cost. A pseudo-code implementation using an active mode list for truncation is shown in Algorithm 5.3. While the example uses the incremental formulation to evaluate modes, the same approach is used for the explicit formulation.

Performance Discussion

The different algorithms were implemented using C++, and a benchmark was performed to evaluate their performance. They were evaluated on a PC with 24GB RAM and an Intel

Xeon X5650 (2.67GHz, 6 cores). The test program was compiled using gcc 4.8, and uses a single core for computation. All computations were performed using single precision floating point numbers. The modal data used for the benchmark is computed by a MA of a sphere of appropriate vertex counts to produce the desired number of modes.

The benchmark results of the different CPU implementations for objects with different numbers of modes are listed in Table 5.1. The results list the computation time of the first sample block after an excitation, where the block length is 256 samples, corresponding to 5.8 ms. If an algorithm uses mode truncation, the percentage of active modes is listed, and the computation time also includes the time required to update the truncation states. Additionally, a mode limit is listed, which provides an estimate of the maximum number of modes in active objects that can be handled by the algorithm. The mode limit provides an estimate for the maximum number of modes that can be handled for a real-time synthesis, and is computed by scaling the number of modes of the benchmarked object such that the computation time would equal the duration of the sound block, which is the maximum available time to still allow a real-time synthesis.

The results show that the incremental algorithm is more than one order of a magnitude faster than the explicit variant, and can handle objects with a total of more than 8000 modes. Assuming an average of around 1000 modes per object, this would allow for around 8 simultaneously active objects.

When using truncation, even more modes can be handled. However, the exact gain depends on the number of inaudible modes, which again depend on the objects geometry, resolution, and material as well as the excitation location and the deactivation thresholds. The provided examples show active modes from 42% to 71%, where objects with fewer total modes typically have fewer modes that can be omitted. However, due to the overhead of checking the modes' activation state, the performance does not increase to the same extent. Still, a notable performance gain can be observed even for the first block after an excitation, as it was measured here. Over time, the sound decays and the number of active modes decreases further and further, so that the effectiveness of mode truncation increases. Figure 5.1 compares the performance of the truncating algorithm CPU-IT against the non-truncating one(CPU-I), computing consecutive blocks while deactivating an increasing number of modes. Assuming that all objects have a similar level of inaudible modes, the algorithm CPU-IT using truncation can handle more than 10 objects with an average of 1000 modes each.

While the results are only listed for an audio block size of 256 samples, other block sizes were also tested. However, the block size has no notable influence on the performance – at least if the mode truncation check is performed equally often, i.e. after the same total sample count.

It can be concluded that the incremental variant of the run-time modal synthesis is far better suited for a CPU implementation than the explicit variant. Furthermore, disabling inaudible modes can further reduce computation times, so that the sound of at least 10 simultaneously sounding objects with 1000 modes each can be synthesized.

As discussed before, the algorithms utilize a single CPU thread. When multiple CPU cores are available for the sound synthesis, one can distribute the different objects onto multiple threads to compute them concurrently. In this case, the number of objects whose sound can be

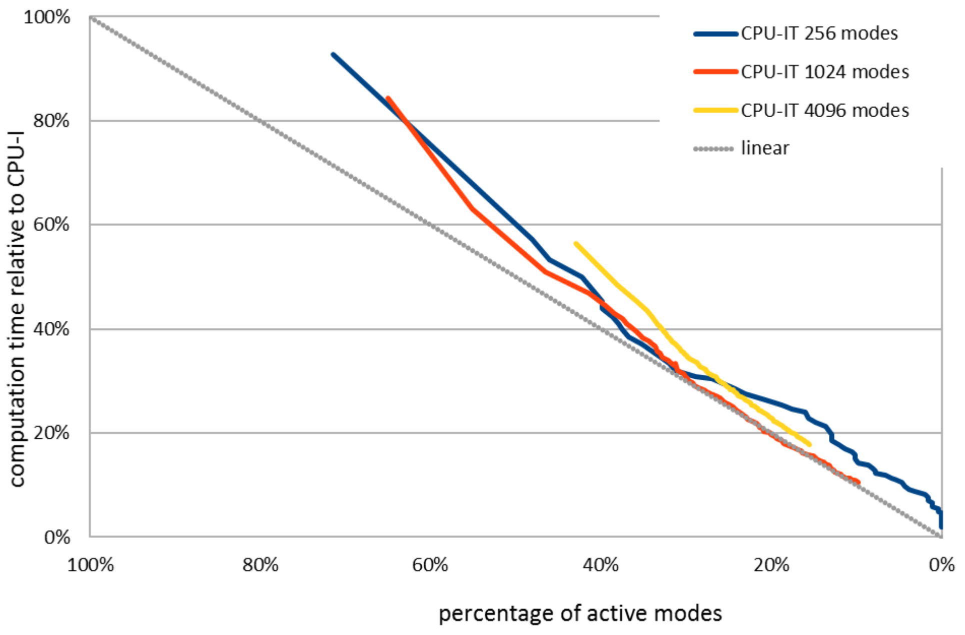


Figure 5.1: Efficiency of the truncating algorithm CPU-IT over time where decay leads to a continuously decreasing number of active modes. Performance is given relative to CPU-I, which considers all modes.

synthesized in real-time scales linearly with the number of utilized CPU cores. Therefore, for the case when no forces are acting on an object, CPU-based algorithms are sufficiently fast to synthesize modal sounds for several objects simultaneously.

5.2.4 GPU-based Sample Computation

Using the CPU synthesis already allows for a decent number of objects and modes to be handled by a run-time sound synthesis. However, utilizing the high parallel computing power of modern graphics cards can further improve the synthesis performance, allowing many more sounding objects.

GPUs provide a high number of processing units that provide a high potential for parallel processing. For the algorithms presented here, CUDA[1] was used, which allows programming general-purpose GPU programs. When designing algorithms for GPU computation, however, the special requirements of the graphics card have to be considered to utilize its potential:

- **Memory Transfer:** While the on-board memory of the GPU is fast, the data bandwidth for transferring data between the computer's main memory and the GPU memory is limited. Even small data transfers come with an overhead. Thus, frequent data transfer

[1] `https://developer.nvidia.com/cuda-zone`, version 5.5

between the CPU and GPU should be avoided, and data required by the GPU during computation should be uploaded and permanently stored on the GPU wherever possible.

- **Memory Size:** While the amount of memory available to the GPU has been increasing in recent years, it typically is still smaller than a PC's main memory. Thus, it should be taken care not to store too much data in the GPU memory. This requirement is conflicting with the memory transfer limitation, since both permanent storage and dynamic memory update rates are limited. Thus, it must be carefully weighed what data to store permanently, and what data must be frequently transferred between GPU and CPU memory.

- **Computation Groups:** On a GPU, the computations are performed using many small computation units, which are grouped on multiple levels. Using CUDA's terminology, a *kernel* defines the program, i.e. the code block to be executed. It is executed by different *threads* that run on different processing units (called CUDA cores). When launching a computation, the threads are grouped on two levels: the *grid* and the *blocks*. One block consists of several threads processed by a single so-called *streaming multiprocessor*. All threads in a block share the streaming multiprocessor's local memory. Since the block size is limited by the available local memory, multiple blocks are distributed across a grid. Additionally, the threads of a block are divided into *warps* of a fixed number of threads, which is 32 for all CUDA versions. The threads of one warp are executed simultaneously step-by-step. The overall number of threads to be computed by one launch is the *grid dimension* times the *block dimension*.

- **Synchronous Execution** As described, threads are grouped in warps, which are executed simultaneously. All instructions are performed synchronously step-by-step. This has to be considered when developing algorithms for the GPU, since branching does not always work as intended. If one branch is chosen by at least on thread of the warp, it delays the computation of all threads until the branches executions have been processed. If different threads of a warp enter different branches, the branches are processed one after the other, which can significantly impact the performance. Thus, the use of conditional branches has to be carefully considered for each instance to determine its applicability for GPU programming.

- **Memory Access:** While modern graphics cards have a high memory access speed, it is important to obey some rules when regarding memory accesses. To achieve a high memory bandwidth, access to the global memory should be *coalesced*. Memory coalescing describes a synchronous memory access of all threads in a block, so that it can be performed as a single operation. However, to allow coalescing, the memory address used by each thread must be aligned, where each thread in a block accesses a 4 bytes word in a block of continuous memory, and the word offset must equal the thread identifier. When non-coalesced access to the global memory occurs, each thread has to retrieve its data separately. This can significantly degrade the performance, so that coalesced memory access should be used wherever possible.

These special requirements of GPU programming should be considered when developing algorithms in order to optimize the performance.

In the following, different algorithms for the modal sound synthesis on GPUs will be presented. To gauge their benefit and problems, their performance was evaluated using the same benchmark as for the CPU-based synthesis (see Section 5.2.3). The benchmark ran on the same PC as for the CPU implementation, using an Nvidia Quadro 6000 and CUDA 5.5.

Explicit Algorithm

When using the GPU for modal sound synthesis, it is first necessary to transfer the mode parameters onto the GPU memory. For the explicit GPU algorithms, the four required values – angular frequency, decay, and the two amplitude values – are stored as two arrays of `float2`, each array containing the entries for all modes. The static angular frequency and decay can be uploaded once, while the complex amplitudes are updated after mode excitations have been applied.

To compute the resulting sound, one thread is launched for each mode-sample combination, which compute that mode's contribution at the sample time, and adds it to the corresponding location of the global output sample buffer. While there are other options to distribute the computation, this configuration proved to be the most efficient one. Te reasons for this are the high number of threads that allow a good distribution, and the full use of coalesced memory access.

The CUDA implementation of the explicit algorithm, GPU-E1, is shown in Algorithm 5.4. This implementation includes some optimizations over a straightforward implementation.

One optimization is the evaluation of the trigonometric functions. Instead of using two separate calls to compute the sine and cosine of Equation 5.5, CUDA provides a combined function `sincosf` that allows retrieving them simultaneously. Another possible way to retrieve the sine and cosine is to pre-compute the results for a range of values, and store them in a lookup table on the GPU. At run-time, the lookup table results are retrieved and interpolated. Using a lookup table with 512 entries only induced a very small error in the resulting signal (0.4%), but the resulting performance was equal to using the built-in functions.

Another optimization is the usage of shared memory for the mode parameters and excitation. In the presented implementation, all threads of a block compute different samples of the same mode. Therefore, the mode parameters and excitation are identical for all threads. Thus, it suffices for the first thread of the block to retrieve the data, and store them in shared memory that is then accessed by all threads of the block. While it is possible to retrieve the values from global memory in each thread, this significantly degrades the performance because this memory access would be non-coalesced.

When starting the computation on the GPU, the grid and block dimensions have to be specified. For the explicit implementation, the best results were achieved by using block dimensions `blockDim = min(sample_size, 512)` equal to the sample size, up to a maximum of 512. Note that the maximum of 512 produced best results on the tested hardware, but is hardware-dependent and may be different for other GPUs. A two-dimensional grid of blocks is used: `gridDim = (num_modes, ceil(block_size / 512))`. Its x dimension equals the number of modes of the object, and the y dimension is the

	block size	256 samples			512 samples			1024 samples		
	#modes	256	1024	4096	256	1024	4096	256	1024	4096
GPU-EX	time [μs]	29	53	148	33	73	208	45	110	374
	mode limit	50807	112489	160998	90203	163105	228841	132929	215211	254248
GPU-E1	time [μs]	37	78	243	54	134	460	67	189	676
	mode limit	39162	76695	98487	55553	89018	103369	89273	126041	140673
GPU-E2	time [μs]	47	101	339	50	115	372	66	182	637
	mode limit	31290	58769	70231	59657	103401	127867	89710	130466	149352
GPU-ET	time [μs]	41 (111%)	63 (81%)	117 (48%)	53 (98%)	93 (69%)	193 (42%)	63 (94%)	123 (65%)	268 (40%)
	mode limit	36062	93942	203744	56224	127439	246037	94741	192884	354768
	active modes	183 (71%)	665 (65%)	1757 (43%)	183 (71%)	665 (65%)	1757 (43%)	183 (71%)	665 (65%)	1757 (43%)
GPU-I1	time [μs]	599	1917	5380	1129	3416	8941	2216	5446	14194
	mode limit	2482	3101	4419	2631	3479	5318	2682	4366	6700
GPU-I2	time [μs]	415	692	1911	779	1198	3228	1509	2191	5624
	mode limit	3582	8592	12440	3817	9927	14731	3938	10853	16912
GPU-I3	time [μs]	40	83	251	55	137	467	69	193	684
	mode limit	37290	71788	94835	54389	86766	101844	86390	123467	138972
GPU-I4	time [μs]	53	98	240	76	140	438	98	234	887
	mode limit	28133	60478	99277	39152	84911	108495	60583	101481	107254
GPU-I5	time [μs]	176	181	336	312	316	605	589	593	1190
	mode limit	8439	32805	70855	9530	37667	78546	10095	40075	79922

Table 5.2: Performance comparison of different GPU synthesis algorithms, using varying block lengths and mode counts. GPU-EX, GPU-E1, GPU-E2, and GPU-ET use explicit algorithms, where GPU-ET also trunates inaudible modes. GPU-I1 to GPU-I5 use an incremental mode evaluation. For truncating algorithms, the number and percentage of active modes are listed, as well as the relative run-time compared to the non-truncating variant. The computation time is measured for the first block after an excitation. Percentage values for the truncation variants display the reduction over the variant without truncation. The mode limit scales the object's mode count to the maximum number that can be computed within the duration of one block.

```
__device__ float CalculateSample(const float2& params,
                                 const float2& excitation,
                                 const float time)
{
  float sin_t, cos_t;
  sincosf(params.y * time, &sin_t, &cos_t);
  return(2 * expf(params.x * time)
         * (excitation.x * cos_t - excitation.y * sin_t));
}

__global__ void CalculateBlock(const float2* mode_param_data,
                               const float2* mode_excitation_data,
                               const float time_stamp,
                               float* output_samples)
{
  const int mode_index = blockIdx.x;
  const int sample_index = blockIdx.y * blockDim.x + threadIdx.x;

  __shared__ TFloat2 mode_params;
  __shared__ TFloat2 mode_excitation;
  if(threadIdx.x == 0)
  {
    mode_params = mode_param_data[mode_index];
    mode_excitation = mode_excitation_data[mode_index];
  }
  __syncthreads();

  float sample_time = time_stamp + sample_index / 44100.0f;
  float sample = CalculateSample(mode_params,
                                 mode_excitation, sample_time);

  atomicAdd(&output_samples[sample_index], sample);
}
```

Algorithm 5.4: CUDA implementation of the explicit algorithm using `atomicAdd` (GPU-E1).

number of sample block subdivisions, in case it exceeds 512. This layout was chosen for performance reasons. Since a block computes samples of a single mode, it can use the described optimization of the mode parameter access. Also, using one block's threads for consecutive samples allows coalesced write access to the output sample buffer.

