Multimedia Information Retrieval †

Dulce Ponceleon (IBM Almaden Research Center)
Malcolm Slaney (Yahoo! Research)

September 6, 2007

† A Chapter for the Book Modern Information Retrieval. Feature complete draft.

0.1 Introduction

0.1.1 What is Multimedia?

Nowadays, we live in a complex and dynamic world, in the shadow of an ever-growing mountain of digital data. Data makes its way to our home from many different paths including cable, satellite, consumer electronic devices, music downloads, video download, personal video cameras, etc. The trend has been strengthened by the proliferation of cellular phones in the last decade, which now come with a camera. This has allowed the widespread taking of digital photos and short movies, as well as their quick upload into Web servers for later retrieval and displaying. As a result, digital media now reaches us not only at home but also while we are on the go. It moves from our home devices to our portable devices as a result of the convergence of the entertainment, communications and computer industries in the e-publishing age (see Chapter **??**).

One of the consequences of this convergence is the rapid growth in recent years of the digital video market. Several factors have contributed to this, such as the falling prices of video-capture devices, the proliferation of digital video and Web cameras, the wide adaptation of video standards, and the quick expansion of broadband Internet access. All of these have made capturing, storing and distributing video easier and more affordable than ever before, which has led to an exponential growth of media and to the rapid development of technology. As a consequence, there is now an urgent need to improve the means of retrieving and manipulating digital data due to the demand of a variety of new applications such as video-on-demand, voice-over-IP, medical imaging, multimedia digital libraries, surveillance, security, distance learning, and event detection. The fundamental implication is that we need to develop better management methods, procedures and tools for *multimedia!* .

But, what is multimedia? It is essentially any digital data, including plain text, mostly unstructured, that we use to communicate or capture information. Beyond text, multimedia is visual and sound data, covering pictures, graphs, images, videos, animations, speech, music, sounds, and even 3D visualizations.

Multimedia data often do not have a well defined structure. While text has a layout with paragraphs and logical structure, a multimedia signal might be very different. As a consequence, for proper storage and retrieval, it is necessary to discover and define the structure of the multimedia data. One way to capture this information is through metadata, which allows us to classify, index and search multimedia data. In the absence of metadata, all we have from which to extract information is the data content itself. In this case one alternative approach is to use content to derive (automatically, semi-automatically or manually) metadata. Because of this the term Content Based Retrieval (CBR) is often used synonymously with Multimedia Retrieval.

Multimedia includes complex spatial, temporal, and semantic structures comprising all phases of the life-cycle of digital media such as capturing, editing, compressing, processing, delivery, consumption, and analysis. In each of these phases, one of the key challenges is to find the underlying structure within the multimedia data.

0.1.2 Multimedia IR

What is Multimedia Information Retrieval (MIR)? What does come to mind when people think about multimedia retrieval? In its most general conception, the multimedia retrieval problem can be stated as follows:

> The task of a Multimedia IR system is to retrieve text, image, video and sound data related to the interest of the user and rank them according to a degree of similarity with regard to the user query. The similarity degree (i.e., the ranking) should be computed in order to improve the likelihood that the user will find the data in the answer set relevant.

A characteristic example would be to allow a user to search for a scene with a particular actor or with a specific passage in a favorite movie. For instance, "Keanu Reeves avoiding bullets in a Helicopter Crash" in the movie "The Matrix." Note that this is not possible using the search engines of today. To process this request we could add manual annotations to the thousands of scenes that comprise each movie, but this is too expensive. There is more value in indexing lightly-produced footage, such as educational videos, surveillance videos, and all the variety of media on the Web, and using these footage annotations to answer the queries posed by users. This is the state-of-the-art in multimedia IR, i.e. retrieving photos, pictures, scenes based on annotated footage.

Since automatic annotation of multimedia has its limits, instead, we often use semi-automatic approaches to leverage automatic techniques as well to keep the user involved. Indeed, combining manual annotation with automatic annotation is the trend in the e-publishing age. State-of-the-art systems like Marvel [**Smith 200X**] show that investing approximately 10% of the total effort in manual annotation and using machine learning approaches to automatically annotate the remaining 90% of the data provides promising results. In short, despite the cost and limitations of manual annotation it is still an important part of indexing videos and images.

