Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2015)
F. Jaillet, G. Zachmann, and F. Zara (Editors)

Level-of-Detail Modal Analysis for Real-time Sound Synthesis

Dominik Rausch[1,2] and Bernd Hentschel[1,2] and Torsten W. Kuhlen[1,2]

[1]Virtual Reality Group, RWTH Aachen University
[2]Jülich Aachen Research Alliance - JARA-HPC

Abstract

Modal sound synthesis is a promising approach for real-time physically-based sound synthesis. A modal analysis is used to compute characteristic vibration modes from the geometry and material properties of scene objects. These modes allow an efficient sound synthesis at run-time, but the analysis is computationally expensive and thus typically computed in a pre-processing step. In interactive applications, however, objects may be created or modified at run-time. Unless the new shapes are known upfront, the modal data cannot be pre-computed and thus a modal analysis has to be performed at run-time.

In this paper, we present a system to compute modal sound data at run-time for interactive applications. We evaluate the computational requirements of the modal analysis to determine the computation time for objects of different complexity. Based on these limits, we propose using different levels-of-detail for the modal analysis, using different geometric approximations that trade speed for accuracy, and evaluate the errors introduced by lower-resolution results. Additionally, we present an asynchronous architecture to distribute and prioritize modal analysis computations.

Categories and Subject Descriptors (according to ACM CCS): H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Audio Output H.5.5 [Information Interfaces and Presentation]: Sound and Music Computing—Signal Synthesis

1. Introduction

In Virtual Reality and many other interactive applications, sound is an important component. To synthesize sounds, the most common approach is to record samples and play them back when a corresponding auditory event occurs. However, especially when applications are highly interactive or contain physically simulated objects, there may be a high number of different auditory event types – different stones, glasses, balls, etc. that possibly collide with one another or the scenery. Furthermore, a single sound per event type may sound repetitive and unrealistic, since in reality differences in material and excitation produce audibly different sounds. For example, when tapping a wine glass at different locations, or when dropping stones of different shapes or sizes, we expect different sounds. Thus, the number of sounds that would require recording can become very high, creating an unreasonable effort.

An alternative approach is physically-based sound synthesis, which calculates the sound of an object based on a physical description. For interactive applications, *modal synthesis* [Adr91, OSG02] offers a suitable solution. By performing a *modal analysis* [Sha96], one can compute the *vibration modes* of an object, which describe characteristic resonant frequencies, decays and deformation shapes. During run-time, these modes are excited by forces acting on the object, and the resulting deformation can be translated into sound. The run-time synthesis is very efficient and can be computed in real-time for many complex objects [RL06, RHK14]. However, a modal analysis is computationally expensive, and is thus usually performed in a pre-processing step.

In order to pre-compute a modal analysis, all objects' geometries and material parameters have to be known upfront. However, in interactive applications, this may not always be the case. For example, existing objects may be deformed by a soft-body simulation or interactively modified by a user, e.g. when scaling or sculpting. Furthermore, completely new objects can be created at run-time, for example when a bottle breaks into several fragments. For such objects, a modal analysis cannot be computed upfront, yet modal data is required for them to produce sounds.

For deforming bodies, it is possible to adjust the modal

DOI: 10.2312/vriphys.20151335

data to a limited degree [Max08], but this cannot handle large deformations. Another option is the use of a database of pre-computed shapes, where the best match for a new or changed object is retrieved [GLM*11]. However, the modal data depends strongly on shape, size and material properties. To avoid approximation that are too coarse, the database would have to be very large, requiring a long initial computation. Furthermore, the modal data can be quite large (∼35 MB for an object with 3000 vertices and 1000 modes). Even though the size may be reduced by discarding modes [vdDKP04, RL06] or by using compression techniques [LAJJ14], a large number of objects in a modal database would still require an unreasonable amount of space.

Therefore, it would be beneficial to be able to perform a modal analysis for new objects at run-time. For interactive applications, however, the time available for a modal analysis to finish is very limited. In the worst case, a new object may start sounding immediately, so that a result should be available within one application frame (∼16 ms for applications running at 60 fps). This becomes even more challenging when multiple objects are created simultaneously. For example, when a window breaks using dynamic fracture computation, several shards are created and immediately collide with each other.

To enable a modal sound synthesis for such a scenario, we have examined the time requirements of a modal analysis, which mainly depends on the complexity of the object's geometry. The main factor of the analysis is the Eigendecomposition of a matrix, for which we compare the performance of different math libraries. This way, we determine how complex a geometry may be to be analyzed in a given time. Based on this, we propose the usage of different sound Levels-of-Detail (LoDs) that trade computation time against accuracy, and examine the impact of the simplification on the resulting sounds. To compute multiple LoDs and to handle several analyses simultaneously within appropriate time frames, we distribute the work among multiple worker instances running locally or on remote clients.

The remainder of this paper is structured as follows. In section 2, we will explain the basics of modal analysis, and provide benchmark results of various Eigendecomposition routines and the whole modal analysis process. The different LoDs that we use are detailed in section 3. Section 4 describes a distributed architecture used to compute the modal analyses. In section 5, we evaluate the error caused by the lower-precision LoDs, followed by conclusions and future work in section 6.

2. Modal Analysis Performance

When objects are excited by forces, the main source of sounds are surface vibrations, which can be modeled physically. While the physical response can be computed explicitly using a time-stepping method [OCE01], this is prohibitively expensive for real-time applications due to the high temporal resolution of audible sounds. Instead, a *modal analysis* can be used to calculate the linearly independent vibration modes of an object. These represent characteristic vibrations with specific frequency, damping and mode shape (see figure 1). Since modes are linearly independent, the total vibration of a surface can be computed as a superposition of the vibration of individual modes. This allows a very efficient synthesis at run-time, where individual modes of an object are excited by interaction forces, and their sound contributions are accumulated to compute the overall sound.