Once a thread has computed the contribution of its mode, the value has to be added to the result sample buffer. To avoid races due to different threads writing simultaneously, this addition must by synchronized. The implementation GPU-E1 uses an `atomicAdd` to perform the addition onto the result sample buffer entry using an atomic operation, preventing conflicts between threads. However, using `atomicAdd` reduces the performance, especially if multiple threads of a block write to the same entry, requiring a sequential execution. By assigning the different threads of a block to compute different samples of a single mode, no write conflicts arise within a block. Thus, `atomicAdd` conflicts can only arise between different blocks. Additionally, with this arrangement the memory access to the output sample buffer is coalesced, further improving performance.

Still, the usage `atomicAdd` incurs a slow-down. Table 5.2 compares the performance of the algorithm using `atomicAdd` (GPU-E1) with an algorithm using unsynchronized sample accumulation (GPU-EX). While GPU-EX does not compute correct results due to data races, it shows the performance loss caused by the use of `atomicAdd`, so that GPU-E1 can take up to twice as long as GPU-EX depending on the configuration size. Thus, a significant amount of performance could be gained by optimizing the accumulation of the final sample buffer.

An alternative approach to accumulating the mode contributions into the output buffer is to use an intermediate buffer. The $m \times s$ threads compute the sample contributions of m modes for s samples. Each sample's contribution can be stored in an intermediate buffer, i.e. a matrix of size $m \times s$, without the need for inter-thread synchronization. After all threads' contributions have been computed, a reduction step is performed to accumulate the columns of the intermediate matrix into a vector of size s, which produces the output sample buffer. To perform the reduction, different variants were tested: using a custom kernel launch to perform the reduction, integrating the reduction into the program that computes the mode contributions, and using the reduction provided by two external libraries, `thrust` and cuBLAS. All of these variants showed similar performance, but using cuBLAS was slightly faster than the other ones. The performance of an explicit algorithm with intermediate buffer and following reduction step (see GPU-E2 in Table 5.2) is only slightly better than GPU-E1 for higher number of modes and block sizes, and slightly slower for smaller problems. Because GPU-E2 also has higher memory requirements than GPU-E1 due to the need to store the $m \times s$ matrix containing the intermediate results, it is usually preferable to use the variant using `atomicAdd`.

Incremental Algorithm

With the presented optimizations, the explicit algorithms using CUDA provide a high performance. For the CPU sound synthesis, the incremental approach showed a much higher performance than the explicit algorithm, promising even higher possible performance for GPU-based incremental modal sound synthesis.

While the implementation of the explicit algorithm is well-suited for a GPU

```
__device__ void ExcitationUpdate(const float2& update,
                                 float2& excitation)
{
  float real = excitation.x;
  float imag = excitation.y;
  excitation.x = real * update.x - imag * update.y;
  excitation.y = real * update.x + imag * update.x;
}

__global__ void CalculateBlock(const float2* mode_update_data,
                               float2* mode_excitation_data,
                               const int num_samples,
                               float* output_samples)
{
  const int mode_index = blockIdx.x * blockDim.x + threadIdx.x;

  const float2& mode_update = mode_update_data[mode_index];
  float2& mode_excitation = mode_excitation_data[mode_index];

  for(int i = 0; i < num_samples; ++i)
  {
    atomicAdd(&output_samples[i], mode_excitation.x);
    ExcitationUpdate(mode_update, mode_excitation);
  }
}
```

Algorithm 5.5: CUDA implementation of the incremental algorithm, computing all samples of a mode per thread and accumulating results using `atomicAdd` (GPU-I1).

parallelization, the incremental variant poses several challenges that have to be overcome to achieve a good performance. A first incremental CUDA algorithm is shown in Algorithm 5.5, which corresponds the CPU algorithm. It first assembles the initial amplitude as well as the per-sample update, and uploads them to the graphics memory as two `float2`-arrays. Then, the computation threads are launched. Because the incremental algorithm updates the excitation at each sample based on the value of the previous sample, it is no longer possible to launch one thread for each mode-sample-combination. Instead, one thread is used for each mode, and computes all of its sample contributions.

However, this approach (see GPU-I1 in Table 5.2) is much slower than the explicit GPU algorithms, and is even worse than the CPU-based implementation. There are multiple reasons for the low performance. Since fewer threads are launched, the distribution of threads across the GPU's computation units is less efficient when few object with few modes are used. Another, more significant problem is the memory access and synchronization. Since each thread computes all threads of a single mode sequentially, it has to access all entries of the output sample buffer, leading to non-coalesced memory access. Additionally, GPU-I1 uses `atomicAdd` to synchronize the accumulation of the different mode's contributions into the sample buffer. Because all threads of a block perform the computations for the same sample simultaneously, they all try to write to the same sample buffer location at the same time, so that the `atomicAdd` cslld produce conflicts that lead to a strong loss of performance.

To achieve a good performance, it is necessary to improve the incremental GPU algorithm to overcome these problems. In order to address the conflicting `atomicAdd` operations, the same approach that was discussed for GPU-E2 can be used: Instead of directly accumulating the results into the output sample buffer, they are first stored in an intermediate buffer, which is reduced afterwards to produce the resulting sound signal. Using this approach (see GPU-I2 in Table 5.2) provides a performance improvement over GPU-I1. However, this implementation still suffers from the low thread count and non-coalesced access to the output sample buffer. Thus, it is still an order of magnitude slower than GPU-E1.

One of the benefits of the explicit GPU algorithm is that it can launch $m \times s$ threads for m modes and s samples, which allows a better use of the GPU's computational resources as well as coalesced access to the result buffer. Thus, another possible optimization is to launch one thread per mode-sample combination for the incremental approach, too. Equation 5.7 specifies the formula to update the excitation for consecutive samples, which requires the sample evaluation for one mode to be performed sequentially. However, instead of using the previous sample as reference, one can compute the amplitude $\underline{a}_k^{(n)}$ for sample n incrementally based on the amplitude $\underline{a}_k^{(0)}$ at the beginning of the computation block.

$$\underline{a}_k^{(n)} = \underline{a}_k^{(0)} \cdot e^{\underline{w}_k \cdot n \cdot \Delta t} \tag{5.9}$$

Thus, each mode-sample combination has its own excitation update factor $e^{\underline{w}_k \cdot n \cdot \Delta t}$, which needs to be pre-computed and uploaded to the GPU memory before the run-time synthesis. While this increases the memory consumption, the computation of a mode's sample values no longer depend on each other, allowing them to be computed in parallel by separate threads. Thus, a better distribution across the GPU cores is achieved. Also, by grouping the

```
 __global__ void CalculateBlock(const float2* mode_update_data,
                                const float2* mode_excitation_data,
                                const int num_samples,
                                float* output_samples)
{
  const int sample_index = blockIdx.y * blockDim.x + threadIdx.x;
  const int mode_index = blockIdx.x;
  const int update_index = mode_index * num_samples + sample_index;

  const float2& mode_update = mode_update_data[update_index];
  const float2& mode_excitation = mode_excitation_data[mode_index];

  float contrib = mode_excitation.x * mode_update.x
                - mode_excitation.y * mode_update.y;
  atomicAdd(&output_samples[sample_index], contrib);
}
```

Algorithm 5.6: CUDA implementation of the incremental algorithm using pre-calculated per-sample mode updates, and accumulating results using `atomicAdd` (GPU-I3).

kernel launch equally as the explicit GPU algorithms – i.e. letting one block compute different sample contributions of one mode – the access to the result buffer is coalesced.

Algorithm 5.6 shows the CUDA implementation of this approach, accumulating the mode components using `atomicAdd` (GPU-I3). This approach overcomes the problems of non-coalesced write access, and provides a higher performance than GPU-I1 and GPU-I2, and is only slightly slower than GPU-E1. While GPU-I3 provides a high performance, it requires more memory to store the $m \times s$ update factors for each mode-sample computation.

One advantage of basic incremental algorithm for modal synthesis is that a sequential computation of the samples is better suited to applying interactive forces (see Section 5.3.5). However, since the sequential computation was abandoned for GPU-I3, it is no longer suitable as a basis for force application.

Therefore, another algorithm GPU-I4 will be presented, which performs more optimizations to allow segments of samples to be computed sequentially, while still maintaining a high performance. It works on smaller groups of mode-sample combinations, and uses shared memory for accumulation of the output sample buffer. Instead of computing all samples in one thread per mode as for GPU-I1, the blocked algorithm separates the output samples into smaller partitions. In total, $m \times s$ sample evaluations are necessary for m modes and a sample block size of s. This grid of $m \times s$ is split into square groups of size $k \times k$. Each of these groups computes k consecutive samples of k different modes, leading to $\lceil \frac{m}{k} \rceil \times \lceil \frac{s}{k} \rceil$ partitions

Using these smaller groups allows to perform the accumulation of the output sample buffer more efficiently. For each of the $\lceil \frac{m}{k} \rceil \times \lceil \frac{s}{k} \rceil$ segments, a block of k threads is launched. For each block, an intermediate buffer of size $k \times k$ is allocated in shared memory, so that it can be accessed by all threads of a block. Each thread works in two steps. First, it computes the mode contributions of its corresponding mode for k consecutive samples, and stores them in the corresponding row of the intermediate shared buffer. After all threads of a block finish this step, each thread then reduces the corresponding column of the intermediate buffer, resulting in the accumulated sample contributions of the k modes handled by this block. These are then added to the global output sample buffer using `atomicAdd`. The CUDA implementation of this two-step algorithm is shown in Algorithm 5.7.

To allow each block to compute only k consecutive samples, it is necessary to pre-compute multiple sets of mode excitation updates. In addition to the update factor $e^{\underline{w}_k \cdot \Delta t}$ that is used to compute sequential samples, a sequence of update factors $e^{\underline{w}_k \cdot n \cdot k \cdot \Delta t}$ are prepared that allow computing each mode's amplitude at the beginning of the nth sample group.

Partitioning the algorithm and working on square blocks of $k \times k$ mode-sample pairs prevents the problems with writing to the output buffer that occurred with CPU-I1. Within each block of threads, each entry of the shared buffer is only accessed twice – once when writing the contribution, and once when reducing into the final result. Thus, the only inter-thread synchronization for access to the shared buffer is the condition that all threads of a block finish the computation of the sample contribution before the reduction step is started. After reduction, the accumulation into the global result sample buffer is again utilizing coalesced memory access, and using `atomicAdd` for the accumulation provides few conflicts because all threads of a block access different entries of the buffer.

One restriction for this synthesis variant (GPU-I4) is that the size of the shared memory is limited. The shared buffer contains $k \times k$ floats, which on the utilized GPU limits k to being at most 64. Thus, the segment size k can be either 32 or 64, of which the latter results in slightly better performance on the examined hardware.

Using the segmented algorithm with a shared buffer provides much better performance than the fully sequential variants GPU-I1 and GPU-I2 (see Table 5.2). In most cases, GPU-I3 still provides a slightly better performance due to the higher number of threads, especially for objects with few modes. GPU-I3, however, requires more GPU memory to store the $m \times s$ excitation updates, while GPU-I4 only requires around $\frac{1}{k}$th of the memory, since only the excitation update for the first sample of a group of k threads is required.

Despite the presented optimizations, the incremental GPU-based synthesis algorithms are still slower than the explicit ones, and also require more GPU memory. Therefore, the explicit approach is favorable for synthesizing modal sounds on the GPU when no forces are ative on the sounding object. However, the incremental algorithms provide an important basis for the computation of dynamic force excitations at sample rate. This requires a fully sequential computation of samples, for which a fifth incremental algorithm, GPU-I5, will be presented in Section 5.3.5.

Truncation

On the CPU, disabling inaudible modes can be achieved efficiently, e.g. by using active flags or a list of active modes, and improve the performance significantly when a high number of modes can be omitted. On the GPU, truncation requires a more specialized approach.

While it would be possible to store an activation flag and skip computation of disabled modes, this would not be beneficial for the performance. Multiple threads are executed together in a warp, performing each computation step synchronously. Thus, if just one of

```
__device__ void ExcitationUpdate(const float2& update, float2& excitation)
{
  float real = excitation.x;
  float imag = excitation.y;
  excitation.x = real * update.x + imag * update.y;
  excitation.y = real * update.x + imag * update.x;
}
__device__ float ReduceBuffer(float sample_buffer[], int num_modes)
  float* local_buffer = &sample_buffer[threadIdx.x * blockDim.x];
  float sample_contrib = 0;
  int mode_count = min(blockDim.x, num_modes - blockDim.x * blockIdx.x);
  for(int mode = 0; mode < mode_count; ++mode)
    sample_contrib += local_buffer[mode];
  return sample_contrib;
}

__global__ void CalculateBlock(const float2* mode_updata_data,
                              float2* mode_excitation_data,
                              const float2* mode_update_offset_data,
                              int samples_per_block,
                              int num_samples, int num_modes,
                              float* output_samples)
{
  extern __shared__ float sample_buffer[];
  int mode_index = blockIdx.x * blockDim.x + threadIdx.x;
  int sample_offset = blockIdx.y * samples_per_block;
  int sample_count = min(samples_per_block, num_samples - sample_offset);

  if(mode_index < num_modes)
  {
    const float2& mode_update = mode_update_data[mode_index];
    const float2 mode_update_offset = mode_update_offset_data[blockIdx.y
                                          * num_modes + mode_index];
    float2 mode_excitation = mode_excitation_data[mode_index];
    ExcitationUpdate(mode_update_offset, mode_excitation);

    for(int sample = 0; sample < sample_count; ++sample)
    {
      sample_buffer[sample * blockDim.x + threadIdx.x] = mode_excitation.x;
      ExcitationUpdate(mode_update, mode_excitation);
    }
  }
  __syncthreads();

  if(threadIdx.x < sample_count)
  {
    float contrib = ReduceBuffer(sample_buffer, num_modes);
    atomicAdd(&output_samples[sample_offset + threadIdx.x], contrib);
  }
}
```

Algorithm 5.7: CUDA implementation of the incremental algorithm using blocked partitions and accumulation in shared buffers (GPU-I4).