Do we want to keep the New Paragraph A current and very important trend in the Web is facilitating and stimulating user-generated content. During the 1990's people could create their own Web pages and link them to other pages of interest. But generating new information and publishing it was restricted to the one own's sytle.

I am not sure the work is sytle.

Around the year 2000, Web sites started allowing external users to more easily contribute their own content. Now anybody can can easily place their photos on the Web (i.e. Flickr), publish a video (i.e. YouTube), or even describe themselves in a social network (i.e. Match.com, Friendster, My Space, Orbitz and FaceBook). All these sites allow users to label their content, but in highly personal ways that makes IR more difficult. Thus a user might take a trip to San Francisco, upload many pictures and label all of them with the tag "SanFrancisco." This is an appropriate tag for this user, it describes their own

San Francisco trip this year, even though most of the images do not represent typical scenes of San Francisco. The good news is that all this multimedia data provides interesting content that is of wide appeal. The bad news is that it is highly personal and not labeled the way a professional might do it. In Section 0.2.4 we discuss other ways to get users to label data.

Multimedia information retrieval (MMIR) deals with searching, indexing, extracting, browsing and summarizing information and semantics contained in multimedia. That is, MMIR is the science of extracting meaning and discovering patterns in a sea of heterogenous and unstructured multimedia. It encompasses different sub-areas, such as:

- content representation and multimedia object representation,
- feature extraction,
- query formulation to map high-level semantic concepts into low-level features,
- query-by-example,
- relevance feedback, interactive queries,
- efficient feature indexing and cataloguing, i.e. extraction of low-level features (color, shape, texture, etc.) in images and videos,
- integrated searching and browsing,
- techniques for searching multimedia based on their contents.

A recent survey [**?**] states that "MMIR is about the search for knowledge in all its forms, everywhere." The grand multimedia challenge is defined as to "make capturing, storying, finding, and using digital media an everyday occurrence in our computing environment" (ACM SIGMM 2005). A solution to this challenge requires facing a key problem—the semantic gap"—which is represented by the distance between high-level concepts that users associate with multimedia objects, such as the description of a scene, for instance, and the limitations of low-level features (color, texture, shape) to capture and express such abstractions. Bridging such a gap is still an open problem and an active research area, as we later discuss.

Because of the semantic gap, the problems faced in multimedia IR tend to be hard. A few examples are as follows. How does a user select her favourite programs or discover new programs of interest out of the many channels being offered? How do we browse an audio collection effectively? If a multimedia query should be more sophisticated than plain keywords, what should it be? Can we get the user to provide annotations as a by-product of other applications? What does the result of a multimedia query should look like? How do we compactly represent a video or audio collection?

It is pretentious to say that one can summarize all of multimedia information retrieval in a book chapter. Instead we have selected several representative areas of multimedia IR to cover. We start by contrasting the differences between

multimedia and text and by discussing the challenges of dealing with multimedia. Why text-retrieval techniques (generally speaking) fall short when extended to multimedia? What are the consequences of having continuous, unstructured media? What is the impact of having media with an inherent time element?

0.1.3 Text IR versus Multimedia IR

There are several aspects that make text retrieval different from image, video or audio retrieval. In text, words are readily available as basic semantic units and structure is provided by punctuation, and paragraphs. Even font characteristics reveal emphasis. In contrast, multimedia data is typically an uninterrupted stream, a linear story, and there are few delimiters. For non-text media, defining the semantic unit is a fundamental step to attain high-quality search. For example in video, time is important—content changes with time—so the continuous stream needs to be broken into manageable chunks using video segmentation, as we will discuss in Section 0.4. There is simple no natural extension to text retrieval that provides for multimedia retrieval. As a result, special techniques are require, some of them very specific to a particular modality: audio or video.

Regarding speech transcripts, advances in speech recognition now allow the generation of good quality transcripts with low error rates. However, even a nearly perfect transcript lacks punctuation, paragraphs, and all the textual elements that provide structure. In short, although retrieval based on a speech transcript seems very close to text retrieval, in practice it is not. Dealing with this problem requires recognizing that the time associated with every word in the speech transcript is valuable information for retrieval. For example, a large gap between words might be indicative of a change of topic.