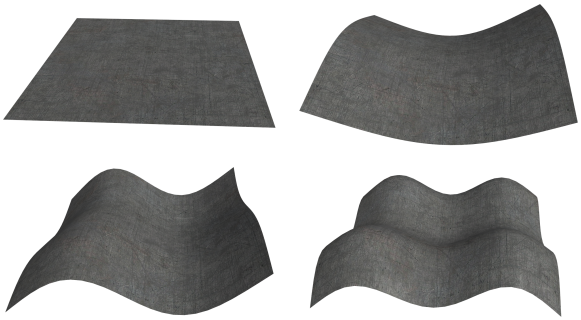

Figure 1: *Different vibration modes of a thin square steel plate.*

2.1. Modal Analysis

A modal analysis is the process of determining an object's modes, either experimentally [AB93, PvdDJ*01], analytically [RF95], or numerically from a physical mesh. While the experimental approach is characterized by a high measurement effort, and an analytic analysis is only possible for few basic shapes, the numerical approach can be used for arbitrary geometries. For these, the dynamics response of an object to external forces is modeled by a linear dynamics system:

$$\mathbf{K}\cdot\mathbf{d}+\mathbf{C}\cdot\dot{\mathbf{d}}+\mathbf{M}\cdot\ddot{\mathbf{d}}=\mathbf{f}_{ext} \tag{1}$$

Here, $\mathbf{d}$ is the displacement of the vertices of the object, $\mathbf{f}_{ext}$ the external force, and $\mathbf{K}$, $\mathbf{M}$ and $\mathbf{C}$ are the stiffness, mass, and damping matrices. They can be assembled using different methods, e.g. from a Spring-Mass System [RL06] or a Finite Element Model [OSG02, PFF*10]. Each of these matrices is a real symmetric matrix of size $3n \times 3n$ for a three-dimensional object with n mesh vertices. Thus, equation (1) forms a system of $3n$ partial differential equations. The goal of a modal analysis is the determination of characteristic vibration modes of this system by transforming it into a system of ordinary differential equations [Sha96]. When using Rayleigh damping $\mathbf{C} = \alpha\cdot\mathbf{K}+\beta\cdot\mathbf{M}$ [SRL45, RYKL13], one can decouple the above equation system by computing the Eigendecomposition of $\tilde{\mathbf{K}} = \mathbf{L}^{-1}\cdot\mathbf{K}\cdot\mathbf{L}^{-T}$. Here, $\mathbf{M} =$

	Eigen	Meschach	LAPACK						ARPACK	
#verts	SelfAdjoint	symmeig	ssyev	ssyevd	ssyevx	ssyevx L	ssyevr	ssyevr L	regular L	shift-invert L
32	0.001	0.167	0.004	0.004	0.004	0.004	0.004	0.003	0.004	0.005
64	0.006	1.148	0.024	0.023	0.024	0.023	0.020	0.020	0.020	0.025
128	0.058	9.698	0.174	0.149	0.175	0.132	0.127	0.131	0.132	0.159
256	0.530	116.128	1.413	1.175	1.422	0.956	0.950	0.961	0.908	0.980
512	3.721	887.559	11.829	9.632	11.935	9.256	8.474	9.055	6.569	6.586
1024	36.776	12345	87.278	—	87.428	61.526	56.067	63.037	76.886	72.419
2048	364.075	—	696.196	—	704.293	519.105	456.138	520.160	691.382	576.269

Table 1: *Benchmark results for Eigen, Meschach, and LAPACK packages and different computation routines for different problem sizes. Times are given in seconds, for Eigendecompositions of matrices of size $3n \times 3n$ for meshes with n vertices. If an L is added to a method name, only eigenvalues in the audible frequency range are computed.*

$\mathbf{L} \cdot \mathbf{L}^T$ is the Cholesky decomposition of the mass matrix, which is easy to compute when using a diagonal lumped-mass matrix. Using the Eigendecomposition $\tilde{\mathbf{K}} = \mathbf{V} \cdot \mathbf{D} \cdot \mathbf{V}^T$ and substituting $\mathbf{r} = \mathbf{V}^{-1} \cdot \mathbf{d}$ and $\mathbf{g} = \mathbf{V}^{-1} \cdot \mathbf{L}^{-1} \cdot \mathbf{f}_{ext}$, the equation system is transformed into *modal space*:

$$\mathbf{D} \cdot (\mathbf{r} + \alpha \cdot \dot{\mathbf{r}}) + \ddot{\mathbf{r}} + \beta \cdot \dot{\mathbf{r}} = \mathbf{g} \tag{2}$$

Equation (2) has been diagonalized, because the only occurring matrix $\mathbf{D}$ is diagonal. This way, the system is decoupled into $3n$ ordinary differential equations. Each of these equations represents a damped harmonic oscillator, i.e. a mode, and can be solved analytically. The corresponding eigenvectors represent the mode shape, and from the eigenvalues and material parameters one can derive the frequency and damping of the mode.

One advantage of modal analysis is that it only requires an object's geometry and physical material parameters. For many interactive applications like Virtual Reality, a visual geometry already exists and can be used directly for sound synthesis, e.g. using a Spring-Mass System [RL06]. However, a Spring-Mass System cannot model bending strain well, but this is important for shell vibrations. Thus, we use a tetrahedral Finite Element Method. While this approach provides better results, it requires a tetrahedral volume representation of the objects. Construction of a solid volume mesh is challenging and time-consuming, especially when using the visual geometry as basis. For modal sound synthesis, a common approach is to restrict oneself to only use thin-shell meshes [CAJ09, RL06, Zam12]. These can be constructed from surface meshes that are non-manifold and non-watertight, and shells commonly produce more recognizable sounds. In our case, we construct a thin-shell tetrahedral mesh from a surface mesh with m vertices by extruding each triangle along its normal, forming a physical mesh with $2m$ vertices.