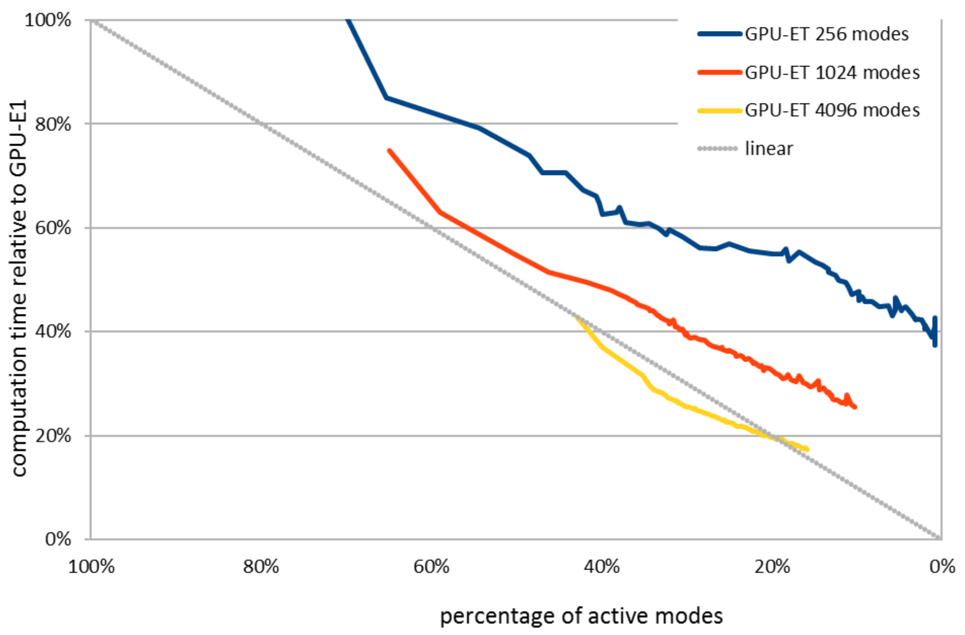


Figure 5.2: Efficiency of the truncating algorithm GPU-ET over the run-time of a synthesis, where decay leads to a continuously decreasing number of active modes, using a block size of 256. Performance is given relative to GPU-E1, which considers all modes.

these threads is not disabled and has to perform the computation, the other threads still have to wait for it to finish before their computational resources can be used again.

Using a list of active modes would not provide a performance boost, either. When the index of a thread's mode would be retrieved from an active list, the additional indirection would result in non-coalesced memory access to the mode data, reducing the performance.

To achieve a good speed-up, the truncation state of each mode is computed on the CPU. Afterwards, the required data – i.e. the mode's angular frequency, decay, and excitation in case of the explicit algorithm – of only the active modes are assembled, and re-uploaded to GPU memory. This way, the computation of the sound on the GPU works exactly as for the non-truncating version, just using fewer modes.

While this speeds up the computation of the actual sound samples, the truncation introduces an additional overhead. The CPU-based activity checks only require little additional computation time. However, for GPU-based synthesis algorithms, it is necessary to transfer the newly assembled modal data onto the graphics memory. Especially for objects with few modes, the memory transfer can be significant in relation to the synthesis time. The performance measurements of the truncating algorithm (GPU-ET) are listed in Table 5.2, and include the time required for activity state computation and the upload of the modal data. The truncation efficiency over the course of a sound with continuously increasing number of inaudible modes is plotted in Figure 5.2. While the overhead is acceptable for objects with many modes, it is more significant for smaller objects and can even lead to a worse performance than without truncation.

However, the truncation overhead only occurs when the activation state is updated and new data is uploaded to the GPU. In the performance measurements, the truncation update was performed before every sound block computation, leading to a higher overhead for smaller block sizes. To reduce the impact of the overhead, one can perform the truncation check after a larger number of samples, e.g. every 1024 samples. Additionally, one can perform the mode activation check regularly, but only upload the reduced mode data to the GPU if a sufficient number of modes was disabled. This way, truncation can provide a good speed-up for GPU-based modal synthesis.

Performance Discussion

The relative performance of the different GPU-based algorithms was already discussed to determine necessary optimization. Relating the performance of the GPU-based algorithms (see Table 5.2) to the CPU-based variants (see Table 5.1) shows the high computation power of the graphics card and its benefit for the modal sound synthesis. However, the special requirements also pose some challenges visible in the performance.

Unlike for the CPU algorithms, the size of the sound sample blocks has a noticeable influence on the result. Thus, benchmark results for three different block sizes were provided: 256 samples (5.8 ms), 512 samples (11.6 ms), and 1024 samples (23.2 ms). These show that the relative performance is worse for smaller block sizes, especially when combined with object with fewer modes. This is mostly due to a constant overhead that comes with every kernel launch as well as the read-back of the result sample buffer, which is less significant when a computation takes longer. Additionally, fewer samples lead to fewer concurrent threads, which can lead to a lower utilization of the GPU's streaming processors. The same is true for the number of modes per object, so that multiple objects with few modes are slower to synthesize than a single object with equal mode count.

Several different algorithms for the GPU-based modal synthesis were presented using both the explicit (GPU-E1 and -E2) and implicit (GPU-I1 to -I4) formulation. The explicit variants provide very similar performance, and can handle much more active modes than any CPU-based algorithm. The incremental algorithm, which performed best on the CPU, required several improvements to achieve a good performance on the GPU. Still, the best-performing implementation GPU-I3 is still slightly slower than the explicit variants. All in all, GPU-E1 proved to be the best choice for modal sound synthesis on the GPU without active forces. While GPU-E2 can be slightly faster than GPU-E1 in some setups, it requires additional memory on the GPU for the intermediate buffers.

For the best-performing variant GPU-E1, a very high number of active objects is possible. Even when using a small sound block size of 256 samples, it is possible to handle more than 75 of objects with 1000 modes each, as opposed of the 8 that can be handled per CPU core by the CPU synthesis algorithms. Using truncation, even more objects can be handled. This is more than sufficient even for highly interactive scenes with lots of simultaneously active sound sources.

5.3 Synthesis with Active Forces

The previous section presented algorithms to compute the modal sound of an object after it has been excited. This excitation of modes occurs when external forces act on the object, thereby inducing a local deformation that causes the object's vibrations. As will be discussed in Section 5.3.2, force-based excitation should be applied at a high sampling rate. Ideally, it is performed at the same rate as the sound synthesis, and is thus performed together with the actual sample evaluation.

5.3.1 Mode Excitation

When an external force acts on an object, the resulting mode excitation can be computed from the force vector, its point of attack, and the modal gains. The external force vector $\mathbf{f}_{ext} \in \mathbb{R}^{3n}$ contains the three force components acting on each of the n vertices of the sound geometry. This vector is typically sparse, since most forces only act on a small area. For the common case of a force acting on a single point on the object, one can compute the corresponding triangle position on the sound mesh, and distribute the force to the vertices of this triangle according to the barycentric coordinates of the attack point.

When the external force vector is computed, it is then transformed into modal space (see Section 2.2.2) using the systems eigenvector matrix $\mathbf{V}$:

$$\mathbf{g} = \mathbf{V}^T \cdot \mathbf{L}^{-1} \cdot \mathbf{f}_{ext} \tag{5.10}$$

The transformed force vector $\mathbf{g} \in \mathbb{R}^m$ then holds the relative force component for each of the m modes.

The forces are then used to update the amplitude of the modes, following the approach described by Raghuvanshi et. al. [Raghuvanshi et al. 2006]. If mode k has angular frequency ω_k, decay γ_k,modal mass m_k and current mode amplitude $\underline{a}_k = a_k + i \cdot b_k$, its amplitude is updated at time t using the corresponding force vector entry g_k as follows:

$$\underline{a}_k \leftarrow \underline{a}_k - i \cdot \frac{g_k \cdot q_k}{m_k \cdot \omega_k \cdot e^{(\gamma_k + i \cdot \omega_k) \cdot t}} \tag{5.11}$$

As discussed before, the acoustic transfer factor q_k is directly included into the amplitude to allow a direct estimate of the resulting sound contribution of the mode.

5.3.2 Force Application Rate

Using Equation 5.11, forces can be applied at any given time t. Forces act over a certain time span – which can be less than a milliseconds for very short impacts, or a long duration for lasting contacts like rolling or sliding. For example, when a collision between two objects occurs, kinetic energy is transferred, and they separate again afterwards. The impulse is not transferred immediately, but over a short time frame due to local deformations of the objects. This contact duration depends on different factors like the geometry of the contact area and the objects' material parameters [Johnson et al. 1987]. It can be estimated for run-time modal synthesis using sphere approximations of the contact geometry. Here, larger or softer objects produce longer contact durations.

The temporally distributed force must be applied at discrete time stamps using Equation 5.11. Thereby, the force is *sampled*. The common approach is to perform the force excitation infrequently, e.g. at the beginning of a block of 256 or 512 samples, which are then synthesized without further regard for active forces, as was done in Section 5.2. However, such a coarse sampling of excitation forces leads to changes in the resulting sound.

The temporal distribution of a contact force has an audible impact on the resulting sounds. When a force acts on a surface that is already vibrating, the resulting excitation depends on the phase between force and vibration. This is modeled in Equation 5.11 by the factor $e^{-(\gamma_k+i\cdot\omega_k)\cdot t}$ that encodes the phase between the current amplitude and the excitation. Thus, if a force acts over a longer duration, it may initially excite a certain mode. However, once the phase of the vibration shifts, the force starts opposing the already present motion of the surface, thereby dampening the vibration mode instead of further exciting it.

For typical collisions, where the contact duration is short, this mostly effects high-frequency modes whose phase switches fast enough for this effect to occur. Since the contact duration is higher for larger, softer objects, these contacts produce less amplitude in higher-frequency modes, and thus pronounces the lower-frequency modes. For this reason, hitting an object like a glass with a soft mallet produces a lower timbre than when using a hard mallet. When objects come into lasting contact, this effect becomes even more important, such as for rolling or sliding.

For modal sound synthesis, the rate at which the mode excitation from forces are applied can have a significant impact. If there is a larger time difference between force application steps, the above-mentioned effects cannot be accurately reproduced, and aliasing artifacts occur. In such cases, transient collisions that last shorter than the interval between force applications all produce a single, sharp force spike and thus all sound alike. For lasting contacts, high-frequency changes in the force – as may originate from surface roughness when an object slides over another – are grouped, and thus their specific sound character is lost.

One instance where such an aliasing effect is very noticeable is for an object resting on another surface. In reality, such an object does not produce any sound because no energy is transferred, Still, a constant force is acting continuously in such a case. If the force is applied at larger time intervals, it produces a regular periodic excitation of the object, so that the object still produces sounds. When applying the excitation at a high enough sample rate, however, all modes may be initially excited, but are dampened again once their phase shifts, resulting in no or extremely low sounds to be produced for resting objects.

For these reasons, it is advisable to evaluate the force excitation at a high frequency. To properly model the phase between the force and current vibrations, the relevant frequency range of 20 Hz to 22 kHz should be covered. Ideally, the force excitation should be computed at the same rate as the synthesized sound samples, which is typically 44.1 kHz.

5.3.3 Dynamic Forces

Interactive applications typically contain a variety of objects that can move according to different rules. For example, their movement can be controlled by a physics engine, or directly by user interaction. Objects often come in contact with one-another, which should result in a corresponding sound to be generated by the modal sound synthesis. The resulting sound should reflect the type of contact, which may differ significantly depending on the contact interaction. Where transient collisions produce localized forces over a short time frame, gliding or rolling contacts produce longer-lasting forces where the force magnitude and application point change over time.

Thus, dynamic contact forces can be described by the forces' magnitude, direction, or point of attack over time. The dynamic forces can be used to model the contact duration and softness of transient contacts, as well as the longer-lasting, more complex forces produced by lasting contacts like scraping, rolling or sliding. The modal synthesis algorithms presented here are designed to work with arbitrary dynamic input forces, so that different approaches can be used to simulate the contact force (see Section 2.2.4) without need for a specialized synthesis algorithm.

In general, forces can act on one or mutliple points of an object. To simplify the force application, the input forces are split to forces acting on individual vertices. If a contact acts on an point inside a triangle, or if it is distributed over multiple vertices in an area, it is modeled using multiple forces acting onto the corresponding vertices. This way, it is possible to model contacts with changing point of attack, by changing the distribution of the total force onto the per-vertex forces over time, e.g. according to changing barycentric coordinates.

While this increases the overall number of forces, it simplifies the evaluation of each indiviudal force excitation sample. By removing the position-dependency of forces, only the direction and magnitude have to be examined for each sample. Additionally, each force always works on its own three rows of the gain matrix.

5.3.4 CPU-based Excitation

When computing the sound from already excited modes on the CPU, the incremental approach proved to be faster than the explicit one (see Section 5.2.3). Additionally, it is more suited as a basis for applying per-sample excitation from forces. When evaluating a force excitation according to Equation 5.11, the current time is used to compute the phase. For the interactive approach, however, this factor can be omitted. At each sample, each mode's amplitude is updated by the incremental approach such that it corresponds to an excitation at $t = 0$, thereby resetting the time offset to zero and encoding the phase in the amplitude. Thus, the corresponding factor in Equation 5.11 reduces to $e^0 = 1$, simplifying the computation. Furthermore, the force acting at a sample must be applied after all previous samples' forces have already been applied, in order to gain the correct amplitude with matching phase. Since the incremental algorithm uses this condition to speed up the actual synthesis, it is well suited for force excitation at sample rate.

To compute the force excitation at sound sample rate, the synthesis algorithm CPU-I (see Section 5.2.3) is extended to also evaluate forces. The excitation is performed before the incremental excitation update, and can be applied in different ways.

