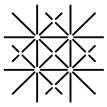


University of Basel

Towards an All-Purpose, Content-Based Multimedia Information Retrieval System

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems Group (DBIS)
https://dbis.dmi.unibas.ch

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: Luca Rossetto, M.Sc.

Ralph Gasser
ralph.gasser@stud.unibas.ch

July 7th 2017

Acknowledgements

First and foremost I would like to thank Prof. Dr. Heiko Schuldt for giving me the opportunity to realise this project. A special thanks goes to Luca Rossetto for trusting me with the Cineast codebase and for being an extraordinary supervisor this entire time. I greatly appreciate all the support, the feedback and input I received. I undoubtedly learned and profited a lot!

Furthermore, I would like to acknowledge Andrea Schäfer and Arno Schneuwly for their invaluable support in terms of proofreading and reviewing. Moreover, Ivan Giangrecco deserves thanks for all the patience he had when helping me with ADAMpro. And last but not least, I want to express my gratitude to all who participated in the evaluation. Thank you for your patience, your impressive drawing skills and your merciless relevance judgements.

Now obviously, there are so much more who would deserve some form of acknowledgement. So collectively, a huge and heartfelt "thank you" to all my friends and especially my boyfriend, who supported me over the past six months.

Abstract

Digital media in its various forms is ubiquitous and plays a crucial role in many different domains ranging from culture and arts to science and our everyday lives. As our media collections grow at an ever-increasing pace — not only in terms of sheer volume but also in terms of variety — management of those collections and making effective use of the knowledge they contain becomes a daunting endeavour. One of the major obstacles is satisfying a particular information need; that is, retrieving an item of interest from such a collection. Traditionally, this is facilitated by a keyword-based search, which requires prior, manual annotation of the data. This approach, however, does not scale well with the size of the collections as the annotation process is bound to human labour.

In this master's thesis we present a second iteration of *Cineast* — a content-based multimedia information retrieval engine. As opposed to the keyword search approach implemented by most media management systems, Cineast makes direct use of the file's content to facilitate different types of similarity search, such as Query-by-Example or Query-by-Sketch, for and across different media types — namely, images, audio, video and 3D models. We explore different feature descriptors for content-based music retrieval and content-based 3D model retrieval. Furthermore, we introduce *Vitrivr NG*, a web-based user interface that enables easy-to-use, multi-modal retrieval from and browsing in mixed media collections.

To the best of our knowledge, the full stack of Cineast and Vitrivr NG is unique in that it is the first retrieval system that integrates support for four different types of media. As such, it paves the way towards an all-purpose, content-based multimedia information retrieval system.

Symbols and abbreviations

Symbols

$\mathbf{v}, \mathbf{M}$ Vectors and matrices are printed in bold

$\mathbf{v}^T, \mathbf{M}^T$ Transpose of a vector $\mathbf{v}$ or matrix $\mathbf{M}$

i The imaginary unit, i.e. $\sqrt{-1}$

a^* The complex conjugate of $a \in \mathbb{C}$

$\overline{a}$ The sample mean

δ_{ij} The Kronecker delta

$\tilde{x}, \tilde{f}$ The tilde indicates a discrete function

L^p The Minkowski distance of order p

L^1 The Manhattan distance

L^2 The Euclidian distance

Z_n^m Zernike polynomial of radial degree n and azimuthal degree m

R_n^m Radial polynomial of radial degree n and azimuthal degree m

Y_l^m Spherical harmonic of order l and m

P_l^m Associated Legendre polynomial of order l and m

Abbreviations

BoF Bag of Features; a method to aggregate local descriptors into global feature vectors. See Section 4.2.3 on page 29 for an in-depth explanation.

BoW Bag of Words; see BoF.

CB3DR Content-Based 3D model Retrieval; a domain in information retrieval

CBIR Content-Based Image Retrieval; a domain in information retrieval

CBMR Content-Based Music Retrieval; a domain in information retrieval

CENS Chroma Energy Normalized Statistics; a chroma feature used in music retrieval. See Section 5.3.3 on page 46 for an in-depth explanation.

CV Computer Vision; a field in computer science that seeks to mimic human sight and perception using computers.

DCG Discounted Cumulative Gain; a measure for retrieval effectiveness of IR systems and algorithms. See Section 7.1.5 on page 61 for an in-depth explanation.

DCT Discrete Cosine Transform; a variant of the Discrete Fourier Transform.

DoH Determinant of Hessian; a method for blob detection in images.

DoG Difference of Gaussian; a method for blob detection in images.

DFT Discrete Fourier Transform; Fourier transform of a discrete signal as opposed to a continuous function.

DSP Digital Signal Processing

FFT Fast Fourier Transform; an algorithm to efficiently calculate the DFT of a signal.

HOG Histogram of Oriented Gradients; an algorithm to detect and describe local features in images. See Section 4.2.2 on page 29 for an in-depth explanation.

HPCP Harmonic Pitch Class Profiles; a chroma feature used in music retrieval. See Section 5.2.1 on page 42 for an in-depth explanation.

IR Information Retrieval; a field in computer science that deals with representing, storing, and finding documents using information technology

LSH Locality Sensitive Hashing; an indexing technique employed by ADAMpro. See [1]

MAP Mean Average Precision; a measure for retrieval effectiveness of IR systems and algorithms. See Section 7.1.4 on page 61 for an in-depth explanation.

MFCC Mel-Frequency Cepstrum Coefficients; a low-level feature used in music retrieval and speech recognition. See Section 5.2.2 on page 43 for an in-depth explanation.

MIR Music Information Retrieval; see Content-Based Music Retrieval (CBMR)

MRR Mean Reciprocal Rank; a measure for retrieval effectiveness of IR systems and algorithms. See Section 7.1.3 on page 61 for an in-depth explanation.

PCA Principal Component Analysis; a statistical procedure to convert a set of linearly correlated variables into a set of uncorrelated values called principal components.

PCM Pulse Code Modulation; a method to digitally represent sampled analog signals (e.g. an audio signal). See Section 5.1.1 on page 36 for an in-depth explanation.

PCP Pitch Class Profiles; a type of chroma feature used in music retrieval.

QbE Query-by-Example; a content-based query method that can be applied to retrieval in all modalities. It involves comparison to an existing reference document.

QbS Query-by-Sketch; a content-based query method used in retrieval of visual modalities. It involves comparison to a user-generated, hand-drawn sketch.

QbH Query-by-Humming; a content-based query method used in retrieval of auditory modalities, more specifically music. It involves comparison to a reference melody sung or hummed by a user.

SH Spectral Hashing; an indexing technique employed by ADAMpro. See [1, 2]

SIFT Scale-Invariant Feature Transform; an algorithm to detect and describe local features in images.

STFT Short-Term Fourier Transform; a technique to obtain a sequence of time-local DFT's of an audio signal. See Section 5.1.2 on page 37 for an in-depth explanation.

SURF Speeded-Up Robust Features; an algorithm to detect and describe local features in images. See Section 4.2.1 on page 29 for an in-depth explanation.

VAF Vector Approximation Files; an indexing technique employed by ADAMpro. See [1, 3]

Table of Contents

1 Introduction

"Multimedia information retrieval (MIR) is about the search for knowledge in all its forms, everywhere." [4] As such, it is an important topic not only in computer science but also in the domain of arts, culture, science and our everyday lives. Media is ubiquitous and it takes many forms in terms of formats as well as semantics. Media documents may include images of X-ray or MRI scans in medical science, music in our personal collections or videos in archives of large TV stations. Moreover, media retrieval plays an important role in digital libraries [5] and a wide range of other business use cases [6].

1.1 Motivation

The advent of cheaper storage and mobile devices has given rise to an increase not only in the size of the collections but also the variety of media types and formats found within [7]. Take for example your personal photo library: Not only does it contain an ever growing number of images but nowadays it also often encompasses videos and audio snippets that are sometimes even annotated with textual metadata. Or have a look at initiatives like *memoriav*[1] or *EUROPEANA*[2], which seek to preserve our audio-visual cultural heritage and make it accessible to the public. People working in these and similar projects are confronted with an enormous number of media files of different types in various formats.
As our media collections grow larger and more diverse, the quest for accessing the knowledge contained within becomes more arduous. This is mainly due to the lack of proper tools for satisfying our information need. The classical approach consists in adding textual annotations to media documents in order to retrieve them later based on this metadata [7]. However, this query paradigm becomes increasingly infeasible for two reasons: Firstly, the sheer amount of data and the pace at which multimedia collections grow makes the laborious task of prior annotation ever more daunting. The authors of [6] estimate, that the data in the "digital universe" will grow by a factor of 300 between 2005 and 2020 from 130 EB to 40 000 EB. This amounts to approximately 5200 GB per person.

[1] http://memoriav.ch
[2] http://www.europeana.eu

Secondly, because textual descriptions tend to be subjective due to personal experience, expertise, language and culture. Furthermore, it is difficult to, for example, describe a video or audio sequence in a way that enables others to retrieve it later. This is due to the temporal progression inherent to those media types. Even though data descriptor standards like Dublin Core[3], LIDO[4] or ID3[5] try to remediate this situation, these conventions can only be applied to specific domains and sometimes even they fail to cover the diversity of the media they seek to describe.

An alternative to metadata-based retrieval is *content-based retrieval*. A content-based retrieval system assists the operator in finding documents not necessarily based on textual annotation but on the content of the documents themselves. Such techniques rely on *feature descriptors* that are extracted from the raw data of the media file and persisted in a database during an offline indexing phase. Commonly, those feature descriptors are vectors in a high-dimensional vector space — this is called the *vector space model* [7]. At query time, reference documents are typically used to find similar documents in a collection. Such a reference document may be an example image, a sketch, an audio excerpt or any other suitable representation of what the user is looking for. Another feature descriptor is extracted from that reference document and subsequently compared to the collection of feature descriptors stored in the database. At this point, some measure of similarity between these descriptors is calculated and results are ranked according to the outcome of this calculation. The reasoning behind this approach is, that if a document in the collection is similar to the reference document — which the user finds relevant for some reason — former document must be relevant too. One of the main challenges in content-based retrieval is finding compact, computationally efficient metrics for similarity that coincide with the human notion thereof. This is sometimes referred to as the *semantic gap* [8].

We argue, that businesses as well as the general public face multimedia collections that grow along the axis of sheer volume as well as variety in terms of media types like images, audio, video and text. Furthermore, new media types like 3D models or motion capture data will play a more important role in the future as new technologies, such as additive manufacturing, emerge and become affordable. Many independent driving forces add to this development. Those flourishing, heterogeneous collections pose a challenge from a data management perspective but also raise questions concerning data retrieval. Both issues are in need of a solution.

Over the past decades, many options have been explored in quest of finding the state of the art in information management and retrieval for the different types of media. However, most of these approaches only work in isolated domains and/or on a specific media type. We opine that, ideally, media of all kind should be manageable in and retrievable from an integrated system. Such a system should be able to store the data and at the same time enable the user to search and find required information quickly and easily in a seamless user experience, exploiting retrieval methods that work within and across the different modalities (e.g. standalone audio vs. audio interlaced with video). In addition, because the notion of similarity

[3] http://dublincore.org
[4] http://www.lido-schema.org
[5] http://id3.org

may be very specific to a concrete retrieval task, such a system should be extendable so that new feature descriptors and even new media types can be added if required.

1.2 Contribution

In this master's thesis, we integrate different, media type specific content-based retrieval techniques into a single system so as to contrive a solution that is capable of managing and searching mixed multimedia collections. This project builds on previous efforts by Luca Rossetto and Ivan Giangreco from the Distributed Information Systems (DBIS) Group at the University of Basel. They have devised a software stack that allows for content-based retrieval in video collections. That software is called *vitrivr* [9]. The vitrivr stack is comprised of a user interface, a feature extraction engine called *Cineast* [10] and a storage layer for feature vectors called *ADAMpro* [1]. In this thesis, this stack is extended in order to support content-based retrieval of images, audio, and 3D models in addition to its current video retrieval capabilities.

1.3 Outline

The remaining document is structured as follows: Chapter 2 provides a brief overview of related work and the current state of the art in the field. Chapter 3 describes Cineast's and Vitrivr NG's system architecture. Chapters 4, 5 and 6 give a theoretical introduction to content-based image-, music- and 3D model retrieval. Furthermore, these chapters describe the feature modules that have been designed and implemented as part of this thesis. Chapter 7 discusses the evaluation and Chapter 8 offers some conclusion and outlook towards potential future work.

2
Related work

The following chapter summarises the history and existing work related to content-based multimedia information retrieval and the state of the art for the individual modalities like audio, 3D models and images.

2.1 Content-Based multimedia information retrieval systems

Currently, there seems to be very little work on integrated solutions for content-based retrieval of different types of media. Most research in the field focuses on a particular modality like audio, video or 3D or even further specialised subdomains, like for example, music, speech or environmental sounds for audio. Nevertheless, a few authors have tried to combine content-based retrieval techniques in a single system.

In 1995, Flickner et al. [11] introduced the QBIC (Query By Image Content) system, which allowed for Query-by-Example and Query-by-Sketch on both image and video databases. QBIC was basically a pure image retrieval system. In order to add support for videos, those were segmented into shots, that is, an uninterrupted series of frames. Representative frames of each shot were then identified and treated as still images.

The closest thing to a general-purpose retrieval system is reported by [12]. The authors created MUVIS, a content-based multimedia indexing and retrieval framework that supported images, video and audio — both stand-alone and interlaced with video. MUVIS was based on previous work by the same group in the field of image retrieval.

In 2014, the Distributed Information Systems (DBIS) research group at the University of Basel introduced *Cineast* [9, 10], a content-based information retrieval engine for video. The team focused on the visual information in videos but already had other modalities like text (subtitles) and audio in mind, when they had designed the system's architecture. The recent transition to *ADAMpro* [1] as Cineast's new storage engine has promoted the potential support for new media types. ADAMpro is being advertised as a database mainly targeted at big multimedia retrieval and allows for efficient *k-Nearest Neighbours (kNN)* search in high-dimensional spaces, which is crucial for content-based multimedia retrieval. It employs efficient indexing strategies like *Spectral Hashing (SH)* [2], *Locality-Sensitive Hashing (LSH)* [13] and *Vector-Approximation (VA) files* [3].

2.2 Content-Based Image Retrieval

Early work on image matching and retrieval started in the late 1970s and it has become a fundamental aspect of many problems in computer vision. Most *Content-Based Image Retrieval (CBIR)* systems make use of colour and texture information in the images whereas some use shape or layout, e.g. for character recognition or classification [14]. General-purpose colour-based CBIR systems very often employ histograms in different colour spaces, colour layout and region-based search, or a combination thereof [15]. Typical techniques for identifying shapes involve edge histograms or image moments, like for instance centroid distances [15, 16]. The aforementioned QBIC system by Flickner et al. [11], for example, leverages a combination of colour features and shape identification.

The team surrounding CANDID (Comparison Algorithm for Navigation of Digital Image Databases) [17], on the other hand, calculates *Gaussian Mixture Models (GMM)* to approximate feature vectors that have previously been obtained at every pixel. These mixtures are used to build a signature for every image and the L^1 norm is calculated to measure the distance between two such signatures. However, even though the method allows for more efficient information representation compared to histogram methods, it also proved to be computationally inefficient for large datasets.

More recent developments gave rise to more advanced techniques like SIFT [18], SURF [19], VLAD [20] and Fisher Vectors [21]. The *Scale-Invariant Feature Transform (SIFT)* transforms an image into a set of local feature vectors, each of which is invariant to scaling, translation and rotation and robust to changes in illumination and local distortions [18]. The algorithm uses *Difference of Gaussians (DoG)* to identify points of interest (key points). For each key point, the histogram of a local, oriented gradient is subsequently calculated and stored in a 128 dimensional vector. Various flavours of the SIFT algorithm have been proposed since its original publication.

Speeded-Up Robust Features (SURF) is an optimised algorithm inspired by SIFT, of which the authors in [19] showed that the former outperforms the latter. They use a Hessian-matrix approximation to identify key points, which is faster than the DoG approach employed by SIFT. After the key points have been identified, SURF describes the intensity content of the area around a key point using a technique that is similar to the gradients calculated by SIFT.

Once local feature descriptors have been obtained by means of SIFT, SURF or a similar approach, a simple *Bag of Words (BoW)* or *Bag of Features (BoF)* model can be applied to create a global, aggregated feature vector. The technique is described and evaluated in [22–24]. It involves obtaining a vocabulary of k "visual words". Once such a vocabulary has been assembled, every local feature in an image is assigned to the closest entry in the vocabulary. The histogram of this assignment is then the BoF, a k-dimensional vector.

Other techniques to compile compound feature vectors from local descriptors are *Fisher vectors* and *VLAD*. Fisher vectors are described in [21] and can be calculated by representing the visual vocabulary (e.g. SIFT descriptors) as a GMM, where each Gaussian represents a word in the visual vocabulary. Every local feature vector is then assigned to a mode in the GMM with a weight given by its posterior probability. The *Fisher Vector (FV)* is obtained by stacking the mean and covariance deviation vectors for each mode in the GMM.

VLAD, in contrast, sees itself as a simplification of the Fisher Kernel [20]. As with the BoF method, a vocabulary of words is obtained by k-means clustering. For each cluster center $\mathbf{c}_{i,j}$ in the vocabulary, the difference $\mathbf{c}_{i,j} - \mathbf{x}_j$ of the vectors assigned to it is accumulated to form a new feature vector. VLAD was shown to be less memory consuming and more efficient than traditional BoF methods.

2.3 Content-Based Audio Retrieval

A major challenge with content-based retrieval of audio is that there are different types of audio and that the strategies to index and query them differ substantially. This is mainly due to the psychological aspects of audio perception — and thus perceived similarity — also known as *psychoacoustics*. One large area of research is audio retrieval of music, referred to as *Content-Based Music Retrieval (CBMR)* or just *Music Information Retrieval (MIR)*. It is a multidisciplinary field that straddles different domains — ranging from computer science to psychology. There is a large community surrounding CBMR organised in the *International Society for Music Information Retrieval (ISMIR)*, which holds the annual MIREX evaluation campaign for MIR algorithms [25].

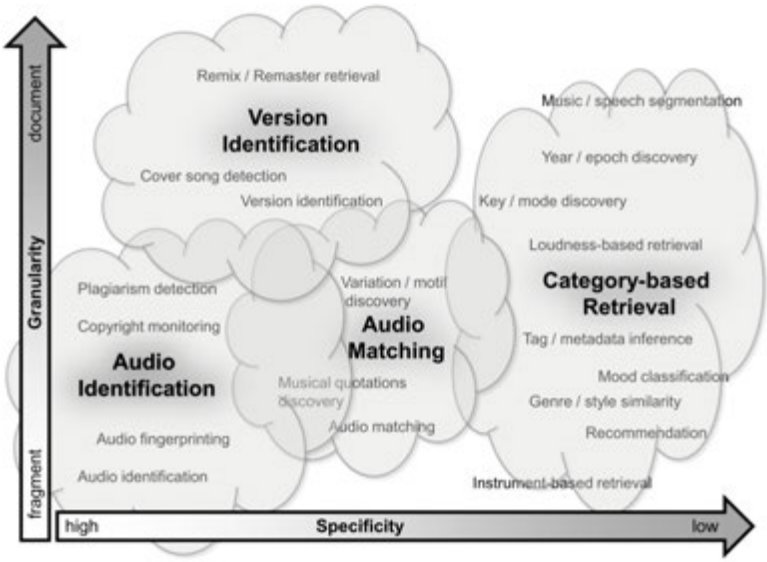


Figure 2.1: Categorisation of MIR tasks along the dimensions of granularity and specificity (Source: [26]).

MIR tasks can be characterised by their *specificity*; that is, the desired degree of similarity between the document and the query, and their *granularity*, i.e. whether comparison takes place on a document level or on the level of fragments thereof. This classification scheme is illustrated in Figure 2.1. Based on these two dimensions, the authors of [26] classify existing Query-by-Example techniques for music into four larger categories: *audio identification (fingerprinting)*, *audio matching*, *version identification* and *category-based retrieval*. We will not go into details about the latter.

Audio identification This is a high-specificity, low-granularity task. Given a small audio fragment as query, the task consists in identifying the particular recording it belongs to. The notion of similarity is hence very close to identity.

Audio matching This is a mid-specificity, mid-granularity task. It consists in finding variations of a given, musical fragment as they occur in different performances or arrangements of the same piece (e.g. a live performance). These variations may differ in aspects like tempo or execution of note groups.

Version identification This is a low-specificity task. It consists in finding versions that may differ considerably from the query fragment in terms of instrumentation, key, tempo and even melody. Changes like this usually occur in cover songs or remixes of a musical piece.

The authors in [26] further conclude that the problem of audio fingerprinting, that is, finding exact matches given an audio fragment, has been largely solved, naming prominent examples like Shazaam [27] and fingerprinting techniques like *Mel-Frequency Cepstrum Coefficients (MFCC)* [28, 29]. However, even though there is a lot of ongoing research on audio matching or version identification, no best practices have evolved yet and a lot of working examples trade retrieval accuracy for scalability or vice versa. Generally, chroma-based features, like *Pitch Class Profiles (PCP)* [30, 31] or variations thereof [32] are often used for these MIR tasks, but some systems also exploit rhythm or melody [33]. For example, [34] proposes a version identification scheme based on previous work by Casey et al. [35, 36]. The proposed technique involves comparing pitch class profiles for overlapping shingles (audio fragments) of fixed length. The authors were able to demonstrate that this technique scales well, especially when combined with LSH. The authors in [37] propose a retrieval system that segments audio and classifies segments into different categories (speech, music, environmental sound, speech over music and speech over environmental sound). They use the *low-level descriptors (LLDs)* defined in the MPEG-7 standard. Namely, they employ *Audio Power*, *Audio Spectrum Centroid*, *Audio Spectrum Spread* and *Audio Spectrum Flatness* descriptors. Additionally, their system allows for Query-by-Example based on the same LLDs. It exploits the aforementioned classification to restrict the search space to the audio segments that match the class of the query.

Reduced specificity beyond that of version identification is commonly referred to as *Query-by-Humming (QbH)* or *Query-by-Singing*. QbH seeks to find a musical piece based on hummed user input. In order to do this, melody must be extracted from both the hummed input as well as the original songs in the database. This is a non-trivial task, especially for polyphonic music, that is, music that comprises multiple instruments and vocals. Generally speaking, melody extraction can be divided into two major subtasks - *pitch estimation* and *pitch tracking*. Pitch estimation consists in identifying candidate pitches that could potentially be part of the melody. Many algorithms have been proposed to do this in a robust fashion. Most of them calculate pitch salience through harmonic summation of fundamental frequencies [33, 38–40]. Nevertheless, phenomena like inharmonicity or octave errors and the fact that multiple sources of sound are difficult to separate make this a daunting task for polyphonic music. Once pitch candidates have been identified one must

find the actual contour of the main melody. This usually requires some knowledge about the underlying musical model, i.e. pitch transitions that are likely to occur as opposed to transitions that are unlikely or even forbidden. This step of melody extraction is usually done by applying some heuristic [33] or machine learning technique [41] like *Hidden Markov Models (HMM)*.

It has been pointed out by a number of authors, that most existing work on QbH systems is based on prior symbolic transcriptions of the music (e.g. MIDI data) that is being stored in a database. The authors in [42], for example, propose such a QbH retrieval system. They use the low-level audio descriptors (LLD) defined in the MPEG-7 standard (*AudioFundamentalFrequency*) descriptor to obtain the melody contour of the hummed user input, which they subsequently transcribe and compare to melody sequences in the database. There is, however, work that assumes that no such symbolic representation of the documents is available beforehand. Instead, the authors in [43] try to extract the information from the raw audio stream using fundamental frequency (F0) estimation. Salamon et al. [33] take this approach one step further. They consider QbH to be a special case of version identification and they suggest that a version identification system for music could extract different representations of the audio data, using a combination of methods and pre-processing steps. Namely, the authors describe a main melody representation (equal loudness filter + F0 estimation), bass line representation (low-pass filter + F0 estimation) and *Harmonic Pitch Class Profiles (HPCP)*. A combination of these representations could yield better accuracy for version identification. Furthermore, the authors argue that by limiting oneself to main melody features, the same system could be extended to support QbH. *Mean Reciprocal Rank (MRR)* values ranging between 0.66 to 0.33 are being reported for their test collection of 2125 songs. As they employ a quadratic programming algorithm for the feature matching, it is questionable, however, whether their approach scales well to larger collections.

To summarise, one can state that with the exception of audio fingerprinting, no state of the art seems to have emerged yet for content-based retrieval of music on a large scale. Existing methods often lack either retrieval accuracy or scalability or they enforce some constraints on the data, that limits the practical use of a system.

2.4 Content-Based 3D model Retrieval

3D models are complex in nature both in terms of the data structures that are used to represent them, as well as in terms of their geometry and topology. Besides obtaining descriptors for a 3D object that are both robust and easy to calculate, there is one principal challenge that must be overcome when tackling content-based multimedia retrieval of 3D models: The degree of freedom that is inherent to them. In a given host coordinate system, 3D models are positioned and oriented unpredictably and scaled in arbitrary units of measurement. Therefore, in order to be robust, any measure for similarity must yield the same results under these transformations, that is, translation, rotation and scaling [44, 45].

One way to tackle this challenge is to rely on feature descriptors that are inherently invariant to aforementioned transformations. This limits the set of possible descriptors, however, as most directly rely on geometric properties of the object itself [46]. Furthermore, the authors

in [47] point out that similarity should obviously be based on the object's appearance, to which their geometric features contribute a great deal. Nevertheless, one very promising approach for *Content-Based 3D model Retrieval (CB3DR)* is based on spherical harmonics descriptors as reported by different authors [44, 48, 49]. These descriptors are invariant to rotation, which already solves a large portion of the problem.

Another solution involves normalisation of the 3D model prior to extracting the features, that is positioning and aligning the model into a canonical coordinate system. This is called *pose estimation* or *pose normalisation* and is often achieved by applying a variant of the *Karhunen-Loeve transformation* [47, 49–51], which is easy to implement and computationally efficient. Under this transformation, a 3D model's center of mass is first moved to the origin of the reference coordinate system. Afterwards, the model is rotated until it is aligned along its principal component axes and subsequently scaled to a unit size. The main drawback of this method is that it does not account for differences in triangle resolution of models. Multiple solutions were proposed to address this issue. One of them was put forward by Vranic et al. [45] and involved generalising the discrete PCA to a continuous version so that all of the points on the mesh surface are equally relevant for computation of the principal axes. Another issue of the PCA approach is that the x-, y- and z-axes do not have an intrinsic direction. The authors in [50] solve this ambiguity by applying the convention that the area on the positive side of the coordinate system must be greater than on the negative side.

Once normalisation has been applied one must find an appropriate descriptor for similarity between 3D models. Similarity can be assessed by means of many different descriptors of which feature vectors, histograms and statistical moments are only three examples. Multiple surveys [50, 52] list and classify the many methods for content-based 3D model retrieval. Their classification is based on various aspects and the nomenclature may therefore differ. Some of the most interesting approaches are briefly described hereinafter.

The authors in [47] apply a strictly geometric approach and calculate distance histograms to measure similarity between objects. This method was shown to be effective for robust retrieval but it was also found to scale badly to large datasets.

D. Vranic and D. Saupe [51] intersect rays emanating from the object's center of mass with its triangle mesh surface. The distance to the farthest intersection of the i-th ray is the i-th component in the resulting feature vector. In their experiments, they used 20 rays that travel in the direction of the vertices of a dodecahedron and they achieved solid retrieval results with this method.

In their work, the authors of [44, 49] use a linear combination of *spherical harmonics* to obtain a function that approximates the 3D model's surface. The weight coefficients of this function serve as components in the feature vector. Due to the nature of spherical harmonics, these feature vectors are inherently invariant to rotation.

Chen et al. [53] propose *light field descriptors* for 3D models. These descriptors are based on projections of the 3D model onto the faces of a circumscribing dodecahedron. Subsequently, classical CBIR techniques are applied on the resulting images to extract feature descriptors, namely calculation of *Zernike moments* and *Fourier descriptors* for the resulting shape. The advantage of this approach is that it can be employed both for Query-by-Example

(comparison of 3D model to 3D model) and Query-by-Sketch (comparison of 2D sketch to 3D model). This was confirmed in [48], where the authors describe a system that combines different feature descriptors. On the one hand, they realised QbE by using a shape descriptor based on spherical harmonics. On the other hand, they implemented QbS based on the aforementioned light field descriptors. Furthermore, they investigated different modes of user interaction for a 3D retrieval system. Their findings suggest that QbE based on existing 3D models and QbS based on hand-drawn 2D sketches are very well accepted by the user, whereas query modes based on ad hoc 3D modelling are too complex.

The survey in [52] evaluates the different techniques with respect to their retrieval effectiveness. They found that a variant of the light field descriptor, called depth-buffer, yields the best overall results, which is in accordance with reports by other authors. Additionally, spherical harmonics proved to be very efficient under conditions where PCA normalisation was not effective.

A more recent and fundamentally different approach to QbS based on 2D sketches is reported by [54]. The authors train a pair of *Convolutional Neural Networks (CNN)*, also referred to as "siamese network". One network is trained with views of the 3D model from two different perspectives and the other one with hand-drawn sketches generated by users. The authors demonstrated that their approach outperforms existing feature descriptors.

3
Architecture

This chapter describes the system architecture of the new version of Cineast and the new user interface, which are the main deliverables of this project. Large parts of Cineast's architecture are identical to the original version, which is described in [10]. This chapter focuses on changes and additions that were developed as part of this project. However, for the sake of completeness and clarity, some aspects are described even though they have not changed significantly since the original publication.

3.1 Cineast

Cineast is a multi-modal feature extraction engine written in Java. A complete list of the frameworks and libraries that were used can be found in Appendix E on page 115.
Cineast comprises two major parts: an offline *ingest workflow* and an online *retrieval workflow* and their respective runtimes. The system architecture outlined in Figure 3.1 has been designed to support those two workflows for multiple media types.
The core functionality of Cineast is provided by *feature modules*. They are responsible for feature extraction both during ingest and retrieval and both the *ingest runtime* and the *retrieval runtime* have access to the same set of feature modules. In order to support the ingest of media files, Cineast also includes numerous classes that take care of file handling, decoding of different formats and segmentation of their content. Those classes are subsumed in the *file handling* module also depicted in Figure 3.1. Additionally, Cineast includes an API, which enables external access to the retrieval runtime. This API currently supports communication via REST[6] and the WebSocket protocol, and it has been utilised to build the user interface, which is otherwise completely detached from Cineast itself. Furthermore, Cineast includes an abstraction layer for communication with the underlying storage engine. For most purposes, that storage engine is ADAMpro [1], a database that has been tailored to the requirements of multimedia retrieval. However, the abstraction layer allows different engines for both lookup (selectors) and persisting (writers) to be connected to Cineast.

[6] **Re**presentational **S**tate **T**ransfer; a way of designing HTTP interfaces

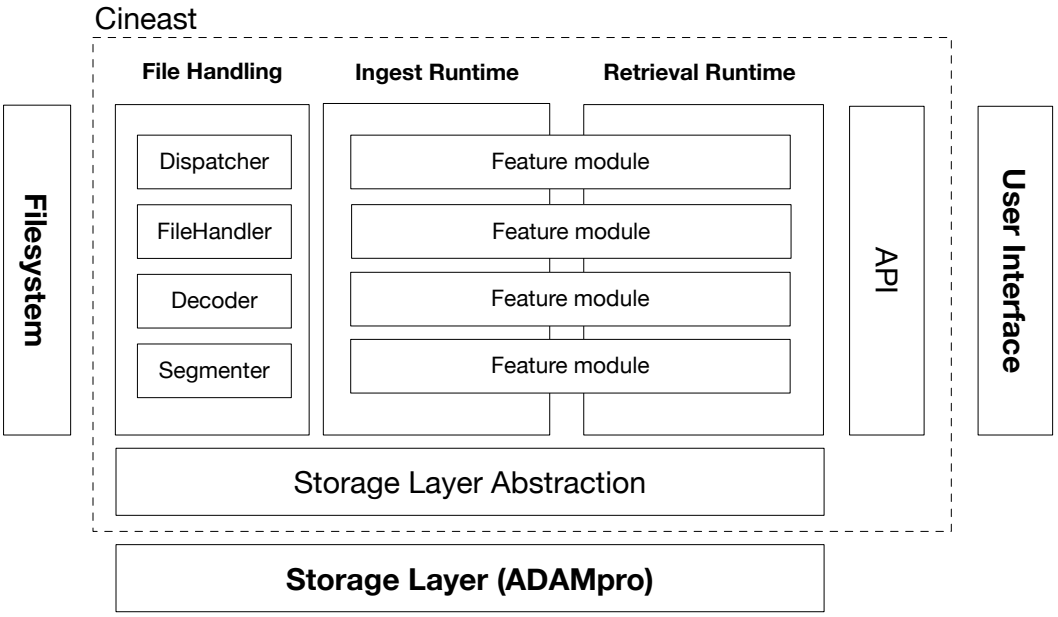


Figure 3.1: Illustration of Cineast's system context. The main modules are the file handling module, the ingest runtime and the retrieval runtime. Those three modules facilitate the offline ingest and the online retrieval workflow supported by Cineast. Outside the system context lies the storage layer for feature vectors and the user interface.

3.1.1 Data model: Media information

The data model for representation of multimedia information has been derived from the pre-existing version and — for the purpose of this thesis — enriched with ideas from [55]. It consists of four main entities, which are called *Multimedia Object*, *Multimedia Segment*, *Metadata* and *Feature*. To keep things short, the prefix "multimedia" is henceforth omitted. The aforementioned entities and their interrelationships are depicted in Figure 3.2.

Object The (multimedia) object entity represents a single media file F (sometimes also called document), such as an image or a video. The entity encapsulates technical information about the file, e.g. its path, its internal, unique ID, its filename and its type in the sense of media type.

Metadata The metadata entity can be used to describe the file and/or its content (on a file level). Metadata entries usually hold textual information and can be derived from file metadata standards like for instance EXIF[7], ID3[8] or IPTC[9].