2.2. Eigendecomposition Benchmark

The Eigendecomposition is the most time-consuming part of the modal analysis, and its complexity is usually $O(n^3)$ for n vertices. While there are algorithms with a lower asymptotic complexity [DDH07], in practice these are not faster for small problem sizes as used here. Thus, the computation time quickly becomes prohibitive for an increasing vertex count. To determine limits for resolutions that are still suitable for a run-time analysis, we benchmarked different math packages and their routines for the symmetric real Eigendecomposition. Specifically, we examined LAPACK (version 3.4.2, default BLAS), the optimized LAPACK and sparse matrix routines of the Intel Math Kernel Library MKL (version 11.0), Eigen (version 3.2.0), and Meschach (version 1.2). Since the matrix $\tilde{\mathbf{K}}$ is inherently sparse, it is also possible to use sparse matrix routines, for which we tested MKL's `feast` routines as well as ARPACK (using ARPACK++ v1.2 and SuperLU v4.3). Furthermore, we used CULA (version R17) to perform the decomposition on a GPU. LAPACK, MKL, and CULA provide multiple Eigendecomposition routines, which follow the LAPACK naming convention: `ssyev` and `ssyevx` use a traditional QR decomposition approach, `ssyevd` uses a divide-and-conquer approach [Cup80], and `ssyevr` uses the relatively robust representation method [PD00].

Although all tested decomposition routines also provide double precision variants, we found that single precision computations only produce a neglectable error. Thus, we only provide results for single-precision computations here. Double-precision computations require about 1.3 to 2 times longer (depending on the problem size and used method), which in the worst case allows for ∼20% smaller vertex counts than single-precision computations.

For the modal analysis, usually all eigenvalues and eigenvector are computed. However, since the eigenvalues equal the square of the undampened natural frequencies of the modes, one can limit the query so that only modes in the audible range (20 Hz – 22 kHz) are computed. Since this can speed up the computation, we also benchmarked these variants. For the objects in this benchmark, the material properties were chosen such that 50% of the computed modes were in the audible range. In real applications, however, this fraction can vary, because an increase in geometric resolu-

	MKL							CULA		
#verts	ssyev	ssyevd	ssyevx	ssyevx L	ssyevr	ssyevr L	feast L	ssyev	ssyevx	ssyevx L
32	0.004	0.001	0.004	0.002	0.002	0.002	0.135	0.026	0.026	0.012
64	0.024	0.005	0.025	0.007	0.007	0.007	0.575	0.059	0.059	0.032
128	0.176	0.025	0.176	0.036	0.036	0.039	2.816	0.142	0.150	0.088
256	1.141	0.160	1.075	0.232	0.209	0.231	14.389	0.476	0.500	0.341
512	6.372	1.635	6.512	2.130	1.945	2.239	71.699	1.657	1.721	1.727
1024	67.442	9.405	72.555	13.899	11.993	14.488	353.991	7.139	7.146	13.784
2048	453.372	86.395	451.400	144.718	102.893	152.114	—	29.204	29.131	121.693

Table 2: *Benchmark results for MKL and CULA packages and different computation routines for different problem sizes. Times are given in seconds, for Eigendecompositions of matrices of size $3n \times 3n$ for meshes with n vertices. If an L is added to a method name, only eigenvalues in the audible frequency range are computed.*

tion mostly adds high-frequency modes so that a higher percentage would be discarded, while it is common for low-resolution objects to have very few inaudible modes.

All results were computed on a PC with two Intel Xeon X5650 (each with 6 cores, 2.67 GHz) and 24 GB RAM. The test program was compiled with gcc 4.8. The CULA benchmark was performed on an Nvidia GeForce GTX 480.

Since the decomposition time can vary depending on the matrix structure, we decomposed the matrices corresponding to 15 different physical meshes, and report the maximum time required. The results for the different libraries and routines are listed in table 1 and table 2. As can be seen, the MKL `ssy*` routines are considerably faster than other CPU-based algorithms. It is also notable that the fastest MKL routine for a full decomposition with all eigenvalues (`ssyevd`) outperforms the variants calculating a limited range of eigenvalues. Only for complex geometries and a high percentage of inaudible modes, limiting the modal range may improve the performance.

For the presented problem sizes, sparse matrix routines show a worse performance that dense solvers. However, while not beneficial for a run-time synthesis, sparse decomposition is still important when pre-processing complex geometries, where the high memory requirements of dense matrix storage would exceed memory limits.

The presented results were computed using a single CPU thread, but MKL also supports multi-threaded computations. Figure 2 shows the performance gain when using multiple threads for problems of different size. As can be seen, for small matrix sizes, the performance gain from using multi-threading is rather lower. For larger matrices, however, a notable improvement is achieved, significantly reducing the required computation time.

Using the GPU-based CULA library, the computation is significantly faster than MKL for large matrices, but slower for small ones. This was to be expected, because GPU computations typically come with a data transfer overhead, and require a high level of parallelism to utilize the full power of the GPU. On the used hardware, the turning point where CULA's `ssyev` outperforms MKL's `ssyevd` (single-threaded) is for geometries with ∼520 vertices (requiring around 1.7 s) Consequently, we chose MKL's `ssyevd` as the primary routine for Eigendecomposition especially for smaller problems, while CULA's GPU-based Eigendecomposition provides a good alternative for larger matrices if a powerful GPU is available.

In addition to the presented methods, we also examined packages that use a distributed cluster for the computation of Eigendecompositions. Specifically, we looked at ScaLAPACK and SlepC, which are both MPI-based. These libraries are tailored to solving large matrices. However, we found that for a run-time modal analysis, where relatively small matrices are used and many eigenvalues and eigenvectors have to be computed, they are not faster than single-node algorithms.