Two different approaches for force application at sample rate will be presented. Their performance will be measured using the same benchmark setup as used in Section 5.2, except that now forces are acting on the object, which consists of 1024 modes. A different number

	forces	1×short	10×short	1×long	10×long
CPU-ID1	time [μs]	826	2039	2741	21329
	rel. time	119%	296%	397%	3095%
	mode limit	7196	2915	2168	278
CPU-ID2	time [μs]	781	1165	2617	8267
	rel. time	113%	169%	379%	1200%
	mode limit	7613	5101	2271	719
CPU-ID2/4	time [μs]	690	794	1118	2551
	rel. time	100%	115%	162%	370
	mode limit	8567	7490	5317	2330

Table 5.3: Performance comparison of different CPU synthesis algorithms using interactive force excitation and incremental computation. The computation time is measured for a single audio block of 256 samples (5805 μs), and objects with 1024 modes. Forces act on one or ten vertices, either over a short (16 samples) or long (full block) duration. For CPIU-ID2/4, mode excitation is updated every fourth sample (/4), other variants update every sample. Relative time lists the computation time as percentage of the time of the method without force distribution (CPU-I). The mode limit is estimated by scaling the configuration's mode count and computation time to the block duration.

of forces per object was tested, using either a single or 10 active forces. Additionally, the duration for which forces are active will be varied. They are either lasting over the full duration of the computation (long), or only for the first 16 samples of the block (short).

To compute the excitation of the different modes, the first implementation (CPU-ID1) maintains a list of active forces (see Algorithm 5.8). Each force acts on a single vertex and provides the index of this vertex, as well as the time-dependent force vector that specifies its magnitude and direction. For each force evaluation, the list of active forces is traversed. Then, the gain vectors for the corresponding force is iterated. Each entry of the gain list specifies the corresponding vertex index, as well as the three-dimensional vector containing the excitation scale for force components acting along the respective Cartesian axes. To compute the excitation, the three force vector entries are multiplied by the corresponding gain, and the resulting force is applied to the mode according to Equation 5.11.

The performance of this implementation is very low (see Table 5.3), and can only handle around 2000 modes with a single continuously acting vertex force each, while a single short force can be used for objects with around 7000 modes. However, most contacts produce forces acting on multiple vertices. In these cases, the performance drops further. For ten long-lasting forces, CPU-I1 is not able to handle the synthesis of even a single object with 1000 modes. Thus, a higher performance has to be achieved.

CPU-ID1 iterates over all forces and excites the corresponding modes directly. Therefore, the excitation computation has to be performed multiple times if the same mode is affected by multiple forces or the different cardinal directions of a single force. These repeated excitations can degrade the performance especially when multiple forces are active. Instead of this direct excitation for each force component, it is possible to first process all forces, and accumulate

```
function apply_forces(time):
  for force in active_force_list do
    if force still active do
      force_vector = force.get_force_at_time(time)
      for i from 0 to 2 do
        vertex_gain_list = vertex_gains[force.vertex_index][i]
        for gain in vertex_gain_list do
          excitation = force_vector[i] * gain.gain_factor)
          mode_states[gain.mode_idx].excite(excitation)
        end
      end
    else
      active_force_list.remove(force)
    end
  end
end

function compute_buffer(buffer, buffer_size):
  for i from 0 to buffer_size - 1 do
    buffer[i] = 0
    apply_forces()
    foreach mode m do
      real = m.amplitude[0]
      imag = m.amplitude[1]
      buffer[i] = buffer[i] + real
      m.amplitude[0] = real * m.delta[0] - imag * m.delta[1]
      m.amplitude[1] = imag * m.delta[0] - real * m.delta[1]
    end
    time = time + 1 / 44100
  end
end
```

Algorithm 5.8: Pseudo code of a CPU algorithm for force excitation using an active mode list (CPU-ID1).

```
function apply_forces(time):
  for force in active_force_list do
    if force still active do
      force_vector = force.get_force_at_time(time)
      for i from 0 to 2 do
        vertex_gain_list = vertex_gains[force.vertex_index][i]
        for gain in vertex_gain_list do
          excitation = force_vector[i] * gain.gain_factor)
          mode_states[gain.mode_idx].excite_accum += excitation
        end
      end
    else
      active_force_list.remove(force)
    end
  end
  for mode m do
    m.excite(m.excite_accum)
    m.excite_accum = 0
  end
end
```

Algorithm 5.9: Pseudo code of a CPU algorithm for force excitation using an active mode list and mode accumulation (CPU-ID2). This algorithm uses the same `compute_buffer` routine as Algorithm 5.8.

their corresponding contributions for each mode. The actual excitation is then only performed once after all forces have been evaluated. This implementation of this algorithm, CPU-ID2, is shown in Algorithm 5.9.

When only a single force is working, this leads to a small performance improvement because the force's three directions are grouped (see Table 5.3). However, the real benefit of this approach is that it can better handle multiple simultaneous forces. Still, the performance is too low to handle complex objects with several active forces.

A significant amount of performance can be gained by reducing the sample rate of the force application. Due to the dependency of the excitation on the phase of the current excitation, it should be applied at a high enough sample rate. A periodic function – such as the mode's surface vibration – should be sampled with at least twice the vibration frequency. Therefore, 44.1 kHz is typically chosen for reproduction of audio for humans, which consists of frequencies up to 22 kHz. Since the relevant modal surface vibrations have the same frequency range, using the sound sampling rate is a reasonable choice. However, to improve the performance, a lower sampling rate can be chosen by only evaluating the excitation every *n*th frame. While reducing the force application rate helps to improve the performance, it may produce inaccuracies and aliasing effects, as discussed in Section 5.3.2. Thus, one can trade a decrease in accuracy against a better performance. However, one can also examine the modes exited by a force, and determine the maximum frequency of these modes. Then, the force sample rate can be chosen as twice this frequency, which may be lower than the sound sample rate. However, when an object is analyzed with a high-resolution geometry, there are typically modes with high frequencies, so reducing the force application sampling rate may not be possible without sacrificing accuracy.

When exciting only every 4th sample (see CPU-ID2/4), the performance is improved over CPU-ID2, which excites every sound sample. With the reduced sampling rate, it is not possible to handle 10 continuous forces acting on two objects with 1000 modes each.

All in all, the CPU-based algorithms for sound synthesis with active forces show a low performance, and allow only few objects with a very limited number of active vertex forces. Even when using multiple CPU cores for the synthesis, complex scenes with many interacting objects may easily exceed these computational capabilities.

5.3.5 GPU-based Excitation

Since the force application at full sampling rate proved to be challenging to compute on a CPU, a different approach is required to allow synthesizing more complex scenes. The GPU-based modal synthesis without force excitation (see Section 5.2.4) already proved to be much faster than the CPU-based variants. Thus, using the GPU to also apply interactive forces is promising a higher performance.

However, for modal synthesis on the GPU, the best results are achieved when computing different samples in different threads, either using the explicit algorithm, or with the incremental variant GPU-I3 that used multiple update factors to allow a parallel sample evaluation. While these approaches provide a high performance, they are unsuited as a basis for synthesis algorithms that also apply forces at sample rate. The force at a specific sample time can only be correctly applied if all previous samples' force contributions have already been evaluated.

Incremental Synthesis with Sequential Samples

Section 5.2.4 already presented the incremental approach GPU-I4, which utilizes segments of size $k \times k$ in order to compute k consecutive sample contributions of k modes. It utilizes an intermediate shared buffer to resolve performance issues when writing to the result sample buffer.

GPU-I4 only computes groups of k samples sequentially, but not all of them. Thus, the algorithm GPU-I4 is extended to ensure fully sequential sample evaluation, as is required for interactive force updates. For GPU-I4, the s samples are separated into $\lceil \frac{s}{k} \rceil$ segments of k samples that are computed consecutively in separate threads for each mode. The limitation to k samples is necessary due to the limited amount of the shared memory. Instead of using multiple threads for the different partitions in parallel, the sequential incremental algorithm GPU-I5 sequentially computes multiple segments of k samples one after the other.

The easiest way to achieve this is to perform different kernel launches for each partition, effectively reducing the sample block size to k. However, this multiplies the number of kernel launches, which produce some overhead each and thus lead to a decrease in performance. Instead, it is faster to compute all sample segments in one kernel launch. For this, the `compute_buffer` routine of GPU-I4 (see Algorithm 5.7) is executed multiple times in a loop, thereby iterating over the $\lceil \frac{s}{k} \rceil$ sample segments.

The performance of this approach (GPU-I5 in Table 5.2) is lower than for GPU-I4. Especially for objects with few modes, only a low number of threads is created, leading to a lower utilization of the available resources. Still, it is much faster than the other variants with sequential sample evaluation and provides a good basis for a GPU-based synthesis with active forces.

Incremental Synthesis with Active Forces

Since GPU-I5 evaluates all samples sequentially, it can be extended to also apply excitation from forces at sample rate. For this, it is necessary to transfer the necessary excitation data to GPU memory.

When external forces act on an object, the mode excitations are computed by using the gain matrix to transform the force into modal space. It is possible to perform this step on the GPU by uploading the full gain matrix. However, since this matrix is rather large – $3v \times m$ for an object with m modes and v surface vertices – this would require a significant amount of memory on the GPU. Uploading the gain matrices of all objects once up-front could quickly exceed the available GPU memory. On the other hand, transferring the whole matrix when the first force starts acting on an object would lead to a delay due to the CPU-to-GPU memory transfer. Additionally, accessing the gain matrix using arbitrary vertex indices of a force application point would lead to non-coalesced memory access.

Instead, it proved to be better to assemble only the required rows of the gain matrix into a single `float3` array of size $f \times m$ when f forces are active. This way, only the columns of the gain matrix that correspond to active forces have to be transferred. While this requires uploading a new force gain vector whenever a force is added or removed, the lower memory size and coalesced memory access to the vertex gain list make up for the higher memory transfer cost. In addition to the force gain data vector, a second `float3` array of size $f \times s$, which contains the force vector of all f forces for each of the s samples.

After uploading the force data onto the GPU, the excitation and the resulting sound can be computed. The GPU-based algorithm with active forces (GPU-ID) works similar to GPU-I4, but at each sample it loops all forces, computes the excitation for the thread's associated mode, and adjusts the mode amplitude accordingly. The CUDA implementation of this algorithm is shown in Algorithm 5.10.

On the CPU, some optimizations could be applied for the force excitation. Since a force does not affect all modes, its gain list for CPU synthesis only contains entries for affected modes. On the GPU, however, listing only a limited set of affected modes would require an indirection when accessing the mode data, which would lead to non-coalesced memory access. Thus, it is faster to provide gain entries for all modes, even if they are zero.

Additionally, the CPU could deactivate forces as soon as they ended. On the GPU, this can only be done at the end of a finished sample block, and thus the force vector data is padded using zero forces. While this allows all threads on the GPU to execute the same code, there is no longer an advantage of shorter forces over forces that last over the duration of the whole sample block. While it would be possible to compute a shorter block of forces, the additional kernel launch and the smaller number of samples per block lead to a lower performance, too.

The performance of GPU-ID was measured analogous to the CPU-based variant. Table 5.4 lists the computation time with different numbers of forces acting on an object with 1024 modes, using a block length of 512 samples. The computation time includes the assembly of the force data arrays and their upload to GPU memory.

Since forces are always considered active over the whole block, there is no performance difference between short and long-lasting forces. Another interesting result is that no performance is gained when evaluating forces only every nth sample, and may even be slower than evaluating them every sample. This is likely due to the fact that the force excitation on the GPU utilizes vector operations, for which the GPU is highly optimized. When skipping modes, the contributions of multiple force samples have to be accumulated, leading to a delay due to the additional memory access.

For GPU-ID, the increase in computation time per active force is moderate. Each acting vertex force increases the run-time by approximately 10% over the run-time of the base algorithm GPU-I5 without active forces. Thus, a large number of forces can act simultaneously and affect a large number of objects. For example, around 20 objects with 1000 modes and 10 continuously acting forces can be handled, which is sufficient for most VR applications.

5.3.6 Truncation

For the modal synthesis without active forces, it proved to be helpful to disable inactive modes to reduce the computational load, both for the CPU- and GPU-based implementations. The same can be done when forces are acting on the object. However, in this case, modes can only

```
__global__ void CalculateBlock(const float2* mode_updata_data,
                               float2* mode_excitation_data,
                               int samples_per_block, int num_samples,
                               int num_modes, int num_forces,
                               const float3* force_gain_data,
                               const float3* force_vector_data,
                               float* output_samples)
{
  extern __shared__ float sample_buffer[];
  int mode_index = blockIdx.x * blockDim.x + threadIdx.x;
  int sample_blocks = ceil(num_samples / samples_per_block);
  float2 mode_update = mode_update_data[mode_index];
  float2 mode_excitation = mode_excitation_data[mode_index];

  int sample_offset = 0;
  for(int sample_block = 0; sample_block < sample_blocks; ++sample_block)
  {
    int sample_count = min(samples_per_block, num_samples - sample_offset);
    if(mode_index < num_modes)
    {
      for(int sample = 0; sample < samples_count; ++sample)
      {
        for(int force = 0; force < num_forces; ++force)
        {
          const float3* force_gain = force_gain_data[force * num_modes
                                                     + mode_index];
          float3 force_vector = force_vector_data[force * num_samples
                                                  + sample_offset + sample];
          mode_excitation.y += dot(force_vector, force_gains);
        }
        int buffer_index = sample * blockDim.x + threadIdx.x;
        sample_buffer[buffer_index] = mode_excitation.x;
        ExcitationUpdate(mode_update, mode_excitation);
      }
    }
    __syncthreads();

    if(threadIdx.x < samples_count)
    {
      float contrib = ReduceBuffer(sample_buffer, num_modes);
      atomicAdd(&output_samples[sample_offset + threadIdx.x], contrib);
    }
    sample_offset += samples_per_block;
    __syncthreads();
  }

  if(mode_index < num_modes)
    mode_excitation_data[mode_index] = mode_excitation;
}
```

Algorithm 5.10: CUDA implementation of the incremental algorithm with force excitations (GPU-ID). It utilizes the same `ExciteUpdate` and `ReduceBuffer` routines as GPU-I4 (see Algorithm 5.7).

	forces	1×short	1×long	10×long	20×long
GPU-ID	time [μs]	346	348	606	906
	rel. time	110%	110%	192%	286%
	mode limit	34371	34137	19628	13128
GPU-ID/4	time [μs]	538	538	600	893
	rel. time	170%	170%	190%	282%
	mode limit	22094	22103	19815	13307

Table 5.4: Performance comparison of the GPU synthesis algorithm using interactive force excitation and incremental computation. The computation time is measured for a single block of 512 samples (11610 μs), and an object with 1024 modes. Forces act on one or ten vertices, either over a short (16 samples) or long (full block) duration. Mode excitation is updated every sample for GPU-ID, and every fourth sample for GPU-ID/4. Relative time is in relation to the method without force distribution (GPU-I5). The mode limit is estimated by scaling the configuration's mode count and computation time to the block duration.

be disabled if none of the active forces can still excite them. Otherwise, a mode with very low excitation could be truncated even though a force might later-on excite it to an audible level.