Segment The segment entity represents a chunk of the entire multimedia object and serves to model temporal progression of the content. The cardinality of this relationship depends on the type of media, the total duration of the file and implementation specific aspects, henceforth referred to as *segmentation strategy*. In case of an image, for

[7] https://www.jeita.or.jp/japanese/standard/book/CP-3451C_E
[8] https://id3.org
[9] https://iptc.org/standards/photo-metadata

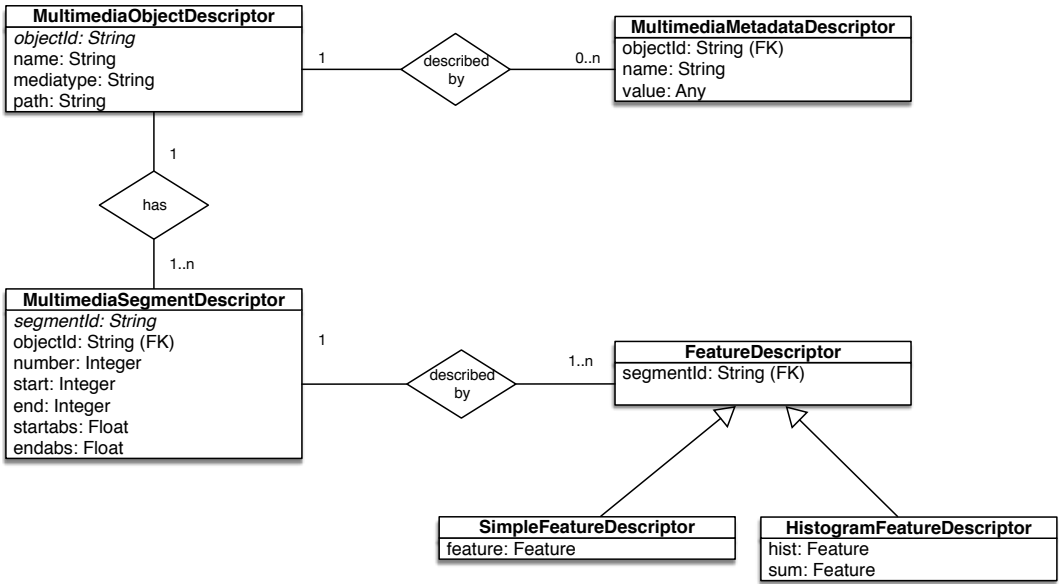


Figure 3.2: Entity relationship model for Cineast. The model depicts the four main entities *Object*, *Segment*, *Metadata* and *Feature* as proposed by [55]. The subclasses of the feature entity indicate, that there are different kinds of features.

instance, only one such segment exists per object, whereas for video and audio an ordered sequence of segments captures the progression of the file's content with time.

Feature The feature entity captures some aspect of a segment's content and, transitively, of the object itself. That aspect is usually referred to as feature descriptor. An arbitrary number of different features can coexist for a single segment. Logically, a feature entity usually stores one or several d-dimensional, numerical feature vectors $\mathbf{f_i}$ — a mathematical representation of the feature descriptor.

Currently, four different media types are supported by Cineast: images, videos, audio and 3D models. The high-level data model is applied to each media type in a slightly different manner, which is described in the following sections.

3.1.1.1 Images

Images are represented by two-dimensional arrays holding colour values for every pixel. The simplest and most common way to encode an image, which is also used by Cineast internally, is in the RGB colour space. RGB is an additive colour model; that is, every color can be created by adding red, green and blue components in some ratio. Therefore, in order to store an RGB image, three 2D arrays are required, each of which holds values for the red, green and blue colour component of every pixel. This representation is tied into the previously described data model as follows:

$$Object \ni Segment \ni Image$$

The cardinality of the relationship between image file (object) and image segment is 1:1, as is the cardinality between image segment and the raw image data. The assumption here is,

that image files contain only a single image and that this image is static. Hence, formats that support image sequences (like GIF) are currently not supported.

3.1.1.2 Audio

At its lowest level, Cineast represents audio as a stream of 16bit samples where each sample is a *signed short* value. This representation is called *Pulse-Code Modulation (PCM)* and is very common in DSP. A more detailed explanation of PCM encoding can be found in Section 5.1 on page 35. The PCM representation of the audio signal is tied into the previously described data model as follows:

$$Object \ni Segment \ni AudioFrame \ni Sample$$

The audio decoder usually returns samples in batches. These batches are referred to as audio frames and usually have a duration of a few milliseconds. The boundaries between such frames are arbitrary and usually determined by the decoder and the codec. An audio segment comprises a sequence of audio frames and accordingly the entire file is represented by a sequence of such audio segments. This captures the temporal progression of the signal. Depending on the segmentation strategy, the duration and composition of a single segment as well as the exact number of segments may vary.

3.1.1.3 Video

Video files potentially encompass both auditory and visual information. This information is usually encoded in separate streams and only a temporal relationship exists between them. The visual information is represented by a sequence of still images. Commonly, 24 or 25 images per second are used but other values exist as well. Such an image is called (raw) video frame. The audio information is again represented as a sequence of audio frames containing PCM data (see previous section). This representation of video data is tied into the previously described data model as follows:

$$Object \ni Segment \ni VideoFrame \begin{cases} \ni Image \\ \ni AudioFrame \ni Sample \end{cases}$$

The video decoder extracts the raw video and audio frames separately. Both hold a presentation timestamp (PTS), based on which the decoder merges the raw frames into a single video frame that packages both the visual and the auditory information. Because the image rate is orders of magnitude smaller than the audio sampling rate, one video frame contains a single image, the raw video frame, and multiple audio frames. Video segments[10] consist of a sequence of video frames and the whole file is represented by a sequence of video segments. This captures the temporal progression in the video file. Depending on the segmentation strategy, the duration and composition of a single segment as well as the exact number of segments may vary.

[10] In the previous version of Cineast, video segments were referred to as Shots. For consistency reasons, this nomenclature was changed.

3.1.1.4 3D models

There are two commonly used representations for 3D models used in computer graphics. On the one hand, there is the *polygon mesh* and on the other hand there is the *voxel grid*. Both concepts are explained in Chapter 6 on page 49. Files that encode 3D models mostly encode polygon mesh information, that is, information about vertices, edges and faces. This representation of 3D model data is tied into the previously described data model as follows:

$$Object \ni Segment \ni \begin{cases} \ni VoxelGrid \ni Voxel \\ \ni Mesh \ni \begin{cases} Vertex \\ Face \end{cases} \end{cases}$$

The 3D model decoders return the native representation of the respective file format. The cardinality of the relationship between file (object) and segment is 1:1 as is the cardinality between segment and/or the voxel or mesh representation. The assumption here is that a single file does only contain a single 3D model and that it does not change over time. Formats that contain multiple models or animations are currently not supported. Because some feature modules require a specific representation, there are also classes that facilitate conversion from polygon mesh to voxel grid (the other direction is currently not supported).

3.1.2 Data model: Queries

The data model used to express similarity queries is based on the five entities described in this section and depicted in Figure 3.3.

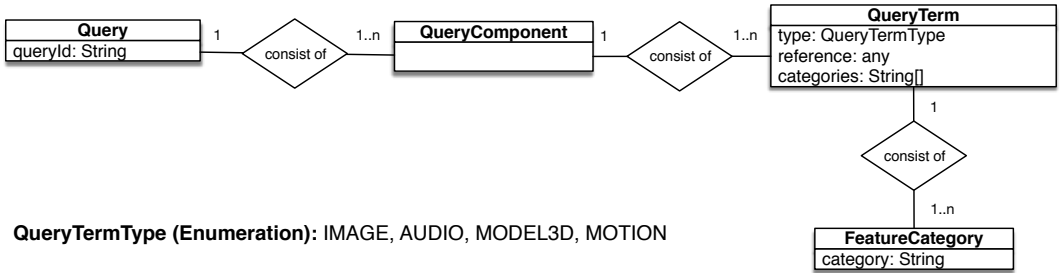


Figure 3.3: Entity relationship model for queries in Cineast. The model depicts the four main entities *Query*, *Query component*, *Query term* and *Feature category*.

The primary use of the data model is exchange of information about queries between Cineast and any consumer of the Cineast API.

Query Represents a single query as requested by any consumer of the Cineast API. It comprises one to many query containers and details all the information required by Cineast in order to invoke the requested feature modules, extract the relevant features and perform the necessary lookups.

Query component A query component groups a set of query terms that should be evaluated conjointly when executing the query. That is, every query term and the partial results of the associated feature modules count towards the final result of that component. A query can consist of multiple query components. One can think of query

terms that share the same component to be connected by a logical AND, whereas query terms in different components are connected by a logical OR.

Query term A query term has a certain type and consists of a single reference document and a set of feature categories that should be considered when evaluating the query.

Reference document A reference document can be anything that serves as a baseline for pairwise comparison. Upon execution, feature vectors in the database are compared to the vectors extracted from the reference document. A reference document could be an example image, an excerpt of an audio file, a hummed melody or a sketch.

Feature category Feature categories group a list of feature modules for a specific retrieval task. Each module in a category is assigned a weight, which is used during the final fusion step to weigh the partial results per module and category. Feature categories can be configured in Cineast and feature modules can be freely mixed and matched within the categories.

Currently, four types of query terms are supported by Cineast - namely, IMAGE, AUDIO, 3DMODEL and MOTION. They differ in the kind of reference document that is being used. From the data model perspective it is straightforward to add support for new types.

3.1.3 Decoding and Segmentation

Decoding is the operation $\mathfrak{D}$ of reading a native media file F and transforming its content into a stream of N data units U_i that can be processed by Cineast.

$$\mathfrak{D} : F \rightarrow \{U_1, U_2, \ldots, U_N\}$$

An example would be reading the samples from an audio file and returning them as audio frames. Numerous differing file formats and codecs exist for audio, video, images and 3D models and they exhibit different characteristics in terms of structure, compression and information loss. That is why the decoding facility in Cineast has been designed in a way that allows for extension if required. Currently, the following formats are supported:

Image All formats supported by the TwelveMonkeys[11] library, which has been integrated into Cineast. Among the most prominent are TIFF, PNG, different JPEG flavours, BMP and PICT.

Video All formats and codecs supported by the FFMPEG[12] library, which has been integrated into Cineast via native (JNI) bindings. Among the supported codecs are MPEG-1, MPEG-2, MPEG-4 and H.264.

Audio All formats and codecs supported by the FFMPEG library, which has been integrated into Cineast via native (JNI) bindings. Among the supported codecs are AIFF, RIFF WAVE, MP3, AAC, OGG Vorbis and FLAC.

[11] https://github.com/haraldk/TwelveMonkeys
[12] https://ffmpeg.org

3D models A modular decoder has been created for 3D models. Currently, that decoder includes drivers for Wavefront OBJ and Stereolithography (STL) files. However, it is straightforward to add support for more formats if required.

Segmentation $\mathfrak{S}$ consists in grouping the N data units U_i returned by a decoder (e.g. audio frames or raw images) into M segment tuples $S_n = (I, U_i, U_{i+1}, \ldots, U_{i+l_n})$ of length l_n, as defined in Cineast's data model. Every such tuple is identified by an identifier I and the length l_n of a segment can differ even within the same file, depending on the applied segmentation strategy.

$$\mathfrak{S} : \{U_1, U_2, \ldots, U_N\} \rightarrow \{S_1, S_2, \ldots, S_M\}$$

For modalities that do not exhibit any temporal progression, like for instance images and 3D models, segmentation is reduced to packing the raw data units into a segment each and assigning them unique IDs. This is referred to as *passthrough segmentation*. Therefore, every raw unit results in a single segment.

$$\mathfrak{S}_{PT}(U_i) = S_i = (I, U_i)$$

For video and audio, segmentation becomes a more complicated matter and different segmentation strategies can be employed. For example, audio frames are currently grouped into segments of constant duration t with a defined overlap o between two successive segments. That is, audio frames U_i are appended to a segment until a certain threshold for total duration is reached, which triggers the creation of a new segment.

$$\mathfrak{S}_{CD_{d,o}}(U_{i-o}, \ldots, U_i, U_{i+1}, \ldots, U_{i+(t-o)}) = (I, U_{i-o}, \ldots, U_i, U_{i+1}, \ldots, U_{i+(t-o)})$$

Other approaches could be implemented as well. For instance, Jonathan Foote [56] proposes an automated audio segmentation algorithm based on novelty in the audio file. It could be used to automatically create segments from an audio signal based on major changes and may offer an alternative to fixed-length fragments. A similar strategy is already employed for videos, where segments are based on calculated shot boundaries (see [10] for more information).

3.1.4 Feature modules

The main responsibilities of the feature modules are performing the extraction $\mathfrak{E}$ of feature descriptors $F_i = (I, \mathbf{f_1}, \mathbf{f_2}, \ldots, \mathbf{f_N})$ and generation of feature vectors $\mathbf{f} \in \mathbb{R}^n$ from a segment S_i during retrieval and ingest. A feature descriptor tuple is always identified by an identifier I that allows association of the descriptor with either the segment or the object.

$$\mathfrak{E} : S_i \rightarrow F_i$$

Feature extraction usually involves orchestration of a wide range of domain specific helper classes, like for instance a DSP pipeline for audio segments or a CV library for image information. Generation of such a descriptor is very specific to the aspect that is being

described, as is the size of the resulting vector and the number of vectors per object or segment. The feature modules designed as part of this thesis and the theory behind them are elaborated in Chapters 4, 5 and 6.

In Cineast, the vast majority of features are represented by vectors $\mathbf{f} \in \mathbb{R}^n$, where the dimensionality n usually lies somewhere between 10 and 500. That is, a feature vector can be thought of as a point in a high-dimensional vector space. All these feature vectors are treated equally by the feature modules once they have been obtained — they can either be persisted during ingest or used for lookup during retrieval.

Using the analogy of $\mathbf{f} \in \mathbb{R}^n$ being a point in a high-dimensional space, lookup can then be expressed as finding the k nearest neighbours $\{\mathbf{f_1}, \ldots, \mathbf{f_k}\}$ of a query vector $\mathbf{f_q}$ using a *dissimilarity function* D in the host space. D is sometimes also referred to as *distance function*.

$$D : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+ \tag{3.1}$$

Hence, finding the nearest neighbours is equivalent to finding the k vectors that minimise the expression in equation 3.2, where N denotes the number of vectors in the database and d_i is the *dissimilarity* or *distance* between $\mathbf{f_q}$ and $\mathbf{f_i}$. This is also called *k-Nearest Neighbour search*, *kNN-search* or *kNN-lookup*.

$$d_i = D(\mathbf{f_q}, \mathbf{f_i}),\ \forall i \in \{1, \ldots, N\} \tag{3.2}$$

Many different dissimilarity functions D exist. Very often, norm-induced metrics of the host vector space are used. The *Minkowski distances* L^p of order p, which include the well-known *Euclidian distance* L^2 and the *Manhattan distance* L^1, belong to that family of dissimilarity functions. However, D does not necessarily have to be a metric.

Picking the right dissimilarity function for a particular feature is again very dependent on the feature itself and in practice, very often, D is empirically selected. However, the Euclidian distance and the Manhattan distance are very common choices. Equation 3.3 illustrates calculation of the dissimilarity d_i between a query vector $\mathbf{f_q}$ and a vector $\mathbf{f_i}$ in the database using the L^2 distance measure.

$$d_i = D_{L^2}(\mathbf{f_q}, \mathbf{f_i}) = \sqrt{\sum_{n=1}^{D}(f_{q,n} - f_{i,n})^2},\ i \in \{1, \ldots, N\} \tag{3.3}$$

Usually, for the purpose of a retrieval system, the dissimilarity itself is not an interesting information as it is very abstract, especially in high-dimensional spaces. Instead, a more approachable indication for relevance of the results is preferred. This is why dissimilarities d_i are usually converted to *similarity scores* $s_i \in [0, 1]$ by some kind of monotonically decreasing *correspondence function* C.

$$C\ :\ \mathbb{R}_0^+ \to [0, 1] \tag{3.4}$$

A score of 0 indicates that the feature has no relevance at all with respect to the aspects that are being considered by the descriptor, whereas a score of 1 is equivalent to a full match. Again, there are many different types of correspondence functions. The simplest approach,

and the one most commonly employed by Cineast, is to normalise the distance by some arbitrary maximum (cut-off) distance d_{max}.

$$s_i = C(d_i) = \begin{cases} 1 - \frac{d_i}{d_{max}}, & \text{if } d_i \leq d_{max} \\ 0, & \text{if } d_i > d_{max} \end{cases} \tag{3.5}$$

It has been mentioned before, that the feature module's main responsibility is generation of the feature vectors from the provided objects both during retrieval and ingest. This is facilitated by a programming interface that serves as a hook for both the ingest and the retrieval runtime. Both runtimes hand the results of their processing directly to the eligible feature modules.
Once features have been generated, the feature modules either persist those features (ingest) or use them to perform a kNN-search (retrieval). In both cases, they make use of the underlying storage layer abstraction. Compact storage of feature vectors and efficiently searching these high-dimensional vector spaces is thereby completely delegated to the storage engine, which resides outside of Cineast's system context. In fact, ADAMpro [1] has been designed to perform those operations and it supports a plethora of dissimilarity functions.

3.1.5 Ingest workflow (offline)

The main steps of the ingest workflow are illustrated in the block diagram in Figure 3.4a. A configuration file and a media file or a folder containing media files serve as input to the workflow. The workflow's main responsibilities are extraction and persisting of relevant features. The appropriate *file handler* implementation depends on the media type and is selected by a *dispatcher* class based on the configuration provided by the user.
Once a *file handler* has been selected, that handler orchestrates the actual ingest process for all files in the list. That is, it hands every file to the respective *decoder*, *segmenter* and ultimately to the *extraction runtime*. This process consists of the following main steps (all system components are printed in italics, see Figure 3.1 for further reference):

1. The *decoder* converts a file from its native format into data units that can be processed by Cineast (e.g. audio samples, images, meshes).

2. The *segmenter* aggregates a set of data units returned by the decoder into a segment entity (see Section 3.1.1, p. 12). Different segmenters may employ different strategies to achieve this.

3. The *file handler* persists all the relevant information about both the media object and the segments through the *storage layer abstraction*.

4. **Optionally:** The *file handler* invokes metadata extractors that extract and persist file metadata.

5. The *file handler* forwards each segment to the extraction pipeline.

6. The extraction pipeline forwards every segment to all the feature modules that have been specified in the configuration.

7. The *feature modules* extract the features and persist them through the *storage layer abstraction*.

At the end of the ingest workflow, information about the ingested files, the segments that were derived for each file and the features that were extracted for each segment are persisted in the storage layer. As a consequence of the *file handler* selection by the *dispatcher* being one of the first steps, and the *file handlers* being specific for a media type, different media types can currently not be mixed in a particular ingest run.

3.1.6 Retrieval workflow (online)

The main steps of the retrieval workflow are illustrated in the block diagram in Figure 3.4b. The retrieval workflow is triggered by a query message, for example, transmitted as WebSocket packet. The query message can be thought of as a request which details the query that should be executed by Cineast, see Section 3.1.2 on page 15 for explanation of the data model.
Once the query object has been received and deserialised, the object is forwarded to the *retrieval runtime*. The retrieval process now consists of the following main steps (all system components are printed in italics, see Figure 3.1 for further reference):

1. The *retrieval runtime* extracts the query components and the query terms from the query object and creates what is called a query container, that is, an internal representation of the reference documents's data (i.e., one query container per query component). The runtime then creates a map of feature categories to query containers based on all the categories in the query.

2. The *retrieval runtime* forwards the query containers to every feature module listed in one of the requested feature categories.

3. Every *feature module* generates its feature vectors from the query container and performs a kNN-lookup through the *storage layer abstraction*. As a result, a list of segment IDs and relevance scores is returned.

4. The *retrieval runtime* fuses the partial results returned by every feature module per category based on the configured weights.

5. The *retrieval runtime* returns the per-category results to the requester alongside information about the relevant segments and objects. The latter are looked up through the *storage layer abstraction*.

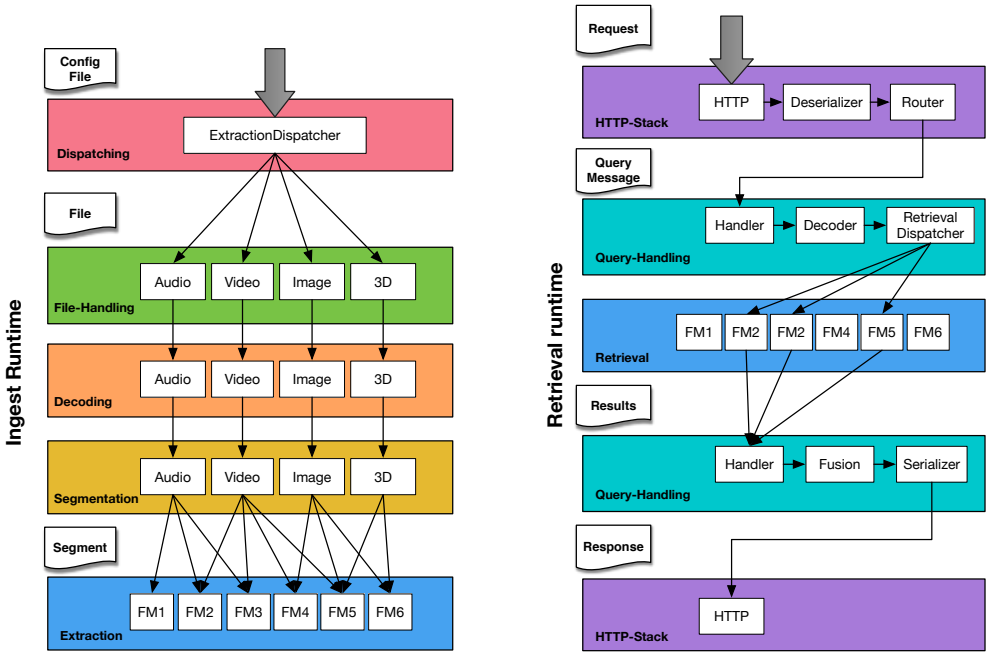
The work performed by the feature modules in step number three can be further formalised into the three following stages. This is done internally, in order to facilitate benchmarking.

Pre-processing Involves extraction of the feature vectors from the provided query object. This is the responsibility of Cineast.

kNN-lookup Consists in sending the vectors obtained in the pre-processing stage to the storage layer in order to perform a lookup. This stage is determined by the storage engine (i.e., ADAMpro).

Post-processing Involves evaluation of the query results created during lookup phase, such as applying the desired correspondence function. This is the responsibility of Cineast.

As a result of the retrieval workflow, the requester receives a list of segment IDs and relevance scores for each segment in every feature category. Furthermore, the requester receives detailed information about the segments and the associated media objects.



(a) Cineast's ingest workflow.

(b) Cineast's retrieval workflow.

Figure 3.4: Block diagram of the two major workflows in Cineast. Colours are used to indicate the similarity of tasks and thus system components being used for a particular set of steps. Note that the feature modules are used for both ingest and retrieval.

3.2 Vitrivr NG

Vitrivr NG is the new user interface for Cineast, which has been re-created from scratch as part of this thesis. The new UI was built using Angular, a client-side JavaScript framework for single page applications, and has been written in TypeScript 2.1. A complete list of the frameworks and libraries that were used to create Vitrivr NG can be found in Appendix E on page 115.

The main design goal for Vitrivr NG was to build a modular, extendible, web-based user interface that maintains the functionality of the original version (see [10]) and extends it, so as to support queries for and across different modalities. This includes not only building such queries but also presenting the different types of results in a consistent way. The system architecture outlined in Figure 3.5 has been conceived with those design goals in mind. It comprises three major building blocks that are described in this section: The *query builder*, the *query service* and the *gallery*.

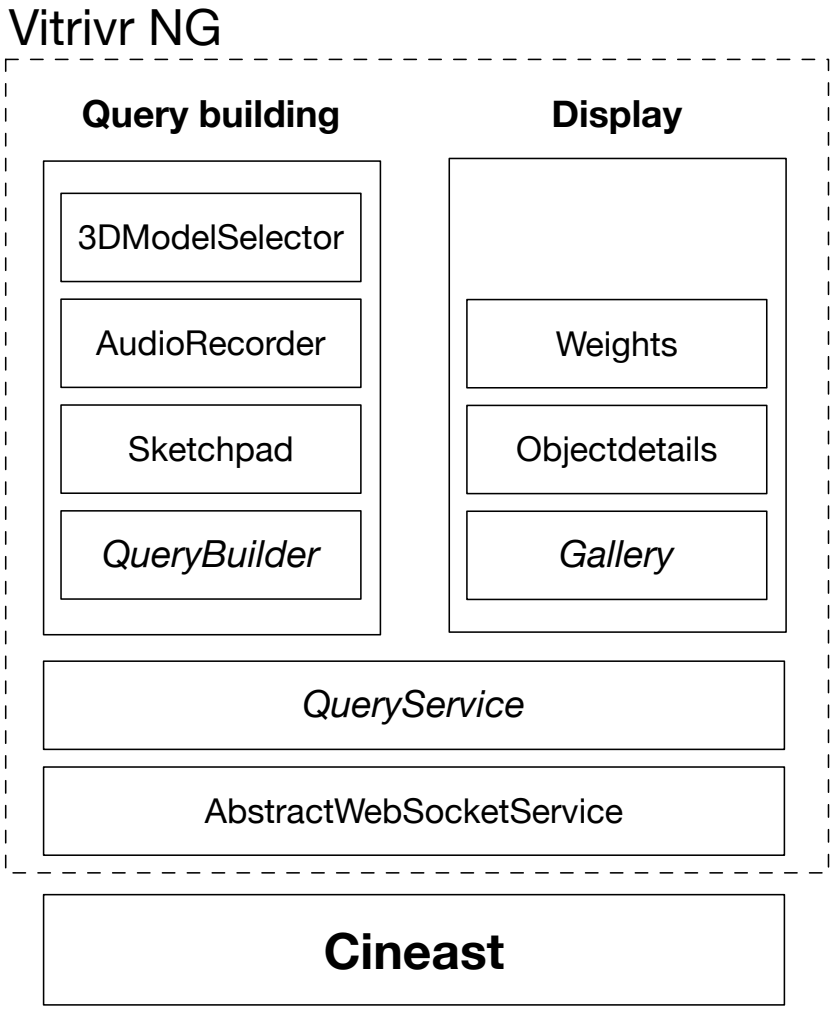


Figure 3.5: Illustration of Vitrivr NG's system context. The major components are the *query builder*, the *query service* and the *gallery*.

3.2.1 Query builder

The query builder's main purpose is to assist the user in formulating similarity queries. It uses the entities described in Section 3.1.2 on page 15. However, the query builder interface depicted in Figure 3.6 tries to abstract these concepts away from the user. In a nutshell, the query builder allows the end user to

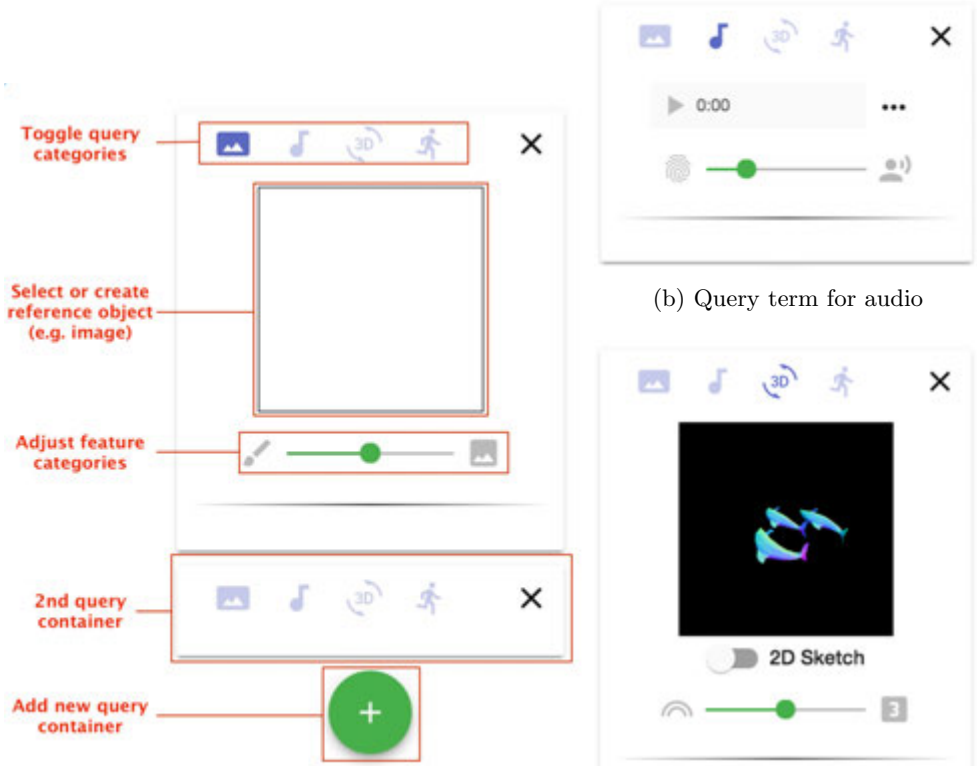
- configure query terms, that is select or create reference documents and setup feature categories.
- combine query terms across modalities using an AND connection, by placing them in the same query component.
- combine query terms across modalities using an OR connection, by placing them in different query components.

Query components can be added directly by the end user. Per component, the UI presents them with a choice of four query categories — image, audio, 3D model and motion. Each category is equivalent to a query term of a specific type and can be switched on and off. It has to be pointed out, though, that each query category can only be selected once per component. Therefore, it is currently impossible to, for instance, connect two image reference documents with an AND relation. However, it is possible to combine an image and an audio clip in a single component, which facilitates cross-modality search.
In every query category, the user can then select the reference document through a query type specific selection dialog. Furthermore, the users can adjust sliders along a certain scale. The scale is different for every type of query term but correlates roughly with the desired specificity of the query. For example, the *rough sketch* setting for images, which is far left on the slider, chiefly includes local and global colour features, whereas the *example image* setting on the other side of the slider encompasses low-level feature descriptors like SURF or HOG. For audio, we used the classification ranging from *fingerprinting* to *audio matching* proposed in [26]. Ultimately, these sliders control the selection of feature categories per query term.

3.2.2 Query service

The query service's main responsibility is to execute queries that have been configured by the query builder through the API that is provided by Cineast. This is done in three steps:

1. The query service receives a complete query from the query builder, that is, a pre-configured graph of query containers and query terms, and serialises it to JSON. Binary data stemming from reference documents is serialised using Base64 encoding.
2. The serialised query is forwarded to the Cineast engine via the WebSocket API. Now, the retrieval workflow described in Section 3.1.6 page 20 takes over and executes the query.
3. Finally, the query service receives and assembles partial results returned by Cineast. Note that Cineast forwards per-category results immediately once they are ready.



(a) Query builder UI with two query components

(b) Query term for audio

(c) Query term for 3D models

Figure 3.6: The query builder UI. Subfigure (a) shows the entire query builder UI. The important UI elements are labelled in the figure. The depicted query component only has one active query term. In the case of multiple terms, the UI elements are stacked vertically. Subfigures (b) and (c) depict the other two types of query terms. A click on the reference document preview opens a separate dialog for selection. I the case of images, this dialog encompasses a simple sketchpad and in the case of audio, a simple audio recorder.

The query service uses pluggable components that take care of the final fusion included in step three. Currently, only one such fusion module is in use, which averages the per-category scores based on adjustable weights per feature category[13]. In the future, modules that support different fusion strategies could be added.
The query service itself leverages a reactive programming pattern based on RxJs[14] to communicate with other system components. Any component interested in query results, such as the gallery, may subscribe to the observable it exposes. Through this observable, these components receive updates about relevant changes to query results and may act on them.

3.2.3 Gallery

The gallery's main responsibility is to display the results of a query. In order to be informed about changes to the result set, it subscribes to the observable exposed by the query service.

[13] There is a dedicated UI component to adjust those weight.
[14] RxJS is integrated into Angular, see http://reactivex.io/rxjs

Whenever the results are updated, the gallery immediately adjusts its visual representation. That representation is centered around *tiles*, which are arranged in a grid. Figure 3.7 depicts a single row of the gallery.

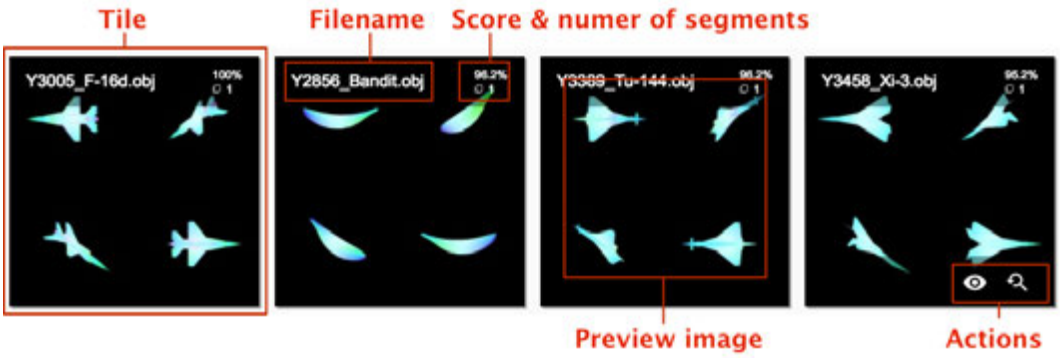


Figure 3.7: First row of the gallery component. The important UI elements are labelled in the figure. Every tile bears information about the media object's filename, its similarity score and the number of segments that were considered to be similar to the current query. Furthermore, a preview image for the media object is displayed. A toolbar provides access to further actions.

A tile in the gallery represents a single media document. This is a deliberate renunciation of the visualisation concept used in the original version of vitrivr, where individual segments were directly listed in the results. The reason we refrain from displaying segments in the gallery is, that some modalities intrinsically comprise a single segment only (see Section 3.1.1). We argue that those documents would be visually outweighed by objects that have multiple segments, regardless of the actual relevance. How many segments were found to be similar to the query is, however, indicated directly under the similarity score.
Furthermore, the tile displays a preview image and the name of the media document. The preview image is determined by the most relevant segment of the document and must be prepared outside of Vitrivr NG[15]. The action bar in a tile provides access to further functionality. Currently, the following actions are supported:

Object details Navigates to the page that depicts details about the selected object. Those details include a list of all segments for that particular object, a preview of the original media file and information about available object metadata. Embedded UI components to preview images, video, audio and 3D models have been added as well.