2.3. Complete Modal Analysis

The Eigendecomposition is the most computationally expensive operation of the modal analysis. Regarding only the Eigendecomposition, objects with ∼94 physical vertices (corrsponding to ∼47 surface vertices) can be handled within an application frame of 16 ms on the examined hard-

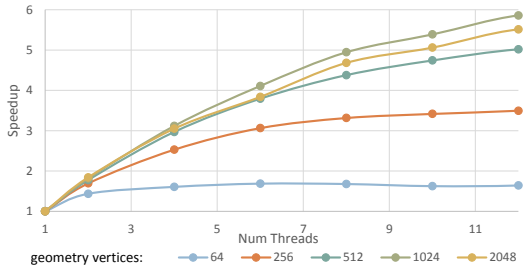

Figure 2: *Multi-threaded performance of MKL `ssyevd` for different numbers of threads and physical mesh vertices.*

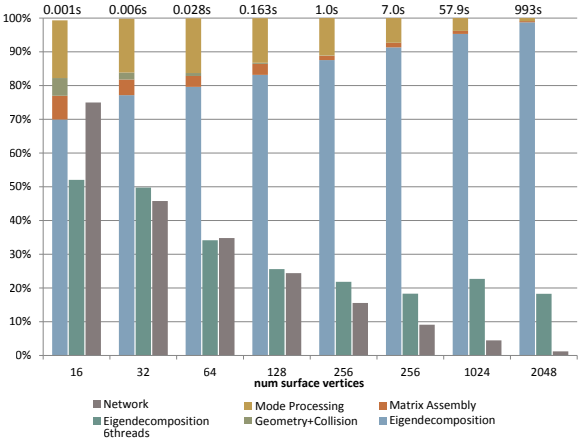

Figure 3: *Relative computation time required by different components of the modal analysis for meshes with different surface vertex counts.*

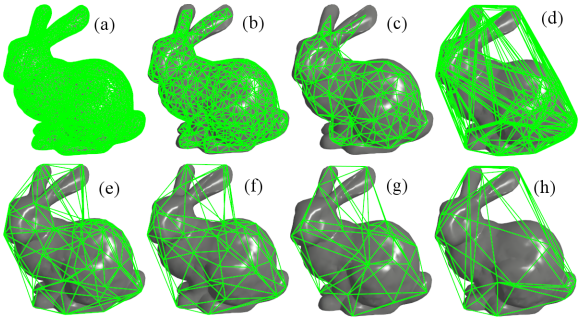

Figure 4: *Different Levels-of-Detail of a Stanford bunny: (a) full geometry, reduced to (b) 400 and (c) 100 vertices, (d) convex hull, Shrink-Hull with (d) 66 (e) 38 and (f) 26 vertices, (h) 26-Hull.*

ware. However, a modal analysis has to perform additional steps to allow a real-time sound synthesis. As a first step, the thin-shell geometry is prepared, and the mass and stiffness matrices have to be assembled. After the decomposition, the results have to be further processed by storing mode and excitation data in suitable run-time data structures. Additionally, inaudible modes are removed, gains are pruned, and acoustic transfer factors are computed. Furthermore, a collision data structure is built in order to map impact forces to mode excitation.

Figure 3 shows the relative computation time for objects with different vertex counts (Stanford bunnies of different resolutions). The modal analysis was performed using MKL `ssyevd` using a single thread, and using 6 threads for comparison. As can be seen, even for small problems the Eigendecomposition time exceeds the time required by other component, and quickly becomes the dominating factor for raising vertex counts. When computing a modal analysis locally, the 16 ms limit allows for an analysis of geometries with approximately 45 surface vertices. Since the Eigendecompositon time varies with the geometry, however, some problems may require longer. From the maximal computation times determined by the Eigendecomposition benchmark, we can conclude that geometries with 40 surface vertices can be analyzed within the given time.

Optionally, one can perform the computation on a remote PC. In this case, the query and result have to be serialized and transmitted over the network. The network transfer duration (using Gb-LAN) is also listed in figure 3 . The query includes the full geometry data, and the result contains the gain matrix with $3 \cdot \#vertices \times 3 \cdot \#modes$ entries. Thus, the network transfer can require a significant amount of time, which is relevant especially for very small objects. Thus, remote computations should only be performed for larger objects, where the impact is less severe and results may take longer.

3. Level-of-Detail Analysis

When new objects are created at run-time, or existing ones are changed, it is necessary to calculate the new modal data within the time constraints of an interactive application. Thus, we have to ensure that modal analysis results are available fast enough. For this, we use different *Levels-of-Detail (LoDs)* (see figure 4). By reducing the complexity of the geometric mesh, we can compute the modal analysis faster. However, the resulting sounds have a lower accuracy, so that higher-quality modal results should be computed, too. Therefore, lower-quality approximations are computed first, and are successively replaced by better results once their slower analyses finish. For this, we propose different LoDs that trade precision for computation time.

While we have to differentiate between physical vertices and surface vertices of a mesh, these have a fixed relation of two-to-one for the used thin-shell meshes. In this section, all vertex counts will refer to the surface mesh.

3.1. Full Mesh

As highest-quality LoD, we use the mesh with full resolution. This mesh does not necessarily have to be the same as the input mesh, but may be refined or remeshed to meet uniformity and edge length criteria. Full-resolution meshes typically consist of thousands of vertices.

3.2. Mesh Simplification LoDs

As has been shown in section 2.3, the required time for a numerical modal analysis mainly depends on the number of vertices. In order to gain faster results, one can use approximated meshes with a controllable amount of vertices.

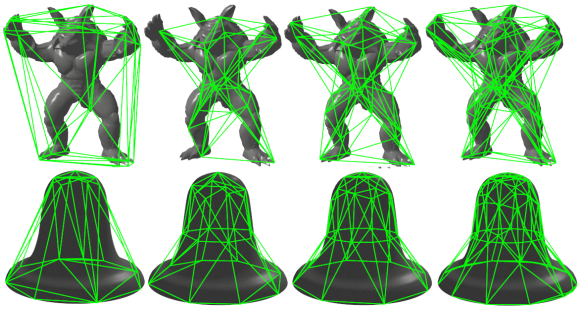

Figure 5: *Stanford armadillo and bell with different hull approximations. From left: 26-Hull, 26-Shrink-Hull, 38-Shrink-Hull, 66-Shrink-Hull.*

The common approach to reduce the amount of vertices is the usage of optimization-based mesh simplification algorithms [CMS98]. In our framework we create several LoDs that simplify the mesh up to a target vertex count. Doubling the vertex count increases the computation time by a factor of approximately eight, so we chose to compute simplifications with $100 \cdot 2^n$ vertices (up to a maximum of half the vertices of the original mesh).