The gain matrix computed by the MA is dense and typically produces very few zero entries. However, one can directly combine the gain values with their modes' acoustic transfer to achieve a relevance factor, and for each vertex, one can only ignore modes with relevant gains that are small compared to the highest gain (see Section 4.3.5). This way, the number of modes affected by a force can be reduced. However, when using conservative thresholds, a single force typically still affects around 50 to 80% of modes. Thus, when forces are active, a high number of modes cannot be considered for deactivation.

Because of the high baseline of modes that are always active, truncation is much less effective for synthesis with active forces. Therefore, best results are achieved by only computing the activation state when a new force is added, but not afterwards. Thereby, modes that are not excited by any existing forces and do not already have an audible excitation are truncated, but the activation check overhead only occurs when adding a new force. After all forces have stopped acting on the object, one can then again switch to the truncating algorithm without active forces for improved performance.

5.4 Discussion

In this chapter, different approaches for computing the sounds of objects using a modal synthesis at real-time have been discussed. These were grouped into two categories: synthesis after an object has been excited, and synthesis while an object is being affected by one or multiple forces.

When using a CPU, the incremental synthesis algorithm proved to be suitable when no forces are active, especially when truncating inaudible modes. Assuming objects with a complexity of $\sim$1000 modes, it is possible to synthesize the sound of around 8 to 9 objects in real-time, and even more when using mode truncation (approximately 12 for the benchmark scenario). Additionally, using multiple CPU cores can further increase the number of objects that can be handled.

However, when the modal excitation induced by external forces should be computed at sound sample rate, the CPU algorithm proves to be too slow. It can only handle $\sim$2 objects with 1000 modes each when a single force continuously acts on one vertex, and cannot handle 10 force-affected vertices for even a single object. In the latter case, it is necessary to reduce the force application rate, leading to inaccurate results. But even when lowering the force application rate, only very few objects with a low number of active forces are possible.

Instead of using the CPU, it is possible to utilize a graphics card for the sound synthesis. By designing algorithms tailored towards the requirements of GPU computations, it was possible to allow a much higher number of actively sounding objects than with CPU-based algorithms. When no forces are active, the explicit algorithm with active truncation can be used. This allows more than 100 objects with 1000 modes each to be synthesized when computing blocks of 512 samples. Using truncation, it was possible to process even more objects (around 245 in the benchmark). When applying force excitation at sample rate using a GPU synthesis, multiple optimizations were necessary to achieve a high performance. With the optimized algorithm, it is possible to handle approximately 20 objects with 1000 modes and ten active forces at full force application rate.

The GPU algorithms proved to be well suited for the task of modal sound synthesis with high force sampling rate. Using the presented approaches and optimizations, more and longer-lasting forces can be used to better model the contact interactions between objects. All in all, a high number of objects can be handled, allowing many objects in a scene to produce modal sounds, enriching even highly interactive VR applications.

Chapter 6 Conclusion

Interactive sound synthesis for VR applications is a challenging, but promising approach. Sound can help to improve the realism of a VE, but is also useful by supporting interaction tasks.

To evaluate the usefulness of synthesized sound, this thesis presented a user study comparing auditory and haptic feedback for a drilling task. While haptic feedback is an important modality, especially for manual interaction tasks, it is difficult to reproduce in a VE. In the study, participants had to perform a virtual drilling task, where a drill was used to remove target material while avoiding damage to the underlying surface. The interaction was supported by different feedback types: auditory and haptic reproduction of the drill vibrations, as well as surface force feedback. The results show that both the auditory and the haptic reproduction of the drill vibrations have a very similar influence on the user performance. This indicates that this kind of haptic feedback can be substituted by auditory feedback if a haptic reproduction of forces is not available, and demonstrates the usefulness of interactively synthesized sounds.

While audio can improve the quality of VR applications, the process of adding sounds to a scene can be cumbersome. Preparing sound samples up front requires a high effort due to the high amount of possible interactions that can produce sounds in highly interactive VR applications. Specialized interactive synthesis algorithms exist for a variety of sound sources, like musical instruments, water, or fire. These approaches produce high-quality sounds and are well suited for objects for which a realistic sound is important, but also require a design effort for each sound. However, unlike certain objects such as musical instruments, most objects in an interactive scenery do not have the primary purpose of creating sounds. To maintain a believable VE, however, they should still produce sounds when excited due to collisions or other contact forces. Producing a specialized sound for such cases would create a high amount of work, especially since the sound of an object changes depending on the location and temporal distribution of contact forces.

For such applications, Modal Synthesis (MS) is a promising approach that has gained increased attention in recent years. It can synthesize the contact sound of an object by simulating its surface vibrations. To allow a real-time synthesis, a MA is performed to determine vibration modes of an object, which define the way it deforms when forces are acting. This approach can synthesize sound based on a physical model and requires only an object's geometry and material as input. While MS only produces approximate results that do not achieve the quality of specially prepared sound samples, it can help to reduce the effort of adding sounds to objects for which approximate sound is sufficient. MS is a promising approach, but some challenges remain. This thesis presented solutions to make MS more viable for interactive applications.

To use MS for an object, a MA has to be performed to determine its modes. This is a time-consuming step and thus commonly performed as a pre-processing step. However, interactively created or modified objects cannot be analyzed upfront. This thesis presented methods that enable a run-time MA of an object. By using multiple LoDs, first approximate results can be computed quickly, and later be replaced by slower but more accurate analyses. Especially for very low-resolution LoDs, a geometric approximation was presented that can be computed quickly and is tailored towards suitability for a MA. The quality of the different LoDs was evaluated using example objects. To reduce the error introduced by lower-resolution approximations, a correction factor was devised that manages to largely compensate resolution-dependent pitch shifts. The LoDs are computed incrementally using an optimized multi-worker architecture, which also allows using remote PCs to increase available resources. Using the presented multi-LoD analysis, it is possible to compute the MA of new objects at run-time, and can even handle several MA requests at the same time. This enables to use MS even for complex situations such as fracturing windows, where many objects are created simultaneously.

Another challenge of MS is the run-time synthesis that computes the sound from modal vibrations. The high number of modes and the required temporal resolution limit the number of objects that can produce sounds simultaneously. Additionally, when forces act on an object, the corresponding excitation of the modes should also be computed at a high frequency, further increasing the computational cost. In this thesis, different algorithms have been presented to increase the performance of the run-time synthesis with a focus on high-resolution force application. The optimized CPU implementation enables synthesis of a few objects after an initial excitation. However, the high-frequency application of force excitation is not sufficiently fast, so that the use of dynamic contact forces is severely limited. To overcome these limitations, this thesis presented different algorithms to compute the modal sound synthesis on a graphics card, with additional focus on the high-frequency application of contact forces. Various optimizations were performed to adopt the algorithms for GPU computation, which achieved very high performance. The presented optimized GPU synthesis algorithms perform very well, and allow synthesizing the sound of more than 100 sounding objects, and also allows multiple forces to act on several of objects. This enables the use of MS event for scenes with many complex objects that are in contact with one another.

The approaches and methods presented in this thesis allow a run-time MA and sound synthesis of a high number of objects, decreasing the effort of adding sounds to VR scenes. For future work, approaches to improve the quality of the sounds produced by modal synthesis can be examined. Specifically, contacts between objects can have a strong influence on the produced sound, by damping certain modes or by transferring vibrations between objects. Simulating these contact coupling effects would require to modeling the contact dynamics at the vibration frequency rate, which is currently too expensive for real-time sound synthesis. Using approximate methods may help to allow adding contact coupling, thereby increasing the realism of the generated sounds.

Appendices

APPENDIX A

PHYSICS MATRIX ASSEMBLY

Given a tetrahedral mesh and material parameters for Young Modulus E, Poisson ratio ν, and density ρ, the mass and stiffness matrices can be assembled. The tetrahedral mesh $M = \{V,T\}$ consisting of vertices $V = \{p_i\}$ and tetrahedra $T = \{t_i\}$, where each tetrahedron $t_i = (p_i^0, p_i^1, p_i^2, p_i^3)$ is formed by four vertices.

A.1 Mass Matrix

To compute the mass matrix, the element mass matrix of each tetrahedral element $M^e \in \mathbb{R}^{12\times 12}$ with volume vol_i and density ρ is computed:

$$M_i = \frac{\rho \cdot vol_i}{20} \begin{bmatrix} 2&0&0&1&0&0&1&0&0&1&0&0 \\ 0&2&0&0&1&0&0&1&0&0&1&0 \\ 0&0&2&0&0&1&0&0&1&0&0&1 \\ 1&0&0&2&0&0&1&0&0&1&0&0 \\ 0&1&0&0&2&0&0&1&0&0&1&0 \\ 0&0&1&0&0&2&0&0&1&0&0&1 \\ 1&0&0&1&0&0&2&0&0&1&0&0 \\ 0&1&0&0&1&0&0&2&0&0&1&0 \\ 0&0&1&0&0&1&0&0&2&0&0&1 \\ 1&0&0&1&0&0&1&0&0&2&0&0 \\ 0&1&0&0&1&0&0&1&0&0&2&0 \\ 0&0&1&0&0&1&0&0&1&0&0&2 \end{bmatrix} \tag{A.1}$$

The entries of all element mass matrices are then accumulated in the corresponding vertices' entries for the global mass matrix.

A.1.1 Lumped Mass Matrix

The mass matrix can be approximated using a lumped mass formulation, where a tetrahedron's mass is evenly distributed to the mass of its vertices. Thus, one first computes the vertex masses m_i by distributing the mass of tetrahedron t with volume vol_t and density ρ onto its vertices.

```
for each tetrahedtron t
    for each vertex i in t
        m[i] += $\rho$ * vol_t / 4
```

The diagonal lumped-mass matrix is then assembled by placing the corresponding vertex mass on the diagonal entry corresponding to the respective vertex.

$$\mathbf{M} = \begin{bmatrix} m_0 & & & & \\ & \ddots & & & \\ & & m_i & & \\ & & & \ddots & \\ & & & & m_{n-1} \end{bmatrix} \tag{A.2}$$

A.2 Stiffness Matrix

The stiffness matrix $\mathbf{K}$ is computed by assembling the elemental stiffness matrices $\mathbf{K}^e \in \mathbb{R}^{12\times12}$. The element stiffness matrix of a linear tetrahedron can be computed as

$$\mathbf{K}^e = \int_{\Omega^e} \mathbf{B}^{e,T} \cdot \mathbf{E}^e \cdot \mathbf{B}^e \, d\Omega^e \tag{A.3}$$

Here, Ω_i is element domain, $\mathbf{B}_i$ is the *stress-strain matrix*, and $\mathbf{E}_i$ element's *elasticity matrix*. The derivation of these matrices is described in detail in the common FEM literature, e.g. [Bathe 2006].

For the computation of the element stiffness matrix, four helper values are defined.

$$n_0 = (p_2 - p_3) \times (p_1 - p_3) \tag{A.4}$$
$$n_1 = (p_0 - p_3) \times (p_2 - p_3) \tag{A.5}$$
$$n_2 = (p_1 - p_3) \times (p_0 - p_3) \tag{A.6}$$
$$n_3 = -n_1 - n_2 - n_3 \tag{A.7}$$

Using these, the different components $k^e_{i,j}$ of the element stiffness matrix $\mathbf{K}^e$ can be computed as follows:

$$k^e_{3k+l,3m+n} = -\frac{1}{6} \cdot (\lambda \cdot n_k[l] \cdot n_m[n] + \mu \cdot (n_m[l] \cdot n_k[n] + \delta_{ln} \cdot n_m \cdot n_k)) \tag{A.8}$$

Here, the Lamé parameters $\lambda = \frac{v \cdot E)}{(1+v)\cdot(1-2\cdot v)}$ and $\mu = \frac{E}{2+2\cdot v}$ are utilized. $\delta_{ln} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$ is the Kronecker Delta.

APPENDIX B

MATERIAL PARAMETERS

	Young Modulus	Poisson Ratio	Density	Mass Damping	Stiffness Damping
	$[\text{GPa}]$		$[\frac{\text{kg}}{\text{m}^3}]$		$[\times 10^{-9}]$
Steel	200	0.29	7850	5.0	30
Bronze	105	0.34	8100	5.0	25
Brass	110	0.357	8525	5.0	20
Ceramic	74	0.19	2700	6.0	100
Granite	52	0.24	2700	15.0	150

APPENDIX C
DRILLING STUDY

This appendix includes written instruction and questionnaires of the presented study comparing haptic and auditory feedback (see Chapter 3). Since all participants were native german speakers, these were only designed in german. The documents are the written pre-study instructions, as well as the post-study questionnaire used to collect participant information and subjective preferences.

C.1 Written Introduction

Vielen Dank für die Teilnahme!

In der hier durchgeführten Studie wird die Interaktion des Benutzers in einer virtuellen Umgebung untersucht. Dazu wird mehrfach unter verschiedenen Bedingungen eine einfache Aufgabe durchgeführt, bei der es um das Abtragen von (gelbgrünem) Material von einer Oberfläche geht (siehe auch Abbildung).