More-Like-This Uses the selected document as a reference for a new query. More-Like-This queries leverage the available feature vectors for the selected segment. Hence, no feature extraction is performed. Instead, Cineast looks those feature vectors up and uses them to execute the similarity search.

The results in the gallery are ordered by similarity score in descending order. There is a dedicated UI component embedded in a sidebar that can be used to re-rank the results by

[15] There is an option in Cineast to produce preview images for segments during extraction.

assigning different weights to the query categories. This re-ranking can be done without actually re-executing the query.

4

Content-Based Image Retrieval

An image is a visual impression obtained by a camera or some other device and/or displayed on a computer or video screen. At its lowest level, an image can be easily described in technical terms. Intrinsically, though, images often convey higher level concepts that are very difficult to grasp for a computer system.
Conceptually, an image can show many different things ranging from a photorealistic representation of reality to a rough sketch of some object. Similarity between images or parts thereof can therefore occur on many different levels. For the purpose of image retrieval, mostly technical descriptors of the image are being used, even though there are also methods that take human perception or abstract concepts into consideration.

4.1 On computer graphics

When talking about images in this thesis, we refer to raster-, or bitmap images unless otherwise stated. This is the most common form to digitally represent graphics that exhibit a high level of detail. A bitmap image takes the form of a two-dimensional array of *picture elements* or *pixels* p_{xy} and, hence, has an extent in x and y direction. Depending on the type of bitmap image, multiple such arrays — so-called channels — are combined to form the final visual.

Binary image Consists of a single channel and $p_{xy} \in \{0, 1\}$ (black or white)

Grayscale image Consists of a single channel and $p_{xy} \in [0, 1]$. The values of p_{xy} give indication about the brightness of the pixel - the higher the value, the brighter.

Colour image Consists of multiple channels c and $p_{cxy} \in [0, 1]$. The values of p_{cxy} give indication about the brightness of the pixel - the higher the value, the brighter.

The most common way to digitally represent a coloured image is in the RGB colour space. RGB is an additive colour model that knows three colour components — red, green and blue. Every colour can be represented by additively mixing these components in a certain ratio. That ratio is expressed by the combined values $p_{cxy} \in [0, 1]$ in each channel. See Figure 4.1 for a visual illustration of both a grayscale and a colour image in the RGB space.

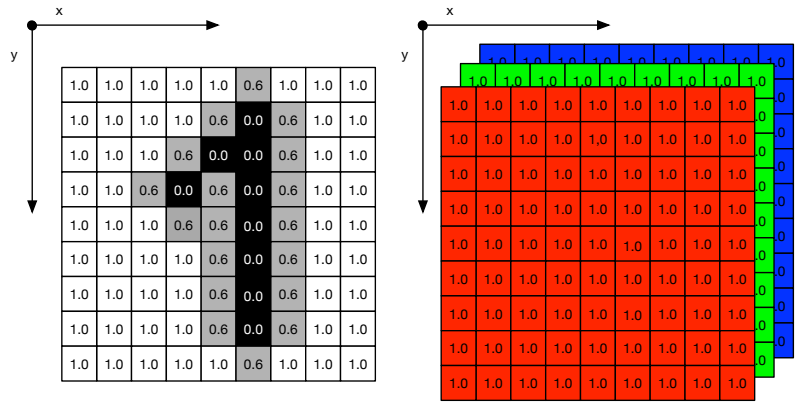


Figure 4.1: Array representation of a 9×9 bitmap image. On the left, one can see the data structure for a grayscale image and on the right the equivalent structure for a coloured image in RGB space.

For the purpose of feature extraction, the aforementioned representation of a bitmap image can be leveraged in several ways. One very simple approach is to interpret every pixel as a discrete three-dimensional colour vector.

$$\mathbf{p_{xy}} = \begin{bmatrix} p_{rxy} \\ p_{gxy} \\ p_{bxy} \end{bmatrix}, \text{ with } p_{rxy}, p_{gxy}, p_{bxy} \in [0, 1]$$

Using this representation, it becomes a straightforward task to, for instance, calculate an average colour (globally or in a specific region), identify the most dominant colours or to derive a colour histogram of the image [57]. In fact, several colour features that have been part of Cineast ever since the original publication are based on this kind of representation. We refer to [9, 10] for more information.
In contrast, one can regard the 2D array of pixel values as a discrete function $\tilde{f}(x, y)$ that returns the colour value of pixel p_{xy} at the indices x and y. This is often done channel-wise or on a grayscale version of the image and it opens the way to a lot of different pattern recognition techniques that can be applied so as to analyse the information in the image. Some of these approaches are briefly described in the following sections. We refer to [14] for more information on pattern recognition in images.

4.2 Image retrieval

For the purpose of information retrieval, different aspects of an image can be considered when designing a feature descriptor. Colour, texture and shape are the most prominent. As we have argued in the previous section, colour information is straightforward to extract directly from the native image representation. Moreover, colour properties are usually invariant to the size of the image; that is, the colour distribution is not affected when resizing it [57].
Texture descriptors, on the other hand, try to identify and describe visual patterns in the image. There is no clear definition of what constitutes a texture, even though we are all able to describe them in terms like “coarse” or “smooth”. It is a difficult concept to represent

and mostly determined by the distribution of grey levels in a particular region [14].
For a wide range of applications, including image retrieval, the shape of an object of interest in an image is another important aspect to consider. Describing such a shape is always a two-step process because in order to describe it, one first needs to be able to identify and extract the shape in a reliable fashion [14] — this is called *image segmentation*.

4.2.1 SURF: Speeded Up Robust Features

SURF was proposed by H. Bay et al. [19] and can be considered an optimised version of the SIFT [58] algorithm. It can be used to detect and describe local features in an image. In a first step, SURF identifies interest points at different scales by looking for local maxima in the determinant of the Hessian Matrix calculated at every point of the image. This step is called *blob detection* and the method is called *Determinant of Hessian (DoH)*. SURF then assigns an orientation to each interest point and constructs a square region centered around that point facing the direction of the aforementioned orientation vector. For every such square region, a sum of *Haar wavelet* responses is obtained for a 4×4 subregion. Concatenation of these values yields a 64-dimensional SURF descriptor for every interest point. As for SIFT, those descriptors are to some extend robust to translation, rotation and scaling as well as changes in illumination.

4.2.2 HOG: Histogram of Oriented Gradients

The concepts behind *Histogram of Oriented Gradients (HOG)* were originally proposed by Robert K. McConnell of Wayland Research Inc. in 1986 [59]. Ever since, it has been used for object detection in images. The idea behind HOG is that an object's appearance can be described by the distribution of intensity gradients. Therefore, in order to compile HOG features, an image is evenly divided into small connected regions, called cells. Those cells may either be circular (C-HOG) or rectangular (R-HOG). For each cell, a histogram of of gradient directions is obtained and the concatenation of those histograms forms the feature vector of a cell. The size of that vector depends on the concrete settings.

4.2.3 Bag of Features

Bag of Features (BoF) or Bag of Words (BoW) refers to a simple model that can be used to aggregate local descriptors into global feature vectors [23]. This technique involves prior generation of a vocabulary of visual words, hereinafter referred to as *codebook*. Such a codebook is usually generated from an existing reference image collection. During codebook generation, local descriptors — such as SURF [19], SIFT [58] or HOG — are extracted for every image in the reference collection, which yields an extensive number (typically between 10^6 and 10^8) of feature vectors $\mathbf{v_i}$ of dimension d. In a second step, those vectors are clustered by k-means clustering, which yields k vectors $\{\mathbf{c_1} \dots \mathbf{c_k}\}$ of dimension d. The vector $\mathbf{c_i} \in \{\mathbf{c_1} \dots \mathbf{c_k}\}$ can be thought of as the center of the i-th cluster and the complete set of those centers forms the codebook.
When generating features from an example image for the purpose of ingest or retrieval, n

local descriptors are extracted for the respective image using the same method that was used during codebook generation. Those descriptors are then assigned to the k entries in the codebook. There are multiple ways of doing this. The simplest approach is to perform a hard assignment; that is, to decide for each local descriptor which of the k words is closest and then assign it to that word. Repeating this for all n descriptors yields a k-dimensional feature vector $\mathbf{f}$ for the image, which is technically a histogram that tracks the assignment of local descriptors to codebook entries.

4.2.4 Centroid distance and Fourier descriptors

The centroid distance was proposed by [60] as a feature for describing shapes in images. It expresses the distance of the boundary points of such a shape, further referred to as *contour*, from the shape's centroid. There are several techniques to obtain shapes and shape contours from images, which shall not be detailed further in this section[16]. However, once a shape has been obtained, the contour $c = \{\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \dots \mathbf{p}_{N-1}\}$ of length N is the subset of all pixels $\mathbf{p}$ in the image that circumscribe the shape. The function $\tilde{c}(n)$ then returns the n-th pixel in that contour. An arbitrary element in the contour can be selected to be n_0, that is, the first element of the contour.

$$\tilde{\mathbf{c}}(\mathbf{n}) = \mathbf{p}_n = \begin{bmatrix} p_{nx} \\ p_{ny} \end{bmatrix}, \ \forall\, n \in \{0, 1, 2, \dots, N-1\} \tag{4.1}$$

The centroid distance $\tilde{r}(n)$ can then be defined as shown in equation 4.2, where c_x and c_y denote the x and y portion of the shape's centroid and p_{nx} and p_{ny} are the x and y coordinates of the pixel $\mathbf{p}_n$. The shape's centroid is simply the arithmetic mean of all the pixel coordinates in the shape's contour.

$$\tilde{r}(n) = \sqrt{(p_{nx} - c_x)^2 + (p_{ny} - c_y)^2} \tag{4.2}$$

In order to obtain a compact descriptor, one calculates the Fourier transform of $\tilde{r}(n)$ to generate its Fourier coefficients a_k (see Chapter 5 for a brief description of the DFT). Using the Fourier coefficients instead of $\tilde{r}(n)$ brings several advantages. Firstly, it's a compact and constant-length representation of $\tilde{r}(n)$. Secondly, the level of detail can be adjusted by including or excluding coefficients. And lastly, the transformation makes the descriptor invariant to the choice of n_0.

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} r(n) e^{\frac{-2\pi i n k}{N}}, \ n \in \{0, 1, 2 \dots, N-1\} \tag{4.3}$$

The Fourier coefficients a_k can now be used to derive a feature vector that describes the shape. One can directly use the magnitude of a_k as feature vector components. Due to the nature of the Fourier transform, low order coefficients capture the low frequency components and thus general properties of the contour while inclusion of higher order coefficients increases the level of detail captured by the descriptor.

[16] We use the functionality included in the BoofCV library for image segmentation.

4.2.5 Zernike polynomials and Zernike moments

Zernike moments can be leveraged as features for pattern recognition or image retrieval as proposed by [60]. They are very well suited to compactly represent the information in an image and perfect candidates for region or shape descriptors.
Zernike polynomials are a set of orthogonal polynomials that are defined on the unit disk [14, 61] and play an important role in the domain of optics. Their definition is given by equation 4.4 (see Figure 4.2 for a visualisation of the 15 first Zernike polynomials).

$$Z_n^m : [0,1] \times [0,2\pi] \to \mathbb{C}, \quad (r,\theta) \rightarrowtail R_n^m(r) e^{im\theta} \text{ with } n, m \in \mathbb{N} \text{ and } |m| \leq n \tag{4.4}$$

The indices n and m are called orders of the Zernike polynomial, sometimes also referred to as *radial and azimuthal degree*. They are constrained such that $n - |m| \mod 2 = 0$ because of the radial polynomials $R_n^m(r)$, which become zero otherwise. $R_n^m(r)$ is generally defined as follows:

$$R_n^m(r) = \sum_{s=0}^{\frac{n-|m|}{2}} (-1)^s \binom{n-s}{s} \binom{n-2s}{\frac{n-|m|}{2} - s} r^{n-2s} \tag{4.5}$$

Zernike polynomials form an orthogonal basis of the space of functions defined on the unit disk. The orthogonality relation is captured in equation 4.6.

$$\int_{\theta=0}^{2\pi} \int_{r=0}^{1} Z_n^m(r,\theta)^* Z_{n'}^{m'}(r,\theta)\, d\theta dr = \frac{\pi}{n+1} \delta_{nn'} \delta_{mm'} \tag{4.6}$$

Due to their orthogonality, a piecewise function $f(x,y)$ that is defined on the unit disk can be expanded into a linear combination of Zernike polynomials as outlined in equation 4.7.

$$f(x,y) = f(r\cos(\theta), r\sin(\theta)) \approx \sum_{n=0}^{N} \sum_{m=-n}^{n} a_{nm} Z_n^m(r,\theta) \tag{4.7}$$

The complex coefficients a_{nm} are called Zernike moments and can be obtained by calculating the projection of the function $f(r\cos(\theta), r\sin(\theta))$ onto the respective Zernike polynomial.

$$a_{nm} = \frac{n+1}{\pi} \int_{\theta=0}^{2\pi} \int_{r=0}^{1} Z_n^m(r,\theta)^* f(r,\theta)\, d\theta dr \tag{4.8}$$

In the case of an image, $\tilde{f}(x,y)$ is a discrete function that returns the pixel value p_{xy} at position (x,y). Hence, the integral in equation 4.8 becomes a sum as in equation 4.9, where $r = \sqrt{x^2+y^2}$ and $\theta = \text{atan2}(x,y)$. The arguments x and y, that is the image coordinates, must be normalised such that $r \leq 1$.

$$a_{nm} = \frac{n+1}{\pi} \sum_{\theta=0}^{2\pi} \sum_{r=0}^{1} Z_n^m(r,\theta)^* \hat{f}(r,\theta) \tag{4.9}$$

Zernike moments can represent an image compactly and can even be used for later reconstruction of the image function $\tilde{f}(x,y)$ [62] as illustrated in Figure 4.3. For the purpose of feature representation, the magnitude of the moments a_{nm} can be directly used as components in the feature vector. Low order moments capture the rough character of the shape whereas higher order moments increase the level of detail.

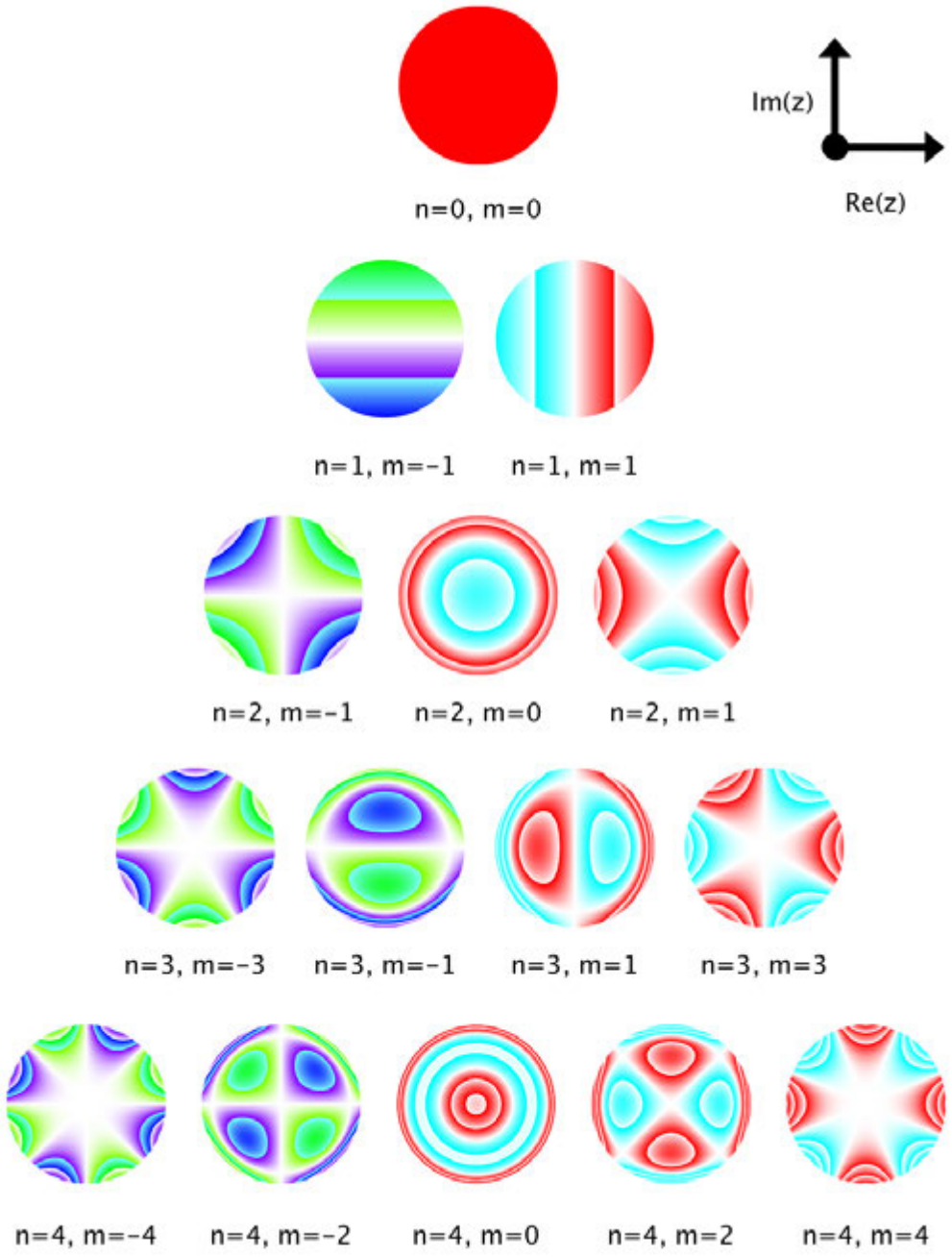


Figure 4.2: Domain colouring plots of the first 15 Zernike polynomials in the complex plane ($n < 5$), ordered vertically by radial degree (n) and horizontally by azimuthal degree (m). Hue gives an indication about the argument of the complex result ($\theta \in [0, 2\pi]$) and the saturation is proportional to its modulus ($r \in [0, 1]$)

.

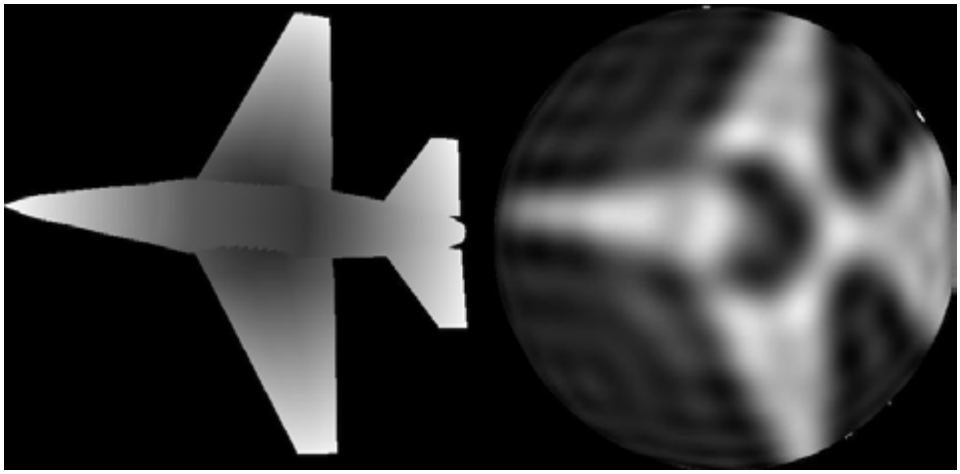


Figure 4.3: Reconstruction (right) of an image (left) from its Zernike moments. Moments up to order $n = 25$ were considered for this reconstruction.

4.3 Implementation

All image features described hereinafter operate on a single image. In the case of video segments, the most representative frame of that segment is used as a proxy for the segment. Therefore, all image features can be leveraged both for pure image retrieval as well as for video retrieval (QbE). This is also true for all features that were created as part of [9, 10]. Cineast currently does not include pure image feature modules that make use of the Zernike moments and centroid distance function methods introduced in the previous sections. Instead, these techniques are leveraged to power feature modules for 3D model retrieval. See sections 6.3.2 and 6.3.3 for more information.

4.3.1 SURF codebook feature module

The SURF feature modules in Cineast combine the Speeded Up Robust Features [19] method with a Bag of Words (BoW) model [23], as described in Section 4.2.3. Two versions of the feature modules exist:

SURFMirflickr25K256 Based on a 256 entry SURF codebook generated from the Mirflickr25k image collection [63].

SURFMirflickr25K512 Based on a 512 entry SURF codebook generated from the Mirflickr25k image collection [63].

The BoofCV [64] library is used to generate the SURF features. We apply the *FH-9* settings for SURF feature extraction — as described in the original paper — both during codebook generation and online/offline feature extraction. A hard assignment strategy is employed to create the histogram from the local SURF descriptors.
The resulting histogram vector (256 dimensional or 512 dimensional) is either persisted during ingest or directly used for lookup during retrieval.

4.3.2 HOG codebook feature module

The HOG feature modules in Cineast combine the Histogram of Oriented Gradients [59] technique with a Bag of Words (BoW) model [23], as described in Section 4.2.3. Two versions of the feature modules exist:

HOGMirflickr25K256 Based on a 256 entry HOG codebook generated from the Mirflickr25k image collection [63].

HOGMirflickr25K512 Based on a 512 entry HOG codebook generated from the Mirflickr25k image collection [63].

We use the BoofCV library to generate the HOG features and apply the HOG settings proposed in [65] both during codebook generation and online/offline feature extraction. This yields a 128 dimensional HOG descriptor per cell. A hard assignment strategy is employed to create the histogram from the local HOG descriptors.
The resulting histogram vector (256 dimensional or 512 dimensional) is either persisted during ingest or directly used for lookup during retrieval.

4.3.3 Feature categories

The feature categories for image retrieval were defined empirically by combining different modules and experimenting with different weights. The following categories were created as part of this thesis:

localfeatures Consists of the *SURFMirflickr25K512* and *HOGMirflickr25K512* features using weights of 1.75 and 1.0 respectively.

localfeatures_fast Consists of the *SURFMirflickr25K256* and *HOGMirflickr25K256* features using weights of 1.75 and 1.0 respectively.

5

Content-Based Music Retrieval

Physically, sound is a vibration that propagates as an audible, longitudinal wave of pressure and displacement through an elastic medium like water or air [66]. When that wave reaches the human ear, it is relayed through the auditory system and converted to an electrical signal on the cochlear nerve, which is ultimately processed and interpreted by the brain. This fundamental, physical and physiological process gives rise to many different perceptual phenomena that include music, speech and noise. Interestingly, the perceptual properties of those phenomena are very different even though the underlying process is similar. This is probably one of the reasons why the domain of content-based audio retrieval is divided into several subdomains, of which CBMR constitutes a major one.

5.1 On audio signal processing

A simple, stationary waveform $y_s(t)$ can be mathematically described as a sinusoidal function as shown in Equation 5.1, where t represents the time in seconds, f denotes the frequency of the wave, ϕ its phase offset and A the maximum of its pressure amplitude [66]. The frequency f in Hertz (Hz) expresses the number of times a cycle is repeated per second. The inverse of the frequency $T = \frac{1}{f}$ is called period.

$$y_s(t) = A\sin(\omega t + \phi),\ \omega = 2\pi f \tag{5.1}$$

A natural audio source produces waveforms that are far more complex than the simple example above. Such a waveform $y_c(t)$ can be thought of as a *linear combination* or *superposition* of umpteen simple sinusoids with different frequencies and phases as indicated by the following equation.[17]

$$y_c(t) = \sum_{k=1}^{K} A_k \sin(\omega_k t + \phi_k) \tag{5.2}$$

An audio signal is an electrical representation of such a complex waveform. For the purpose

[17] The difference between a pure sinusoid and a (relatively simple) superposition of partial signals is also illustrated in figures A.1 and A.2 in Appendix A.

of information processing, such a signal can be converted to a stream of discrete values through a process called *sampling*. This results in a time-domain representation of the signal, which is suitable to extract information about its temporal evolution. The stream can also be converted to a frequency-domain representation, through a method called Discrete Fourier Transform. In the frequency domain, the distribution of energy across different frequencies becomes visible.

5.1.1 Time-domain representation

In the time domain, audio is often modeled as PCM data, which is a method to convert an analog signal into a discrete sequence of numeric values. In order to do so, the signal is sampled at a regular interval T_s, where $f_s = \frac{1}{T_s}$ is called *sampling rate*. For each sample point, the momentary amplitude of the signal is captured and expressed as a discrete value, which is called *sample*.[18]

According to the *Nyquist-Shannon sampling theorem*, which is mathematically expressed by Equation 5.3, the sampling rate limits the highest frequency that can be resolved in a signal. For instance, at a sampling rate of 44 100 Hz, frequencies above 22 050 Hz are lost. It is worth noting here, that the human hearing range is commonly given as 20 Hz to 20 000 Hz [67]. This explains why common sampling rates for music lie between 22 050 Hz (low quality) and 48 000 Hz (high quality).

$$f_{max} = \frac{f_s}{2} \tag{5.3}$$

In Cineast, interleaved 16-bit PCM is used as internal time-domain audio representation. That is, each audio sample is stored as a signed short value between -2^{15} and $2^{15} - 1$. For multichannel audio, samples that belong to the same time point but stem from different channels are stored adjacent to one another. The following expression illustrates a set of samples for an audio signal $\tilde{x}$ with a left and a right channel (stereo sound).

$$\tilde{x} = \{s_{1l}, s_{1r}, s_{2l}, s_{2r}, s_{3l}, s_{3r}, \ldots, s_{nl}, s_{nr}\}$$

Throughout this section, $\tilde{x}$ shall refer to a sequence of samples as illustrated above and the discrete function $\tilde{x}(n)$ shall return the n-th sample of that signal. For the sake of simplicity, we assume that we operate on single-channel audio. Note that in this case, the relationship between time t in the analog signal, sample index n and the sampling interval T_s is as follows:

$$x(t) \approx \tilde{x}(T_s n) \tag{5.4}$$

The duration d of an audio file can then be expressed as the product of the total number of samples N and the sampling interval T_s:

$$d \approx T_s N \tag{5.5}$$

[18] The sampling process is also illustrated by figures A.1 and A.2 in Appendix A.

5.1.2 Frequency-domain representation and Fourier analysis

A lot of information in an audio signal cannot be directly inferred in the time domain. Therefore, Fourier analysis is an important tool in audio signal processing as it facilitates the conversion of a time-domain signal into the frequency domain.
The theory of Fourier expansion states that any piecewise continuous, periodic function $f(t) = f(t+T)$, with $T = \frac{2\pi}{\omega_0}$, can be expanded into a linear combination of trigonometric functions, namely sines and cosines, as outlined in Equation 5.6.

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\omega_0 t) - b_k \sin(k\omega_0 t),\ a_k, b_k \in \mathbb{R} \tag{5.6}$$

This representation is also known as Fourier series. Using Euler's formula, it can be restated in its more common, complex form where $c_k = a_k + ib_k$

$$f(t) = c_0 + \sum_{k=-\infty}^{\infty} c_k e^{ik\omega_0 t},\ c_k \in \mathbb{C} \tag{5.7}$$

The Fourier coefficients c_k can be obtained by evaluating the following integral for different values of $k \in \mathbb{N}$.

$$c_k = \frac{1}{\phi} \int_0^{\phi} f(t) e^{-ik\omega_0 t}\, dt \tag{5.8}$$

The theory of the Fourier integral generalises the concept of Fourier expansion to non-periodic functions. It states that for any integrable function $f(t)$ there exists a Fourier transform $F(\omega)$ as defined in Equation 5.9. $F(\omega)$ is also called *spectral function* or *frequency distribution*.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt \tag{5.9}$$

Conceptually, the Fourier integral calculates the projection of the function $f(t)$ onto the set of trigonometric basis functions (sines and cosines). This yields $F(\omega)$, which quantifies the contribution of the different basis functions to the function $f(t)$. This works because the space in which $f(t)$ resides has the properties of an *inner product space* and sine and cosine functions constitute an infinite, orthogonal basis of that vector space.
There is also an inverse transform that allows reconstruction of the original function from its spectral function as shown in Equation 5.10.

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t}\, d\omega \tag{5.10}$$

As it turns out, the Fourier transform can be obtained for both continuous as well as discrete functions, like for instance an audio signal. The latter is referred to as *Discrete Fourier Transform (DFT)* and a common DFT algorithm is the *Fast-Fourier Transform (FFT)*. In the discrete case, the integrals in Equation 5.9 and 5.10 become sums and $F(\omega)$ again becomes a discrete sequence of Fourier coefficients c_k. According to equation 5.10, the original function $f(t)$ can then be expanded into a linear combination of the trigonometric basis functions using the coefficients c_k as weights.

If we now consider Equation 5.2 again it becomes apparent that audio signals are per definition linear combinations of trigonometric basis functions, sinusoids in our case. Hence, the k-th Fourier coefficient c_k gives an indication of how much the k-th sinusoid contributes to the entire signal. From Equation 5.1 we can directly infer that the sinusoids differ in two attributes: frequency and phase. These are the properties captured by the DFT and encoded in c_k.[19] They indicate how much energy is contained in a specific frequency band.

$$\Delta f = \frac{\Delta\omega}{2\pi} = \frac{f_s}{N} \tag{5.11}$$

Here, N denotes the number of samples used to calculate the FFT. The width of a frequency band Δf is called the *resolution* of the discrete spectrum and the bands themselves are sometimes referred to as *frequency bins*.

5.1.3 Short-Term Fourier Transform

Audio signals are statistically non-stationary; that is, they vary with time. Calculating the DFT for the entire signal of a piece of music yields an energy distribution for the entire piece. Time-local variations are averaged and lost during the transformation [14].
The *Short-Term Fourier Transform (STFT)* is a variant of the DFT that allows for time-local frequency analysis of an audio signal. It offers a trade-off between time-domain and frequency-domain representation. The basic assumption behind the STFT is that the signal becomes stationary if the window one analyses is narrow enough. The mathematical definition of the STFT is captured in Equation 5.12, where $X(\tau, \omega)$ denotes the value of the STFT for a specific time point and a given angular frequency ω, $\tilde{x}$ denotes the audio signal, w a window function and τ is the time point on which the observation is centered.

$$X(\tau, \omega) = \sum_{n=-\infty}^{\infty} \tilde{x}(n) w(n - \tau) e^{-i\omega n} \tag{5.12}$$

In words, the STFT is obtained by multiplying the audio signal, that is, the sequence of PCM audio samples, with a window function of size N, which is non-zero only for a specific interval - the window. That operation yields scaled sample values in a window of constant length, centered around τ, and zero outside of that window. The center of the window is moved along the entire audio signal using a constant hop size $h = \tau_2 - \tau_1$ between two successive windows. For every windowed signal, the DFT is calculated. By doing so, a sequence of time-local Fourier transforms of the audio signal is compiled. This allows for reasoning about the temporal development of the energy distribution across frequency bins. In practice, the choice of window function, the window size N (in samples) and the overlap between two successive windows o (in samples) are parameters for the STFT. Often in literature, N and o are expressed in seconds instead of samples (see Equation 5.5 for conversion). The simplest window function is the *rectangular window* given in Equation 5.13.

[19] The form of an FFT of a signal is also illustrated by figures A.1 and A.2 in Appendix A.

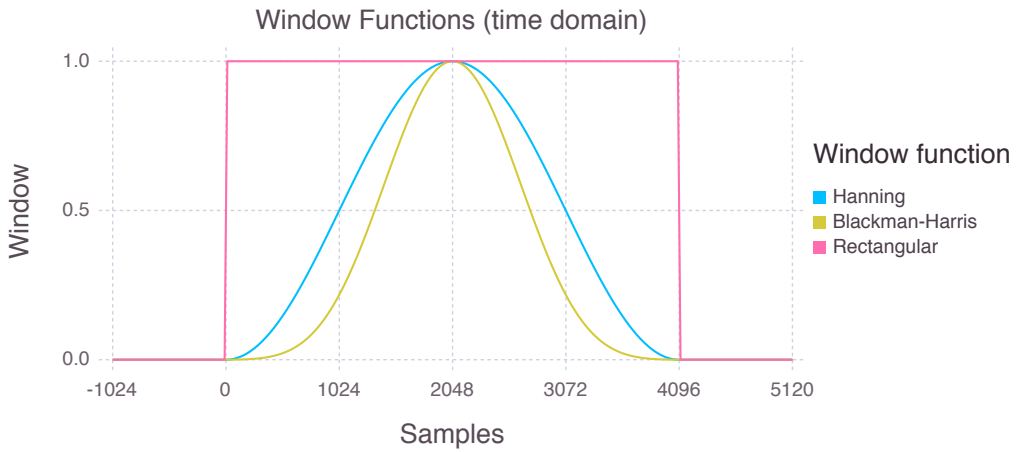


Figure 5.1: Illustration of the three common window functions used in digital signal processing: *Rectangular window*, *Hanning window* and *Blackman-Harris window*. The depicted windows have a length of 4096 samples.

$$w(n) = \begin{cases} 1, & \text{if } 0 \leq n \leq N \\ 0, & \text{otherwise} \end{cases} \tag{5.13}$$

Despite its simplicity, though, it is rarely used because it has poor *spectral leakage* characteristics. Spectral leakage is a phenomenon that arises due to the finite analysis window used in STFT and DFT in general. A specific frequency component of the signal may not be periodic in the window that is being analysed and the periodic extension of such a component may cause discontinuities at the window boundary. Upon expansion of the signal into the linear combination of the trigonometric basis functions, projections of the signal onto these basis functions are calculated. Frequency components in the signal that are not members of the basis - hence, the non-periodic frequency components in the signal - give a non-zero contribution to the entire basis set. Therefore, we observe frequency components in the spectrum that are not actually part of the signal [68].