However, the mesh simplification requires some time, and – depending on the initial complexity of the geometry – can be too time-consuming to ensure a completion within the given time constraints. For example, reducing a Stanford bunny mesh with 8000 vertices to 512 vertices requires 82 ms using `vtkQuadricDecimation` of the Visualization Toolkit (version 6.1.0), and 846 ms using `Surface_mesh_simplification` of CGAL (version 4.4). Thus, simplification is suitable for higher-quality approximations, but may be too slow to calculate an initial approximation.

3.3. Hulls

Optimally, a first numerical analysis should be available fast enough to generate sounds in the next application frame (i.e. within 16 ms). This time span allows for an analysis of a geometry with approximately 40 surface vertices (see section 2.3). Thus, a suitable approximation should reliably reduce the vertex count to this limit, while producing reasonable geometries and being computationally efficient.

One type of approximations that are commonly used in collision detection are bounding volumes. Simple volumes, like bounding spheres or boxes, would only coarsely approximate the geometry, but a convex hull is a reasonable fit for many objects. However, the vertex distribution of a convex hull is often very irregular, with many vertices around round features and large faces along concave areas (see figure 4 (d)). Furthermore, one cannot control the target vertex count for a convex hull, which in the worst case may equal the original mesh (e.g. for a sphere).

k-DOPs [KHM*98] are another suitable option. They calculate a bounding volume from the convex intersection of k half-planes with fixed normals. Using k-DOPs to calculate a hull provides an upper limit on the number of vertices. However, the vertex count may still vary significantly, and the extraction of a mesh from the half-plane formulation is complicated. Instead, we designed different methods to build simplified hulls.

k-Hull Since bounding volumes are often used for collision detection, it is important that they fully envelope the volume. To approximate a mesh for a modal analysis, however, this property is not required so that the reduced mesh may intersect the original mesh. This can even be beneficial, because envelopes overestimate the objects' volume, which usually produces modes with lower frequencies.

We use a hull-construction approach that combines aspects of k-DOPs and convex hulls, which we call k-Hulls. These are formed by at most k convex vertices of the original mesh. To compute these vertices, we define a set of $\frac{k}{2}$ uniformly distributed axes. For each axis, we determine the two vertices whose projection onto the axis is minimal or maximal. These points form a convex set, so we can efficiently construct a mesh from them. The k-Hull can also be computed quickly: a mesh with more than 10000 vertices can be converted to a 26-Hull in $\sim$0.4 ms.

k-Hulls provide a mesh similar to a convex hull, but with fewer vertices so that some features are cut off. They offer a controllable vertex count and a slightly better vertex distribution. However, they still suffers from several problems (see figure 4 (h) and figure 5). Due to the convexity of the result, the enclosed volume can include empty regions inside concavities and thus be significantly larger than the original mesh, especially if the object has protruding features like the armadillo's arms and legs. Furthermore, faces are still frequently badly shaped, and the vertices can cluster at feature points, e.g. the ears of the bunny or the hands of the armadillo. Lastly, because the k-Hull only uses vertices from the original mesh, it cannot subdivide large faces, potentially creating large unsampled areas.

Shrink-Hull To reduce the problems of the k-Hulls, we propose another alternative, which we call Shrink-Hull. In a first design iteration, it was constructed by emitting k rays from the object's center-of-gravity and determining their intersection points with the mesh. The outermost intersection point per ray is used as vertex for the approximate mesh. The result is a (possibly non-convex) mesh with exactly k vertices, and can be seen as an enclosing sphere whose vertices have been moved towards the center until they touch the original mesh. The resulting mesh has a more uniform vertex distribution than the k-Hull, and is not restricted to the original mesh sampling. While the k-Hull tends to overestimate the volume, the Shrink-Hull approximation has the opposite problems: while sometimes the size is similarly overestimated,

more often features that are not within the sampling resolution are lost, and thus the resulting object is often smaller than the original mesh. Furthermore, many features are not sampled by the rays and are thus lost, which makes this basic approach problematic.

To enhance the Shrink-Hull further, we extended it to first search for extremal points along each ray like for k-Hulls, but limited to points within a small cone (with resolution-dependent angle) around the axis. This prevents the choice of identical or similar extremal points for pronounced features, e.g. the bunny's ears, as it occurred for k-Hulls. If no suitable extremal point is found, a ray cast is used to determine an intersection point. If neither an extremal or intersection point is found, the corresponding vertex is discarded – thus, Shrink-Hulls can even handle holes in an object, like the bottom of the bell, and generally produces a better vertex distribution than the k-Hull (see figure 5).

The Shrink-Hull can better follow non-convex regions, like the curve of the bell or the waist of the armadillo, than the k-Hull. Since fewer concave regions are included, the volume is usually much closer to the original mesh, and the vertices are more evenly sampled. All in all, the Shrink-Hull gives a good low-resolution approximation for most meshes. The explicit ray casts slightly increase construction cost, which requires $\sim$1 ms to build a 38-Shrink-Hull for an object with 10000 vertices. For the bell example, the full analysis of Shrink-Hulls with 26, 38, and 66 vertices require 4.3 ms, 7.4 ms, and 25.6 ms. Thus, we use a Shrink-Hull with 38 vertices as lowest-quality numerical analysis, and another one with 66 vertices for a more precise result. If required, we also allow computing a 26-Shrink-Hull, e.g. when using a slow processor or if a high number of modal analyses are requested simultaneously.