Ziel der Aufgaben:

- Es soll in der vorgegeben Zeit so viel vom **gelbgrünen Material** wie möglich entfernt werden.
- Dabei soll unbedingt darauf geachtet werden **kein graues Material von der Oberfläche zu entfernen!**

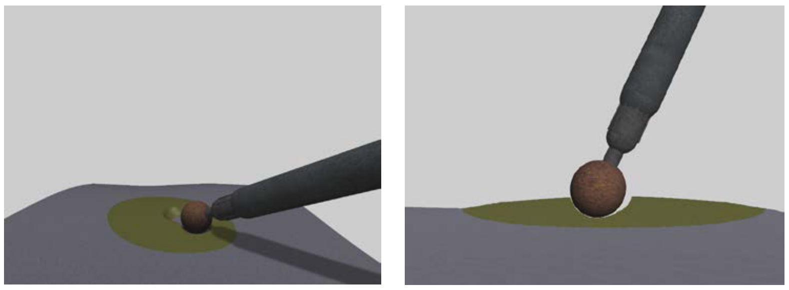


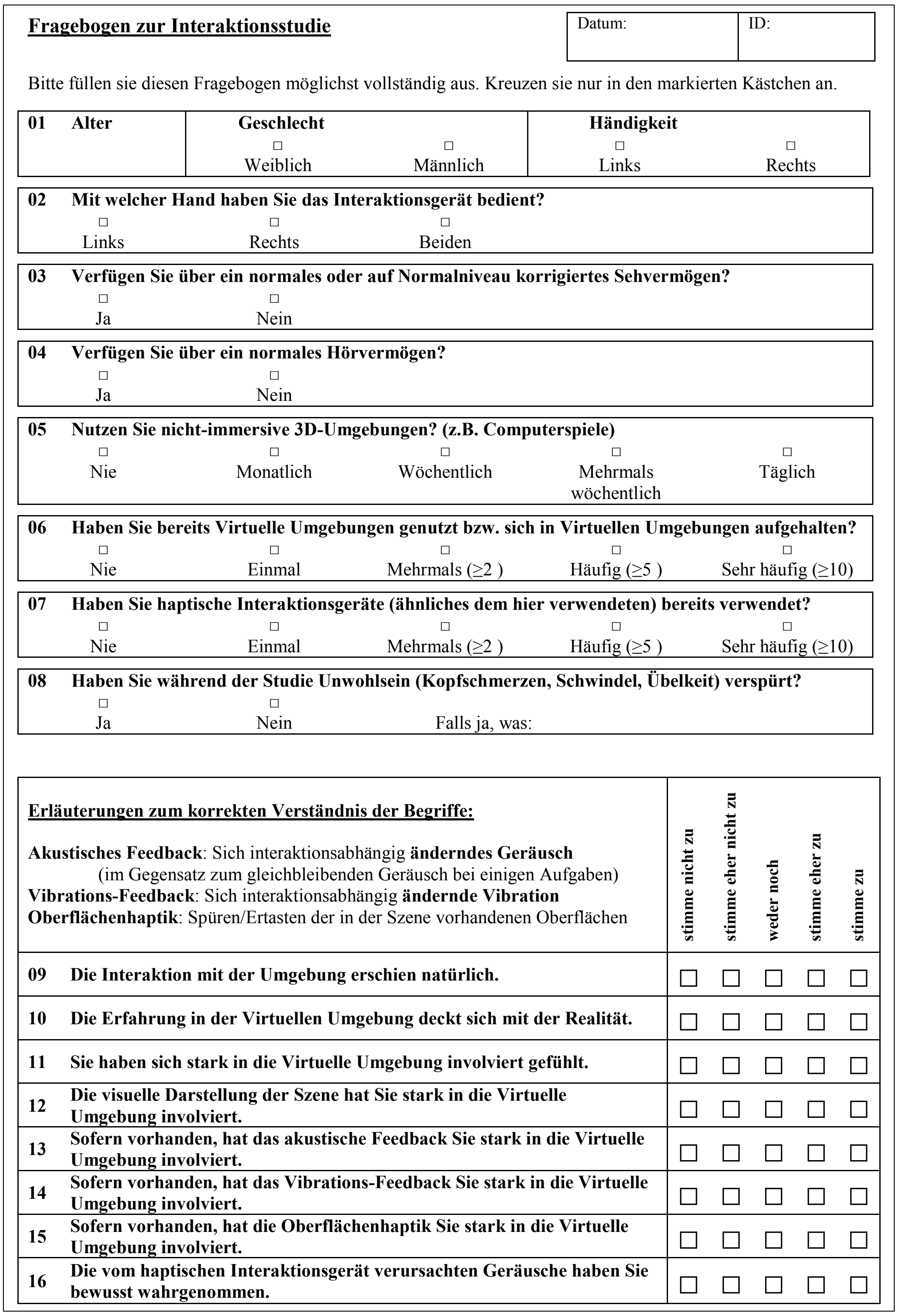
Abbildung: Studienszenario (links) und Querschnitt des abzutragenden Materials (rechts)

Informationen zum Ablauf:

- Während der gesamten Studie muss die bereitliegende **3D-Brille** (kann über einer normalen Brille getragen werden) und der **Kopfhörer** getragen werden.
- Die Aufgaben unterscheiden sich durch verschiedene Feedback-Kombinationen.
- Zunächst erfolgen **sechs Trainingsaufgaben**, wovon die Erste nach 60 Sekunden beendet wird, die fünf weiteren Aufgaben nach 25 Sekunden.
- Zum **Start** jeder Aufgabe muss das virtuelle Werkzeug **zwei Sekunden lang** in die **rote Kugel** gehalten werden, welche sich daraufhin grün färbt.
- Jede Aufgabe wird nach 25 Sekunden **automatisch beendet.**
- Nach Abschluss der Trainingsphase und während der eigentlichen Aufgabe erfolgen mehrere **Pausen**. Sie können hier das Interaktionsgerät aus der Hand legen.
- Sollten Sie sich beim Durchführen der Aufgabe unwohl fühlen (z.B. Kopfschmerzen oder Schwindel), unterbrechen Sie die Studie und geben mir bitte Bescheid.
- Am Ende der Studie erhalten Sie von mir noch einen Fragebogen. **Generelle (inhaltliche) Fragen** zur Studie bitte ich **erst im Anschluss** zu stellen.

Viel Spaß,
Lukas

C.2 Post-study Questionnaire



<u>Fragebogen zur Interaktionsstudie</u>

Datum:	ID:

Bitte füllen sie diesen Fragebogen möglichst vollständig aus. Kreuzen sie nur in den markierten Kästchen an.

01 Alter	Geschlecht		Händigkeit	
	☐ Weiblich	☐ Männlich	☐ Links	☐ Rechts

02 Mit welcher Hand haben Sie das Interaktionsgerät bedient?
☐ Links ☐ Rechts ☐ Beiden

03 Verfügen Sie über ein normales oder auf Normalniveau korrigiertes Sehvermögen?
☐ Ja ☐ Nein

04 Verfügen Sie über ein normales Hörvermögen?
☐ Ja ☐ Nein

05 Nutzen Sie nicht-immersive 3D-Umgebungen? (z.B. Computerspiele)
☐ Nie ☐ Monatlich ☐ Wöchentlich ☐ Mehrmals wöchentlich ☐ Täglich

06 Haben Sie bereits Virtuelle Umgebungen genutzt bzw. sich in Virtuellen Umgebungen aufgehalten?
☐ Nie ☐ Einmal ☐ Mehrmals (≥2) ☐ Häufig (≥5) ☐ Sehr häufig (≥10)

07 Haben Sie haptische Interaktionsgeräte (ähnliches dem hier verwendeten) bereits verwendet?
☐ Nie ☐ Einmal ☐ Mehrmals (≥2) ☐ Häufig (≥5) ☐ Sehr häufig (≥10)

08 Haben Sie während der Studie Unwohlsein (Kopfschmerzen, Schwindel, Übelkeit) verspürt?
☐ Ja ☐ Nein Falls ja, was:

<u>Erläuterungen zum korrekten Verständnis der Begriffe:</u>

Akustisches Feedback: Sich interaktionsabhängig **änderndes Geräusch** (im Gegensatz zum gleichbleibenden Geräusch bei einigen Aufgaben)
Vibrations-Feedback: Sich interaktionsabhängig **ändernde Vibration**
Oberflächenhaptik: Spüren/Ertasten der in der Szene vorhandenen Oberflächen

		stimme nicht zu	stimme eher nicht zu	weder noch	stimme eher zu	stimme zu
09	Die Interaktion mit der Umgebung erschien natürlich.	☐	☐	☐	☐	☐
10	Die Erfahrung in der Virtuellen Umgebung deckt sich mit der Realität.	☐	☐	☐	☐	☐
11	Sie haben sich stark in die Virtuelle Umgebung involviert gefühlt.	☐	☐	☐	☐	☐
12	Die visuelle Darstellung der Szene hat Sie stark in die Virtuelle Umgebung involviert.	☐	☐	☐	☐	☐
13	Sofern vorhanden, hat das akustische Feedback Sie stark in die Virtuelle Umgebung involviert.	☐	☐	☐	☐	☐
14	Sofern vorhanden, hat das Vibrations-Feedback Sie stark in die Virtuelle Umgebung involviert.	☐	☐	☐	☐	☐
15	Sofern vorhanden, hat die Oberflächenhaptik Sie stark in die Virtuelle Umgebung involviert.	☐	☐	☐	☐	☐
16	Die vom haptischen Interaktionsgerät verursachten Geräusche haben Sie bewusst wahrgenommen.	☐	☐	☐	☐	☐

Erläuterungen zum korrekten Verständnis der Begriffe:

Akustisches Feedback: Sich interaktionsabhängig **änderndes Geräusch**
(im Gegensatz zum gleichbleibenden Geräusch bei einigen Aufgaben)
Vibrations-Feedback: Sich interaktionsabhängig **ändernde Vibration**
Oberflächenhaptik: Spüren/Ertasten der in der Szene vorhandenen Oberflächen

		stimme nicht zu	**stimme eher nicht zu**	**weder noch**	**stimme eher zu**	**stimme zu**
17	**Die Veränderung des akustischen Feedbacks während einiger Aufgaben haben Sie wahrgenommen.**	☐	☐	☐	☐	☐
18	**Die Veränderung des Vibrations-Feedbacks während einiger Aufgaben haben Sie wahrgenommen.**	☐	☐	☐	☐	☐
19	**Die visuelle Darstellung der Szene hat bei den Aufgaben geholfen.**	☐	☐	☐	☐	☐
20	**Die stereoskopische Projektion (3D-Sicht) hat bei den Aufgaben geholfen.**	☐	☐	☐	☐	☐
21	**Die Möglichkeit, durch Kopfbewegung die Ansicht der Szene zu ändern, hat bei den Aufgaben geholfen.**	☐	☐	☐	☐	☐
22	**Das akustische Feedback (sofern vorhanden) hat bei den Aufgaben geholfen.**	☐	☐	☐	☐	☐
23	**Die Oberflächenhaptik (sofern vorhanden) hat bei den Aufgaben geholfen.**	☐	☐	☐	☐	☐
24	**Das Vibrations-Feedback (sofern vorhanden) hat bei den Aufgaben geholfen.**	☐	☐	☐	☐	☐
25	**Das Vibrations-Feedback hat mehr geholfen als das akustische Feedback.**	☐	☐	☐	☐	☐
26	**Die Oberflächenhaptik hat mehr geholfen als das akustische Feedback.**	☐	☐	☐	☐	☐
27	**Die Oberflächenhaptik hat mehr geholfen als das Vibrations-Feedback.**	☐	☐	☐	☐	☐
28	**Die Durchführung der Aufgaben war einfach.**	☐	☐	☐	☐	☐
29	**Die Aufgaben <u>ohne</u> Akustik- und <u>ohne</u> Vibrations-Feedback waren einfach.**	☐	☐	☐	☐	☐
30	**Die Aufgaben <u>mit</u> Akustik-, aber <u>ohne</u> Vibrations-Feedback waren einfach.**	☐	☐	☐	☐	☐
31	**Die Aufgaben <u>ohne</u> Akustik-, aber <u>mit</u> Vibrations-Feedback waren einfach.**	☐	☐	☐	☐	☐
32	**Die Aufgaben <u>mit</u> Akustik- und <u>mit</u> Vibrations-Feedback waren einfach.**	☐	☐	☐	☐	☐
33	**Das Vorhandensein von Oberflächenhaptik macht die Aufgaben einfacher.**	☐	☐	☐	☐	☐
34	**Sie haben sich bewusst auf Ihr Gehör konzentriert (sofern entsprechendes Feedback vorhanden).**	☐	☐	☐	☐	☐
35	**Sie haben sich bewusst auf Ihren Tastsinn konzentriert (sofern entsprechendes Feedback vorhanden).**	☐	☐	☐	☐	☐
36	**Sie haben bewusst auf mehrere Sinne gleichzeitig geachtet (sofern entsprechendes Feedback vorhanden).**	☐	☐	☐	☐	☐

Anmerkungen / Kommentare:

APPENDIX D

ACRONYMS

VR Virtual Reality

VE Virtual Environment

HIP haptic interaction point

VIP virtual interaction point

SMS Spring-Mass System

FEM Finite Element Method

BEM Boundary Element Method

MA Modal Analysis

MS Modal Synthesis

FFT Fast Fourier Transform

PCM Pulse Code Modulation

PCA Principal Component Analysis

LoD Level-of-Detail

DoF Degree-of-Freedom

APPENDIX E

MATHEMATICAL SYMBOLS

Formatting	
$\mathbf{M} \in \mathbb{R}^{n \times m}$	Matrix
$\mathbf{v} \in \mathbb{R}^{n}$	Vector
$s \in \mathbb{R}$	Scalar number
$\underline{c} \in \mathbb{C}$	Complex number
$\mathfrak{Re}\,(a+i\cdot b)$	Real part of complex number
$\mathfrak{Im}\,(a+i\cdot b)$	Imaginary part of complex number

Geometry State	
p	Global position
$\mathbf{p} = \begin{pmatrix} \vdots \\ p_i \\ \vdots \end{pmatrix}$	Global position vector
d	Displacement
$\mathbf{d} = \begin{pmatrix} \vdots \\ d_i \\ \vdots \end{pmatrix}$	Displacement vector
q	Substituted Displacement
$\mathbf{q} = \begin{pmatrix} \vdots \\ q_i \\ \vdots \end{pmatrix} = \mathbf{L}^T \cdot \mathbf{d}$	Displacement vector transformed into mode space

r	Displacement transformed into mode space
$\mathbf{r} = \begin{pmatrix} \vdots \\ r_i \\ \vdots \end{pmatrix} = \mathbf{V}^{-1} \cdot \mathbf{q} = \mathbf{V}^{-1} \cdot \mathbf{L}^T \cdot \mathbf{d}$	Displacement vector transformed into mode space