As every real signal is in fact finite, spectral leakage cannot be fully avoided. An appropriate choice of window function, however, can minimise spectral leakage effects. Those window functions smoothly fade-out the data at the boundaries of the window, so that the periodic extension of the data becomes continuous in many orders of derivative. This is known as as *tapering* [68, 69].

$$w(n) = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi n}{N-1}\right) \tag{5.14}$$

Equations 5.14 and 5.15 show the definitions of two very common window functions and Figure 5.1 illustrates their form in the time domain. Equation 5.14 is known as *Hann window* or *Hanning window* and Equation 5.15 is called *Blackman-Harris window*. Both functions minimise the effect of spectral leakage but exhibit different characteristics in terms of the energy distribution in the resulting spectrum.

$$w(n) = a_0 - a_1 \cos\left(\frac{2\pi n}{N-1}\right) + a_2 \cos\left(\frac{4\pi n}{N-1}\right) - a_3 \cos\left(\frac{6\pi n}{N-1}\right) \\ a_0 = 0.35875, a_1 = 0.48829, a_2 = 0.14128, a_3 = 0.01168 \tag{5.15}$$

The choice of parameters for the STFT emphasises different aspects of the audio signal. For example, larger window N and overlap o result in higher frequency resolution at the cost of time resolution, whereas the choice of window function can be used to influence the quality of the STFT itself. Therefore, those parameters may vary depending on the feature that is being derived from the STFT.

5.1.4 Magnitude and power spectrum

The magnitude $M(\omega)$ and power spectrum $P(\omega)$ of a signal describes the distribution of energy across the different frequency bins. They can be directly obtained from a DFT or STFT. For the latter, the spectrum is again a time-local representation of the energy distribution. Equations 5.16 and 5.17 define the magnitude and power spectrum for a STFT, where N denotes the number of frequency bins yielded by the DFT, which equals to the window size, and α denotes a normalisation constant that depends on the window function.

$$M(\tau, \omega) = \frac{2\sqrt{\|X(\tau, \omega)\|}}{\alpha N} \tag{5.16}$$

$$P(\tau, \omega) = \frac{2\|X(\tau, \omega)\|}{\alpha N} \tag{5.17}$$

Each spectrum encompasses a list of frequency bins (ranging from 0 Hz to some maximum) and the (squared) energy content for each bin. The width of a bin in Hz is given by Equation 5.11. The maximum frequency of a spectrum is limited by the sampling rate of the original audio signal according to the Nyquist-Shannon sampling theorem and the fact that the DFT of a real-valued signal is mirrored at $\frac{N}{2}$. The resolution and hence the number of frequency bins per spectrum depends on the number of samples used for the DFT and, transitively, on the window size N for STFTs.

5.2 Music retrieval

As outlined in the previous section, audio can be seen as a waveform and music is a very complex, perceptual manifestation of such a waveform, as it consists of many different component signals that together make up what humans perceive. In this section, we delve into some concepts surrounding music but we neglect the psychological aspects of its perception, even though it is considered to be important for MIR. The information in this section is largely taken from [31], [67] and [66].

Music is governed by its own set of rules and systematics and it can be characterised in terms of loudness, tempo, timbre, pitch, chroma, rhythm or melody. All these aspects and/or a combination thereof give rise to many different measures for similarity.

The most fundamental attribute of tonality in music is *pitch*, which allows for ordering of a tone on a scale from low to high. As such, it is a proxy for the tone's frequency. For example the concert pitch — the A above the middle C, which is often used as reference pitch — has a fundamental frequency f_0 of 440 Hz. It is worth noting, that the human perception of loudness and different pitches is not linear. This is reflected in the existence of different, psychoacoustic scales like the Bark-scale [70] or the Mel-scale [71] for pitches.
Pitches stemming from a natural source, like for example an instrument, are not just simple sinusoids but complex waveforms that consist of multiple components, called *partials*, which can be divided into a *fundamental frequency* f_0 and *overtones*. Note that the most salient overtones of a pitch are often (but not always) integer multiples of the fundamental frequency, that is $f_n = nf_0, n \in \mathbb{Z}$. These overtones are called *harmonics* of f_0. The difference in energy distribution across partials of a pitch give rise to *timbre*, which is what makes two sounds dissimilar even though both have the same loudness and pitch. For instance, timbre allows us to discern identical pitches played by different sources (e.g., two instruments).
In Western music, the pitch scale is logarithmic, that is adding an interval to a pitch is equal to multiplying the fundamental frequencies by a factor. For the chromatic scale, that factor is $2^{\frac{1}{12}}$ for two adjacent pitches, regardless of where those pitches are located. The constant distance between pitches is a property of an *equal-tempered scale*. Generally, the difference in semitones n between two fundamental frequencies can be expressed as follows.

$$n = 12 \log_2 \left(\frac{f_2}{f_1} \right) \tag{5.18}$$

There are different notations for pitch, like the *Helmholtz pitch notation* [72], which is used for naming musical notes on the Western chromatic scale, or the MIDI notation, which assigns an integer number to the pitch based on its position on the chromatic scale [73]. Equation 5.19 can be used to convert a frequency to the associated MIDI number, where f_{ref} is the frequency of the A above the middle C (A440, MIDI: 69), i.e. 440 Hz.

$$d = 69 + 12 \log_2 \left(\frac{f}{f_{ref}} \right) \tag{5.19}$$

In the musical context, pitch is often represented by a two-dimensional model called the pitch helix (see Figure 5.2) as proposed by [74]. The underlying observation is, that pitches one octave (i.e. 12 semitones) apart from one another have similar, perceptual colour. The dimensions in this model are called *height* and *chroma* or *pitch class*.
Chroma features capture harmonic and melodic characteristics of a musical piece while at the same time being robust to changes in timbre. This is why such features are good candidates for MIR [30, 75].
A sequence of tones or pitches that are perceived as a single entity is referred to as *melody*. Simply put, melody is a combination of pitch and rhythm. A musical piece may consist of a single melody (*monophonic music*) or may comprise multiple, discernible melody lines (*polyphonic music*). Melody representations are especially important when it comes to QbH. Unfortunately, extraction of melody from polyphonic music is a non-trivial task [33, 40, 41].

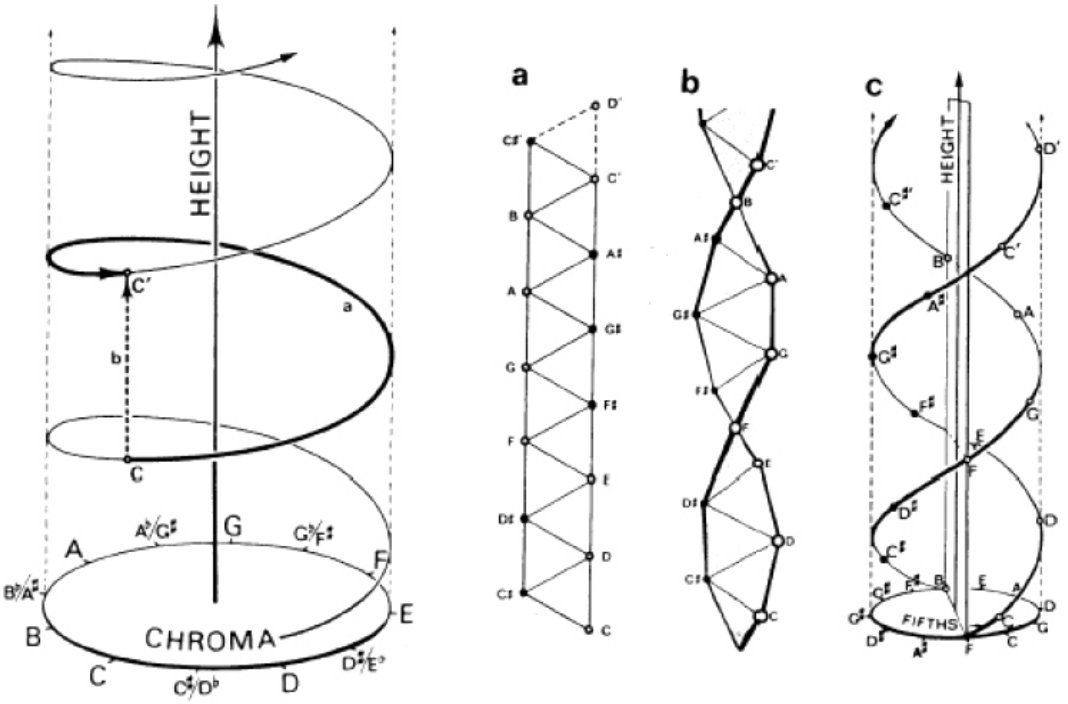


Figure 5.2: The pitch helix proposed by [74]. If one takes the tones on the 12-semitone chromatic scale (A,A#,B,C,C#,D,D#,E,F,F#,G,G#), the different harmonic overtones of a pitch belong to the same pitch class. As one goes up in the scale, the height of the pitch increases steadily whereas chroma recurs. (Source: R.N. Shepard, 1982)

5.2.1 HPCP: Harmonic Pitch Class Profiles

Pitch Class Profiles (PCP) also known as pitch class histograms or Pitch Class Distributions (PCD) are chroma features widely used in MIR. The different terms often refer to the same thing conceptually, even though the calculation details may vary significantly between implementations.

At their core, PCPs capture the intensity of the different tones (or pitches) in an audio signal. Fujishima et al. [30] proposed a twelve-dimensional PCP vector in which each component represents the energy in one of the twelve semitone pitch classes on the equal tempered scale (A,A#,B,C,C#,D,D#,E,F,F#,G,G#). The vector is obtained by mapping each frequency bin l from the power spectrum to one of the twelve pitch classes according to Equation 5.20. That equation is derived from Equation 5.19, which can be used to assign a frequency to the MIDI index of the corresponding pitch. The value of f_{ref} is usually chosen to be 440 Hz, which corresponds to the A above the middle C and is often used as pitch reference.

$$M(l) = \begin{cases} -1, & \text{if } l = 0 \\ ||12 \log_2 \left(\frac{f_s l}{N f_{ref}} \right) \bmod 12||, & \text{if } l \in \{1, 2, \ldots, \frac{N}{2}\} \end{cases} \tag{5.20}$$

Harmonic Pitch Class Profiles (HPCP) [75] are an optimised variant of PCP feature. Instead of doing a hard assignment of a frequency bin to one of the twelve pitch classes, each bin can contribute to multiple classes. The HPCP is given by Equation 5.21, where $w(n, f_i)$ denotes a weight function and a_i is the linear magnitude of the i-th peak in the power spectrum.

$$HPCP(n) = \sum_{i}^{\#peaks} a_i w(n,i),\ n \in \{1 \dots 12\} \tag{5.21}$$

Only local maximum peaks above a certain threshold are considered when calculating the HPCP. The weight function assures that the contribution of a frequency peak to a pitch class decays rapidly as the distance of the peak's frequency f_i to the pitch classes' center frequency grows larger. That distance is given by Equation 5.22, where m is the integer value that minimises $|d(n,i)|$ and f_n is the center frequency of the n-th pitch class.

$$d(n,i) = 12 \log_2 \left(\frac{f_i}{f_n} \right) + 12m,\ \text{with } m = \arg\min_{m \in \mathbb{N}} |d(n,i,m)| \tag{5.22}$$

Provided the distance in 5.22, the weight function is then given by Equation 5.23, where l is an empirical value set to $\frac{4}{3}$ semitones [75].

$$w(n,i) = \begin{cases} \cos^2 \left(\frac{\pi d(n,i)}{l} \right), & \text{if } |d(n,i)| \leq \frac{l}{2} \\ 0, & \text{if } |d(n,i)| \geq \frac{l}{2} \end{cases} \tag{5.23}$$

5.2.2 MFCC: Mel-Frequency Cepstral Coefficients

Since their proposal by J. Foote in 1994 [28], Mel-Frequency Cepstral Coefficients (MFCC) have been a standard feature for MIR [29]. They offer a compact representation of an audio signal's frequency spectrum.

MFCCs can be derived from a signal's magnitude spectrum. First, the frequency is scaled logarithmically by applying Mel filter banks $H(k,b)$ according to Equation 5.24, where k denotes the index of a frequency bin in the spectrum and $b \in \{1, 2, \dots, B\}$ with B being the number of Mel filters in the filter bank.

$$M'(\tau, b) = \ln \left(\sum_{k=0}^{N-1} |M(\tau,k)| H(k,b) \right) \tag{5.24}$$

The definition of the Mel filter bank is given by Equation 5.25. It is a collection of triangular filters that are centered around a frequency f_c.

$$H(k,b) = \begin{cases} 0, & \text{if } f(k) < f_c(b-1) \text{ or } f(k) \geq f_c(b+1) \\ \frac{f(k) - f_c(b-1)}{f_c(b) - f_c(b-1)}, & \text{if } f_c(b-1) \leq f(k) < f_c(b) \\ \frac{f(k) - f_c(b+1)}{f_c(b) - f_c(b+1)}, & \text{if } f_c(b) \leq f(k) < f_c(b+1) \end{cases} \tag{5.25}$$

The center frequencies are obtained by approximating the frequency range of interest on the Mel-scale, which is a perceptual scale of pitches empirically judged to be equidistant to one another, and then parititioning that range into equidistant parts of size Δf_m according to Equation 5.26. Here, $f_{m,u}$ and $f_{m,l}$ denote the upper- and lower bound frequency of the range on the Mel scale. Equation 5.27 can be used to convert ordinary frequencies to Mel frequencies.

$$m_c(b) = b\Delta m = i \frac{(f_{m,u} - f_{m,l})}{B+1}, b \in \{1, 2, \dots, B\} \tag{5.26}$$

To obtain the center frequencies in Hz, Equation 5.27 is solved for f and the resulting equation is applied to every $m_c(b)$.

$$m = 2595 \log_{10} \left(\frac{f}{700} + 1 \right) \tag{5.27}$$

Finally, the Discrete Cosine-Transform (DCT) is applied to $M'(\tau, b)$ (Equation 5.28), where $c(b)$ is the b-th cepstra of the MFCC.

$$c(l) = \sum_{b=1}^{B} M'(\tau, b) \cos \left(\frac{l\pi}{B} \left(b - \frac{1}{2} \right) \right) \tag{5.28}$$

5.2.3 F0 estimation and pitch tracking

In this thesis, we employ the F0 estimation algorithm proposed in [40], which involves a harmonic summation approach. That is, a *salience function* $s(\tau)$, as in Equation 5.29, is calculated for every F0 candidate in a certain range by summing the intensities of the pitch and its harmonic partials $M(f_{t,m})$, where M is the magnitude spectrum of a time frame. The function $g(\tau, m)$ denotes a weight function and $f_{t,m} = \frac{m f_s}{\tau}$ is the frequency of the m-th harmonic partial of a F0 candidate f_0. Note that we adopt the notation used in the original publication in which $s(\tau)$ is not a function of the fundamental frequency f_0 but the fundamental period τ with $f_0 = \frac{f_s}{\tau}$ and f_s being the sampling rate of the audio excerpt.

$$s(\tau) = \sum_{m=1}^{M} g(\tau, m) M(f_{t,m}) \tag{5.29}$$

In [40] the authors use a weight function $g(\tau, m)$ that has the general form outlined in Equation 5.30. They further use machine learning to determine the parameters α and β. For an analysis window of 96 ms they propose values of $\alpha = 27.0$ and $\beta = 320.0$.

$$g(\tau, m) = \frac{f_s/\tau + \alpha}{m f_s/\tau + \beta} \tag{5.30}$$

The algorithm then consists in calculating the pitch salience from a time-local magnitude spectrum according to equations 5.29 and 5.30. The weighted intensities of the pitch candidate and its harmonics are subsequently subtracted from the spectrum. This is repeated for all pitches in a frequency range (usually between 50 Hz and 5000 Hz). Once the pitch salience for all F0 candidates has been obtained, the candidates with the highest salience are selected. To compensate for F0 estimation error, one usually selects more than one candidate per frame.

In order to connect the F0 candidates per segment into a melody contour, we employ the approach described in [76]. That technique uses statistics, such as mean and standard deviation of pitch salience and the pitch frequency, in order to reject pitch candidates until only one contour remains. We refer to [76], Section II for more details.

5.3 Implementation

All audio features described in this section are based on the STFT derived from a segment's audio signal. For offline processing, all signals are re-sampled to 44 100 Hz stereo and channels are merged into a single channel by averaging during the actual feature extraction. The STFT settings used for each feature are described in the respective sections. All feature modules described herinafter can be used for both pure audio segments as well as video segments with audio.

5.3.1 Audio fingerprint feature module

The audio fingerprint feature module is inspired by the Shazam fingerprinting algorithm [27]. The module identifies the intensity peaks in a power spectrum, which has been derived from a STFT (Hanning window, $N = 0.2\,\text{s}$, $o = 0.0\,\text{s}$). The following frequency ranges are being considered: 30 Hz to 40 Hz, 40 Hz to 80 Hz, 80 Hz to 120 Hz, 120 Hz to 180 Hz, 180 Hz to 300 Hz and 300 Hz to 480 Hz.

The set of peak frequencies for each range forms a characteristic, six-dimensional fingerprint vector for a window of 0.2 seconds. Twenty of those local fingerprints are concatenated into a 120-dimensional shingle vector and multiple such vectors are produced per segment by moving the shingle window by one vector (resulting in $n-20$ shingle vectors for n time-local power spectra).

During retrieval, the set of fingerprints derived from the query-object is compared to the persisted fingerprints. The L^1 distance is used for the kNN-lookup.

5.3.2 HPCP shingle feature module

The HPCP shingle feature modules combine HPCPs [75] with the audio shingle approach proposed in [32]. Their intended use is the identification of different version of a piece of music (e.g. live recording, cover song, remix)

Currently, three versions of the HPCP shingle feature module exist. They consider different frequency bands when they extract the HPCP feature. This was inspired by an idea found in [33].

HPCP12Shingle Considers the full frequency range between 50.0 Hz and 5000.0 Hz.

HPCP12BasslineShingle Focuses on bass line frequencies between 10.0 Hz and 262.0 Hz.

HPCP12MelodyShingle Focuses on melodic frequencies between 262.0 Hz and 5000.0 Hz.

The module concatenates 30 time-local HPCP vectors into a 360-dimensional feature vector, which covers approximately 3 seconds of playback. Multiple vectors are produced per segment by shifting the shingle window by one vector (resulting in $n - 30$ shingle-vectors for n time-local HPCP vectors). The HPCPs are derived from an STFT (Blackman-Harris window, $N = 0.2\,\text{s}$, $o = 0.1\,\text{s}$) of the audio segment. All feature vectors are normalised with respect to the L^2 norm.

Upon retrieval, the set of HPCP shingles derived from the reference document is compared to the persisted HPCP shingles. The L^2 distance is used for the kNN-lookup. Hits are counted per segment and the score is derived from the number of hits for a segment.

5.3.3 CENS shingle feature module

The *Chroma Energy Normalized Statistics (CENS)* shingle combines the idea of CENS features [77] with the shingle approach found in [32]. This combination has been reported by [34]. CENS are derived from chroma features like HPCP. The intended use of CENS shingles is the identification of different versions of a piece of music (e.g. live recording, cover song, remix). One of its strengths is its robustness against speed variations.
Currently, three versions of the CENS shingle feature module exist. They consider different frequency ranges when they generate the CENS feature. This was inspired by an idea found in [33].

CENS12Shingle Considers the full frequency range between 50.0 Hz and 5000.0 Hz.

CENS12BasslineShingle Focuses on bass line frequencies between 10.0 Hz and 262.0 Hz.

CENS12MelodyShingle Focuses on melodic frequencies between 262.0 Hz and 5000.0 Hz.

In our implementation, CENS features are derived from HPCP vectors which in turn are derived from an STFT of the segment (Hanning window, $N = 0.2\,\text{s}$, $o = 0.1\,\text{s}$). This is a different approach than the one reported in the original paper. Once a sequence of normalised HPCP vectors $\mathbf{v}$ has been obtained for a segment, those vectors are quantised component wise according to Equation 5.31 and subsequently convolved, again component wise, with a Hanning window of size $w \in \mathbb{N}$.

$$q(v_i) = \begin{cases} 4, & \text{if } v_i > 0.4 \\ 3, & \text{if } 0.2 \leq v_i < 0.4 \\ 2, & \text{if } 0.1 \leq v_i < 0.2 \\ 1, & \text{if } 0.05 \leq v_i < 0.1 \\ 0, & \text{if } v_i < 0.05 \end{cases} \tag{5.31}$$

The vectors that result from the convolution are downsampled by a factor $d \in \mathbb{N}$. This results in a new sequence of twelve-dimensional vectors. Those vectors are referred to as CENS_d^w and they represent a statistic of the energy distribution over a window of w consecutive vectors. Ten of those CENS vectors are concatenated into a 120-dimensional CENS shingle, which is normalised with respect to the L^2 norm. Multiple such vectors are produced per segment by shifting the shingle window by one vector (resulting in $n - 10$ shingle-vectors for n CENS vectors).
During the offline ingest workflow, a set of CENS_5^{21} shingles is generated and persisted. Upon retrieval, CENS_2^{11}, CENS_5^{21}, CENS_{10}^{41} and CENS_{20}^{81} shingles are derived from the reference document and compared to the persisted CENS shingles using the L^2 distance. The different choices of w and d simulate variations in tempo.

5.3.4 MFCC shingle feature module

The MFCC shingle features combine the idea of MFFC features [28] with the shingle approach found in [32]. In our implementation, we concatenate 39 cepstral coefficients into a single 39-dimensional shingle vector. That vector is then normalised with regard to the L^2 distance. Note that one shingle vector represents approximately 0.5 seconds of audio playback. Multiple vectors are produced per segment by shifting the shingle window by one vector (resulting in $n - 39$ shingle-vectors for n cepstral coefficients).
The MFCCs are derived from a STFT (Hanning window, $N = 0.2\,\text{s}$, $o = 0.1\,\text{s}$) of the audio segment. For MFFC calculation, 23 mel-filter banks are used and frequency contributions lower than 133.0 Hz are ignored. The MFFC module returns 13 cepstral coefficients per time-local DFT.
During the offline ingest workflow, the resulting feature vectors are persisted. Upon retrieval, a kNN-search under the L^2 distance is performed with the feature vectors extracted from the reference document up to a maximum of ten lookups. The results are combined into a ranked list of hits by counting the number of hits per segment.

5.3.5 HPCP average feature module

The HPCP average feature module derive short-term statistics from third tone HPCPs [75] and combines those into a sequence of feature vectors. Its intended use is the identification of different version of a piece of music (e.g. live recording, cover song, remix)
Currently, two versions of the HPCP average feature module exist. They differ in the number of frames that are being considered when calculating the HPCP mean and variance.

AverageHPCP20F36B Combines 20 consecutive HPCP vectors (approximately 2 seconds) into a single feature vector.

AverageHPCP30F36B Combines 30 consecutive HPCP vectors (approximately 3 seconds) into a single feature vector.

The two modules calculate the mean and variance over 20 and 30 consecutive, 36-dimensional third tone HPCP vectors respectively. The HPCP vectors are derived from an STFT (Hanning window, $N = 0.2\,\text{s}$, $o = 0.1\,\text{s}$). Both the mean and variance are stored in the resulting 72-dimensional feature vector. Multiple vectors are produced per segment until all HPCPs derived from a segment have been considered.
During the offline ingest workflow, the resulting feature vectors are persisted. Upon retrieval, a kNN-lookup under the L^2 distance is performed with the feature vectors extracted from the reference document. The results of the lookup are subsequently combined into a ranked list of hits. If a segment has multiple hits, the mean distance is calculated.

5.3.6 Feature categories

The feature categories for audio retrieval were defined empirically by combining different modules and experimenting with different weights. The following categories were created:

audiofingerprint Consists of the *AudioFingerprinting* feature module.

audiomatching Consists of the *CENS12Shingle*, *HPCP12Shingle* and *MFCCShingle* features using weights of 2.0, 1.0 and 0.5 respectively.

hpcpaverage Consists of the *AverageHPCP20F36B* and *AverageHPCP30F36B* features using weights of 1.5 and 0.75 respectively.

5.3.7 Rejected feature modules

After some experimentation, we decided to not continue our efforts on the *HPCP12BasslineShingle*, *HPCP12MelodyShingle*, *CENS12BasslineShingle*, *CENS12MelodyShingle* as especially the bass line versions did not produce accurate results. Instead we included the *HPCP12Shingle* and *CENS12Shingle* in the final version.
Furthermore, we have experimented with a *MelodyEstimate* feature module for QbH but we did not manage to create a functional version for this thesis. Unfortunately, the current versions of melody extraction and pitch tracking described in Section 5.2.3 do not produce correct results in a reliable fashion.

6

Content-Based 3D model Retrieval

In computer graphics, a 3D model is a mathematical representation of a three-dimensional surface. The process of creating 3D models after real-world objects can be compared to sculpting. 3D models have applications in several areas from computer games to the movie industry and, more recently, in additive manufacturing. For the purpose of retrieval, topological and geometrical properties of the models can be exploited.

6.1 On 3D models and computer graphics

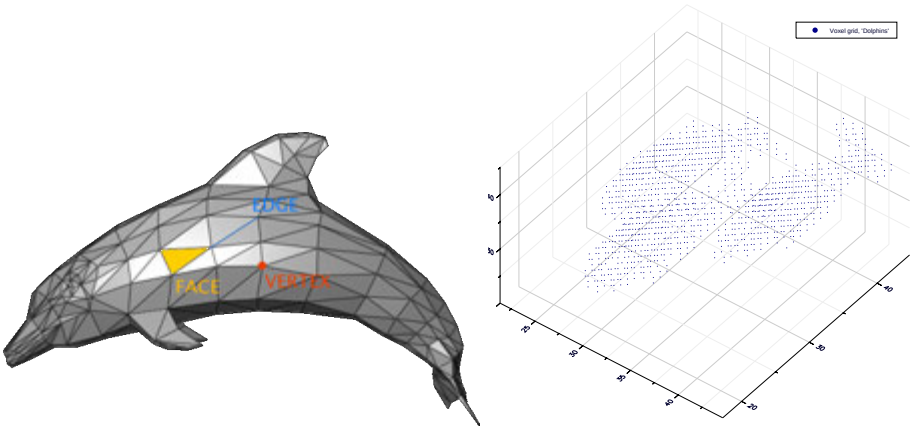
3D models come in many different flavours and exhibit different levels of detail. The most common representation is that of a *polygon mesh*, for which an example is provided in Figure 6.1a. A polygon mesh is a collection of vertices $V = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_i}\}$, edges $E = \{\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_j}\}$ and faces $F = \{f_1, f_2, \ldots, f_k\}$ that conjointly define the surface of a polyhedral object.

Vertex A vertex is a position or point in 3D space. For the purpose of rendering, vertices are often associated with additional information, for example, vertex normals or colour information. In this section, however, we consider a vertex to be a simple position vector $\mathbf{v_i} \in \mathbb{R}^3$.

Edge An edge is a connection between two vertices. Together, the set of vertices and edges define a graph.

Face A face is a closed set of edges. Commonly, triangle meshes are used and a face therefore consists of three vertices and three edges. However, there also are examples of quadrilateral faces, that is, four vertices and four edges.

A less common yet useful representation of a 3D model is that of a *voxel grid*, for which again an example is provided in Figure 6.1b. A voxel is the three-dimensional equivalent of a pixel; that is, a small cube that represents a value on a regular grid in space. A 3D model can then be represented as a three-dimensional, binary function $\tilde{V}(x, y, z)$, for which a voxel in the grid is active and thus $\tilde{V}(x, y, z) = 1$, if the voxel defines the surface of the model. If the voxel does not define the surface of the model, it is inactive and hence $\tilde{V}(x, y, z) = 0$.



(a) polygon mesh (b) voxel grid

Figure 6.1: (a) depicts a polygon mesh representation of a dolphin. The vertices, edges and faces are well visible and one example for each is highlighted (Source: WikiCommons/User:Chrschn, 2007, original image is public domain). (b) shows a 3D scatter plot of a voxel grid representing three dolphins. The blue dots in the plot correspond to active voxels in the grid.

There are techniques to transform polygon meshes into voxel grids and vice versa (for example [78]). However, these methods usually come at a price in terms of computational power and information loss.

6.2 3D model retrieval

For the purpose of information processing and retrieval, different aspects of a 3D model can be taken into consideration. The survey in [50] discusses and classifies existing techniques and we use their classification to structure this section. More information about some of the methods can also be found in Chapter 2. In this thesis we employ one method based on spatial maps and one view based method.

Spatial map based similarity Spatial maps capture locations or sections of the 3D model in a manner that preserves their relative position. Techniques include shape histograms, ray intersection and spherical harmonics descriptors.

View based similarity The paradigm behind view based similarity is that two models are similar if they look similar from different perspectives. Possible techniques include leveraging z-buffer images or the light field function of a model.

Unless specifically devised, both view based and map based descriptors are not invariant to either rotation, translation or scaling, which is why prior pose normalisation is often necessary.

6.2.1 Pose normalisation

One of the challenges in 3D model retrieval is the degree of freedom that is inherent to the objects of interest. In absence of any prior knowledge, a 3D model may have arbitrary

scale, position or orientation in three-dimensional space [50]. As most feature descriptors are not invariant under these transformations (i.e. scaling, translation or rotation), a pose normalisation may be required before a meaningful feature can be extracted.
In order to normalise a model's size it is common to scale it so that the largest side of the mesh's bounding box takes a fixed value (usually 1.0) in the host coordinate system. An alternative approach involves using the model's bounding sphere instead. Either way, one ends up with a 3D model scaled to a certain unit size. It is important to note that by employing this strategy, one ends up with equivalent sizes for all models regardless of how the objects they represent relate to each other in reality.
A common approach to normalise a model's position is to determine its center-of-mass or *barycenter* and move it to the origin. One could again use the mesh's bounding box and assume its center-of-mass to coincide with that of the mesh itself. This approach is computationally simple but was shown to be very sensitive to outliers [50]. A more stable approach involves calculating the mean of all vertex positions in the mesh and use the resulting vector as barycenter.

$$\overline{\mathbf{c}} = \frac{1}{N} \sum_{n=0}^{N-1} \mathbf{v_n} \tag{6.1}$$

The assumption here is a uniform distribution of vertices in all directions, which is a gross simplification in a lot of cases. As soon as a model contains local features that require a higher level of detail and hence a higher vertex density, the calculated barycenter moves away from the actual barycenter of the model.
To compensate for this effect, Vranic et al. [51] proposed introducing an additional weight that depends on the mesh's surface. The barycenter could then be expressed by Equation 6.2, where S_n denotes the total surface area of all faces that have $\mathbf{v_n}$ as vertex and S denotes the surface area of the mesh (i.e. the sum of all faces).

$$\overline{\mathbf{c}} = \frac{1}{N} \sum_{n=0}^{N-1} w_n \mathbf{v_n},\ w_n = \frac{N S_n}{3S} \tag{6.2}$$

This introduces additional stability, as vertices belonging to small faces contribute less than large ones. In the previous example, where a model locally exhibits high levels of details, this would reduce the contribution of vertices from such high density areas.
Normalisation of the model's orientation is often achieved by principal component analysis (PCA), also known as *Karhunen–Loeve transformation* [45, 50]. It involves calculation of the model's principal component axes and subsequent application of two rotations. The first rotation superimposes the PCA axis exhibiting the largest spread with the x-axis of the host coordinate system. The second rotation then aligns the PCA axis with the second largest spread and the y-axis of the host coordinate system. This results in a new coordinate system that depends on the vertex distribution of the 3D model.
To find the PCA axes, the 3×3 covariance matrix $\mathbf{\Sigma_m}$ of the mesh is calculated as shown in Equation 6.3, where $\mathbf{v_n}$ is the n-th vertex in the mesh and $\overline{\mathbf{c}}$ is the mesh's barycenter.

$$\mathbf{\Sigma_m} = \frac{1}{N}\sum_{n=0}^{N-1}(\mathbf{v_n} - \overline{\mathbf{c}})(\mathbf{v_n} - \overline{\mathbf{c}})^T \tag{6.3}$$

The eigenvectors of the covariance matrix are the principal component axes of the model. The eigenvalues are directly proportional to the spread along the corresponding axis.
Combining the aforementioned techniques results in a normalised, canonical coordinate system that allows for comparison of 3D models.

6.2.2 Light field descriptors

The idea of light field descriptors for 3D models has been proposed by different authors like in [53] or [48]. Those descriptors are based on the projection of the model onto some surface, like for instance, the surrounding sphere. That projection is called *light field*[20]. Feature extraction then takes place at the level of that projection, which is a two-dimensional image. Consequently, techniques known from 2D image retrieval can be applied now.
For example, Chen et al. [53] capture images using cameras attached to the vertices of a dodecahedron. This yields twenty different perspectives of the model. They then use a combination of two image features, namely Zernike moments (see Chapter 4.2.5 on page 31) and a Fourier descriptors (see Chapter 4.2.4 on page 30), to facilitate shape matching.

6.2.3 Spherical harmonics descriptors

Funkhouser at al. [44] proposed to leverage spherical harmonics in order to derive a rotation invariant, compact shape descriptor for 3D models.
Spherical harmonics are a set of orthonormal functions defined on the surface of a sphere. Mathematically, they are the angular portion of the solution to Laplace's equation in spherical coordinates and eigenfunctions of the Laplace operator. Therefore, they play an important role in physics. Throughout this thesis, we use the definition in Equation 6.4 and 6.5, where P_l^m denotes the *Associated Legendre polynomials* and N_l^m is a normalisation constant. The indices l and m are called order or degree and must satisfy the constraints as given.

$$Y_l^m : [0,\pi] \times [0,2\pi] \to \mathbb{C}, \quad (\theta,\phi) \rightarrowtail N_l^m P_l^m(\cos\theta)e^{im\phi} \tag{6.4}$$

$$N_l^m = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} \text{ with } l,m \in \mathbb{N} \text{ and } |m| \leq l \tag{6.5}$$

For the sake of completeness, we have also added the definition of the Associated Legendre polynomials in Equation 6.6.

$$P_l^m(x) = (-1)^m \frac{1}{2^l l!}(1-x^2)^{\frac{m}{2}}\frac{d^{m+l}}{dx^{m+l}}(1-x^2)^l \tag{6.6}$$

All spherical harmonics under the definition of Equation 6.4 satisfy the following orthogonality relation and constitute an orthonormal basis of the space of functions that are defined

[20] A light field is a vector function that describes the flow of light through any point in space. See plenoptic function.

on the surface of a sphere.

$$\int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} Y_l^m(\theta,\phi)^* Y_{l'}^{m'}(\theta,\phi)\sin\theta\, d\theta d\phi = \delta_{ll'}\delta_{mm'} \tag{6.7}$$

Hence, spherical harmonics can be used to expand any piecewise, continuous function $f(\theta,\phi)$ that is defined on a sphere by calculating the projection of $f(\theta,\phi)$ onto the spherical harmonics in the basis and using the obtained coefficients a_{lm} as weights. The coefficients a_{lm} capture the contribution of Y_l^m to $f(\theta,\phi)$.

$$a_{lm} = \int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} Y_l^m(\theta,\phi)^* f(\theta,\phi)\, d\theta d\phi \tag{6.8}$$

Funkhouser at al. [48] chose spherical harmonics to compactly represent a 3D model descriptor. In their proposed method, they decompose the 3D model into a linear combination of spherical harmonics functions, which yields a sequence of complex coefficients per model. According to their paper, the magnitude of those coefficients is a provable lower bound for the L^2 distance between two models. The following steps are being described in the paper:

1. The polygon mesh is rasterised into a $2R \times 2R \times 2R$ voxel grid $\tilde{V}(x,y,z)$, with $R \in \mathbb{N}$ and $x,y,z \in [-R,R]$.