3.4. Analytic Solution

The numerical analysis of a single Shrink-Hull approximation can be computed fast enough to allow a sound synthesis in the next frame. However, there may still be situations where this is too slow, e.g. when multiple objects are created simultaneously and immediately start to produce sounds. For such scenarios, we require an even faster LoD.

For some simple geometric shapes, like beams, membranes, and plates, modes can be computed analytically [RF95]. Such an analytic solution can be computed very fast, but due to the limited shapes available, strong approximations are necessary. In our implementation, we approximate objects using rectangular plates. Flat objects are modeled using a single palte, while thicker objects are approximated by six plates forming a box. To align the plates with the object, one could use the axis aligned bounding box of the object, but this would only coarsely relate to the geometry, depend on its orientation and often significantly overestimate the size. Instead, we use a principal component analysis of the object's vertex distribution to determine the orientation of the box, similar to the approximate computation of oriented bounding boxes [GLM96]. The extents are determined from the variance of the vertices along the axes.

For each plate, we analytically compute the first 100 modes. Both the box alignment and the analytic mode computation can be performed very fast ($\sim$0.2 ms for an object with 10000 vertices). This allows us to compute the analytic result on-the-fly whenever sound synthesis is requested for objects for which no numerical modal result has been computed yet.

4. Distributed LoD Analysis

Using the different LoDs, it is now possible to calculate incrementally better results for the modal analysis. For the use-case of interactive sound synthesis, the modal data is required on a system (the application host) that performs the run-time modal synthesis, which detects interaction forces, excites modes, and computes the resulting output. Since the application host usually also handles other application features, like visual rendering and physical simulation, only limited resources are available for the modal analysis computation. Thus, large modal analyses would take too long, especially if multiple analyses are performed simultaneously. Also, long-running analyses of high-quality LoDs could delay computations of high-priority initial LoDs. Therefore, we designed an architecture that computes different modal LoDs asynchronously using multiple analysis workers, both in concurrent threads on the application host and on remote clients.

We define multiple priority classes, and assign a set of workers to each of these classes. In our setup, we found three priority classes to be sufficient. High-priority jobs process the hull approximations with 26, 38, or 66 vertices. Medium-priority jobs compute simplified and full meshes with at most 512 vertices. Low-priority jobs calculate all analyses with more than 512 vertices. Optionally, depending on the system configuration, lower-priority workers can also compute jobs of a higher priority.

For each priority class, we assign a specific set of workers. Local workers run in a concurrent thread on the application host, while remote workers calculate an analysis on PCs connected over via network. While local workers can only use limited resources due to requirements of the main application, dedicated remote workers can use more processing power, but suffer from network transfer overhead, especially for smaller jobs (see section 2.3). Thus, small, high-priority jobs should be computed by local workers, and larger jobs by remote workers.

In our setup, we use one primary PC and four worker nodes, each with two six-core CPUs. On the primary PC we use four to eight concurrent worker threads for high-priority

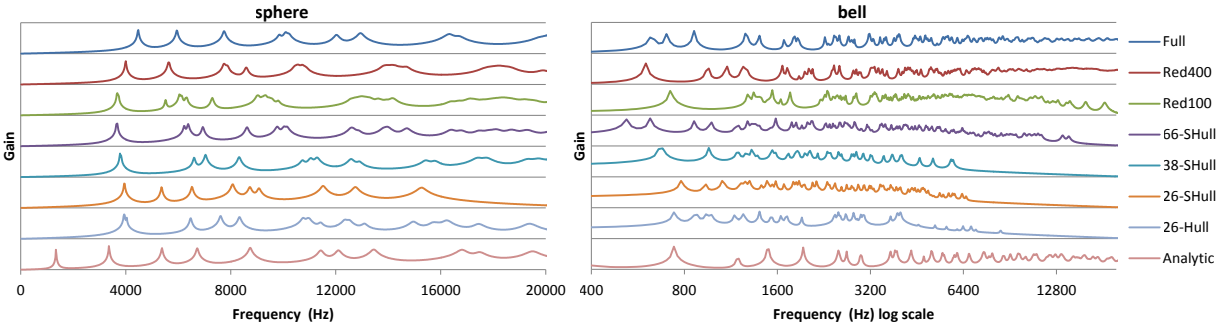

Figure 6: *Comparative frequency spectrums of sounds produced by different LoDs of a sphere and a bell.*

tasks, depending on the resource requirements of other application components. Each of these workers uses a single-threaded Eigendecomposition because small jobs show only a minor speedup from additional threads. On each remote node, two worker processes are running, using one CPU with 6 MKL threads each. Of the eight remote workers, each four are assigned to the medium- and low-priority tasks. With this setup, we can compute analyses of multiple hull approximations within a single application frame, while several larger jobs can be handled remotely. If even more performance is required, one can utilize the GPUs of the remote PCs, or add more nodes.

If only a single PC is available, one can still achieve acceptable performance. In this case, the majority of the threads should be reserved for high-priority tasks, and the medium and low workers' thread priority should be reduced to ensure that high-priority tasks are started without delay. While this delays more precise results, it ensures that at approximate results are available as soon as possible.

When a new modal result arrives for an object, it replaces the currently used result. However, if the object is currently excited, we have to prevent discontinuities in the generated sound. Thus, we maintain the previous result until its excitation has receded, but all following excitation forces are applied to the new result.

5. LoD Approximation Quality

The LoDs reduce the geometric resolution to allow an interactive modal analysis. However, low-resolution geometries produce different results than high-resolution ones. While a certain change is tolerable, the general characteristics of the sound should not change too much.

We examined different meshes to determine the influence of the geometry resolution. This showed that – even when no features are lost – reducing the resolution of a mesh produces a shift towards higher frequencies. A similar observation has been reported by [OSG02], and the authors proposed to simply adjust material parameters to counter the effect. This, however, would require manual adjustment for each object an LoD, which does not work well for automatic analyses with multiple resolutions.