Eigen Decomposition

v	Entry of EigenVector
$\mathbf{v} = \begin{pmatrix} \vdots \\ v_i \\ \vdots \end{pmatrix}$	EigenVector
$\mathbf{V} = [\ldots \mathbf{v}_j \ldots] = [v_{i,j}]$	Matrix of EigenVectors
λ	EigenValue
$\boldsymbol{\lambda} = \begin{pmatrix} \vdots \\ \lambda_i \\ \vdots \end{pmatrix}$	Vector of EigenValues
$\mathbf{\Lambda} = \begin{bmatrix} \ddots & \vdots & \vdots & \\ \cdots & \lambda_i & 0 & \cdots \\ \cdots & 0 & \lambda_{i+1} & \cdots \\ & \vdots & \vdots & \ddots \end{bmatrix}$	Diagonal Matrix of EigenValues

Dynamics Matrices

$\mathbf{I}$	Unit Matrix
$k_{i,j}$	Stiffness matrix entry
$\mathbf{K} = [k_{i,j}]$	Stiffness Matrix
m	Mass
$m_{i,j}$	Mass matrix entry
$\mathbf{M} = [m_{i,j}]$	Mass Matrix
$c_{i,j}$	Damping Matrix entry
$\mathbf{C} = [c_{i,j}]$	Damping Matrix
f_i	External Force entry
$\mathbf{f}_{ext} = \begin{pmatrix} \vdots \\ f_i \\ \vdots \end{pmatrix}$	External Force Vector

g_i	Entry of transformed force Vector
$\mathbf{g} = \mathbf{V}^T \cdot \mathbf{L}^{-1} \cdot \mathbf{f}_{ext} = \begin{pmatrix} \vdots \\ g_i \\ \vdots \end{pmatrix}$	Force vector transformed into mode space
$\mathbf{L} \quad \vert \quad \mathbf{M} = \mathbf{L} \cdot \mathbf{L}^T$	Lower Cholesky Factorization of Mass Matrix
$\mathbf{Q} = \mathbf{L}^{-1} \cdot \mathbf{K} \cdot \mathbf{L}^{-T}$	Mass-normalized stiffness matrix

Material Parameters

$\rho \in \mathbb{R}$	Density
$E \in \mathbb{R}$	Young Modulus
$\nu \in \mathbb{R}$	Poisson Ratio
$\alpha \in \mathbb{R}$	Stiffness damping
$\beta \in \mathbb{R}$	Mass damping

Mode Parameters

$\underline{a} = a + i \cdot b \in \mathbb{C}$	Complex mode amplitude
$a = \mathfrak{Re}(\underline{a}) \in \mathbb{R}$	Real component of mode amplitude
$b = \mathfrak{Im}(\underline{a}) \in \mathbb{R}$	Imaginary component of mode amplitude
$\underline{w} = \gamma + i \cdot \omega \in \mathbb{C}$	Complex angular frequency of a mode
$\omega \in \mathbb{R}$	Angular Frequency of a mode
$\gamma \in \mathbb{R}$	Mode damping
$r \in \mathbb{R}$	Mode state of a mode
$\underline{r} \in \mathbb{C}$	Complex-valued mode state of a mode
$\mathbf{s} \in \mathbb{R}^k$	Surface deformation caused by a mode
q	Mode's acoustic transfer factor

Chapter 6

Bibliography

Adrien, Jean-Marie (1991). "The Missing Link: Modal Synthesis". In: *Representations of Musical Signals*. MIT Press, pp. 269–298.

Allemang, Randall J.; Brown, David L. (1993). "Experimental Modal Analysis". In: *Handbook on Experimental Mechanics*, pp. 635–750.

Alliez, Pierre; De Verdire, Éric Colin; Devillers, Olivier; Isenburg, Martin (2003). "Isotropic Surface Remeshing". In: *Shape Modeling International*, pp. 49–58.

Aspöck, Lukas (2012). "Vergleich von akustischem und haptischem Feedback bei Interaktionsaufgaben in Virtuellen Umgebungen". Term Paper (Studienarbeit). RWTH Aachen University.

Assenmacher, Ingo; Kuhlen, Torsten (2008). "The ViSTA Virtual Reality Toolkit". In: *SEARIS Workshop on IEEE VR*, pp. 23–26.

Avanzini, Federico; Rocchesso, Davide (2001). "Modeling Collision Sounds: Non-linear Contact Force". In: *Conference on Digital Audio Effects (DAFx-01)*, pp. 61–66.

Avanzini, Frederico; Crosato, Paolo (2006). "Haptic-auditory Rendering and Perception of Contact Stiffness". In: *Haptic and Audio Interaction Design*, pp. 24–35.

Barbič, Jernej; James, Doug L. (2005). "Real-time Subspace Integration for St. Venant-Kirchhoff Deformable Models". In: *ACM Transactions on Graphics*. Vol. 24. 3. ACM, pp. 982–990.

Barbič, Jernej; James, Doug L. (2008). "Six-DoF Haptic Rendering of Contact Between Geometrically Complex Reduced Deformable Models". In: *IEEE Transactions on Haptics*, pp. 39–52.

Barzel, Ronen; Hughes, John R.; Wood, Daniel N. (1996). "Plausible Motion Simulation for Computer Graphics Animation". In: *Computer Animation and Simulation*, pp. 183–197.

Bathe, Klaus-Jürgen (2006). *Finite Element Procedures*.

Berkhout, Augustinus J.; de Vries, Diemer; Vogel, Peter (1993). "Acoustic Control by Wave Field Synthesis". In: *The Journal of the Acoustical Society of America* Vol. 93, p. 2764.

Bilbao, Stefan (2008). "Cymbal Synthesis". In: *The Journal of the Acoustical Society of America* Vol. 123:5, pp. 3521–3521.

Bilbao, Stefan (2009). *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*.

Bonneel, Nicolas; Drettakis, George; Tsingos, Nicolas; Viaud-Delmon, Isabelle; James, Doug (2008). “Fast Modal Sounds with Scalable Frequency-domain Synthesis”. In: *ACM SIGGRAPH*.

Bonneel, Nicolas; Suied, Clara; Viaud-Delmon, Isabelle; Drettakis, George (2010). “Bimodal Perception of Audio-Visual Material Properties for Virtual Environments”. In: *ACM Transactions on Applied Perception (TAP)* Vol. 7:1, p. 1.

Bormann, Karsten (2005). “Presence and the Utility of Audio Spatialization”. In: *Presence: Teleoperators and Virtual Environments* Vol. 14:3, pp. 278–297.

Brogni, Andrea; Caldwell, Darwin G; Slater, Mel (2011). “Touching Sharp Virtual Objects Produces a Haptic Illusion”. In: *Virtual and Mixed Reality-New Trends*, pp. 234–242.

Bruyns-Maxwell, Cynthia (2007). “A Simple Simulation of Acoustic Radiation from a Vibrating Object”. In: *Audio Engineering Society Convention 123*.

Chadwick, Jeffrey N; An, Steven S; James, Doug L (2009). “Harmonic Shells: A Practical Nonlinear Sound Model for Near-Rigid Thin Shells”. In: *ACM Transactions on Graphics* Vol. 28:5, 119:1–119:10.

Chadwick, Jeffrey N; James, Doug L (2011). “Animating Fire with Sound”. In: *ACM Transactions on Graphics*. Vol. 30. 4, p. 84.

Chadwick, Jeffrey N.; Zheng, Changxi; James, Doug L (2012a). “Faster Acceleration Noise for Multibody Animations using Precomputed Soundbanks”. In: *ACM SIGGRAPH*, pp. 265–273.

Chadwick, Jeffrey N; Zheng, Changxi; James, Doug L (2012b). “Precomputed Acceleration Noise for Improved Rigid-body Sound”. In: *ACM Transactions on Graphics* Vol. 31:4, 103:1–103:9.

Chaigne, Antoine; Touzé, Cyril; Thomas, Olivier (2005). “Nonlinear Vibrations and Chaos in Gongs and Cymbals”. In: *Acoustical Science and Technology* Vol. 26:5, pp. 403–409.

Chueng, Priscilla; Marsden, Phil (2002). “Designing Auditory Spaces to Support Sense of Place: The Role of Expectation”. In: *Proceedings of the CSCW Workshop: The Role of Place in Shaping Virtual Community*.

Cignoni, Paolo; Montani, Claudio; Scopigno, Roberto (1998). “A Comparison of Mesh Simplification Algorithms”. In: *Computers & Graphics* Vol. 22:1, pp. 37–54.

Ciskowski, Robert D; Brebbia, Carlos Alberto (1991). *Boundary Element Methods in Acoustics*.

Coles, Timothy; Meglan, Dwight; John, Nigel W (2011). “The Role of Haptics in Medical Training Simulators: A Survey of the State-of-the-Art”. In: *IEEE Transactions on Haptics*, pp. 51–66.

Colgate, J. Edward; Stanley, Michael C; Brown, J. Michael (1995). “Issues in the Haptic Display of Tool Use”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 3, pp. 140–145.

Cremer, Lothar; Heckl, Manfred (1985). *Structure-Borne Sound: Structural Vibrations and Sound Radiation at Audio Frequencies*.

Cruz-Neira, Carolina; Sandin, Daniel J; DeFanti, Thomas A; Kenyon, Robert V; Hart, John C (1992). “The CAVE: Audio Visual Experience Automatic Virtual Environment”. In: *Communications of the ACM* Vol. 35:6, pp. 64–72.

Cuppen, JJM (1980). “A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem”. In: *Numerische Mathematik* Vol. 36:2, pp. 177–195.

Demmel, James; Dumitriu, Ioana; Holtz, Olga (2007). “Fast Linear Algebra is Stable”. In: *Numerische Mathematik* Vol. 108:1, pp. 59–91.

Díaz, Iñaki; Hernantes, Josune; Mansa, Ignacio; Lozano, Alberto; Borro, Diego; Gil, Jorge J; Sanchez, Emilio (2006). “Influence of Multisensory Feedback on Haptic Accessibility Tasks”. In: *IEEE Virtual Reality*, pp. 31–40.

Dobashi, Yoshinori; Yamamoto, Tsuyoshi; Nishita, Tomoyuki (2003). “Real-Time Rendering of Aerodynamic Sound using Sound Textures based on Computational Fluid Dynamics”. In: *ACM Transactions on Graphics* Vol. 22:3, pp. 732–740.

Edwards, Gregory W; Barfield, Woodrow; Nussbaum, Maury A (2004). “The Use of Force Feedback and Auditory Cues for Performance of an Assembly Task in an Immersive Virtual Environment”. In: *IEEE Virtual Reality* Vol. 7:2, pp. 112–119.

Ewins, David J (1995). *Modal Testing: Theory and Practice*. Vol. 6.

Freed, Adrian; Rodet, Xavier; Depalle, Philippe (1992). “Synthesis and Control of Hundreds of Sinusoidal Partials on a Desktop Computer without Custom Hardware”. In: *ICSPAT (International Conference on Signal Processing Applications & Technology*.

Garland, Michael; Heckbert, Paul S. (1997). “Surface Simplification using Quadric Error Metrics”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

Gerzon, Michael A (1985). “Ambisonics in Multichannel Broadcasting and Video”. In: *Journal of the Audio Engineering Society* Vol. 33:11, pp. 859–871.

Glondu, Loeiz; Legouis, Benoit; Marchal, Maud; Dumont, Georges, et al. (2011). “Precomputed Shape Database for Real-Time Physically-Based Simulation”. In: *VRIPHYS*, pp. 47–54.

Glondu, Loeiz; Marchal, Maud; Dumont, Georges (2013). “Real-time Simulation of Brittle Fracture using Modal Analyis”. In: *IEEE Transactions on Visualization and Computer Graphics* Vol. 19:2, pp. 201–209.

Gottschalk, Stefan; Lin, Ming; Manocha, Dinesh (1996). “OBBTree: A Hierarchical Structure for Rapid Interference Detection”. In: *Proceedings of the ACM Conference on Computer Graphics*, pp. 171–180.

Hearn, George; Testa, Rene B (1991). “Modal Analysis for Damage Detection in Structures”. In: *Journal of Structural Engineering* Vol. 117:10, pp. 3042–3063.

Hsu, Bill; Sosnick-Pérez, Marc (2013). “Real-time GPU audio”. In: *Communications of the ACM* Vol. 56:6, pp. 54–62.

James, Doug L; Pai, Dinesh K (2002). “DyRT: Dynamic Response Textures for Real Time Deformation Simulation with Graphics Hardware”. In: *ACM Transactions on Graphics* Vol. 21:3, pp. 582–585.

James, Doug L; Barbič, Jernej; Pai, Dinesh K (2006). “Precomputed Acoustic Transfer: Output-Sensitive, Accurate Sound Generation for Geometrically Complex Vibration Sources”. In: *ACM Transactions on Graphics* Vol. 25:3, pp. 987–995.

Johnson, Kenneth Langstreth; Johnson, Kenneth Langstreth (1987). *Contact Mechanics*.

Kang, Jae-Hoon; Leissa, Arthur W (2008). “Vibration Analysis of Solid Ellipsoids and Hollow ellipsoidal Shells of Revolution with Variable Thickness from a Three-dimensional Theory”. In: *Acta Mechanica* Vol. 197:1-2, pp. 97–117.

Karjalainen, Matti; Välimäki, Vesa; Jánosy, Zoltán (1993). “Towards High-Quality Sound Synthesis of the Guitar and String Instruments”. In: *Proceedings of the International Computer Music Conference*, pp. 56–56.

Karjalainen, Matti; Ansalo, Poju; Mäkivirta, Aki; Peltonen, Timo; Välimäki, Vesa (2002). “Estimation of Modal Decay Parameters from Noisy Response Measurements”. In: *Journal of the Audio Engineering Society* Vol. 50:11, pp. 867–878.

Kim, Seung-Chan; Kwon, Dong-Soo (2007). “Haptic and Sound Grid for Enhanced Positioning in a 3-D Virtual Environment”. In: *2nd International Conference on Haptic and Audio Interaction Design*, pp. 98–109.