2. The resulting voxel grid is treated as a binary function $\tilde{f}(r,\theta,\phi)$, where r, θ and ϕ are spherical coordinates (radius, azimuth and elevation).

$$\tilde{f}(r,\theta,\phi) = \tilde{V}(r\sin\theta\cos\phi + R, r\cos\theta + R, r\sin\theta\sin\phi + R),$$
$$\text{with } r \in [0,R]\,, \theta \in [0,\pi] \text{ and } \phi \in [0,2\pi]$$

   The resulting function is then sampled at the surface of a series of concentric spheres. That is, functions $\tilde{f}_r(\theta,\phi) = \tilde{f}(r,\theta,\phi)$ are derived for different, fixed radii $r \in [0,R]$. This sampling process is further illustrated in Figure 6.2.

3. Each function $\tilde{f}_r(\theta,\phi)$ is then expanded into a linear combination of spherical harmonics. This yields a unique signature (i.e. a sequence of a_{lm}) for every radius.

$$\tilde{f}_r(\theta,\phi) \approx \sum_l \sum_{m=-l}^{l} a_{lm} Y_l^m(\theta,\phi)$$

4. Combining the different signatures over the different radii into a single matrix yields the spherical harmonics descriptor for the model.

Note that in the original paper, the proposed descriptor is a two-dimensional matrix whereas we use a one-dimensional feature vector by stacking the signatures for the different radii, i.e. $|a_{rlm}|$ as follows:

$$\mathbf{f} = \{a_{000}, a_{010}, a_{011}, \ldots, a_{100}, a_{110}, a_{111} \ldots, a_{rlm}\}$$

It is worth noting that only coefficients for positive values of m are considered in the feature vector. This is due to the fact that the a_{rlm} for $m < 0$ are simply the complex conjugates of those for $m > 0$ if the input function is real-valued. Two such descriptors can be compared by simply computing the L^2 distance between them.

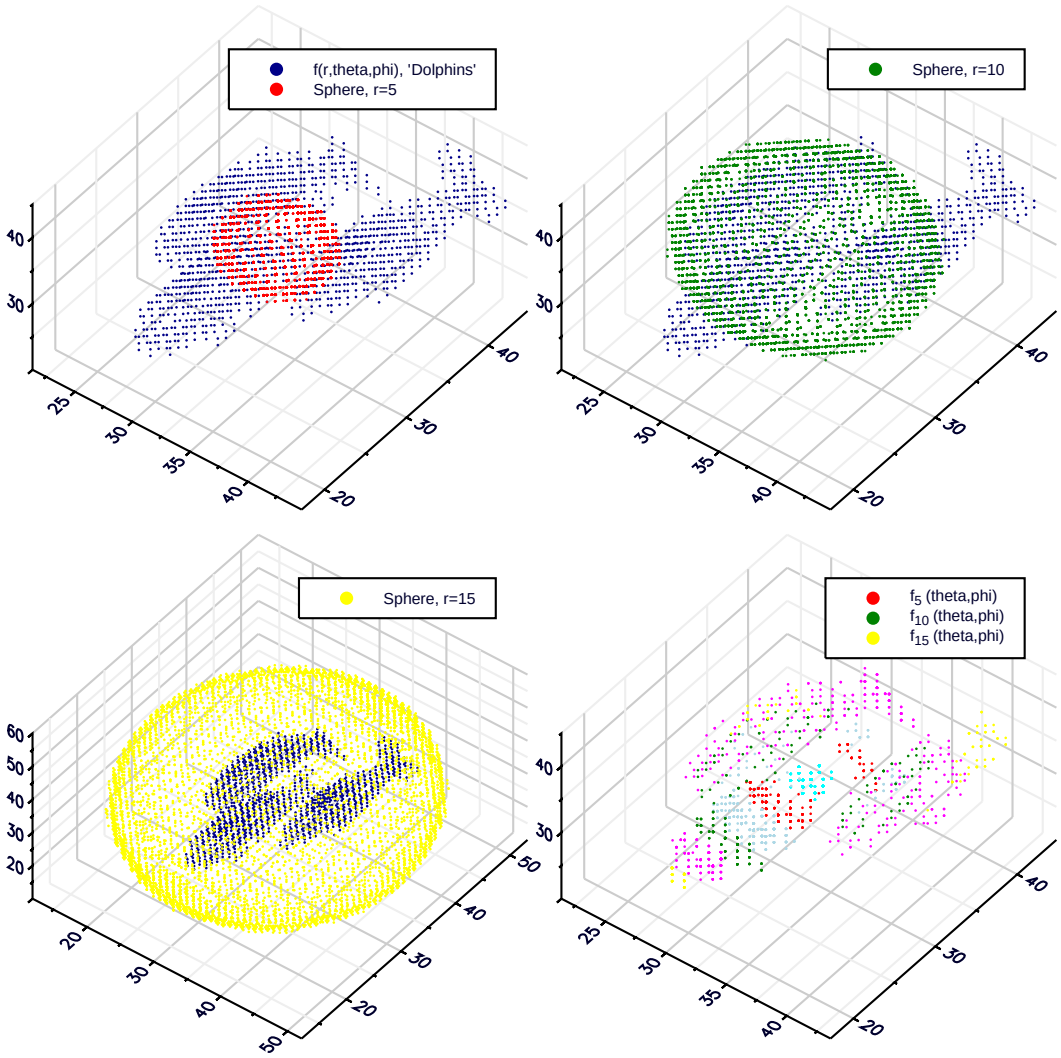


Figure 6.2: Illustration of the sampling process described in [48]. The voxelised model $\tilde{f}(r, \theta, \phi)$, for example the three dolphins in subplot 1 to 3 (dark blue), is evaluated at the surface of different, concentric spheres of radius $r \in [0, R]$, which is also indicated in subplots 1 to 3 (red: $r = 5$, green: $r = 10$, yellow: $r = 15$). Doing so for different radii yields different parts of the original voxel grid, that is $\tilde{f}_r(\theta, \phi)$, as shown in the 4th subplot.

6.3 Implementation

All 3D model related features described in this section use the normalised polygon mesh as starting point. That is, prior to extraction, the mesh is scaled to unit size 1.0, its barycenter is moved to the origin and it is aligned along its principal component axes.

6.3.1 Spherical harmonics feature module

The spherical harmonics feature module applies the approach proposed by Funkhouser et al. in [48] and described in the previous section. Three different versions of the descriptor exist using slightly different settings in terms of voxel grid resolution R and the orders l of the spherical harmonics that are being used.

SphericalHarmonicsLow The mesh is rasterised into a $64 \times 64 \times 64$ voxel grid and harmonics between $l_{min} = 0$ and $l_{max} = 3$ (10 coefficients per radius) are considered. This results in a 220-dimensional feature vector.

SphericalHarmonicsDefault The mesh is rasterised into a $64 \times 64 \times 64$ voxel grid and harmonics between $l_{min} = 0$ and $l_{max} = 4$ (15 coefficients per radius) are considered. This results in a 330-dimensional feature vector.

SphericalHarmonicsHigh The mesh is rasterised into a $74 \times 74 \times 74$ voxel grid and harmonics between $l_{min} = 1$ and $l_{max} = 5$ (20 coefficients per radius) are considered. This results in a 540-dimensional feature vector.

We use the algorithm described in [78] for the voxelisation step. Subsequently, we evaluate the voxel function $f_r(\theta, \phi)$ on the surface of concentric spheres with radii $r \in \left[0, \frac{R}{2}\right]$ and expand those functions into linear combinations of spherical harmonics between l_{min} and l_{max}. The magnitudes $|a_{rlm}|$ of those coefficients are then concatenated into a single feature vector, which is normalised with respect to the L^2 norm.
During extraction, that feature vector is persisted. During retrieval, a kNN-search under the L^2 dissimilarity is performed with the resulting feature vector.

6.3.2 Light field Zernike feature module

The *LightfieldZernike* feature module combines light field images with Zernike moments [60, 79] as proposed by [53] and described in Section 4.2.5 on page 31. The module yields at least twenty 36-dimensional feature descriptors per 3D model. Invariance with respect to rotation, translation and scaling of the model is achieved through prior pose normalisation. The following steps are carried out in order to obtain the feature vectors from a 3D model. In case of an image (e.g. for QbS during retrieval), the first step is skipped.

- Light field images of the 3D model are generated from 20 different camera perspectives. To do so, the camera is positioned at the vertices of a regular Dodecahedron that is concentric with the model.
- Every generated light field image is then segmented by thresholding and subsequent application of a contour detection algorithm. We use the *BoofCV* library to facilitate these image operations. This results in a list of shape contours per image.
- Each contour above a certain size is used to generate a new binary image of the shape by colouring pixels that are enclosed by the contour white and the rest of the pixels black.

- For each resulting binary image, the Zernike moments on the unit disk that circumscribes the shape are calculated up to order 10. This yield 36 complex Zernike coefficients a_{nm} of which we use the magnitude $|a_{nm}|$ as components in the feature vector.

The steps above yield at least twenty feature vectors per model (one per perspective) and at least one feature vector for an example image. However, because the segmentation step may identify multiple contours in an image, one might end up with more than one descriptor per image. This is especially true if the original model consists of disconnected parts.
During extraction, the resulting feature vectors are persisted together with an internal perspective-index that identifies the camera position used in the image generation step. During retrieval, two different cases must be considered.

QbE with 3D model For QbE with a provided 3D model, a kNN-search under the L^2 distance is performed for each feature vector extracted from the model. If a potential hit's perspective-index is equal to the perspective-index of a result, that fact is used to boost the final score in the result set. Only the best result per model is kept and returned.

QbS with image For QbS with a provided image, a kNN-search under the L^2 distance is performed for each feature vector extracted from the image. Only the best result per model is kept and returned.

6.3.3 Light field Fourier feature module

The *LightfieldFourier* feature module combines light field images with Fourier descriptors [60] as proposed by [53]. The module yields at least twenty, 128-dimensional feature descriptors per 3D model. Invariance with respect to rotation, translation and scaling of the model is achieved through prior pose normalisation.
The following steps are carried out in order to obtain the feature vectors from a 3D model. In case of an image (e.g. for QbS during retrieval), the first step is skipped.

- Light field images of the 3D model are generated from 20 different camera perspectives. To do so, the camera is positioned at the vertices of a regular dodecahedron that is concentric with the model.

- Every generated light field image is then segmented by thresholding and subsequent application of a contour detection algorithm. We use the *BoofCV* library to facilitate these image operations. This results in a list of shape contours per image.

- For each contour above a certain size, the centroid distance function $r(n)$ is calculated. $r(n)$ returns the distance from the n-th point in the contour (x_n, y_n) to the centroid (x_c, y_c) of the entire contour, with $n \in \{0 \dots N\}$ where N denotes the total number of points in the contour.

- The Fourier transform of the resulting centroid distance function $r(n)$ is obtained. The magnitudes $|a_k|$ of the top 128 Fourier coefficients a_k serve as components of the feature vector.

The steps above yield at least twenty feature vectors per model (one per perspective) and at least one feature vector for an example image. However, because the segmentation step may identify multiple contours in an image, one might end up with more than one descriptor per image. This is especially true if the original model consists of disconnected parts.
During extraction, the resulting feature vectors are persisted together with an internal perspective-index that identifies the camera position used in the image generation step. During retrieval, two different cases must be considered.

QbE with 3D model For QbE with a provided 3D model, a kNN-search under the L^2 distance is performed for each feature vector extracted from the model. If a potential hit's perspective-index is equal to the perspective-index of a result, that fact is used to boost the final score in the result set. Only the best result per model is kept and returned.

QbS with image For QbS with a provided image, a kNN-search under the L^2 distance is performed for each feature vector extracted from the image. Only the best result per model is kept and returned.

6.3.4 Feature categories

The feature categories for 3D model retrieval were defined empirically by combining different modules and experimenting with different weights. The following categories were created as part of this thesis:

sphericalharmonicslow Consists of the *SphericalHarmonicsLow* feature.

sphericalharmonicsdefault Consists of the *SphericalHarmonicsDefault* feature.

sphericalharmonicshigh Consists of the *SphericalHarmonicsHigh* feature.

lightfield Consists of the *LightfieldFourier* and *LightfieldZernike* features using weights of 1.0 and 2.5 respectively.

7 Evaluation

The primary goal of an IR algorithm or IR system, regardless of being for CBIR, CBMR, CB3DR or any other domain, is satisfying a particular information need. That is, from a collection of documents it should retrieve a subset of documents that are particularly relevant to a user in a given context [80]. When evaluating the performance of such a system or algorithm, an experimenter is required to measure its ability to perform that task.

7.1 On effectiveness evaluation of information retrieval systems

In general, the effectiveness of an IR method or system is often expressed using the concept of *relevance* [80–82]. Despite being so central for evaluation in IR, however, there is no unambiguous definition of the term [80]. For example, authors in [82] differentiate between two interpretations of relevance. On the one hand, it can be regarded as a direct, logical relation between a query and a document, for instance, topical in nature. On the other hand, relevance can also be seen as a proxy for the subjective usefulness or utility of a result, that is, a measure for user preference. TREC[21] [83] — an annual workshop on text retrieval — uses a clear working definition of relevance as follows: "If you were writing a report on the subject of the topic and would use the information contained in the document in the report, then the document is relevant." [84, 85].

In any case, an IR system tries to predict relevance of a document given a particular query [80]. As we have seen in Section 3.1.4, relevance can be expressed as similarity between a reference document — like a sketch or an audio file — and the documents in the collection. The implicit assumption here is, that if a reference document is considered relevant, documents similar to that reference can be considered relevant as well [80].

Regardless of the underlying model, in an experimental setup relevance is usually assessed in the form of *relevance judgements* performed by some group of users [85]. Those judgements can either be binary, that is, a document is relevant or it is not, or one can allow for different degrees of relevance.

[21] http://trec.nist.gov

Two evaluation metrics typically found in literature are *precision* and *recall* and derived measures like *P-R curves* or the *F-measure* [86, 87]. They are closely related to the binary nature of relevance and the direct relation between query and document [87, 88].
However, some authors have pointed out the shortcomings of precision and recall [89] especially as collections become larger and relevance judgements fail to cover the entire collection [90]. To address this issue, new measures like the *Discounted Cumulative Gain (DCG)* have been proposed [91]. They are based on the notion that relevance indicates user preference and is not simply a binary matter. Hence, documents may be valuable to some extend even though they are not fully relevant for a particular information need.
Evaluation of content-based multimedia retrieval systems still faces many challenges today. Partially, because the information contained in audio, images and video is far more complex than text, and the determination of what objectively constitutes a relevant result is therefore more challenging [4]. In addition, there are only few standardised collections that can be used for evaluation and creating such collections is difficult because content is often protected by intellectual property rights [4, 92]. Furthermore, it is still disputed in the community as to whether particular collections are well suited for particular evaluation tasks [4]. This is presumably why different authors often use different collections and evaluation metrics, which makes direct reproduction and comparison of results very difficult.

7.1.1 Formalising the experimental setup

To establish a foundation for the remaining chapter, we introduce some terms and symbols to formalise an IR system in general and Cineast in particular. Most of the concepts and terms introduced here are also described in [80].
Formally, an IR system always operates on a *corpus* or *collection* $C = \{d_1, d_2, \ldots, d_N\}$ of *documents* d_i. The value N denotes the size of the collection, which can easily lie in the range of 10^6 and 10^9 depending on the type of collection.
A user of the IR system usually seeks a subset of that collection. This is called an *information need*. For the purpose of retrieval, that information need is translated into a query Q for which the system then generates a list of *results* $X_Q \subset C$ in which the documents occur in order of the relevance predicted by the system. The position of a document in this list is called *rank* $j \in \mathbb{N}_{\geq 0}$ of the document and the lower the rank, the more relevant the document according to the IR algorithm[22]. For practical reasons, the size of X_Q is usually orders of magnitude smaller than the size of the collection. This is sometimes enforced in a process called *pruning*.
In an experimental setup, given a particular query Q, one can theoretically define a set of documents $Y_Q \subset C$ that are considered to be relevant to that query, regardless of whether they are contained in the result set X_Q. This requires manual review and annotation of all the documents in the corpus C. These annotated collections are sometimes referred to as *ground truth* and they can be used to automatically decide whether a particular result X_Q contains relevant documents or not. Another common approach involves user-driven

[22] It is worth noting here that the rank usually determines the position and prominence of a document in the user interface. That is, documents with lower ranks are shown more prominently.

relevance judgement or *rating* r_j of results X_Q. This involves manual feedback by the user as to whether a retrieved document is considered relevant or not. As we have seen in the beginning of this chapter, ratings r_j can either be binary, i.e., 0 for irrelevant documents and 1 for relevant documents, or on an arbitrary scale that allows for *marginal relevance*. Regardless of the method, given the rank j and the ratings r_j for each document in X_Q, different evaluation metrics can be derived from those values. These metrics are being described in the following sections.

7.1.2 Precision and Recall

Precision $\mathcal{P}$ expresses the fraction of retrieved documents that are relevant to a query and hence measures a system's ability to reject irrelevant results. In contrast, recall $\mathcal{R}$ expresses the fraction of relevant documents that were successfully retrieved and hence measures a system's ability to retrieve all relevant documents [86]. Definitions of $\mathcal{P}$ and $\mathcal{R}$ are provided in Equation 7.1 where Y_Q denotes the set of documents relevant to a query and X_Q the set of retrieved documents for a query (i.e., the result).

$$\mathcal{R} = \frac{|Y_Q \cap X_Q|}{|X_Q|}, \quad \mathcal{P} = \frac{|Y_Q \cap X_Q|}{|Y_Q|} \tag{7.1}$$

By themselves, neither precision nor recall are sufficient to accurately describe an IR algorithm. For example, it is easy to obtain a recall of 1.0 simply by retrieving the entire corpus C, that is, by making the result set large enough. On the other hand, precision can be maximised by only retrieving the first relevant document. Therefore, both values are required to characterise a system. Typically, in the literature, this is done through P-R-curves, where precision $\mathcal{P}$ is expressed as a function of recall $\mathcal{R}$ [86].

From the terms in Equation 7.1 two things become immediately apparent: Firstly, precision and recall are purely set-based. Therefore, the ranking of results in X_Q is not taken into account. Secondly, in order to obtain precision and recall values, the documents that are relevant for a specific query, that is Y_Q, must be known beforehand. In practice, this is achieved by labelling the documents in a corpus and using those labels to determine whether a document is relevant for a particular query or not. Obviously, this is a laborious, manual task that must be carried out in preparation of an experiment. This is why the use of standardised, pre-labelled collections is so important. In addition to simplifying the process itself, such collections also allow for direct comparison of evaluation results as the choice of labels greatly affects the outcome of an evaluation.

To account for ranking, there also exist modified versions of precision and recall called *precision@K or p@K* and *recall@K or r@K* [86]. These values capture precision and recall at a specific rank k. However, the choice of the rank k is always arbitrary and has a huge impact on the result which is why other measures that do not require a prior choice of k are usually preferred.

7.1.3 Mean Reciprocal Rank (MRR)

The *Reciprocal Rank* is the multiplicative inverse of the rank of the first relevant document. The *Mean Reciprocal Rank (MRR)* can then be calculated by averaging the reciprocal rank values for a sample of N different queries Q_n.

$$MRR = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{rank_n} \tag{7.2}$$

7.1.4 Mean Average Precision (MAP)

The *Average Precision* accounts for precision and recall in a list of ranked results without having to choose the rank k. It can be calculated by traversing the results X_Q starting at the lowest rank. If the entry at rank k is considered relevant, the p@K value p_k is calculated, otherwise p_k is zero. Finally, all p@K values are averaged (see Equation 7.3). Obviously, L non-zero p_k values are obtained with L denoting the number of relevant documents in the corpus, that is, $|Y_Q|$

$$AP = \frac{1}{L} \sum_{k=0}^{K} \begin{cases} p_k, & \text{if } d_k \text{ is relevant} \\ 0, & \text{otherwise} \end{cases} \tag{7.3}$$

The *Mean Average Precision (MAP)* can then be calculated by averaging the AP values for a sample of N different queries Q_n.

$$MAP = \frac{1}{N} \sum_{n=1}^{N} AP_n \tag{7.4}$$

7.1.5 Discounted Cumulative Gain (DCG)

The reasoning behind *Discounted Cumulative Gain (DCG)* is that the greater the rank j of a relevant document, the less valuable it is for the user. The underlying model assumes that users chiefly consider the top items in a list of results. The higher the rank of a document, the less likely it is that a user will actually examine it due to the time, effort and information accumulated by reviewing all the preceding documents. This is expressed in Equation 7.5, where r_i denotes the rating assigned by the user to the item at rank j and k denotes the rank up to which the DCG is calculated [91].

$$DCG_k = \sum_{j=1}^{k} \frac{r_j}{\log_2(j+1)} \tag{7.5}$$

In the literature, the DCG is usually normalised by the *Ideal Discounted Cumulative Gain (IDCG)*. This IDCG can be obtained by re-ranking the documents according to the rating assigned by the user instead of the relevance predicted by the IR system. This re-ranking yields a modified result X'_Q for which again the DCG according to Equation 7.5 is calculated in order to obtain the IDCG.

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{7.6}$$

7.2 Evaluation setup

For this thesis, we evaluate the system's performance along two dimensions. Firstly, we measure the retrieval effectiveness of the system as a whole in terms of utility for the end user. We decided not to evaluate individual feature modules, with the exception of the Princeton Shape Benchmark (see Section 7.2.4). Secondly, we capture the retrieval performance in terms of execution time per feature module. The latter is interesting, because it gives indication about the performance of the extraction engine and the underlying storage (i.e., ADAMpro). This allows us to identify potential focus areas for future work.

The evaluation is based on two sets that comprise twelve similar scenarios each. There are three scenarios per domain, that is, image retrieval, audio retrieval, video retrieval and 3D model retrieval and each focuses on different aspects within the respective domain. A scenario comprises a simple objective (information need), which the users is expected to carry out using Vitrivr NG. This approach reflects our opinion, that retrieval is an interactive process and that an IR system like Vitrivr NG should support the user in finding what they are looking for. Therefore, we are ultimately interested in the user's relevance judgement for the results produced by Cineast and Vitrivr NG.

In general, users are free to execute as many queries as they please until they are either satisfied with the result or they decide to give up. Here we apply the principle that *searching is an iterative process* [80]. Unless otherwise stated, users are allowed to leverage all of Vitrivr NG's capabilities, namely QbE, QbS and More-Like-This queries. Furthermore, they may also use the refinement functionality provided by Vitrivr NG. Applying the *principle of the least effort* [80, 93], we expect the users to take the course of action that they believe to be connected with the least expenditure.

Once a user has obtained and accepted a result for a scenario, they are required to rate the top 15 documents on the following scale:

- 0 — Resulting document is not considered to be relevant at all.
- 1 — Resulting document is considered slightly relevant.
- 2 — Resulting document is considered very relevant.
- 3 — Resulting document is considered highly relevant, close to identity.

The relevance judgements are then aggregated into *MAP*, *MRR*, *NDCG@15* and *p@15* values per scenario. For binary metrics, ratings of 2 and 3 are considered to be hits and values of 0 and 1 are considered to be misses. Furthermore, we decide for each scenario whether the user was able to fulfill the objective based on the presence of at least one high-relevance rating.

During the entire evaluation session, Cineast logs the execution (wall clock) time of all new feature modules described in chapters 4, 5 and 6. The logging is broken down into the three stages: pre-processing, kNN-lookup and post-processing as described in Chapter 3, Section 3.1.6. These values are used to calculate average execution times per feature module and stage.

7.2.1 Evaluation scenarios

An in-depth description of the scenarios as presented to the user can be found in Appendix B. Basically, each scenario encompasses the following information:

Description A short description of the scenario that formulates a simple objective like "Retrieve a variant of the example A." or "Find a scene that looks similar to the illustration." Some scenarios restrict the functionality that can be used, other leave that completely open. Some scenarios allow the use of external tools such as Google in order to obtain a reference document.

Material (Optional) Some scenarios include accompanying material like a short audio excerpt or a 3D model. The users are expected to utilise those as reference documents. Sometimes, they get to choose between multiple variants.

Illustration (Optional) Some scenarios include illustrative imagery to clarify what is requested. For example, a scene may be depicted and the user is expected to sketch that scene.

The scenarios in the two test sets are similar with respect to the aspect of the IR system that is being tested. The concrete information need, however, is different.

7.2.2 Test collections

For the evaluation we have combined collections from different sources. When possible, we tried to use standardised test collections. However, we found a lot of those collections to be unsuitable for our scenarios either due to their size, their content or both. Especially in the domain of audio retrieval, there unfortunately are very few collections that exhibit the desired variety while containing documents that allow for testing of specific features like, for example, audio matching algorithms.
This is why we have assembled our own test collections for audio. We selected random items from public sources such as the Freemusicarchive[23], Pixabay[24] and Thingyverse[25]. In order to construct examples for audio matching, we included specific items from our personal music collections for scenarios A4, A5, A6, B4, B5 and B6.
Table 7.1 lists all the collections that were used in the evaluation. All in all, we ended up with a total of 4397 audio files, 200 video files, 12969 3D models and 164512 images. All items were stored in the database at all time during the evaluation.

7.2.3 Environment

For the entire evaluation, Cineast and ADAMpro were run on the same machine (8-core Intel Core i7-4770@3.4 GHz, 32.0 GB memory and 512.0 GB disk space with Ubuntu 16.04 LTS). ADAMpro was started with a heap size of 32.0 GB and Cineast had up to 4.0 GB

[23] https://freemusicarchive.org
[24] https://pixabay.com
[25] https://www.thingiverse.com

Table 7.1: List of all the collections used during the evaluation. The table contains information regarding the size and source.

Domain	Name	Entries	References
Image	Pixabay	164512	https://pixabay.com
Audio	Freemusicarchive	4335	https://freemusicarchive.org
Audio	Personal music collection	62	-
Video	Open Short Video Collection	200	See [94]
3D	NTU 3D (1-4)	4003	See [95]
3D	Thingyverse	8966	https://www.thingyverse.com

of heap memory at its disposal. The Vitrivr NG user interface was served from the same machine through an *Apache Web Server*. Thumbnails and media objects were served from a different node mainly due to disk space restrictions.

7.2.4 Princeton Shape Benchmark for 3D models

For 3D models, we were able to obtain an evaluation collection called the *Princeton Shape Benchmark (PSB)* [96]. We decided to use that collection in order to test the individual feature modules related to CB3DR, namely the *LightfieldZernike*, *LightfieldFourier* and *SphericalharmonicsDefault* and *SphericalharmonicsHigh* feature modules introduced in Chapter 6. This allows for direct comparison with results reported by other authors using the same collection.

The PSB v1 collection comprises 1814 3D models divided into two subsets à 907 models each. Every model has a unique identifier that is included in its file name. Furthermore, every model was manually classified in a separate classification file. The classification dictionary contains 197 different leaf classes that primarily reflect the function of the object and secondarily its form. The dictionary includes simple terms like “airplane”, “pig”, “spider”, “ship” or “tree”.

We indexed all 1814 models with Cineast and extracted the provided base classification into a ground truth dictionary. Only leaf classes were considered. Subsequently, we executed queries through Cineast using each of the 1814 models from the PSB collection. From the results, we calculated precision and recall values by comparing the class of the reference document to the respective class of the retrieved documents. If the classes coincided, the document was considered relevant, otherwise it was not. The process was fully automated and only the provided classification was used in determining the relevance of the results.

7.3 Results: Retrieval effectiveness

The outcome of the user-driven evaluation is summarised in Table 7.2. In total, 25 datasets where gathered. Thirteen participants did test set A and the other twelve worked through test set B. We use the following sections to comment on the results per scenario.

7.3.1 Image: Query-by-Example

Scenarios A1, A2 and B1, B2 aimed at QbE for images. In the first task, users were free to pick a reference image of their choice whereas for the second task, users were provided with a variant of an image of which we knew that it existed in the database.

7.3.1.1 Scenarios A1 and B1

Interestingly, not all users considered the results returned in A1 and B1 to be relevant. A success rate of 92 % and 66 % respectively is not great for a simple task like this and neither is a MRR of 0.77 and 0.33. As it turns out, the results are very dependent on the reference image. This is also reflected by the p@15 value, which is considerably higher for A1 than for B1. It seems that if the general colour setting of the reference remotely matches an image in the database, the latter image is set to rank high in the list of results despite not depicting the same thing conceptually.
Generally, colour-based feature categories — namely, *localcolor*, *globalcolor* and *quantized* — outweigh other features most of the time. For photographs of a mountain scenario (B1), possible colour distributions vary greatly between instances from gray to blue and sometimes even brown or red and yellow. Therefore, the likelihood of unrelated hits occurring is higher than for meadows (A1), for which the dominant colours are mostly green and blue.

7.3.1.2 Scenarios A2 and B2

For scenarios A2 and B2, 92 % and 100 % of the users were able to obtain the copy of the reference image and the MRR of 0.92 and 1.0 indicates, that the desired image was in first place most of the time.
The reason why the score for A2 is not 100 % is probably due to a user mistake. The provided reference image in A2 is a grayscale image whereas the original in the database is in colour. The users were advised to switch off the colour features through the query refinement section of the interface. If they failed to do so, however, the query indeed did not list the desired image in the top 15 ranks due to, again, the colour features completely outweighing all the others.
From the p@15 we must deduce, that a majority of the remaining results were considered to be irrelevant or only slightly relevant. The NDCG@15 implies, however, that the ranking coincided pretty well with the rating of the users.

7.3.2 Image: Query-by-Sketch

Scenarios A3 and B3 involved QbS tasks. In both cases, the users had to find a particular logo or icon based on a sketch. Success rates of 69 % and 100 % respectively and a MRR of 0.63 and 0.80 indicate, that the majority of users were able to retrieve the item of interest and that the relevant item was placed in the top half of the result set. However, not all users managed to retrieve it in the case of A3.
The reason for this can be reproduced in a simple experiment (see Appendix C). The illustrative image provided with the scenario was such that the logo filled most of the canvas and touched its edges. Naturally, a majority of users sketched the logo in the exact same

way. If one does so, Cineast indeed will not return the logo itself in most cases. This is due to the images in the database both having a white border so that the red to white ratio is different. This leads to lower scores for most of the (very dominant) colour features.
It is also worth noting that it took the users 3.3 and 2.2 queries on average to obtain the results. This is approximately one query more than for scenarios A1, A2, B1 and B2, which is likely due to the fact that most users required multiple attempts at sketching the item of interest. Furthermore, according to feedback, a lot of users employed More-Like-This to push the desired item from higher ranks to the top.

7.3.3 Audio: Fingerprinting

Scenarios A4, A5, B4 and B5 were pure audio fingerprinting tasks of which A4 and B4 encompassed a simple, 3 second audio segment as reference document and A5 and B5 encompassed a noisy audio excerpt. With the exception of A5, the success rate was a 100 % for those tasks, which means that the audio segment in question could always be retrieved. The MRR value for those tasks — again with the exception of A5 — was 1.0, indicating that the relevant document had top rank.
The remaining results in the top 15 ranks can generally be considered irrelevant hits, which explains the low p@15 values. As the users mostly agreed with the ranking of Vitrivr NG (3 for the first item, 0 for the rest) the NDCG@15 tends to be close to 1.0
In summary, the above shows that the fingerprinting works and exhibits some robustness to noise. However, in the case of A5 the noise seems to impair the ranking.

7.3.4 Audio: Matching

Scenarios A6 and B6 were audio matching scenarios. In both scenarios the users were supposed to find the original version and a cover version of the same musical piece. Again, the high success rate of 100 % for both tasks indicates, that at least one of the versions could be retrieved. The MRR here lies between 1.0 and 0.92, which indicates a rank between 1 and 3 for the first, highly relevant item in the list.
The cover version was also retrieved in most cases. Furthermore, some of the other top 15 items were considered to be highly or at least very relevant. Both these facts contribute to a p@15 between 0.13 and 0.14. The NDCG@15 indicates, that the ranking presented by Vitrivr NG coincides with the user rating in many instances. It is, however, not perfect.

7.3.5 Video: Query-by-Sketch

Scenarios A7 and B7 were pure QbS tasks for video and obviously challenging for users to complete. This is indicated by the comparatively low success rates of 73 % and 33 % respectively and the low MRR values. In fact, the success rate of B7 was the worst of all scenarios in the entire evaluation.
The explanation for why the success rate was so low in the case of B7 as compared to A7 is very likely provided by the reference image itself (see Appendix B). The B7 reference image uses a very disadvantageous colour palette which is difficult to reproduce. In hindsight, we

should have used a different image for this scenario.
However, these examples — together with A3 and B3 — confirm the difficulties of QbS, especially for complex imagery. Again, it is worth noting that both A7 and B7 required 1.69 to 2.25 queries per image — more than the QbE-based tasks.

7.3.6 Video: Combining modalities

In scenarios A8 and B8 participants were tasked to combine an audio excerpt with a reference image of choice to find a particular scene in a video. The success rate of 100 % is positively surprising, especially as it is higher than for A9 and B9, where users were only allowed to use a reference image alone. This indicates, that adding another modality to the mix indeed brings some advantages even though the relative contribution of the audio features depend on the provided reference image.
However, the ranking of the results does not always seem to agree with the user ratings as we can read from the NDCG value between 0.74 and 0.77 and precision tends to be rather low. The latter can be attributed to the fact, that we were actually looking for a particular scene which is unique in the entire collection both in terms of the visual as well as the auditory part. As with scenarios A4, A5, B4 and B5, the audio fingerprinting feature, which was used most of the time, reliably produces one hit and a lot of seemingly unrelated results.