Especially when a low shell thickness is combined with large triangles, the resulting extruded tetrahedra are badly shaped, resulting in very high deviations from the original result. To determine the extent of the frequency shift, we examined different models at decreasing resolutions. This showed that the frequency shift depends on the edge ratio of the tetrahedra, which led us to an implementation of a correction formula: When assembling the global stiffness matrix, we scale the tetrahedron stiffness matrix by $\frac{1}{sqrt(r)}$, where r is the ratio between the average edge length and the shortest edge of the tetrahedron. While not fully negating the frequency shift, it strongly reduces the impact.

Apart from the frequency shift, lower-resolution LoDs have fewer features and potentially missing holes and different volumes, which may alter the produced sounds further. To evaluate the introduced error, we compare different example geometries and their LoDs. Figures 6 and 7 show the frequency spectrums produced by different LoDs of a sphere (diameter 250 mm, shell thickness 10 mm), a bell (base diameter 638 mm, shell thickness 10 mm), a Stanford bunny (height 154 mm, shell thickness 5 mm), and a Stanford armadillo (height 151 mm, shell thickness 1 mm), all using material parameters for steel (Young Modulus 200 GPa, Poisson ratio 0.29, damping parameters $\alpha = 3 \cdot 10^{-8}$ and $\beta = 5$). Sounds produced by the examples can be found in the supplemental material.

For the sphere, all numerical results produce similar sounds, with only small variations in the frequency and distribution of modes. Due to the simple, convex geometry of the sphere, the non-analytical LoD geometries are also accurate, uniformly sampled sphere representations that only vary in resolution, which represents a best-case scenario. As a more complex example, the larger and thicker bell produces lower-frequency sounds, which shows a general problem of the low-resolution approximation: Fewer vertices provide fewer degrees-of-freedom, and thus fewer modes.

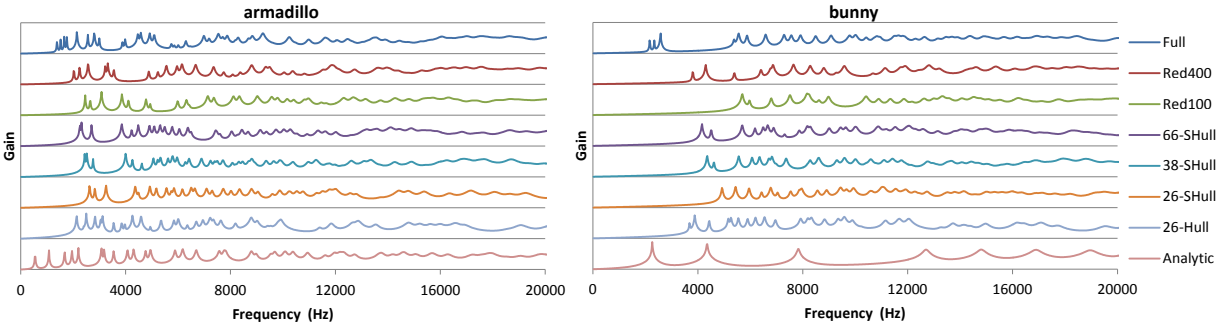

Figure 7: *Comparative frequency spectrums of sounds produced by different LoDs of a Stanford bunny and a Stanford armadillo.*

Additional modes typically model higher frequencies, so that for *k*-Hull and Shrink-Hull LoDs, the high-frequency spectrum shows no peaks. While this slightly alters the timbre, the effect is rather small since high-frequency modes are strongly damped. A more significant effect that produces audible differences in the bell sound is the frequency shift, as well as the less pronounced base modes which are characteristic for a bell's sound. Here, it is notable that the full (visual) mesh shows a worse result than the simplified meshes due to a less uniform triangulation, which emphasizes the fact that full-resolution meshes should be further processed to match high mesh quality criteria.

Representing more complicated, non-convex geometries with many features, we analyzed the bunny and armadillo. When comparing the full-resolution model with the simplified variants, we again see the aforementioned shift towards higher frequencies in general, but even for the armadillo with a very thin shell (1 mm), it is acceptably small due to the correction factor. A more significant impact is the loss of features. For the armadillo, the full and simplified meshes show a cluster of low-frequency modes, representing vibrations of the limbs, that are separated from the modes representing body vibrations. The *k*-Hull approximation does not show this separation, although the significantly overestimated volume of the hull leads to a general shift of all modes towards lower frequencies. The Shrink-Hull, on the other hand, manages to coarsely sample the extremities, thus producing similar – although fewer – limb modes separated from the main torso modes.

The Stanford bunny shows the highest error for the LoDs. Like the armadillo's limbs, its ears produce distinct low-frequency modes, but neither *k*-Hull nor Shrink-Hull manages to capture these adequately. Even when using mesh simplification, it reduces the sampling and volume of the ears, so that they are less separated from the torso modes.

For all tested objects, the different numerical results produce audible differences, but the general characteristics of the sounds are quite similar, so that the synthesized sounds still appear plausible for the corresponding object and material. When examining the analytic results, however, we see significant deviations from the numerical results. Because the object is only very coarsely approximated by plates, this was to be expected, and emphasizes that analytical results should only be used as a fall-back if a first numerical result cannot be provided in time.

6. Conclusion

We have presented a system to calculate modal analyses at run-time, in order to allow a modal sound synthesis for dynamically added or modified objects. We have compared the performance of different Eigendecomposition packages, and determined the computation time for modal analyses of objects with different complexity. Based on these limits, we proposed different LoD approximations that provide successively more accurate approximations. By using Shrink-Hull approximations, a first approximation can be computed in $<$16 ms. Higher-resolution analyses of the full geometry and several simplified meshes are computed, too, and replace lower-resolution LoDs as soon as they finish. If the lowest LoDs still takes too long (e.g. when many objects are created simultaneously), we use a coarse analytic solution.