Klatzky, Roberta L; Pai, Dinesh K; Krotkov, Eric P (2000). “Perception of Material from Contact Sounds”. In: *Presence: Teleoperators and Virtual Environments* Vol. 9:4, pp. 399–410.

Klosowski, James T; Held, Martin; Mitchell, Joseph SB; Sowizral, Henry; Zikan, Karel (1998). “Efficient Collision Detection using Bounding Volume Hierarchies of k-DOPs”. In: *Visualization and Computer Graphics, IEEE Transactions on* Vol. 4:1, pp. 21–36.

Langlois, Timothy R; An, Steven S; Jin, Kelvin K; James, Doug L (2014). “Eigenmode Compression for Modal Sound Models”. In: *ACM Transactions on Graphics* Vol. 33:4, p. 40.

Larsson, Pontus; Västfjäll, Daniel; Kleiner, Mendel (2001). “Ecological Acoustics and the Multi-Modal Perception of Rooms: Real and Unreal Experiences of Auditory-Visual Virtual Environments”. In: *Proceedings of the Conference on Auditory Display*, pp. 245–249.

Larsson, Pontus; Västfjäll, Daniel; Kleiner, Mendel (2002). “Auditory-Visual Interaction in Real and Virtual Rooms”. In: *Proceedings of the Forum Acusticum, 3rd EAA European Congress on Acoustics*.

Lécuyer, Anatole; Mégard, Christine; Burkhardt, Jean-Marie; Lim, Taegi; Coquillart, Sabine; Coiffet, Philippe; Graux, Ludovic (2002). “The Effect of Haptic, Visual and Auditory Feedback on an Insertion Task on a 2-Screen Workbench”. In: *Proceedings of the Immersive Projection Technology Symposium*.

Leissa, Arthur W (1969). *Vibration of Plates*. Tech. rep. DTIC Document.

Lent, Keith (1989). “An Efficient Method for Pitch Shifting Digitally Sampled Sounds”. In: *Computer Music Journal* Vol. 13:4, pp. 65–71.

Lentz, Tobias; Schröder, Dirk; Vorländer, Michael; Assenmacher, Ingo (2007). “Virtual Reality System with Integrated Sound Field Simulation and Reproduction”. In: *EURASIP Journal on Applied Signal Processing* Vol. 2007:1, pp. 187–187.

Lloyd, D Brandon; Raghuvanshi, Nikunj; Govindaraju, Naga K (2011). “Sound Synthesis for Impact Sounds in Video Games”. In: *Symposium on Interactive 3D Graphics and Games*. I3D ’11, 55–62 PAGE@7.

Martín, Javier; Savall, Joan; Díaz, Inaki; Hernantes, Josune; Borro, Diego (2008). “Evaluation of Sensory Substitution to Simplify the Mechanical Design of a Haptic Wrist”. In: *HAVE 2008: Haptic Audio Visual Environments and Games*. IEEE, pp. 132–136.

Maxwell, Cynthia (2008). “Sound Synthesis from Shape-Changing Geometric Models”. PhD thesis. University of California at Berkeley.

Maxwell, Cynthia Bruyns; Bindel, David (2007). “Modal Parameter Tracking for Shape-Changing Geometric Objects”. In: *Proceedings of the 10th International Conference on Digital Audio Effects (DAFx07),* pp. 253–260.

Meijden, O.A.J. van der; Schijven, MP (2009). “The Value of Haptic Feedback in Conventional and Robot-Assisted Minimal Invasive Surgery and Virtual Reality Training: A Current Review”. In: *Surgical Endoscopy* Vol. 23:6, pp. 1180–1190.

Ménélas, Bob; Picinalli, Lorenzo; Katz, Brian F G; Bourdot, Patrick (2010). “Audio Haptic Feedbacks for an Acquisition Task in a Multi-Target Context”. In: *IEEE Symposium on 3D User Interfaces (3DUI)*. IEEE, pp. 51–54.

Morse, Philip McCord (1948). *Vibration and Sound*. Vol. 2.

Moss, William; Yeh, Hengchin; Hong, Jeong-Mo; Lin, Ming C; Manocha, Dinesh (2010). “Sounding Liquids: Automatic Sound Synthesis from Fluid Simulation”. In: *ACM Transactions on Graphics* Vol. 29:3, p. 21.

Nealen, Andrew; Müller, Matthias; Keiser, Richard; Boxerman, Eddy; Carlson, Mark (2006). “Physically Based Deformable Models in Computer Graphics”. In: *Computer Graphics Forum*. Vol. 25. 4, pp. 809–836.

O’Brien, James F; Cook, Perry R; Essl, Georg (2001). “Synthesizing Sounds from Physically based Motion”. In: *ACM SIGGRAPH*, pp. 529–536.

O’Brien, James F; Shen, Chen; Gatchalian, Christine M (2002). “Synthesizing Sounds from Rigid-Body Simulations”. In: *ACM SIGGRAPH*, pp. 175–181.

Pai, Dinesh K; van den Doel, Kees; James, Doug L; Lang, Jochen; Lloyd, John E; Richmond, Joshua L; Yau, Som H (2001). “Scanning Physical Interaction Behavior of 3D Objects”. In: *Conference on Computer Graphics and Interactive Techniques*, pp. 87–96.

Parlett, Beresford N; Dhillon, Inderjit S (2000). “Relatively Robust Representations of Symmetric Tridiagonals”. In: *Linear Algebra and its Applications* Vol. 309:1, pp. 121–151.

Pentland, A.; Williams, J. (1989). “Good Vibrations: Modal Dynamics for Graphics and Animation”. In: *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* Vol. 23:3, pp. 207–214.

Petzold, Bernd; Zaeh, Michael F; Faerber, Berthold; Deml, Barbara; Egermeier, Hans; Schilp, Johannes; Clarke, Stella (2004). “A Study on Visual, Auditory, and Haptic Feedback for Assembly Tasks”. In: *Presence: Teleoperators and Virtual Environments* Vol. 13:1, pp. 16–21.

Picard, Cécile; Tsingos, Nicolas; Faure, François, et al. (2008). “Audio Texture Synthesis for Complex Contact Interactions”. In: *Virtual Reality Interaction and Physical Simulation*.

Picard, Cécile; Faure, François; Drettakis, George; Kry, Paul, et al. (2009a). “A Robust and Multi-Scale Modal Analysis for Sound Synthesis”. In: *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*.

Picard, Cécile; Tsingos, Nicolas; Faure, François (2009b). “Retargetting Example Sounds to Interactive Physics-Driven Animations”. In: *Audio Engineering Society Conference: 35th International Conference: Audio for Games*.

Picard, Cécile; Frisson, Christian; Faure, François; Drettakis, George; Kry, Paul G. (2010). "Advances in Modal Analysis using a Robust and Multiscale Method". In: *EURASIP Journal on Advanced Signal Processing* Vol. 2010, 7:1–7:12.

Pulkki, Ville (1997). "Virtual Sound Source Positioning using Vector Base Amplitude Panning". In: *Journal of the Audio Engineering Society* Vol. 45:6, pp. 456–466.

Rabenstein, Rudolf; Trautmann, Lutz (2001). "Digital Sound Synthesis by Physical Modelling". In: *International Symposium on Image and Signal Processing and Analysis*, pp. 12–23.

Raghuvanshi, Nikunj; Lin, Ming C. (2006). "Interactive Sound Synthesis for Large Scale Environments". In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D '06, pp. 101–108.

Rausch, Dominik; Aspöck, Lukas; Knott, Thomas; Pelzer, Sönke; Vorländer, Michael; Kuhlen, Torsten (2012). "Comparing Auditory and Haptic Feedback for a Virtual Drilling Task". In: *Joint Virtual Reality Conference of ICAT-EGVE-EuroVR*, pp. 49–56.

Rausch, Dominik; Hentschel, Bernd; Kuhlen, Torsten (2014). "Efficient Modal Sound Synthesis on GPUs". In: *IEEE VR Workshop: Sonic Interaction in Virtual Environments (SIVE)*, pp. 13–18.

Rausch, Dominik; Hentschel, Bernd; Kuhlen, Torsten W (2015). "Level-of-Detail Modal Analysis for Real-time Sound Synthesis". In: *Virtual Reality Interaction and Physical Simulation (VRIPHYS)*, pp. 61–70.

Reddy, Junuthula N (2006). *Theory and Analysis of Elastic Plates and Shells*.

Ren, Zhimin; Yeh, Hengchin; Lin, Ming C (2010). "Synthesizing Contact Sounds between Textured Models". In: *IEEE Virtual Reality*, pp. 139–146.

Ren, Zhimin; Yeh, Hengchin; Klatzky, Roberta; Lin, Ming C (2013a). "Auditory Perception of Geometry-Invariant Material Properties". In: *IEEE Transactions on Visualization and Computer Graphics* Vol. 19:4, pp. 557–566.

Ren, Zhimin; Yeh, Hengchin; Lin, Ming C. (2013b). "Example-Guided Physically based Modal Sound Synthesis". In: *ACM Transaction on Graphics* Vol. 32:1, 1:1–1:16.

Richard, Paul; Burdea, Grigore; Gomez, D.; Coiffet, P. (1994). "A Comparison of Haptic, Visual and Auditive Force Feedback for Deformable Virtual Objects". In: *Proceedings of ICAT*. Vol. 94, pp. 49–62.

Richards, EJ; Westcott, ME; Jeyapalan, RK (1979). "On the Prediction of Impact Noise, I: Acceleration Noise". In: *Journal of Sound and Vibration* Vol. 62:4, pp. 547–575.

Rossing, Thomas D; Fletcher, Neville H (1995). *Principles of Vibration and Sound*.

Savioja, Lauri; Välimäki, Vesa; Smith III, Julius O (2010). "Real-time Additive Synthesis with one Million Sinusoids using a GPU". In: *Audio Engineering Society Convention 128*. Audio Engineering Society.

Schröder, Dirk (2011). "Physically Based Real-Time Auralization of Interactive Virtual Environments". PhD thesis. RWTH Aachen University.

Shabana, Ahmed (1996). *Theory of Vibration II: Vibration of Discrete and Continuous Systems*. Vol. 2.

Smyth, Tamara; Scott, Frederick S (2011). "Trombone Synthesis by Model and Measurement". In: *EURASIP Journal on Advances in Signal Processing* Vol. 2011:1, pp. 1–13.

Strutt, John William; Rayleigh, Baron; Lindsay, Robert Bruce (1945). *The Theory of Sound*.

Teschner, Matthias; Heidelberger, Bruno; Müller, Matthias; Gross, Markus (2004). “A Versatile and Robust Model for Geometrically Complex Deformable Solids”. In: *Computer Graphics International*. IEEE, pp. 312–319.

Tolonen, T; Välimäki, V; Karjalainen, M (1998). *Evaluation of Modern Sound Synthesis Methods*. Tech. rep. Helsinki University of Technology.

Truax, Barry (1988). “Real-time Granular Synthesis with a Digital Signal Processor”. In: *Computer Music Journal* Vol. 12:2, pp. 14–26.

Tsai, Pei-Yin; Wang, Tien-Ming; Su, Alvin (2010). “GPU-based Spectral Model Synthesis for Real-Time Sound Rendering”. In: *13th International Conference on Digital Audio Effects*, pp. 1–5.

Ullrich, Sebastian; Rausch, Dominik; Kuhlen, Torsten W (2011). “Bimanual Haptic Simulator for Medical Training: System Architecture and Performance Measurements”. In: *Joint Virtual Reality Conference of EGVE/EuroVR*, pp. 39–46.

van den Doel, Kees (2005). “Physically based Models for Liquid Sounds”. In: *ACM Transactions on Applied Perception (TAP)* Vol. 2:4, pp. 534–546.

van den Doel, Kees; Pai, Dinesh K. (1998). “The Sounds of Physical Shapes”. In: *Presence: Teleopertors and Virtual Environments* Vol. 7:4, pp. 382–395.

van den Doel, Kees; Kry, Paul G; Pai, Dinesh K (2001). “FoleyAutomatic: Physically-based Sound Effects for Interactive Simulation and Animation”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 537–544.

van den Doel, Kees; Knott, Dave; Pai, Dinesh K (2004). “Interactive Simulation of Complex Audiovisual Scenes”. In: *Presence: Teleoperators and Virtual Environments* Vol. 13:1, pp. 99–111.

Van Duyne, Scott; Smith III, Julius (1993). “The 2-D Digital Waveguide Mesh”. In: *Applications of Signal Processing to Audio and Acoustics*, pp. 177–180.

Van Santen, Jan; Sproat, Richard; Olive, Joseph; Hirschberg, Julia (2013). *Progress in Speech Synthesis*.

Västfjäll, Daniel (2003). “The Subjective Sense of Presence, Emotion Recognition, and Experienced Emotions in Auditory Virtual Environments”. In: *CyberPsychology & Behavior* Vol. 6:2, pp. 181–188.

Vorländer, Michael (2008). *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*.

Wan, Ming; McNeely, William A (2003). “Quasi-static Approximation for 6 Degrees-of-Freedom Haptic Rendering”. In: *Proceedings of the 14th IEEE Visualization 2003*, pp. 257–262.

Wilkinson, James Hardy; Wilkinson, James Hardy; Wilkinson, James Hardy (1965). *The Algebraic Eigenvalue Problem*. Vol. 87.

Zambon, Stefano (2012). “Accurate Sound Synthesis of 3D Object Collisions in Interactive Virtual Scenarios”. PhD thesis. Universita degli Studi di Verona.

Zhang, Qiong; Ye, Lu; Pan, Zhigeng (2005). “Physically-based Sound Synthesis on GPUs”. In: *Proceedings of the 4th international Conference on Entertainment Computing*. ICEC’05, pp. 328–333.

Zhang, Ying; Fernando, Terrence; Xiao, Hannan; Travis, Adrian R L (2006). “Evaluation of Auditory and Visual Feedback on Task Performance in a Virtual Assembly Environment”. In: *Presence: Teleoperators and Virtual Environments* Vol. 15:6, pp. 613–626.

Zheng, Changxi; James, Doug L. (2010). “Rigid-Body Fracture Sound with Precomputed Soundbanks”. In: *ACM SIGGRAPH*, 69:1–69:13.

Zheng, Changxi; James, Doug L. (2011). “Toward High-Quality Modal Contact Sound”. In: *ACM SIGGRAPH*, 38:1–38:12.