7.3.7 Video: Query-by-Example

In scenarios A9 and B9 users had to retrieve a specific scene based on a provided but distorted image (A9) or example image of their choice (B9). Unsurprisingly, the success rate for A9 was almost 30 % higher than for B9. Also, the MRR is both cases was relatively low as it ranged between 0.39 and 0.64. This indicates, that it was difficult for the users to bring the desired video to the top rank.
Furthermore, it must be noted that the p@15 value is very low in both cases (0.07 and 0.05). Accordingly, the false positive rate must have been very high. Again, this can mainly be attributed to the dominance of the colour features for which the overall distribution of colours is sufficient for a result to be considered relevant. However, it was also noted by many users, that some hits were actually black images. This is presumably an artifact from the *edge* feature.

7.3.8 3D models: Query-by-Sketch

Scenarios A10 and B10 were QbS tasks for 3D model retrieval. From the success rate of 69 % and 100 % respectively, and the MRR of 0.69 and 1.0, we can deduce that most of the users succeeded in finding a relevant model and that if they found it, it was ranked at top position.
However, as for all the QbS tasks so far, the number of queries is comparatively higher than for the other tasks. In fact, users required an average of 2.8 and 4.5 queries in order to fulfill A10 and B10 respectively. Firstly, this indicates that usually several attempts at sketching were taken. Secondly, again, the More-Like-This functionality seems to have allowed for

pushing of high-ranking results to the lower ranks. This is in line with qualitative feedback we have received.
The p@15 values of 0.12 and 0.10 are relatively low, indicating that on average at the most one additional item was considered highly relevant in the top 15 ranks. The NDGC values of 0.83 and 0.93 imply that, again, the ranking of Cineast coincided pretty well with the rating of the users.

7.3.9 3D models: Query-by-Example

Scenarios A11 and B11 were QbE tasks for 3D model retrieval. This task was apparently straightforward in both cases, judging from the duration — the lowest in the entire evaluation — and the number of queries required to fulfill it. Both scenarios show a 100 % success rate and a MRR value of 1.0, which means that highly relevant items could always be obtained and were always placed at the top rank.
What is interesting, though, is the large discrepancy of p@15 values, which was 0.69 for A11 and 0.23 for B11. The reason for this is probably two-fold:

1. It was reported by other authors [50, 52] that retrieval performance of the spherical harmonics descriptors differs object classes. Furthermore, those authors reported a remarkable retrieval accuracy for the class of *airplanes* without giving a specific reason. And indeed, in A11 we provided a selection of airplane models as reference documents, whereas for B11 we used chess pieces.

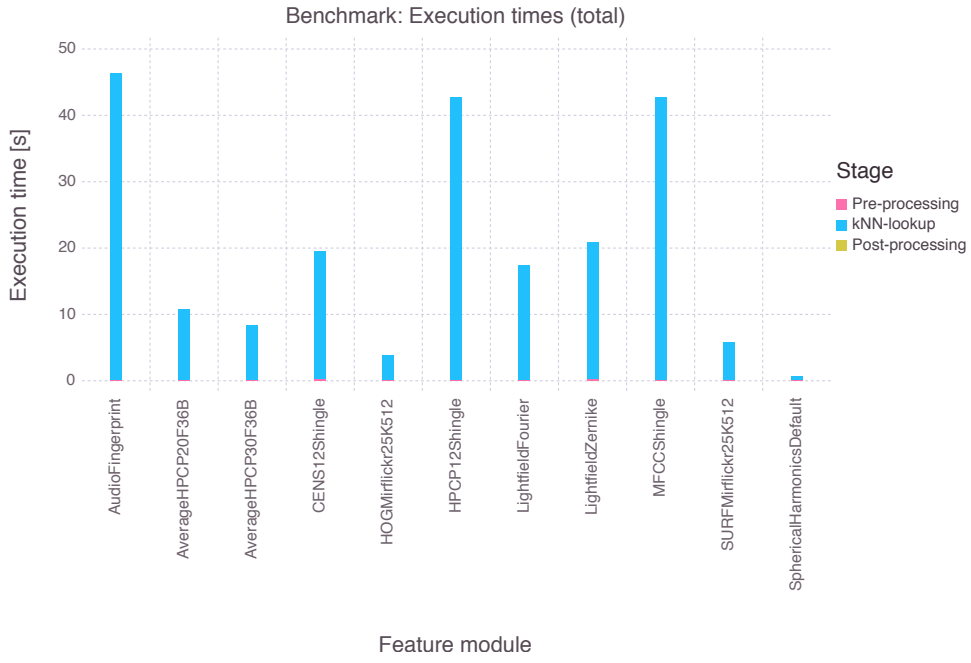
2. Due to the structure of the collection, the number of airplane models, and therefore the number of potential matches, is higher than for chess pieces. Naturally, this also has an influence on the result set.

7.3.10 3D models: Free choice

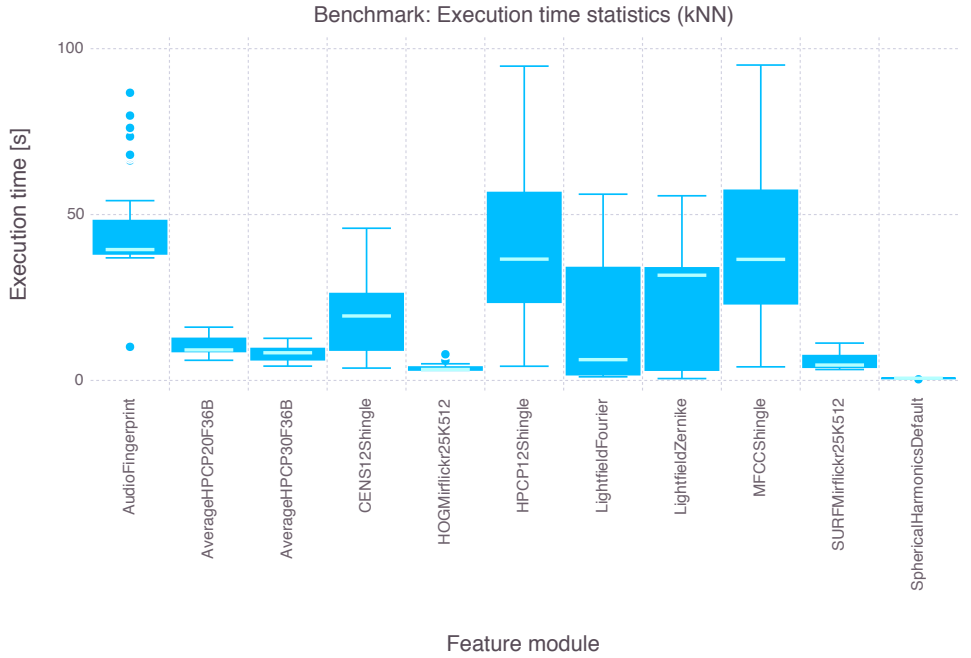
In scenarios A12 and B12, we asked the user to find a particular object of interest by whatever means they prefer. They were allowed to use external resources like Google. Interestingly, most users chose the QbS mode here and were able to obtain relevant items on most of the cases (100 % success rate for B12 and 77 % success rate for A12).
The p@15 values of 0.26 and 0.34 indicate, that some additional items were found that were rated two or higher. Judging from the NDCG@15, the ranking by Cineast coincided pretty well with the rating provided by the users.

7.4 Results: Retrieval performance

The results of the performance benchmarks are presented in Figure 7.1a and 7.1b. Furthermore, we have listed the average execution time per module in Table 7.3 alongside some additional information regarding general characteristics of the modules themselves. Their names correspond to those listed and described in Chapters 4, 5 and 6.
It is to be noted, that the feature modules under test differ in vector size, index structures used, number of entries in the respective entity and number of lookups that are being



(a) Mean execution times of the individual feature modules broken down to the stages.



(b) Execution times reported for the *kNN-lookup* stage of the individual feature modules.

Figure 7.1: Results of the performance benchmark. Figure 7.1a plots the mean execution time per feature module and stage in seconds. The total execution time is split into three stages as described in Section 3.1.4: pre-processing, kNN-lookup and post-processing. The execution time of the pre-processing and post-processing stage is determined by Cineast whereas the kNN-lookup stage is dominated by the storage layer (i.e., ADAMpro). The box plot in Figure 7.1b illustrates the variance of the execution time during the kNN-lookup stage.

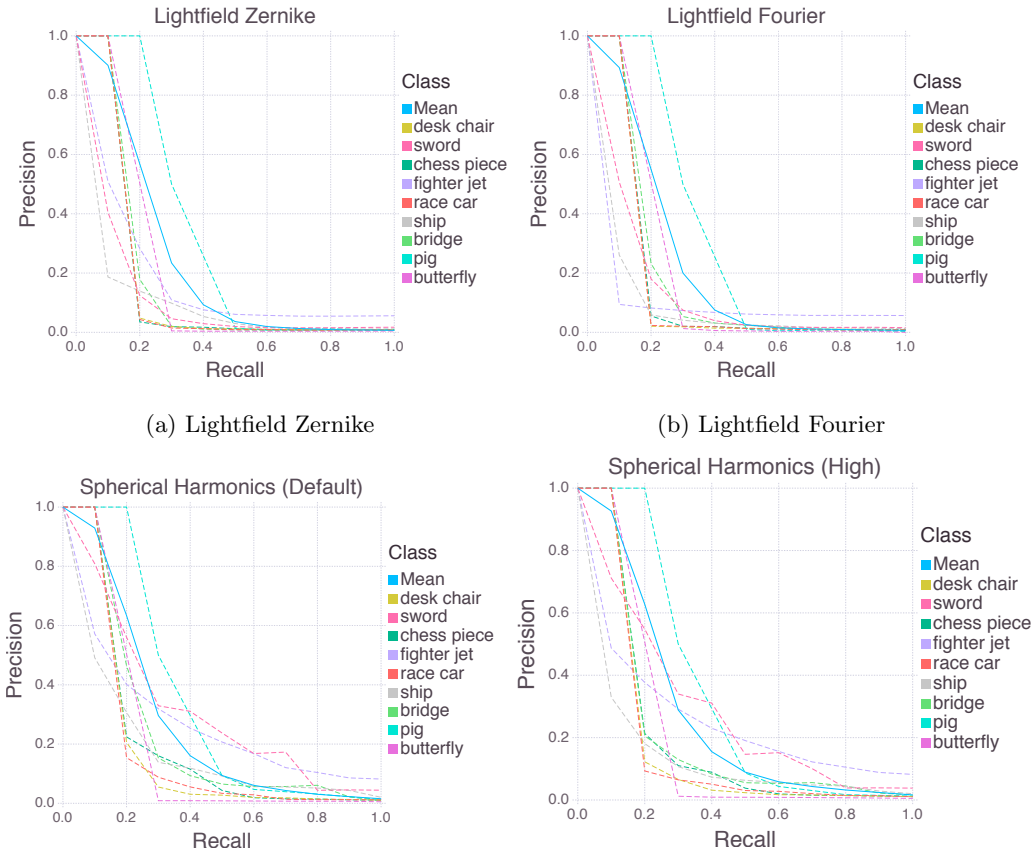
performed. For that reason, the resulting metrics cannot be compared directly. The main purpose of this assessment is not to appraise the ADAMpro lookup performance but to identify focus areas for further improvements. And indeed there are some general trends we would like to draw attention to:

- The execution time of the pre-processing stage, that is, the stage during which feature vectors are generated from the reference object, is negligible compared to the time required to perform the kNN-lookup. The same is true for the post-processing stage, which is barely measurable in most cases. This is even true for the fastest module — the *SphericalHarmonicsDefault* feature — where kNN-lookup still takes 2.3 times longer than pre-processing (see Table 7.3).
- The execution time for the kNN-lookup stage increases approximately linearly with the number of vectors that are being used for lookup. As querying multiple vectors simultaneously is a requirement for most audio features, the total lookup time suffers considerably for those feature modules. This becomes apparent when comparing lookup times for audio features to lookup times of features like *SURFMirflickr25K512* or *SphericalHarmonicsDefault* (see Figure 7.1a), which only use a single lookup.

All things considered, lookup seems to be the limiting factor for all the new feature modules. This can be attributed to the storage layer ADAMpro. Unfortunately, lookup speed for audio features must be characterised as slow, which is confirmed by the subjective user experience during the evaluation. Performance seems to degrade quickly as dimensionality of the feature vectors and the number of vectors increase. As the shingeling approach employed by most audio features results in both many and rather high-dimensional vectors ($d \geq 100$), this immediately becomes an issue. In addition, conducting multiple lookups in the same entity with different vectors does not seem to be handled in an optimised fashion. This explains the long execution times for *MFCCShingle*, *HPCP12Shingle*, *CENS12Shingle*, *AudioFingerprint*, *AverageHPCP20F36B*, *AverageHPCP30F36B*, *LightfieldFourier* and *LightfieldZernike*, for which multiple lookups are performed. It must be noted, that the number of lookups for the audio features is directly proportional to the duration of the reference document up to a feature specific maximum. The *LightfieldFourier* and *LightfieldZernike* feature modules, in contrast, perform 20 lookups in the case of 3D model to 3D model comparison and only one lookup when comparing a 2D sketch to 3D models.

7.5 Results: Princeton Shape Benchmark (PSB)

The results of the Princeton Shape Benchmark (PSB) [96] are presented in Figure 7.2. The maximum precision p per level is plotted against ten equidistant recall levels between 0.0 and 1.0. To demonstrate the between-class performance difference, results for randomly selected classes have been included in the figure.
The graphs in Figure 7.2 and some additional statistics reveal some interesting findings regarding the retrieval effectiveness of the examined feature modules with respect to the utilised test collection:



(a) Lightfield Zernike

(b) Lightfield Fourier

(c) Spherical Harmonics (Default)

(d) Spherical Harmonics (HighRes)

Figure 7.2: Precision vs. Recall graphs for the four feature modules: (a) Light field Zernike, (b) Light field Fourier, (c) Spherical Harmonics (Default), (d) Spherical Harmonics (High-Res). The maximum precision p is plotted against 10 equidistant recall levels between 0.0 and 1.0. The dashed lines represent results averaged for different, randomly selected classes whereas the blue line is the mean over all 197 classes in the PSB.

1. All feature modules are capable of finding at least the indexed copy of the reference document (the identity) with a 99.3 % accuracy. This explains the initial precision of 1.0 in most cases. However, in a handful of cases, the identity had rank 2 and instead, another object of the same class was ranked at position 1.

2. The *SphericalHarmonicsDefault* feature module returns at least one relevant result in the top 5, 10 and 20 ranks, 30.2 %, 49.2 % and 59.4 % of the time, respectively. The increase when using the (high resolution) *SphericalHarmonicsHigh* feature module is negligible (approximately 1 %).

3. The *LightfieldZernike* feature module returns at least one relevant result in the top 5, 10 and 20 ranks, 20.7 %, 28.4 % and 40.8 % of the time, respectively.

4. The *LightfieldFourier* feature module returns at least one relevant result in the top 5, 10 and 20 ranks, 15.8 %, 22.0 % and 30.1 % of the time, respectively.

Generally, one can state that the retrieval effectiveness of all four feature modules differs greatly between classes. Some classes (like e.g. pigs, see Figure 7.2) exhibit a much higher accuracy than others (like e.g. trees). This is in line with the findings in [50] and [52]. The results in [52] are especially interesting, as they are also based on the PSB. One can also summarise, that the feature modules based on spherical harmonics perform better overall than the light field based feature modules in the case of 3D model to 3D model comparison. This makes sense, as spherical harmonics features are not prone to PCA alignment errors. In fact, [52] showed that spherical harmonics features perform best, albeit not great, on classes that are sensitive to differences in PCA like for instance the *desk chair* class (see Figure 7.2).

7.6 Interpretation

To start with, it is worth noting, that for the user-driven evaluation results to be significant, it would be necessary to conduct it with a much larger user base. That being said, the following tendencies can nevertheless be deduced from the data:

1. The new *AudioFingerprinting* feature module seems to work reasonably well both for pure audio retrieval as well as video retrieval. It also shows some robustness to noise. However, in addition to the actual hits it produces too many arbitrary results. Those should ideally be filtered. The main issue, though, is retrieval speed.

2. The new audio matching feature modules seem to be able to identify cover versions reliably. However, when it comes to similarity in a broader sense, the features often fail to produce meaningful results. This can be attributed to the fact, that feature modules only consider chroma, with the exception of *MFCCShingle*. As we have elaborated in Chapter 5, there is much more to music than chroma. Again, however, the major issue with those feature modules is also retrieval speed.

3. The new 3D model features also seem to work reasonably well. When it comes to 3D model to 3D model comparison, the modules based on spherical harmonics outperform the light field features both in terms of speed and effectiveness. However, 2D sketch to 3D model comparison can only be facilitated by the light field based features. The user-driven evaluation has shown though, that it is not straightforward to find a 3D model based on a sketch without knowing exactly what the desired model looks like. This weakness can to some extend be compensated by means of the More-Like-This functionality.

4. The image related feature modules designed and described in the original paper [10] still work in the new version of Cineast. Unsurprisingly, they can be used for both video retrieval as well as image retrieval. They also still exhibit the same weaknesses, like for instance the dominance of colour based features.

   Although difficult to back by the available data, it seems that the new image feature modules — namely, *SURFMirflickr25K512* and *HOGMirflickr25K512* — very often fail to contribute to the results in a useful way. This is presumably due to the immense

variety of interest points found in a collection like Mirflickr which we try to squeeze into a tiny 512 element codebook. The technique might be more appropriate for domain-specific use cases, such as a medical image database, rather than a general purpose retrieval system.

With regards to the issue of lookup speed, one can see in Table 7.3 that the need for multiple lookups impairs the execution speed of the respective feature module's kNN-lookup stage. In fact, there seems to be a roughly linear relationship between the number of lookups and the time required to execute them all. So far, we have identified two potential reasons for this: Firstly, multiple lookups — even if submitted in a single batch — are executed sequentially by the current version of ADAMpro, meaning that the available hardware is not fully exploited. Secondly, it seems that necessary data structures are not cached between lookups despite the fact that they could be re-used.

Table 7.2: Results of the user-driven evaluation. The different metrics are captured per scenario (first column). The success rate indicates in how many cases at least one highly relevant item was obtained.

Scenario	**NDCG**	**p@15**	**MRR**	**MAP**	**Success**	**Duration (s)**	**Queries (s)**
A1	0.91	0.60	0.77	0.33	92 %	304.96	1.8
A2	0.96	0.13	0.92	0.10	92 %	220.50	2.2
A3	0.67	0.11	0.63	0.08	69 %	337.34	3.3
A4	0.97	0.08	1.0	0.07	100 %	241.29	1.0
A5	0.61	0.07	0.47	0.06	85 %	224.38	1.8
A6	0.87	0.14	1.0	0.10	100 %	490.43	1.2
A7	0.85	0.11	0.56	0.08	77 %	273.37	1.7
A8	0.74	0.14	0.63	0.10	100 %	268.43	1.0
A9	0.73	0.07	0.64	0.07	92 %	159.63	1.7
A10	0.83	0.12	0.70	0.09	69 %	254.50	2.8
A11	0.97	0.69	1.0	0.38	100 %	96.63	2.6
A12	0.89	0.26	0.77	0.16	77 %	159.45	2.2
B1	0.49	0.17	0.33	0.12	67 %	298.20	1.7
B2	0.94	0.16	1.0	0.11	100 %	116.60	1.1
B3	0.87	0.08	0.80	0.08	100 %	291.02	2.2
B4	0.98	0.07	1.0	0.07	100 %	193.16	1.0
B5	0.97	0.08	1.0	0.08	100 %	162.05	1.0
B6	0.86	0.13	0.92	0.1	100 %	545.428	1.3
B7	0.30	0.03	0.17	0.03	33 %	584.52	2.25
B8	0.77	0.09	0.66	0.08	100 %	296.05	1.2
B9	0.54	0.05	0.39	0.05	67 %	200.83	2.0
B10	0.93	0.1	1.0	0.08	100 %	385.47	4.5
B11	0.92	0.23	1.0	0.15	100 %	80.07	1.6
B12	0.90	0.34	1.0	0.20	100 %	192.47	2.5

Table 7.3: Results of the performance benchmark. All the feature modules are listed alongside with the size of the feature vector, the number of entries, the type of index being used, the number of database lookups performed per query and the average execution time in seconds per stage. Some feature modules perform a constant number of lookups whereas for others, the number of lookups scales with the duration of the query track. Note that for *SphericalHarmonicsLow* and *SphericalHarmonicsHigh* no data was recorded because nobody used those features during the evaluation. **Indices:** VAF = Vector Approximation File, SH = Spectral Hashing, LSH = Locality Sensitive Hashing. See [1] for further information.

Feature	**Size**	**Entries**	**Index**	**Lookups**	**Pre-proc. (ms)**	**kNN (ms)**	**Post-proc. (ms)**
AudioFingerprint	120	519983	LSH	10 per second	96.06	46194.40	1.69
AverageHPCP20F36B	72	519983	VAF	0.50 per second	116.62	10719.80	0.40
AverageHPCP30F36B	72	311599	VAF	0.33 per second	113.70	8181.65	0.035
CENS12Shingle	120	921951	SH	$10 - 30$, depending on duration	207.12	19341.30	1.59
HPCP12Shingle	300	409748	SH	8 per second, max. 50	159.90	42626.40	2.21
MFCCShingle	325	413596	SH	8 per second, max. 50	155.23	42648.40	2.00
SphericalHarmonicsHigh	540	12962	VAF	1	n/a	n/a	n/a
SphericalHarmonicsDefault	330	12962	VAF	1	135.71	58.88	2.85
SphericalHarmonicsLow	220	12962	VAF	1	n/a	n/a	n/a
LightfieldFourier	129	259399	VAF	1 (sketch) or 20 (3D model)	17378.50	17.22	7.62
LightfieldZernike	37	259399	VAF	1 (sketch) or 20 (3D model)	20692.90	20.90	7.65
SURFMirflickr25K512	512	179868	VAF	1	161.94	5653.73	1.13
HOGMirflickr25K512	512	179868	VAF	1	94.82	3835.85	2.25

8

Conclusion and Future work

In thesis we were able to demonstrate that indeed, Cineast — an engine originally designed for content-based video retrieval — can be extended to support additional modalities like audio or 3D models. To a large extent, this was possible due to the power and flexibility of the underlying storage engine ADAMpro and the modular architecture of Cineast itself. To the best of our knowledge, we have thereby created the first true, integrated, content-based multimedia retrieval stack — taking the idea behind [12, 17] one step further.

8.1 Conclusion

In the course of this project, we have re-designed Cineast's extraction pipeline and data model so as to support images, audio and 3D models in addition to the existing video support. Both aspects were constructed so as to remain open to potential future extensions. We have also devised 19 additional feature modules that describe different properties of image, music and 3D model data and build on ideas from various authors. Meanwhile, we laid the foundation for future work in those domains in the form of a simple 3D model and audio processing framework. Furthermore, we have added support for different query modes for the new modalities including Query-by-Example, Query-by-Sketch and a first, not yet functional, prototype of Query-by-Humming. Those modes are fully supported by the new Vitrivr NG user interface, which at the same time maintains the functionality of the original UI. The choice of Angular as framework for Vitrivr NG makes the user interface more maintainable and easily extendible.

The evaluation has shown that most of the new feature modules fulfil their purpose. However, it has also exposed weaknesses, namely, the extremely long lookup times for audio features and the difficulties associated with QbS for 3D model retrieval. Moreover, all features — old and new — sometimes tend to produce results that are incomprehensible for the user. These are issues that ought to be addressed.

8.2 Future work

A project like this is never finished and a lot of questions remain unanswered at the end of this thesis and new questions have been raised. This section summarises some of the topics that merit for a more detailed investigation.

8.2.1 ADAMpro and feature vector lookup

The evaluation has clearly shown that the query performance, in terms of speed, drops dramatically as soon as multiple lookups become necessary. For the shingeling approach [35, 36] applied in all audio related feature modules, multiple lookups are, however, a necessity. It would therefore be worth investigating how ADAMpro can be optimised so as to offer better support and faster speed for those types of queries. In fact, this is probably one of the major obstacles that ought to be addressed and it may not only be relevant for audio retrieval but for all modalities that exhibit temporal progression. Potentially, one could exploit the parallel nature of those shingle lookups. It would also be worth investigating, whether caching of necessary data structures could offer benefits in terms of speed. A more fundamental change would involve offloading the distance calculations to the GPU as proposed by [97, 98]. In the very recent publication by Johnson et al., the authors report 8.5x speedups compared to the current state of the art on billion-scale data sets.

In terms of features, one could further explore ADAMpro's fuzzy set operations and how those could be applied to optimise lookup in general both in terms of performance as well as effectiveness. Last but not least, one must consider adding support for additional distance functions. For instance, distances based on *Dynamic Time Wrapping (DTW)* might be very useful for retrieving versions of musical pieces that differ in tempo. Furthermore, a lot of papers that report on QbH algorithms use the *Earth Mover's Distance* to compare the melody transcriptions. Of course the main challenge here will be having index structures that support efficient lookup with those distances in high-dimensional spaces.

On the Cineast side, additional changes could be made to exploit ADAMpro's more advanced query functionalities. We have already added support for batched queries as part of this thesis. In a second step, one could also add support for ADAMpro's progressive queries. In the progressive query mode, intermediate results are forwarded as they become available. This would allow us to notify users about partial results and thereby make the application more responsive. Currently, progressiveness is only implemented at the level of Cineast by forwarding complete result sets per feature module.

8.2.2 Additional music features

The majority of music features that have been added as part of this thesis are based on chroma and melody. It would be worth investing in additional features based on, for instance, rhythm and tempo, and assessing their influence on the retrieval effectiveness for mid-specificity tasks like version identification or audio matching. The evaluation showed that both work reliably as long as the collection contains actual cover versions of a reference piece. However, most of the remaining results seem arbitrary to the user because they usually do not sound very similar, despite the correspondence in chroma.

In addition, the features associated with QbH require additional work. We have experienced that melody extraction is not a trivial task for polyphonic music and that the transcription accuracy is poor for complex musical pieces. Moreover, in order to create a fully functional QbH system, additional steps will be required like normalisation of the playback speed both during extraction and retrieval, and compensation mechanisms for off-key singing. Adding full QbH support could be a potential project for a dedicated thesis and it would presumably require additions both to Cineast and ADAMpro.

8.2.3 General audio support

General audio support has been deliberately excluded from the scope of this project. This is a non-trivial task for two reasons. Firstly, support of general audio means additional feature modules for different types of audio, such as speech or sound effects. Each of those domains are governed by their own rules and come with their own challenges. Secondly, in order to handle all types of audio efficiently we would require a classification of audio segments as proposed by [37] and dynamic querying based on this classification. This is necessary because without, Cineast would generate and persist a lot of superfluous features — for example a music feature for a pure speech segment. This would inevitably impair the retrieval performance both in terms of speed as well as effectiveness. A segmentation strategy for audio based on self-similarity as described in [56] might be a viable strategy to that end. Such an approach would, however, raise new questions like how one can combine different segmentation strategies for the auditory and visual parts in video.

8.2.4 Optimising 3D model retrieval

In general, the performance of 3D model to 3D model comparison based on the spherical harmonics descriptors seems to work reasonably well. However, it is remarkable that the retrieval effectiveness differs very much between classes of 3D models. This requires additional investigation. Moreover, it would be interesting to systematically explore how different extraction settings like resolution of the voxel grid, number and orders of the harmonics and interval between radii influences the retrieval effectiveness.

When searching a collection of 3D models for a particular item, however, 3D model to 3D model comparison is usually not the approach of choice because naturally, the user lacks a reference object at that point. Therefore, more effort should be put into the light field based feature modules that facilitate 2D sketch to 3D model comparison. Again, the evaluation has shown that the QbS approach works for 3D models in theory. However, in practice it is often difficult to find an object based on a sketch alone because the feature modules seem to be very sensitive to minor deviations of the shape. On the one hand, this could be addressed by further fine-tuning the existing features by, for example, experimenting with the orders of the Fourier and Zernike coefficients or the weights used during distance calculation. On the other hand, support for new features like the ones listed in [50, 52] could be added to Cineast and combined with the existing feature modules. Namely, it might be worth testing the *depth buffer descriptor*.

Last but not least it would be interesting to take the QbS paradigm for 3D models one step

further and add support for QbE with arbitrary images. This would allow for completely different use cases, like finding 3D models that were used in a video. However, it would also require more advanced image segmentation.

8.2.5 Combination with boolean retrieval

As part of this thesis, we have added support for extracting and storing file metadata. This functionality could be combined with existing frameworks for file content description, such as EXIF[26], ID3[27] or IPTC[28]. In a next step, one could think about adding support for querying that metadata and combining such boolean queries with kNN-search.

8.2.6 Evaluation

Evaluation is probably the achilles heel of this thesis. The proposed method of user driven evaluation (see Chapter 7) should be further refined and carried out on a much larger scale so as to obtain meaningful results with regards to retrieval effectiveness. Furthermore, one should probably complement such high-level evaluation with traditional retrieval experiments carried out for individual feature modules, as we did with the Princeton Shape Benchmark. The challenge here is preparation of collections that are suitable for those experiments. For instance, a collection for testing audio matching features must satisfy different criteria than one for testing audio fingerprinting. This is why preparing the different test sets for all the different features would have been beyond the scope of this thesis' time frame. However, maybe future efforts can profit from recent developments like the FMA data set for music retrieval [99].

8.2.7 User interface

The new user interface applies new concepts for formulating and setting up queries. Some of these concepts seem to work very well, others need further refinement. For instance, the idea of tuning the specificity of a query on a simple slider seems to be very approachable. However, the selection of the appropriate feature categories in the background is not as straightforward as we first thought and more work could be invested into fine-tuning these settings and the categories themselves. In addition, one could experiment with different weighting schemes for the final, late fusion step that takes place at the level of the UI.
Adding support for additional query modes might also be an interesting project to consider. For music retrieval, Query-by-Keyboard might be a viable alternative to QbH and it would certainly simplify some of the online pre-processing steps. Another mode would be Query-by-Tapping — which requires features related to rhythm. Both these query modes are already offered by *musipedia*[29].
Finally, we have received a lot of qualitative feedback during the evaluation all of which

[26] https://www.jeita.or.jp/japanese/standard/book/CP-3451C_E
[27] https://id3.org
[28] https://iptc.org/standards/photo-metadata
[29] http://www.musipedia.org

could be included in a second iteration of the user interface. The feedback included minor things like an “undo” functionality for sketching but also major changes and additions. Among other things, it was pointed out that the UI needs further optimisation in order to fully support tablets. One could even go so far as to add support to the full range from mobile to desktop device.

8.3 Vision

We believe that content-based multimedia retrieval will only add to its already considerable importance for reasons that have been explained in Chapter 1. More and more institutions and companies will be facing the question, how large, heterogenous multimedia collections can be stored and managed and how particular items can be retrieved without the need of manual annotation. But even in the presence of textual annotation, content-based retrieval paves the way for completely different use cases that cannot possibly be satisfied by classical text-based retrieval alone.

As Apache Lucene[30] and related projects like Apache Solr[31] or Elastic Search[32] made text retrieval available to a broad audience, so could a software stack based on Cineast and ADAMpro have the potential to make content-based multimedia retrieval accessible to everyone. For that reason, it might be worth pursuing a path where Cineast becomes the center of an integrated, extendible open-source framework targeted at efficient indexing and retrieval of multimedia collections.

[30] https://lucene.apache.org
[31] http://lucene.apache.org/solr
[32] https://www.elastic.com

Bibliography

[1] Giangreco, I. and Schuldt, H. ADAMpro : Database Support for Big Multimedia Retrieval. *Datenbank-Spektrum*, 16(1):17–26 (2016).

[2] Weiss, Y., Torralba, A., and Fergus, R. Spectral Hashing. *Advances in Neural Information Processing Systems*, (1):1–8 (2008).

[3] Weber, R., Blott, S., Ave, M., and Hill, M. Similarity-Search Analysis Methods and Performance Study for in High-Dimensional Spaces. In *Proceedings of the 24th VLDB Conference*, pages 194–205 (1998).

[4] Lew, M. S., Sebe, N., Djeraba, C., and Jain, R. Content-based multimedia information retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2(1):1–19 (2006).

[5] Wan, G. G. and Liu, Z. Content-Based Information Retrieval and Digital Libraries. *Information Technology and Libraries*, 27(1):41–47 (2013).

[6] Gantz, J. and Reinsel, D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(December 2012):1–16 (2012).

[7] Amato, F., Greco, L., Persia, F., and Poccia, S. R. Content-Based Multimedia Retrieval. In Colace, F., De Santo, M., Moscato, M., Picarello, A., Schreiber, F., and Tanca, L., editors, *Data Management in Pervasive Systems*, pages 291–310. Springer (2015).

[8] a.W.M. Smeulders, Worring, M., Santini, S., Gupta, A., and Jain, R. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380 (2000).

[9] Rossetto, L., Giangreco, I., Tanase, C., and Schuldt, H. vitrivr: A Flexible Retrieval Stack Supporting Multiple Query Modes for Searching in Multimedia Collections. In *ACM Multimedia* (2016).

[10] Rossetto, L. *Cineast: a Content-based Video Retrieval Engine*. Master thesis, University of Basel (2014).

[11] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Qian Huang, Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., and Yanker, P. Query by image and video content: the QBIC system. *Computer*, 28(9):23–32 (1995).

[12] Kiranyaz, S., Caglar, K., Guldogan, E., Guldogan, O., and Gabbouj, M. MUVIS: A Content-based Multimedia Indexing and Retrieval Framework. *IEEE* (2003).

[13] Indyk, P. and Motwani, R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613 (1998).

[14] Theodoridis, S. and Koutroumbas, K. *Pattern Recognition*. Academic Press Inc., 4 edition.

[15] Singhai, N. and Shandilya, S. K. A Survey On: Content Based Image Retrieval Systems. *International Journal of Computer Application*, 4(2):1827–1836 (2010).

[16] Zhang, D. and Lu, G. A Comparative Study of Fourier Descriptors for Shape rRepresentation and Retrieval. In *ACCV2002: The 5th Asian Conference on Computer Vision*, pages 1–6. Melbourne, Australia (2002).

[17] Kelly, P. M. and Cannon, M. Query by image example: the CANDID approach. *Integration The Vlsi Journal*, (April):20–24 (2000).