To ensure that modal results are available as soon as possible, we presented a system using multiple workers, both in concurrent threads and on remote PCs connected over a network. This way, sounding objects can be added to dynamic interactive environments during run-time and can produce approximate sounds immediately, while the results become more accurate when longer computations are finished.

The presented system uses different approximations of the geometry, and a correction factor was defined to reduce the frequency shift of lower geometry resolutions. The quality of the results was evaluated by examining the frequency spectrums of example objects, which showed that lower resolutions lead to small frequency shifts, as well as artifacts from missed features and volume deviations. The examples indicate that the numerical approximations sound similar to the

full resolution, and provide a meaningful result while computing the higher-quality results. The analytic results, however, produce significantly different sounds, and should only be used as last option if no initial numeric LoD can be computed in time.

In Virtual Environments, sounds do not necessarily have to be exactly realistic, but only need to be plausible – therefore, the results of the numeric LoDs seem to work reasonably well. For future work, we are planning to perform a further evaluation of the legitimacy of these approximation in the form of a quantitative user study comparing recorded sounds and sounds that have been synthesized from different LoDs.

Another interesting approach that can be examined in the future is the consideration of latency tolerances. By delaying the sound produced by a newly created modal object, one could increase the resolution of the lowest LoD, yielding a better result. This would reduce the approximation error at the cost of an increased audio latency.

References

[AB93] ALLEMANG R. J., BROWN D. L.: Experimental modal analysis. *Handbook on Experimental Mechanics* (1993), 635–750. 2

[Adr91] ADRIEN J.-M.: The missing link: Modal synthesis. In *Representations of Musical Signals* (1991), MIT Press, pp. 269–298. 1

[CAJ09] CHADWICK J. N., AN S. S., JAMES D. L.: Harmonic Shells: A practical nonlinear sound model for near-rigid thin shells. *ACM Transactions on Graphics 28*, 5 (2009), 119:1–119:10. 3

[CMS98] CIGNONI P., MONTANI C., SCOPIGNO R.: A comparison of mesh simplification algorithms. *Computers & Graphics 22*, 1 (1998), 37–54. 6

[Cup80] CUPPEN J.: A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik 36*, 2 (1980), 177–195. 3

[DDH07] DEMMEL J., DUMITRIU I., HOLTZ O.: Fast linear algebra is stable. *Numerische Mathematik 108*, 1 (2007), 59–91. 3

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the ACM Conference on Computer Graphics* (1996), pp. 171–180. 7

[GLM*11] GLONDU L., LEGOUIS B., MARCHAL M., DUMONT G., ET AL.: Precomputed shape database for real-time physically-based simulation. In *VRIPHYS* (2011), pp. 47–54. 2

[KHM*98] KLOSOWSKI J. T., HELD M., MITCHELL J. S., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of k-DOPs. *Visualization and Computer Graphics, IEEE Transactions on 4*, 1 (1998), 21–36. 6

[LAJJ14] LANGLOIS T. R., AN S. S., JIN K. K., JAMES D. L.: Eigenmode compression for modal sound models. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 40. 2

[Max08] MAXWELL C.: *Sound Synthesis from Shape-Changing Geometric Models*. PhD thesis, University of California at Berkeley, 2008. 2

[OCE01] O'BRIEN J. F., COOK P. R., ESSL G.: Synthesizing sounds from physically based motion. In *Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, pp. 529–536. 2

[OSG02] O'BRIEN J. F., SHEN C., GATCHALIAN C. M.: Synthesizing sounds from rigid-body simulations. In *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation* (2002), pp. 175–181. 1, 2, 8

[PD00] PARLETT B. N., DHILLON I. S.: Relatively robust representations of symmetric tridiagonals. *Linear Algebra and its Applications 309*, 1 (2000), 121–151. 3

[PFF*10] PICARD C., FRISSON C., FAURE F., DRETTAKIS G., KRY P. G.: Advances in modal analysis using a robust and multiscale method. *EURASIP Journal on Advanced Signal Processing 2010* (2010), 7:1–7:12. 2

[PvdDJ*01] PAI D. K., VAN DEN DOEL K., JAMES D. L., LANG J., LLOYD J. E., RICHMOND J. L., YAU S. H.: Scanning physical interaction behavior of 3D objects. In *Conference on Computer Graphics and Interactive Techniques* (2001), pp. 87–96. 2

[RF95] ROSSING T. D., FLETCHER N. H.: *Principles of Vibration and Sound*. Springer, 1995. 2, 7

[RHK14] RAUSCH D., HENTSCHEL B., KUHLEN T.: Efficient modal sound synthesis on GPUs. In *IEEE VR Workshop: Sonic Interaction in Virtual Environments (SIVE)* (2014), pp. 13–18. 1

[RL06] RAGHUVANSHI N., LIN M. C.: Interactive sound synthesis for large scale environments. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (2006), I3D '06, pp. 101–108. 1, 2, 3

[RYKL13] REN Z., YEH H., KLATZKY R., LIN M. C.: Auditory perception of geometry-invariant material properties. *IEEE Transactions on Visualization and Computer Graphics 19*, 4 (2013), 557–566. 2

[Sha96] SHABANA A.: *Theory of Vibration II: Vibration of Discrete and Continuous Systems*, vol. 2. Springer, 1996. 1, 2

[SRL45] STRUTT J. W., RAYLEIGH B., LINDSAY R. B.: *The Theory of Sound*. MacMillan, 1945. 2

[vdDKP04] VAN DEN DOEL K., KNOTT D., PAI D. K.: Interactive simulation of complex audiovisual scenes. *Presence: Teleoperators and Virtual Environments 13*, 1 (2004), 99–111. 2

[Zam12] ZAMBON S.: *Accurate Sound Synthesis of 3D Object Collisions in Interactive Virtual Scenarios*. PhD thesis, Universita degli Studi di Verona, 2012. 3