[18] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110 (2004).

[19] Bay, H. and Ess, A. Speeded-Up Robust Features (SURF). 110(September):346–359 (2008).

[20] Jégou, H., Douze, M., Schmid, C., and Pérez, P. Aggregating local descriptors into a compact image representation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311. IEEE, San Francisco, CA, USA (2010).

[21] Perronnin, F. and Dance, C. Fisher Kernels on Visual Vocabularies for Image Categorization. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Minneapolis, MN, USA (2007).

[22] Yang, J., Jiang, Y.-G., Hauptmann, A. G., and Ngo, C.-W. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval - MIR '07*, volume 63, pages 197–206. Augsburg, Germany (2007).

[23] Liu, J. Image Retrieval based on Bag-of-Words model. *CoRR (Computing Research Repository)*, abs/1304.5 (2013).

[24] Ahmad, N. Evaluation of SIFT and SURF using Bag of Words Model on a Very Large Dataset. *Sindh University Research Journal (Science Series)*, 45(3):492–495 (2013).

[25] Downie, J. S., Ehmann, A. F., Bay, M., and Jones, M. C. *The music information retrieval evaluation eXchange: Some observations and insights*. Springer (2010).

[26] Grosche, P., Müller, M., and Serrà, J. Audio Content-Based Music Retrieval. In Müller, M. and Schedl, M., editors, *Multimodal Music Processing*, volume 3, pages 157–174 (2012).

[27] Wang, A. The Shazam Music Recognition Service. *Communications of the ACM*, 49(8):44–48 (2006).

[28] Foote, J. T. Content-based retrieval of music and audio. In *Proc. SPIE 3229, Multimedia Storage and Archiving Systems II*, pages 138–147 (1997).

[29] Sigurdson, S., Petersen, K. B., and Larsen, J. Mel Frequency Cepstral Coefficients: An Evaluation of Robustness of MP3 Encoded Music. In *Proceedings of the 7th International Conference on Music Information Retrieval*, pages 3–6 (2006).

[30] Fujishima, T. Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music. In *ICMC Proceedings*, volume 9, pages 464–467 (1999).

[31] Gómez, E. *Tonal Description of Music Audio Signals*. Doctoral dissertation, Universitat Pompeu Fabra, Barcelona (2006).

[32] Kurth, F. and Müler, M. Efficient Index-Based Audio Matching. *IEEE Transactions on Audio, Speech and Language Processing*, 16(2):382–395 (2008).

[33] Salamon, J., Serrà, J., and Gómez, E. Tonal representations for music retrieval: from version identification to query-by-humming. *International Journal of Multimedia Information Retrieval*, 2(1):45–58 (2013).

[34] Grosche, P. and Muller, M. Toward characteristic audio shingles for efficient cross-version music retrieval. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 473–476. IEEE (2012).

[35] Casey, M. and Slaney, M. Song Intersection by Approximate Nearest Neighbor Search. In *Proc Int Society Music Information Retrieval Conf (ISMIR)*, pages 144–149 (2006).

[36] Casey, M., Rhodes, C., and Slaney, M. Analysis of Minimum Distances in High-Dimensional Musical Spaces. *IEEE Transactions on Audio, Speech and Language Processing*, 16(5):1015–1028 (2008).

[37] Doğan, E., Sert, M., and Yazici, A. A Flexible and Scalable Audio Information Retrieval System for Mixed-Type Audio Signals. *International Journal of Intelligent Systems*, 26(10):952–970 (2011).

[38] Klapuri, A. P. Multiple Fundamental Frequency Estimation Based on Harmonicity and Spectral Smoothness. *IEEE Transactions on Speech and Audio Processing*, 11(6):804–816 (2003).

[39] Klapuri, A. A Perceptually Motivated Multiple-F0 Estimation Method. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. New Paltz, NY (2005).

[40] Klapuri, A. Multiple Fundamental Frequency Estimation by Summing Harmonic Amplitudes. In *Proceedings of the International Symposium/Conference on Music Information Retrieval (ISMIR)*, pages 216–221 (2006).

[41] Ryynänen, M. and Klapuri, A. Transcription of the Singing Melody in polyphonic music. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 222–227 (2006).

[42] Batke, J.-M., EIsenberg, G., Weishaupt, P., and Sikora, T. A Query by Humming System Using MPEG-7 Descriptors. In *116th Audio Engineering Society Convention*, page 8. Audio Engineering Society, Berlin, Germany (2004).

[43] Ito, A., Kosugi, Y., Makino, S., and Ito, M. A Query-by-Humming Music Information retrieval from Audio Signals based on Multiple F0 Candidates. In *International Conference on Audio, Language and Image Processing*, pages 1–5 (2010).

[44] Kazhdan, M., Funkhouser, T., and Rusinkiewicz, S. Rotation Invariant Spherical Harmonic Representation of 3D Shape Descriptors. In *Eurographics Symposium on Geometry Processing*, volume 43, pages 156–164 (2003).

[45] Vranic, D., Saupe, D., and Richter, J. Tools for 3D-object retrieval: Karhunen-Loeve transform and spherical harmonics. In *2001 IEEE Fourth Workshop on Multimedia Signal Processing (Cat. No.01TH8564)*, pages 293–298. IEEE (2001).

[46] Bustos, B., Keim, D., Saupe, D., and Schreck, T. Content-Based 3D Object Retrieval. *IEEE Computer Graphics and Applications*, 27(4):22–27 (2007).

[47] Novotni, M. and Klein, R. A geometric approach to 3D object comparison. In *Proceedings International Conference on Shape Modeling and Applications*, pages 167–175 (2001).

[48] Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., and Jacobs, D. A search engine for 3D models. *ACM Transactions on Graphics*, 22(1):83–105 (2003).

[49] Saupe, D. and Vranic, D. 3D model retrieval with spherical harmonics and moments. In *Proceedings of the 23rd DAGM-Symposium on Pattern Recognition*, volume 2191, pages 392–397. Springer, London, UK (2001).

[50] Tangelder, J. W. H. and Veltkamp, R. C. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications*, 39(3):441–471 (2007).

[51] Vranić, D. and Saupe, D. 3D Model Retrieval. In *Spring Conference on Computer Graphics and its Applications*, pages 89–93 (2000).

[52] Bustos, B., Keim, D., Saupe, D., Schreck, T., and Vranić, D. An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries*, 6(1):39–54 (2006).

[53] Chen, D.-Y., Tian, X.-P., Shen, Y.-T., and Ouh. On Visual Similarity Based 3D Model Retrieval. In *Eurographics*, volume 22, pages 313–318 (2003).

[54] Wang, F., Kang, L., and Li, Y. Sketch-based 3D Shape Retrieval using Convolutional Neural Networks. *CoRR*, abs/1504.0 (2015).

[55] Dübelin, P. *SanDBox - A Modular Component-based and Scalable Multimedia Information Retrieval Framework*. Master thesis, University of Basel (2015).

[56] Foote, J. Automatic Audio Segmentation Using A Measure of Audio Novelty. In *International Conference on Multimedia and Expo*, pages 452–455. IEEE, New York, NY, USA (2000).

[57] Swain, M. J. and Ballard, D. H. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32 (1991).

[58] Lowe, D. G. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 1150–1157. Kerkyra, Greece (1999).

[59] McConnell, R. Method of and apparatus for pattern recognition (1986).

[60] Zhang, D. and Lu, G. An Integrated Approach to Shape Based Image Retrieval. In *ACCV2002: The 5th Asian Conference on Computer Vision*. Melbourne, Australia (2002).

[61] Hwang, S.-K. and Kim, W.-Y. A novel approach to the fast computation of Zernike moments. *Pattern Recognition*, 39(11):2065–2076 (2006).

[62] Padilla Vivanco, A., Martínez-Ramírez, A., and Granados-Agustín, F. Digital image reconstruction by using Zernike moments. *Proceedings of SPIE - The International Society for Optical Engineering* (2004).

[63] Huiskes, M. and Lew, M. S. The MIR Flickr Retrieval Evaluation. In *ACM International Conference on Multimedia Information Retrieval (MIR'08)*. Vancouver, Canada (2008).

[64] Abeles, P. BoofCV (2012).

[65] Velmurugan, K. and Santosh Baboo, S. Image Retrieval using Harris Corners and Histogram of Oriented Gradients. *International Journal of Computer Applications*, 24(7):6–10 (2011).

[66] Tipler, P. A. and Mosca, G. Physik. Elsevier GmbH, München, 2. auflage edition (2004).

[67] Cutnell, J. D. and Kenneth W. Johnson. *Physics*. New York: Wiley, 4th edition (1998).

[68] Harris, F. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform (1978).

[69] Grandke, T. Interpolation Algorithms for Discrete Fourier Transforms of Weighted Signals. *IEEE Transactions on Instrumentation and Measurement*, 32(2):350–355 (1983).

[70] Zwicker, E. Subdivision the Audible Frequency range into Critical Bands (Freqzebt. *The Journal of the Acoustical Society of America*, 33(2):248 (1961).

[71] Stevens, S. S., Volkmann, J., and Newman, E. B. A Scale for the Measurement of the Psychological Magnitu Pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190 (1937).

[72] Helmholtz, H. On the Sensations of Tone (1954).

[73] UNSW, S. o. P. Note names, MIDI numbers and frequencies. URL http://newt.phys.unsw.edu.au/jw/notes.html.

[74] Shepard, R. N. *Structural representations of musical pitch*. Swets & Zeitlinger, first edit edition (1982).

[75] Gómez, E. Tonal Description of Polyphonic Audio for Music Content Processing. *INFORMS Journal on Computing*, 18(3):294–304 (2006).

[76] Salamon, J. and Gomez, E. Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech and Language Processing*, 20(6):1759–1770 (2012).

[77] Muller, M., Kurth, F., and Clausen, M. Chroma-based statistical audio features for audio matching. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.*, pages 275–278. IEEE, New Paltz, NY, USA (2005).

[78] Huang, J. H. J., Yagel, R. Y. R., Filippov, V. F. V., and Kurzion, Y. K. Y. An Accurate Method for Voxelizing Polygon Meshes. In *IEEE Symposium on Volume Visualization*, pages 119–126. IEEE, Research Triangle Park, NC, USA (1998).

[79] Teague, M. R. Image analysis via the general theory of moments. *Journal of the Optical Society of America (1917-1983)*, 70(8):920–930 (1980).

[80] Jansen, B. J. and Rieh, S. Y. The Sventeen Theoretical Constructs of Information Searching and Information Retrieval. *Journal of the American Society for Information Science and Technology*, 61(8):1517–1534 (2010).

[81] Bookstein, A. Relevance. *Journal of the American Society for Information Science*, 30(5):269–273 (1979).

[82] Cooper, W. S. A definition of relevance for information retrieval. *Information Storage and Retrieval*, 7(1):19–37 (1971).

[83] Voorhees, E. M. and Harman, D. K. TREC: Experiment and Evaluation in Information Retrieval. *Digital Libraries and Electronic Publishing*, page 462ff (2005).

[84] TREC. English Relevance Judgements (2006). URL http://trec.nist.gov/data/reljudge_eng.html.

[85] Müller, H., Müller, W., Squire, D. M., Marchand-Maillet, S., and Pun, T. Performance evaluation in content-based image retrieval: overview and proposals. *Pattern Recognition Letters*, 22(5):593–601 (2001).

[86] Arguello, J. Evaluation Metrics (2013). URL https://ils.unc.edu/courses/2013_spring/inls509_001/lectures/10-EvaluationMetrics.pdf.

[87] Yao, Y. Y. Measuring Retrieval Effectiveness Based on User Preference of Documents. *Journal of the American Society for Information Science*, 46(2):133–145 (1995).

[88] Järvelin, K. and Kekäläinen, J. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00*, pages 41–48. Athens, Greece (2000).

[89] Sebe, N., Hujismans, N., Tian, Q., and Gevers, T. Toward complete performance characterization in content-based retrieval. In *Proc. SPIE 5670, Internet Imaging VI, 213* (2005).

[90] Buckley, C. and Voorhees, E. M. Retrieval Evaluation with Incomplete Information. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32 (2004).

[91] Järvelin, K. and Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446 (2002).

[92] Schedl, M., Gomez, E., and Urbano, J. *Music Information Retrieval: Recent Developments and Applications*, volume 8 (2014).

[93] Zipf, G. K. *Human Behavior and the Principle of Least Effort*. Addison-Wesley (1994).

[94] Rossetto, L., Giangreco, I., and Schuldt, H. OSVC-Open Short Video Collection 1.0. Technical report, University of Basel, Basel (2015).

[95] Chen, D.-Y., Tian, X.-P., Shen, Y.-T., and Ouhyoung, M. On Visual Similarity Based 3D Model Retrieval. *Computer Graphics Forum*, 22(3):223–232 (2003).

[96] Shilane, P., Min, P., Kazhdan, M., and Funkhouser, T. The princeton shape benchmark. In *Proceedings of International Conference on Shape Modeling and Applications (SMI'04)*, pages 167–178. IEEE CS Press (2004).

[97] Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *CoRR*, abs/1702.0 (2017).

[98] Garcia, V., Debreuve, E., and Barlaud, M. Fast k nearest neighbor search using GPU. In *Computer Vision and Pattern Recognition*, 2, pages 1–6. IEEE, Anchorage, AK, USA (2008).

[99] Defferrard, M., Benzi, K., Vandergheynst, P., and Bresson, X. FMA: A Dataset For Music Analysis. *CoRR*, abs/1612.0 (2016).

A

Illustrations regarding DSP

The following figures illustrate some of the concepts introduced in section 5.1 on page 35.

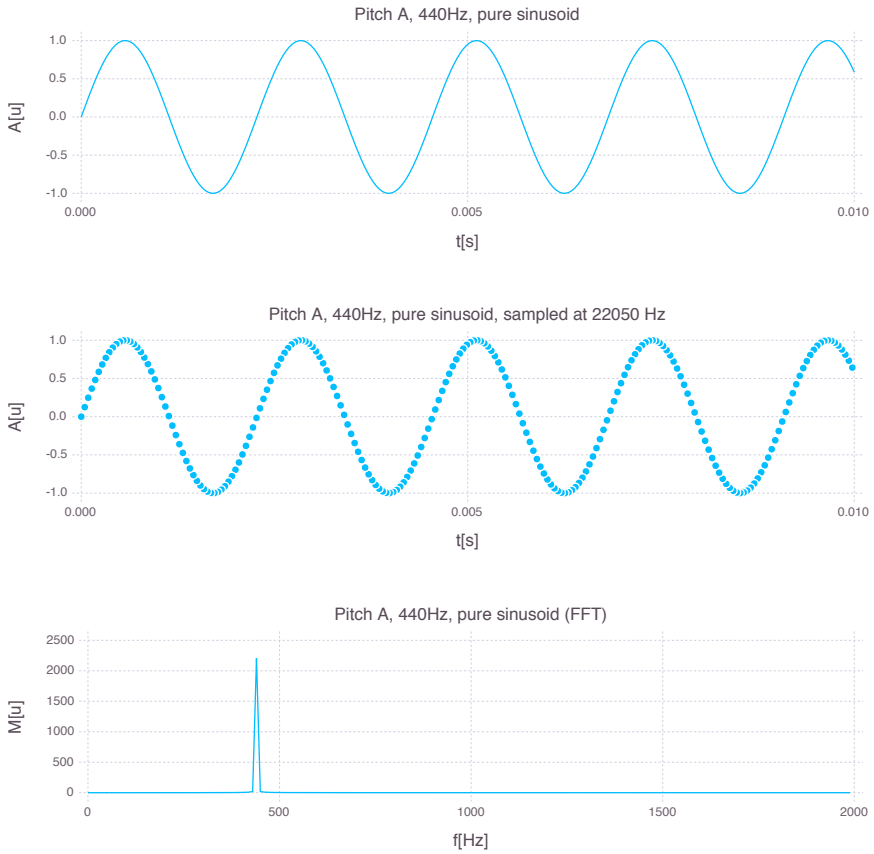


Figure A.1: Pure sinusoid wave of 440 Hz (pitch A above the middle C). The top graph visualises the temporal development of the signal amplitude between $t_0 = 0.0\,\text{s}$ and $t_1 = 0.010\,\text{s}$ (time domain). The graph in the middle illustrates the sampling process at a sampling rate $f_s = 22\,050\,\text{Hz}$, again in the time domain. The last graph depicts the magnitude spectrum of the signal (frequency domain). Note that it contains a single peak centered at approximately 440 Hz, which is the frequency of the signal.

Figure A.2 depicts an artificial, sinusoid function of 440 Hz that contains higher level partials. It still has a very simple form both in terms of the number of partials it contains as well as because it does not exhibit any phase differences. However, the example illustrates the concept of superposition and how such a signal can be decomposed into frequency components by the FFT. The signal is defined by the following equation:

$$y(t) = 40.0\sin(880\pi t) + 22.6\sin(1760\pi t) + 11.45\sin(3520\pi t) + 3.8\sin(7040\pi t)$$

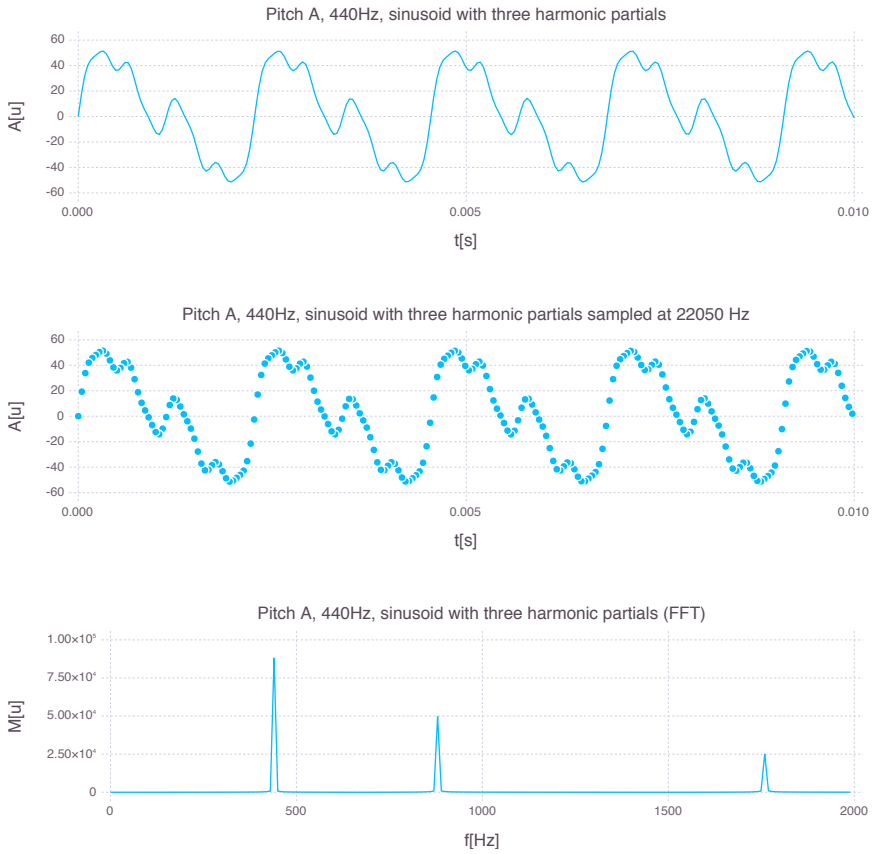


Figure A.2: Sinusoid wave of 440 Hz (pitch A above the middle C) that contains three higher level, harmonic partials at 880 Hz, 1760 Hz and 3520 Hz. The top graph visualises the temporal development of the signal amplitude between $t_0 = 0.0\,\text{s}$ and $t_1 = 0.010\,\text{s}$ (time domain). The graph in the middle illustrates the sampling process at a sampling rate of $f_s = 22\,050\,\text{Hz}$. The last graph depicts the magnitude spectrum of the signal (frequency domain). Note that there are three visible peaks. The peak at approximately 440 Hz belongs to the fundamental frequency f_0 and the other peaks to its higher partials.

B

Evaluation scenarios

B.1 Scenario A

The following objectives A1 to A12 are part of the evaluation scenario A.

A1: Green meadows

Description Take an image depicting a meadow and use that image as a reference image for Query-by-Example. Rank the results with respect to their subjective relevance. You can use any means to obtain the reference image e.g. Google.

Material None

Illustrations Example image of a meadow. See figure B.1.

A2: Nuremberg Castle

Description We are looking for the coloured version of the provided image, which should be contained in the database. Using the provided grayscale image as reference, find that image in the collection and rank the remaining results according to their subjective relevance. You can use both Query-by-Example and More-Like-This.

Hint: Colour features are obviously not helpful for this task. You can switch them off after having executed a query through Vitrivr's query refinement functionality.

Material Grayscale example image Nuremberg castle. See figure B.2

Illustrations None

A3: Pro 7 Logo

Description We are looking for an image depicting the logo of Pro7 (German TV station). Try to find that image using Query-by-Sketch and More-Like-This and rate the remaining results according to their subjective relevance.

Material None

Illustrations Example image depicting the Pro 7 Logo. See figure B.3

A4: Delusion Rain

Description We have provided you with a reference clip of the song 'Delusion Rain'. Try to find the original soundtrack in the collection using the clip and rate the remaining results according to their relevance.

Hint: This is a high specificity task. Use the (far left) audio fingerprinting setting.

Material 3.5 seconds clip (0:15) of the song 'Delusion Rain' by 'Mystery' (Format: WAV, 44 100 Hz, 16 bit, stereo)

Illustrations None

A5: Delusion Rain (Distorted)

Description We have provided you with a reference clip for the song 'Delusion Rain', which exhibits some degree of noise. Try to find the original soundtrack in the collection using the clip and rate the remaining results according to their relevance.

Hint: This is a high specificity task. Use the (far left) audio fingerprinting setting.

Material 3.0 seconds clip (0:15) of the song 'Delusion Rain' by 'Mystery' that exhibits a mixture of white and rose noise (Format: WAV, 44 100 Hz, 16 bit, stereo)

Illustrations None

A6: Swan Lake

Description The collection contains multiple versions of 'Swan Lake' by Tchaikowski. Using an excerpt of your choice, try to find the other versions. Rate the results with respect to their relevance.

Hint: This is a mid specificity task which is why you are likely to receive results other than the piece we're looking for. Try to assess the relevance of those pieces as well by listening to the most relevant segment.

Material 5.0 seconds and 7.0 seconds clip of Piotr Ilyich Tchaikovsky's 'Swan Lake' performed by the Berlin Symphonic Orchestra (Format: WAV, 44 100 Hz, 16 bit, stereo)

Illustrations None

A7: Big Buck Bunny

Description We are looking for the depicted scene from the movie 'Big Buck Bunny'. Try to find it using Query-by-Sketch only. Rank the remaining results according to their subjective relevance.

Figure B.1: Illustration used in objective A1 (Source: Wikimedia Commons, Nikater).

Figure B.2: Example image to use for Query-by-Example in objective A2 (Source: pixabay.com).

Figure B.3: Example image to use for Query-by-Example in objective A3 (Source: pixabay.com).

Material None

Illustrations Illustration of the desired scene. See figure B.4

A8: Land of the Longears

Description We are looking for a scene from a movie 'Land of the Longears'. This time, however, you should combine an image with a Query-by-Example search using the provided audio excerpt. You can either use the provided image or another, similar image of your choice.

Hints: Use the version identification mode for the audio Query-by-Example (slider setting right after audio fingerprinting).

Material 10.0 seconds clip (0:11) of Piotr Ilyich Tchaikovsky's 'Dance of the Prince & The Sugar Plum Fairy' performed by the Berlin Symphonic Orchestra (Format: WAV, 44 100 Hz, 16 bit, stereo)

Illustrations Example image that is similar to the desired scene. See figure B.5

A9: The Killing Joke

Description Use this distorted example image and try to find the video scene it depicts in the collection. Rank the remaining results according to their subjective relevance. You can use Query-by-Example and More-Like-This for this task.

Material Distorted example image from the desired scene. See figure B.6

Illustrations None

A10: U.S.S. Enterprise

Description The collection contains a 3D model of the starship U.S.S Enterprise (See illustration image). Try to retrieve that 3D model using **only** Query-by-3D-Sketch and More-Like-This and rank the remaining results with respect to their relevance.

Material None

Illustrations Illustration image of the starship U.S.S. Enterprise. See figure B.7

A11: Airplanes

Description Pick an airplane 3D model from the provided collection and perform a Query-by-Example with that model. Rank the results with respect to their relevance. If required you are also allowed to use More-Like-This.

Figure B.4: Example image used as template for objective A7 (Source: Movie, Big Buck Bunny, Blender Foundation, 2008).

Figure B.5: Example image used as illustration for objective A8 (Source: http://www.islay.org.uk, May 2017).

Figure B.6: Example image used for objective A9 (Source: The Killing Joke, Sebastian Lopez).

Figure B.7: Illustrative image used for objective A10 (Source: http://www.cygnus-x1.net, May 2017).

Material Five different airplane models from the NTU 3D collection [95], namely Y3015, Y3031, Y3058, Y3151 and (Format: Wavefront OBJ).

Illustrations None

A12: A Bolt

Description We are looking for a 3D model (STL or OBJ file) of a bolt in the database. Try to find it in the collection by whatever means you prefer. You could for instance perform a Query-by-Example or Query-by-3D-Sketch and any external resource you think can help you (e.g. from Google).

Material None

Illustrations Illustration of a bolt. See figure B.8

B.2 Scenario B

The following objectives B1 to B12 are part of scenario B.

B1: Mountains

Description Take an image depicting some mountains and use that image as an example for Query-by-Example. Rank the results with respect to their subjective relevance. You can use any means to obtain the reference image, e.g. Google.

Material None

Illustrations Example image of mountains. See figure B.9

B2: Thunderstorm

Description We are looking for the version of the provided image, which should be contained in the database. Using the distorted reference image, retrieve the image we are looking for from the collection and rank the remaining results according to their subjective relevance. You can use both Query-by-Example and More-Like-This.

Material Example image of a thunderstorm to be used for Quer-by-Example in objective B2. See figure B.10

Illustrations None

B3: Recycling Icon

Description We are looking for an image depicting the green Recycling Icon (see Illustration). Try to find that image using **only** Query-by-Sketch and More-Like-This and rate the remaining results according to their subjective relevance.

Figure B.8: Illustrative image used for objective A12 (Source: http://www.in.all.biz, May 2017).

Figure B.9: Illustration used in objective B1 (Source: Pixabay).

Figure B.10: Example image used in objective B2 (Source: Pixabay).

Material None

Illustrations Illustration of the desired recycling icon. See figure B.11

B4: Fade to Black

Description We have provided you with a reference clip for the song 'Fade to Black'. Try to find the original soundtrack in the collection using the clip for a Query-by-Example and rate the remaining results according to their relevance.

Hint: This is a high specificity task. Use the (far left) audio fingerprinting setting.

Material 3 seconds clip (1:10) from the song 'Fade to Black' by Metallica (Format: WAV, 44 100 Hz, 16 bit, mono).

Illustrations None

B5: Fade to Black (Distorted)

Description We have provided you with a reference clip for the song 'Fade to Black', which exhibits some degree of noise. Try to find the original soundtrack in the collection using the clip and rate the remaining results according to their relevance.

Hint: This is a high specificity task. Use the (far left) audio fingerprinting setting.

Material 3 seconds clip (1:15) from the song 'Fade to Black' by Metallica that exhibits a mixture of white and rose noise (Format: WAV, 44 100 Hz, 16 bit, mono).

Illustrations None

B6: Gimme! Gimme! Gimme!

Description The provided segment of an ABBA song has been re-used by Madonna in her song 'Hung up!'. Try to retrieve both versions using the provided reference clip and rate the results according to their relevance.

Hint: This is a mid specificity task — use the middle setting on the slider. You are likely to receive results other than the piece we are looking for. Try to assess the relevance of those pieces as well by listening to the most relevant segment.

Material 9 seconds clip (0:18) from the song 'Gimme! Gimme! Gimme' by ABBA (Format: WAV, 44 100 Hz, 16 bit, stereo).

Illustrations None

B7: Sintel

Description We are looking for the depicted scene from the movie 'Sintel'. Try to find it using Query-by-Sketch **only**. Rank the remaining results according to their subjective relevance.

Material None

Illustrations Illustration of the desired scene. See figure B.12

B8: Sintel (revisited)

Description Again, we are looking for a scene from them movie 'Sintel'. This time, however, you shall combine the example image with one of the provided audio excerpts for a Query-by-Example search.

Hint: Use the version identification mode for the audio Query-by-Example (slider setting right after audio fingerprinting).

Material An example image depicting the desired scene (see figure B.13) and a 5 seconds and 7 seconds audio excerpt (Format: WAV, 44 100 Hz, 16 bit, stereo).

Illustrations None

B9: Phantom of the Floppera

Description There is a video in the collection that depicts an oscilloscope, pretty much fullscreen. Try to find that scene using a reference image of your choice. You can use Query-by-Example and More-Like-This for this task.

Material None

Illustrations An example image depicting an oscilloscope (see figure B.14)

B10: The 'R' in retrieval

Description We are looking for a 3D model of the uppercase letter 'R'. Try to find it in the database using **only** Query-by-Sketch and More-Like-This queries.

Material None

Illustrations None

B11: Chess Pieces

Description Pick a chess piece 3D model from the provided examples and perform a Query-by-Example with that model. Rank the results with respect to their subjective relevance. If required you are also allowed to use More-Like-This.

Material Three different chess piece models from the Thingyverse collection (Format: Stereolithography STL).

Illustrations None

Figure B.11: Illustration used as template for Query-by-Sketch in objective B3 (Source: Pixabay).

Figure B.12: Illustration used as template for Query-by-Sketch in objective B7 (Source: Sintel, Blender Foundation, 2010).

Figure B.13: Example image of the desired scene used as reference object in objective B8 (Source: Sintel, Blender Foundation, 2010).

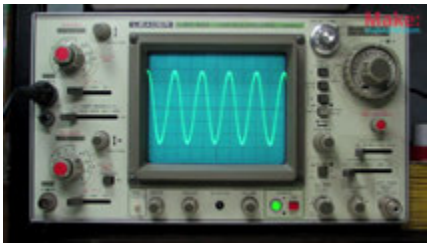


Figure B.14: Illustrative image of an oscilloscope for objective B9 (Source: https://i.ytimg.com, May 2017).

B12: Gears

Description You're looking for a 3D model (STL file) of a gear in the database. Try to find it in the collection by whatever means you prefery. You could for instance use Query-by-Example or Query-by-3D-Sketch and any external resource you think can help you (e.g from Google).

Material None.

Illustrations Illustrative image of a gear (see figure B.15).

Figure B.15: Illustrative image of a gear for objective B12 (Source: http://i154.photobucket.com, May 2017).

C

Query images

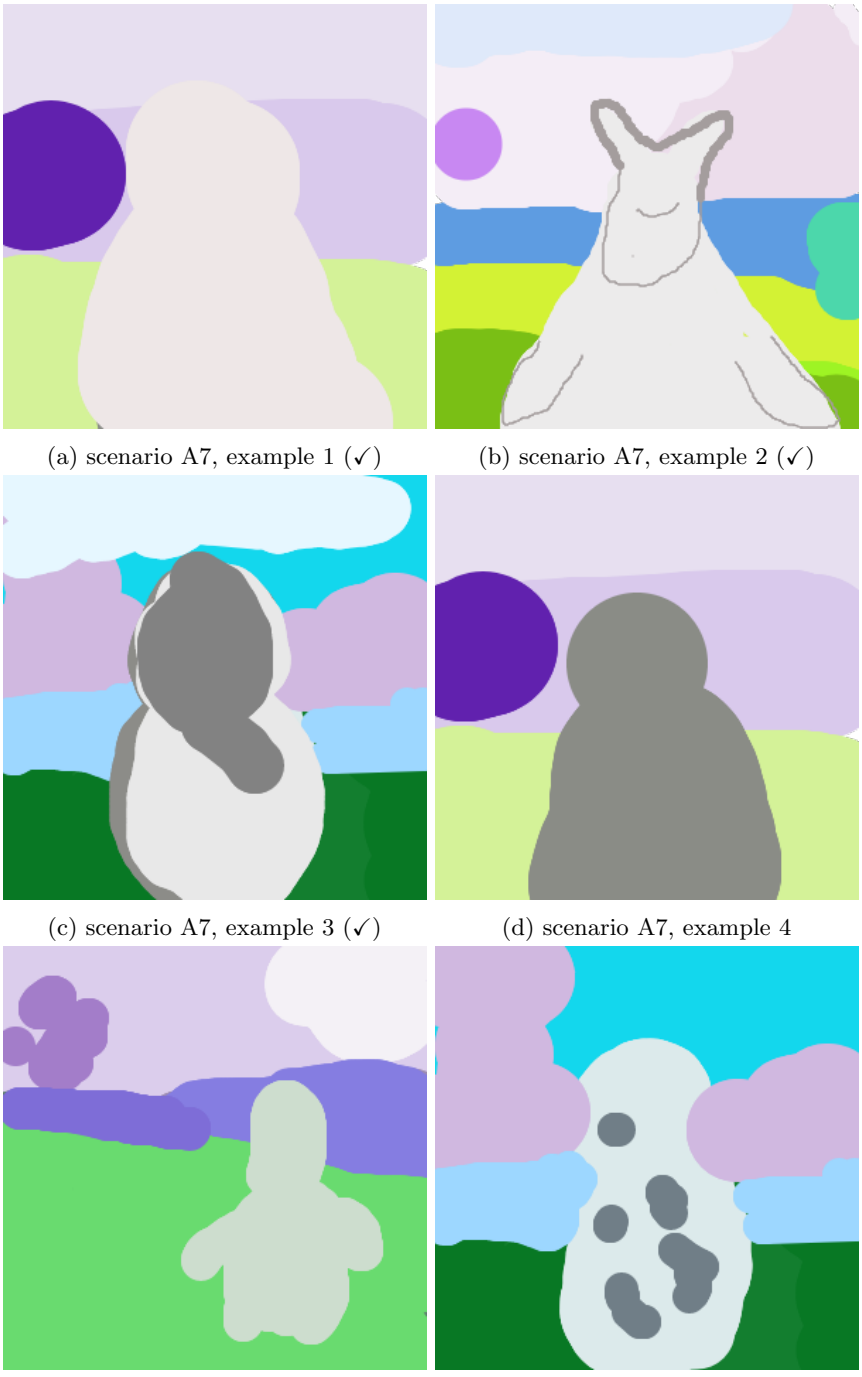
The chapter gives some examples of reference images that were used in the different evaluation scenarios. We use checkmarks (✓) to mark images that produce highly relevant results in the top 15 positions right away; that is, without the need for More-Like-This.

(a) scenario A1, example 1 (✓) (b) scenario A1, example 2 (✓)

Figure C.1: Example images used as reference documents in scenario A1. Regardless of choice, most images were dominated by green and blue colours.

(a) scenario A3, example 1 (✓) (b) scenario A3, example 2

Figure C.2: Sketches used as reference documents in scenario A3. The two examples look very similar but despite, only the first image produces highly relevant results in the top 15.



(a) scenario A7, example 1 (✓)

(b) scenario A7, example 2 (✓)

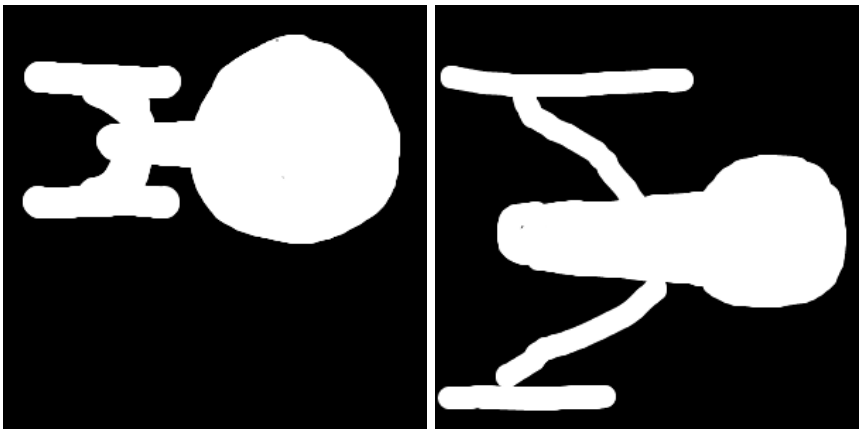
(c) scenario A7, example 3 (✓)

(d) scenario A7, example 4

(e) scenario A7, example 5

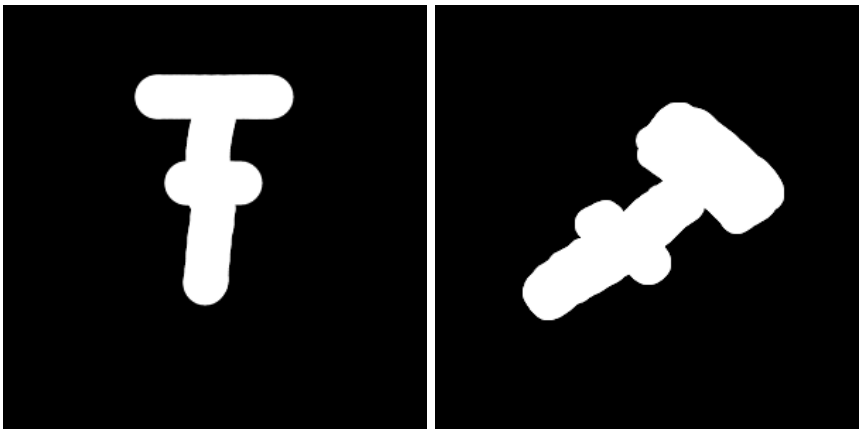
(f) scenario A7, example 6

Figure C.3: Sketches used as reference documents in scenario A7. Small deviations in colour could make the difference between success and failure. Obviously, the level of detail in terms of shape is not important.



(a) scenario A10, example 1 (b) scenario A10, example 2 (✓)

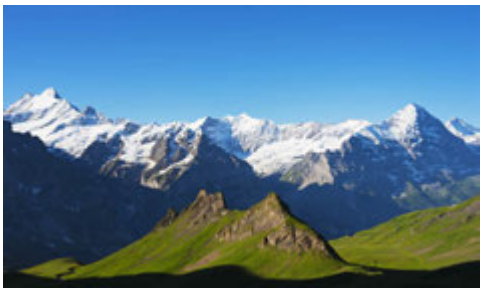
Figure C.4: Sketches used as reference documents in scenario A10. Surprisingly, only the second example produces highly relevant results right away. However, both images ultimately succeeded in producing meaningful results.



(a) scenario A12, example 1 (b) scenario A12, example 2

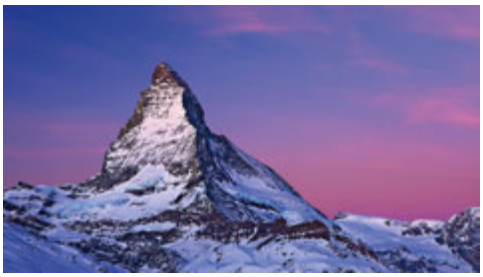
Figure C.5: Sketches used as reference documents in scenario A12. Neither produced highly relevant results in the top 15 right away. However, users managed to find relevant documents through More-Like-This.

(a) scenario B1, example 1 (✓)



(b) scenario B1, example 2 (✓)

(c) scenario B1, example 3 (✓)

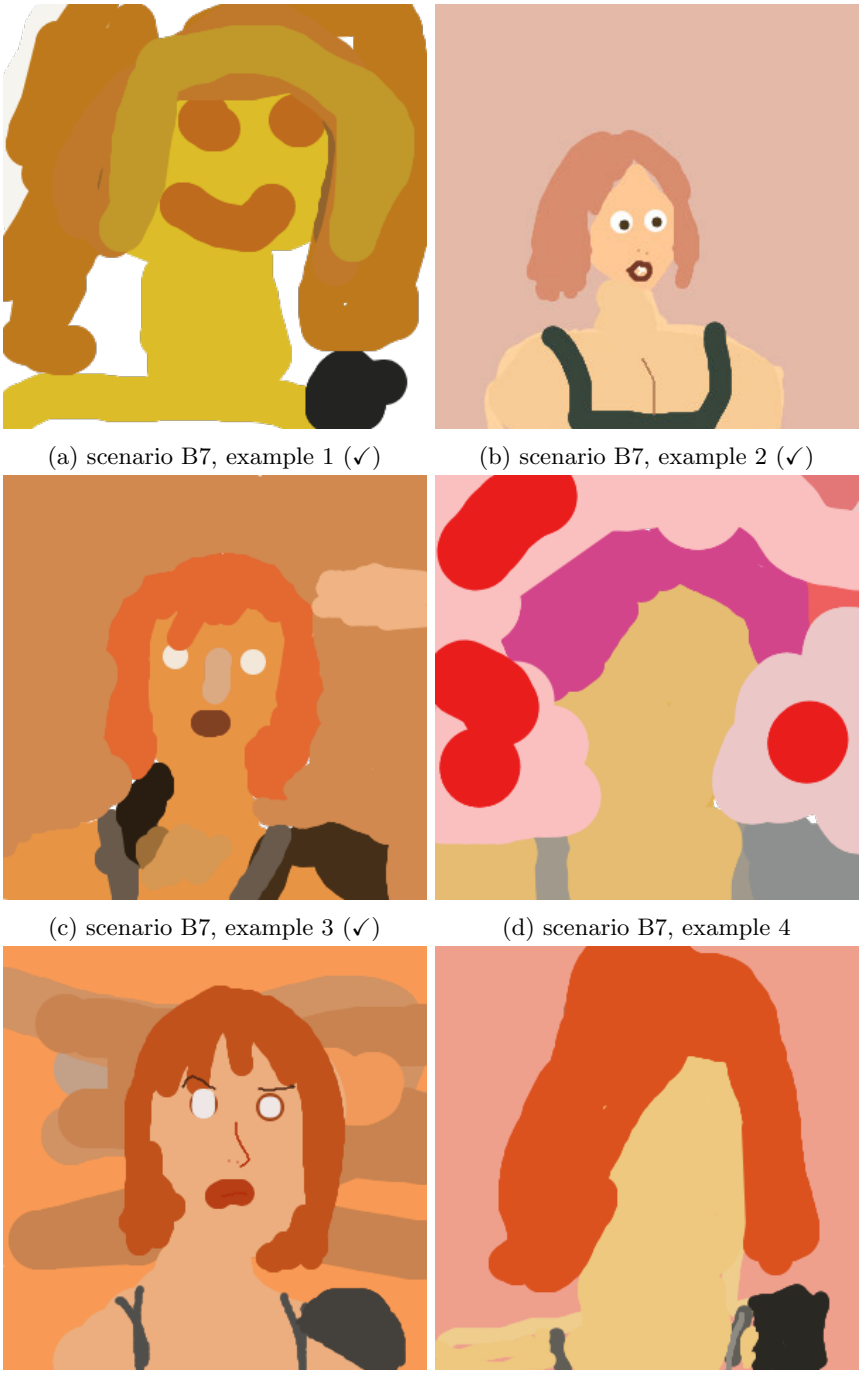


(d) scenario B1, example 4

Figure C.6: Example images used as reference documents in scenario B1. The four images illustrate the great variety in the general colour setting when using images "depicting mountains". The last example did not produce any highly relevant results according to the user rating.

(a) scenario B3, example 1 (✓)

Figure C.7: An example sketch used in scenario B3. As it turned out, most users were able to retrieve the icon based on sketches like this.



(a) scenario B7, example 1 (✓)

(b) scenario B7, example 2 (✓)

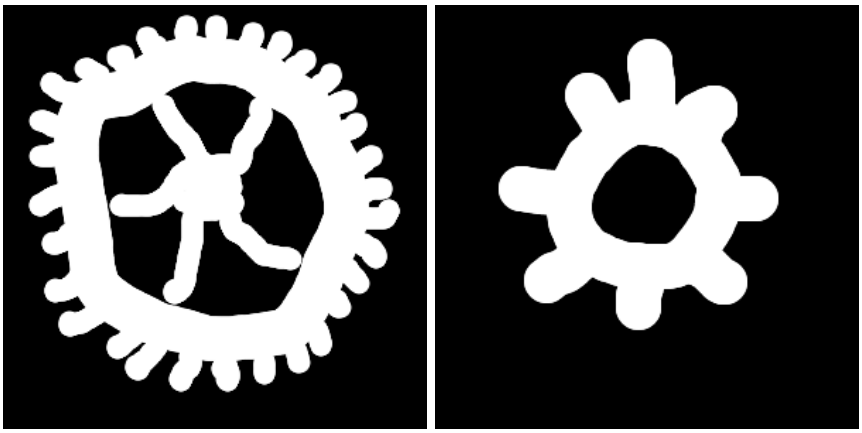
(c) scenario B7, example 3 (✓)

(d) scenario B7, example 4

(e) scenario B7, example 5

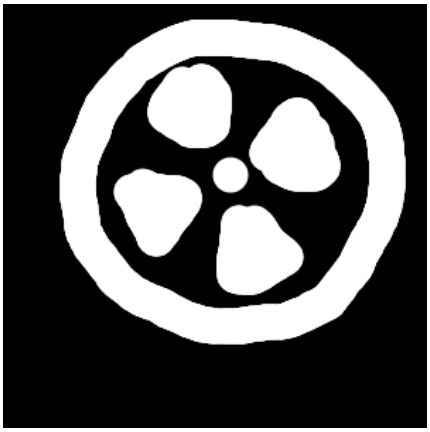
(f) scenario B7, example 6

Figure C.8: Sketches used as reference documents in scenario B7. The colours from the original image were difficult to reproduce. Poor drawing, like the first example, seem to outperform detailed drawings that miss the actual colours.



(a) scenario B12, example 1 (✓)

(b) scenario B12, example 2 (✓)



(c) scenario B12, example 3

(d) scenario B12, example 4

Figure C.9: Sketches used as reference documents in scenario B12.

D

Vitrivr NG Developers Manual

The new user interface — Vitrivr NG — is based on *Angular 4.0*[33] and the *Angular Material*[34] UI framework. A complete list of all dependencies can be found in section E.1 on page 115. All the source code is in *Typescript 2.1*[35] which is why currently, a compiler is required to create JavaScript files. This chapter briefly describes the most important aspects to get started with Vitrivr NG development quickly.

D.1 Development environment

For development, you require *NodeJS* and the *npm* package manager[36]. The Vitrivr NG project was created with *Angular CLI*[37] so you have to install that as well. Once *npm* is available, you can install Angular CLI globally using the following command in your console:

```
npm install -g @angular/cli
```

With *npm* and Angular CLI you can do a lot of things that are beyond the scope of this chapter. We refer to the respective documentation. In the context of Vitrivr NG, you will chiefly use these tools to:

1. Install, update and remove packages and dependencies (*npm*)
2. Start the typescript compiler (Angular CLI)
3. Create a production build (Angular CLI)

For this purpose, you should always execute *npm* and Angular CLI from within the Vitrivr NG project folder (top-level directory). To start Typescript compilation, you can then execute the following command in your console:

```
ng serve
```

[33] https://angular.io
[34] https://material.angular.io
[35] https://www.typescriptlang.org
[36] https://nodejs.org/en/download
[37] https://cli.angular.io

This will start the typescript compiler and make Vitrivr NG available under http://localhost:4200. Note that while the compiler is running, it will automatically detect changes to files in the project folder. Those will cause re-compilation and a subsequent reload of the application. In order to create a standalone version of Vitrivr NG, type:

```
ng build --prod
```

This will compile and bundle the source files and dependencies into a folder *dist* located in the top-level directory of the project folder. That folder can then be copied to a webserver and used as a standalone version of the application. Neither NodeJS nor npm are required to run this standalone version.

D.2 Project structure

Figure D.1 lists the most important files and folders (bold) and gives an overview of the general structure of the Vitrivr NG project. We will use this chapter to briefly describe the relevant files, folders and their function.

src/app Contains all the source code related to the Angular application, that is, modules, components, services and helper classes. This is where you can add new classes.

src/app/app.module.ts The module definition for the Vitrivr NG (top-level) application module.

src/app/app.component.ts The component definition for the Vitrivr NG (top-level) application component.

src/app/app.component.html The HTML view file for the Vitrivr NG (top-level) application component.

src/app/material.module.ts Imports and exports all modules that are provided by the Angular Material framework and used by Vitrivr NG. This module can be imported by any other module that requires components from the Angular Material library[38]. Here, you can also add more imports from the framework if needed.

src/app/app-routing.module.ts Defines the routes for the application. Note that for routing to work in the standalone version, *mod_rewrite* must be activated.

src/app/core Contains service classes like the query service or the evaluation service.

src/app/evaluation Contains all the Angular modules and components related to the evaluation module.

src/app/gallery Contains all the Angular modules and components related to the gallery view.

[38] Note that since version 2.0.0-beta4, the *MaterialModule* included in the Angular Material framework is deprecated. Instead, developers are required to create their own modules to have more finely grained control over which components are being imported.

src/app/objectdetails Contains all the Angular modules and components related to the object details view.

src/app/refinement Contains all the Angular modules and components related to the query refinement view (right sidebar).

src/app/research Contains all the Angular modules and components related to the research view (left sidebar).

src/app/shared Contains classes that are shared between different components and services in the application, like for instance model files, interfaces and utility classes.

src/assets Contains necessary assets like fonts, images etc.

src/config.json JSON file that holds the Vitrivr NG application configuration. See section D.3 for more details.

src/index.html The index file of the Vitrivr NG application. It contains the top-level component of the Angular application. Usually, you should not be required to change this file after its initial creation.

src/styles.scss Contains the global Vitrivr NG style definitions. The file also includes and bundles auxiliary SCSS files, like the *vitrivr-theme.scss*.

src/vitrivr-theme.scss Defines the Angular Material theme. You can overwrite the variables in this file to change Vitrivr NG's colour scheme and general look and feel.

node_modules Contains all the dependencies of Vitrivr NG. This folder is managed by *npm* and therefore, you should not make manual changes. However, sometimes it can help to delete and let *npm* rebuild it completely.

package.json Defines all the dependencies for Vitrivr NG. You can add and remove entries manually. However, when you use *npm* to manage your dependencies and use the *–save* flag, it will store all changes in this file automatically.

tsconfig.json Contains settings for the typescript compiler. You should not need to change this file except when adding JavaScript dependencies with separate, explicit typings. In this case you are required to declare the respective module in this file (Examples: ThreeJS and JSZip).

D.3 Application configuration

The application configuration is contained in the *src/config.json* file. The file is read by an *Angular* service called *ConfigService* (see *src/app/core/basics/config.service.ts*) and made available to the other components and services through that service. This section describes the keys in config.json and the effects of changing an entry. **Note:** We use a dot-syntax to navigate in the JSON object structure of the configuration file.

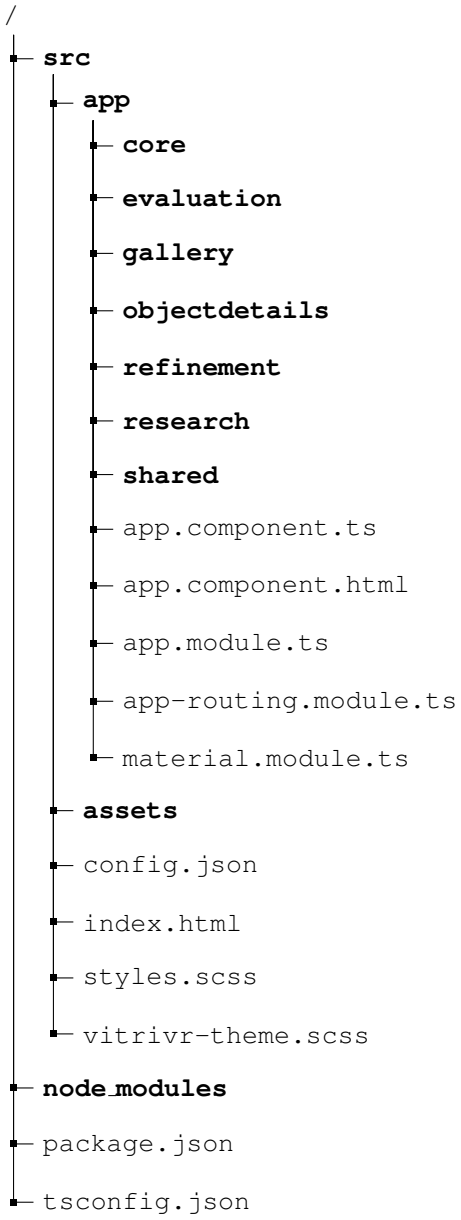


Figure D.1: Overview of the Vitrivr NG project structure. It shows the most important top-level files and folders that are being described in this chapter.

api.host IP address or hostname under which the Cineast WebSocket API is available.

Type: string **Default:** 127.0.0.1

api.port Port at which the Cineast WebSocket API is listening.

Type: number **Default:** 4567

api.protocol_ws Protocol that should be used for communication with WebSocket API.

Type: string **Default:** ws **Values:** ws, wss

api.protocol_http Protocol that should be used for simple HTTP communication with the API (RESTful). Property is currently not in use.

Type: string **Default:** http **Values:** http, https

api.ping_interval Interval in milliseconds at which the PingService should try to contact the WebSocket API.

Type: number **Default:** 10000

resources.host_thumbnails URL relative to which thumbnail paths will be resolved.

Type: string **Default:** http://localhost/vitrivr

resources.host_object URL relative to which media object paths will be resolved.

Type: string **Default:** http://localhost/vitrivr

resources.suffix_default Default suffix and therefore format for thumbnail images. The dot can be ommited.

Type: string **Default:** jpg

resources.suffix An object that can be used to override the default suffix for preview images per media type. The object contains the media types (*IMAGE*, *VIDEO*, *AUDIO* and *MODEL3D*) as key and the desired suffix as value. The dot can be ommited.

Type: JSON object **Default:** null

```
{ IMAGE : "jpg", AUDIO : "jpg", VIDEO : "png", MODEL3D : "png" }
```

evaluation.active Indicates whether the evaluation module should be active or not.

Type: boolean **Default:** false

evaluation.templates A JSON array listing names and URLs to evaluation templates in a dedicated JSON object per entry. Those templates will be made available in the evaluation module.

Type: JSON Array **Default:** null

```
[{ name : "Name", url : "http://www.url.com/evaluation.json" }]
```

D.4 Structure of an Angular application

The purpose of this section is not to give an in-depth explanation about Angular and its inner workings. Again, we refer to the official documentation[39] instead. However, we use this section to briefly explain the most important concepts and mention how they are being applied in Vitrivr NG. A lot of the information in this section can also be found in the official glossary[40].

[39] https://angular.io/docs/ts/latest/guide
[40] https://angular.io/docs/ts/latest/guide/glossary.html

Module A class that forms a unit of organisation within an Angular application and usually bundles building blocks — like components — that belong together functionally. A module identifies the components, directives, services and pipes that are used (imported) by the module and thus available to all the components contained within[41]. In addition, the module also declares the components, directives, services and pipes that are made available by the module for other modules to import. Module files can be identified by their filename, as they usually end with ***.module.ts**. They always contain the **@NgModule** annotation. Every Angular application contains at least one module — the application module.

Examples (Vitrivr NG): GalleryModule, EvaluationModule, ResearchModule

Component A class that is responsible for exposing data to a view. The view data is updated by the framework through data bindings. Furthermore, components handle user interaction with those views. Component files can be identified by their filename, as they usually end with ***.component.ts**. They always contain the **@Component** annotation. Sometimes, they are accompanied by a dedicated view file and/or a CSS file (***.component.html** and ***.component.css**). However, it is also possible to define the actual view in the same file as the component itself.

Examples (Vitrivr NG): GalleryComponent, RefinementComponent, ResearchComponent

Service A class that contains data or logic (or both) that is not bound to a specific view or that should be available to multiple components. Service files can be identified by their filename, as they usually end with ***.service.ts**. Usually, services are made available to components by means of *dependency injection (DI)* and therefore marked with the **@Injectable** annotation. The DI is transparently handled by the *Angular* framework.

Example (Vitrivr NG): QueryService, ConfigService

In Vitrivr NG, we tried to stick to the offical Angular guidelines for project structure and naming of the files. The major building blocks of the user interface, namely the *gallery*, the *research area* (left sidebar), *the query refinement* area (right sidebar) and the *object details* view are defined in their own, respective modules. As one can see in figure D.1, this is reflected in the folder structure.

D.4.1 Services

All services that are part of Vitrivr NG are defined in the *CoreModule*, which can be found in the *src/app/core* folder and bundles different sub modules and service classes. All services are decorated with the **@Injectable** annotation, which makes them eligible for DI. The entire CoreModule is imported by the *AppModule*, which has two effects:

[41] Note that usage of a component or service by another component must be declared in the module the latter component belongs to. Failing to do so will result in a runtime error!

1. The services in the CoreModule, including all the services in submodules imported by the CoreModule, are available to all components in the entire application.

2. The Angular framework assures that the a single instance of the respective service is injected into the components. Hence, the services act as singletons. If one wants to build a service that has separate instances for different components, the import must take place in the respective component's module instead of the AppModule.

Currently, the following services are part of Vitrivr NG. Their functionality can be extended and new services can be added:

CineastAPI The service that handles raw WebSocket communication with the Cineast endpoint. The service simply forwards messages from Vitrivr NG to the Cineast API and in turn broadcasts messages sent by Cineast to all classes that are interested in those messages. An RxJS subscription pattern is used to allow other classes to subscribe to those messages. Subscribers decide autonomously what types of messages they are interested in.

QueryService Uses the Cineast API to execute similarity queries and receive query results. Furthermore, the QueryService invokes helper classes to perform fusion of partial results and it communicates query results to components using an RxJS subscription pattern. Hence, the QueryService acts both as subscriber to the CineastAPI and as publisher to other components and services.

MetadataLookupService Uses the Cineast API in order to perform metadata lookups for particular media objects. It communicates results to components using an RxJS subscription pattern. Hence, the MetadataLookupService acts both as subscriber to the CineastAPI and as publisher to other components and services.

ConfigService Loads the application configuration and provides access to settings.

ResolverService Generates paths to media objects and associated thumbnails by applying a defined set of rules based on the media type of the object, its filename and its ID.

EvaluationService Provides access to certain functionalities surrounding the evaluation module, like e.g. persistent storage through the Index DB API.

D.4.2 Shared classes

Shared classes can be found in the *src/app/shared* folder in the Vitrivr NG projects. These classes mostly involve plain data model classes and associated interfaces. There are also some utility classes that were not realised as services[42].

[42] It is worth noting here, that any ordinary Typescript class can be used from within Angular components without import through modules. Module import is only necessary for classes residing in the Angular domain.

Most importantly, however, it contains a handful of Angular components that do not strictly belong to one of the functional areas described in the previous section and might even be useful outside of the Vitrivr NG context. These components include *SketchCanvasComponent*, *AudioRecorderComponent* and *M3DLoaderComponent*.

D.5 Communication protocol

The QueryService in Vitrivr NG (see section D.4.1) communicates with Cineast via a WebSocket protocol. See figure D.2 for a sequence diagram of a typical message exchange. By default, the communication takes place on port 4567. This is, however, configurable (see section D.3). The protocol is based on JSON; that is, message objects are serialised to JSON, transferred over the wire, and deserialised and processed by the receiving endpoint. Every message has an object and/or interface equivalent in Cineast and Vitrivr NG. Note, that the routing of a message to the handling class in Cineast and Vitrivr NG is based on the type of message. The types of messages that Cineast currently supports are listed in table D.1.

Table D.1: Message types that are currently supported by the WebSocket protocol for communication between Vitrivr NG and Cineast.

Message	**Sender**	**Receiver**	**Description**
Q_SIM	Vitrivr NG	Cineast	A request for execution of a new similarity query.
Q_MLT	Vitrivr NG	Cineast	A request for execution of a new More-Like-This query.
M_LOOKUP	Vitrivr NG	Cineast	A request for looking up metadata for one or many media objects.
QR_START	Cineast	Vitrivr NG	Notifies Vitrivr NG that execution of a query has started.
QR_ERROR	Cineast	Vitrivr NG	Notifies Vitrivr NG that an error occured during query execution and that it has been aborted.
QR_OBJECT	Cineast	Vitrivr NG	Notifies Vitrivr NG about media object information that has become available.
QR_SEGMENT	Cineast	Vitrivr NG	Notifies Vitrivr NG about segment information that has become available.
QR_SIMILARITY	Cineast	Vitrivr NG	Notifies Vitrivr NG about similarity scores that have become available.
QR_METADATA	Cineast	Vitrivr NG	Notifies Vitrivr NG about media object metadata that has become available.
QR_END	Cineast	Vitrivr NG	Notifies Vitrivr NG, that the execution of a query has ended regularly. When this message is received, all results should be available

In Vitrivr NG, the interface definitions for messages can be found under *src/app/shared/messages*. In Cineast, the message classes are located under the *org.vitrivr.cineast.core.data.messages* package. The different types of messages are defined in *mesage-type.model.ts* and in the *MessageTypes* enumeration respectively.

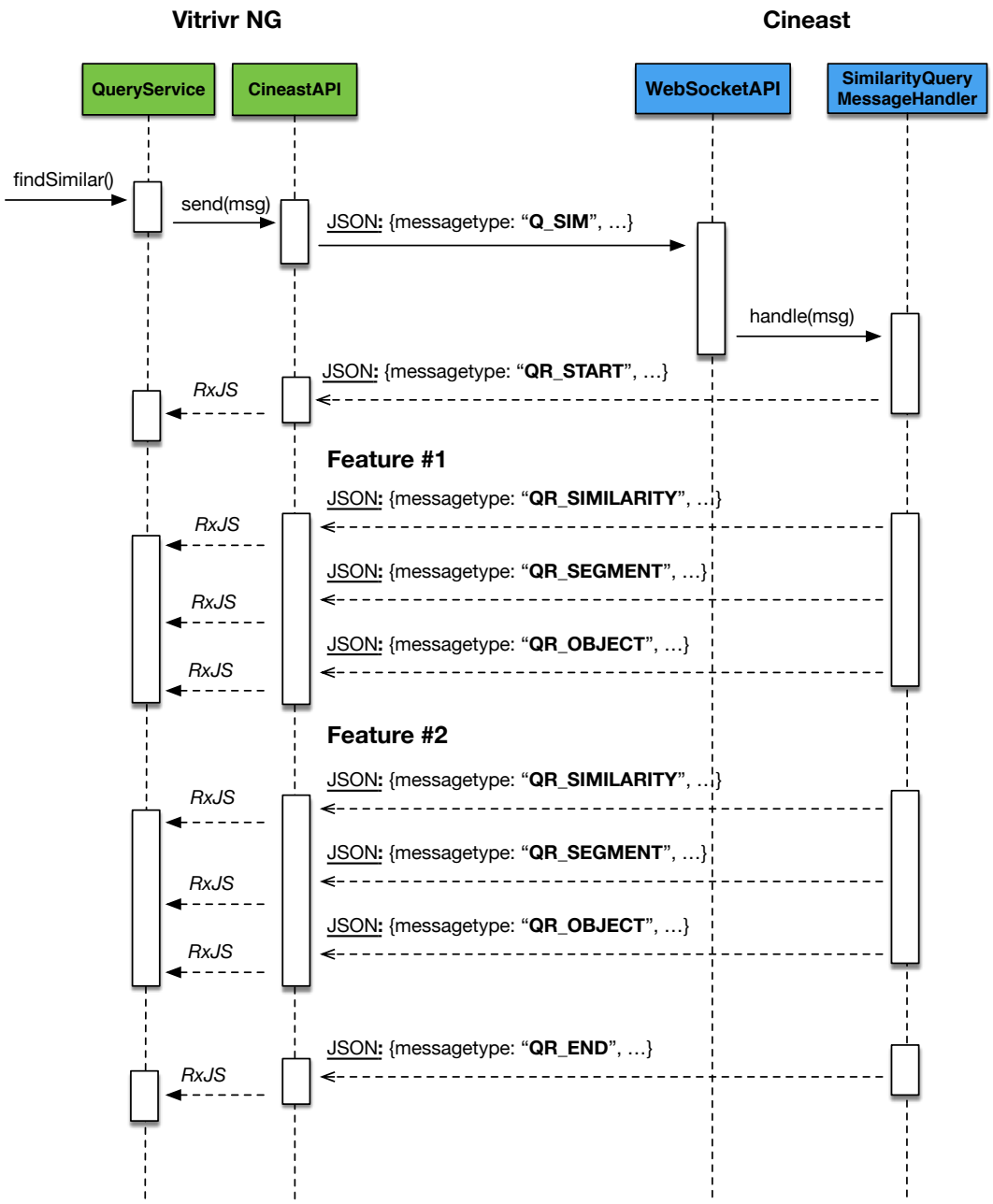


Figure D.2: A sequence diagram showing the typical exchange of messages for a simple similarity query. The similarity search is triggered by a call to the *QueryService::findSimilar()* method. The QueryService then packages the query container and serialises it to JSON, which in turn is sent through the *CineastAPI::send()* method. The resulting Q_SIM message is received by Cineast, deserialised and relayed to the *SimilarityQueryMessageHandler* where the search is executed. As results become available, Cineast notifies Vitrivr NG with QR_SIMILARITY, QR_OBJECT and QR_SEGMENT messages. Those are handled by the CineastAPI and forwarded to the QueryService through RxJS. End of query execution is marked by a QR_END message.

E

Dependencies and Libraries

E.1 Cineast

The following list briefly describes the most important frameworks and libraries that were used to build Cineast.

Jackson (2.8.8) A JSON processor and object mapper.

See: https://github.com/FasterXML/jackson

Spark Java (2.5.5) A lightweight web framework that allows for quick realisation of RESTful and WebSocket interfaces.

See: http://sparkjava.com

Apache Commons CLI (1.3.1) API for parsing command line options passed to programs.

See: https://commons.apache.org/proper/commons-cli

Apache Commons Codec (1.10) API for parsing command line options passed to programs.

See: https://commons.apache.org/proper/commons-codec

Apache Commons Lang (3.5) Provides implementations of common encoders and decoders such as Base64, Hex, Phonetic and URLs.

See: https://commons.apache.org/proper/commons-lang

Apache Commons Math (3.6) Contains lightweight, self-contained mathematics and statistics components addressing the most common problems not available in the Java programming language or Commons Lang

See: http://commons.apache.org/proper/commons-math

Trove4J (3.0.3) High speed regular and primitive collections for Java.

See: http://trove.starlight-systems.com

TwelveMonkeys Image IO (3.3.2) A collection of plugins and extensions for Java's ImageIO that allows to read more advanced file formats.

See: https://github.com/haraldk/TwelveMonkeys

BoofCV (0.2.6) An open source Java library for real-time computer vision and robotics applications.

See: https://boofcv.org

Metadata Extractor (2.10.1) A Java library for reading metadata from image files

See: https://github.com/drewnoakes/metadata-extractor

JOML (1.9.2) A math library for calculations related to Open GL rendering.

See: https://github.com/JOML-CI/JOML

JOGL (2.3.2) Java binding for OpenGL.

See: https://jogamp.org

JavaCPP Presets FFMPEG (3.2.1-1.3) Java binding for FFMPEG.

See: https://github.com/bytedeco/javacpp-presets/tree/master/ffmpeg

E.2 Vitrivr NG

The following list briefly describes the most important frameworks and libraries that were used to build Vitrvir NG.

Angular (4.2.4) A JavaScript framework for single page applications.

See: https://angular.io

Angular Material (2.0.0-beta6) A material design UI framework for Angular. It provides a lot of default UI components.

See: https://material.angular.io

Angular Flex Layout (2.0.0-beta8) An Angular module that provides an API for CSS Flexbox and MediaQueries.

See: https://github.com/angular/flex-layout

Three.js (r86) A JavaScript WebGL (3D) library that includes rendering support and importers for various 3D model formats.

See: https://threejs.org

Dexie (2.0.0-beta.10) A minimalistic wrapper and API for IndexedDB.

See: http://dexie.org/

NGX Color Picker (4.0.1) A colour picker component for Angular 2.

See: https://github.com/zefoy/ngx-color-picker

JSZip (3.1.3) A JavaScript library for creating, reading and editing ZIP files.

See: https://stuk.github.io/jszip/

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Ralph Gasser

Title of work — Titel der Arbeit
Towards an All-Purpose, Content-Based Multimedia Information Retrieval System

Type of work — Typ der Arbeit
Master's Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, July 7th 2017

Signature — Unterschrift