The sheer difference in the size of text documents and multimedia objects also matters. In a multimedia environment, we deal with a number of different types of objects, each with its own characteristics and native data types, which cannot be represented as standard tables in a database since they are not structured data. These data types also require high bandwidth for retrieval and delivery. While a one-hundred page document requires typically 200k bytes of storage, a 75 minute-long audio signal requires 60M bytes when compressed as an MP3 file. Further, a one-hour-long video compressed with MPEG-1 requires 600M bytes of storage (and a bandwidth of 1.5Mbps, mega bits per second, for delivery without hiccups).

While a considerable number of people perform text searches daily, using their favorite search engine, image and video retrieval are not yet as accessible and ubiquitous as text, although growing rapidly. By comparison with text retrieval, multimedia retrieval is a relatively new discipline.

Browsing is also different in multimedia. Despite the impressive graphics displays available nowadays, traditionally we have a strong technological culture that focuses on text, which leads to a well defined and relatively well understood culture around words. To illustrate, the concepts of summarizing and highlighting are much better understood for text. For multimedia there is no

canonical or universally agreed approach for defining the material that should be in a summary, neither there is consensus on the best way to display it to the user.

The emergence and growth of image and video search engines, as well as the proliferation of applications and sites for people to manage and/or share their photos, and music, are indications that multimedia is here to stay.

Add figure from the first cover page, i.e. the High-level System Architecture

0.2 The challenges

0.2.1 Machine-generated Data

Multimedia is both challenging and interesting because of the growth of the data. Figure 0.1 shows an estimate of how multimedia production is growing and how much data will be out there. The trends shown here are all approximate. The text estimate was based on each person typing 8 hours a day, 50 words per minute, every day of the year—and this figure is flat, unchanging, and certainly an upper bound. The surveillance data was based on a 2002 estimate of the number of fixed cameras in central London and extrapolating that to all the cities in the United States. The medical data was calculated based on the average number of bytes collected per patient in an all-digital hospital and then assuming that all hospital records will be digital in the near future. Clearly people are producing and consuming a large quantity of multimedia data, which will require highly complex server farms for proper storage and retrieval.

Space between paragraph and figure is too large. The figure itself needs to be made larger to appreciate the content.

0.2.2 The semantic gap

replace figure with updated figure as indicated in caption notes

There is often a large gap between the contents of a multimedia signal and its meaning. This is known as the *semantic gap* and is one of the hardest aspects of multimedia retrieval.

Figure 0.2 illustrates symbolically the problem. While there is no semantic gap between a table of salary numbers and their meaning, there is a gap between the words in a document and its overall information and meaning. Further, there is a much larger gap between a video of a husband shrugging his shoulders and what this gesture implies to his wife. This gap is hard to address in multimedia both because the objects are hard to find and recognize, and because multimedia tends to cover topics with more emotional content. We will address both of these reasons in turn, and describe their effect on multimedia IR.

Object recognition is one of the hardest problems in image and audio processing. All of us can look at an image, even for small fractions of a second,

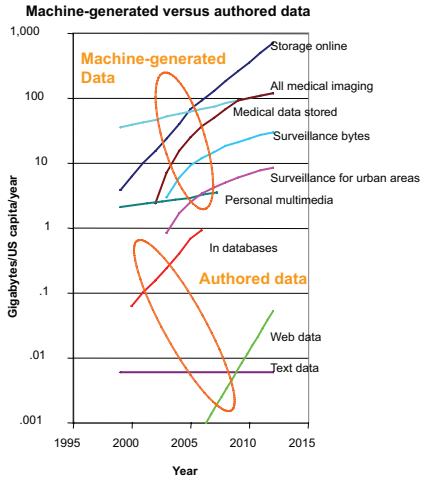

Figure 0.1 The growth of machine-generated data. Note that the vertical axis is measured in gigabytes/person/year. **Fit this into the text. Perhaps side-by-side with the next figure.**

and identify faces and types of objects. While optical character recognition and (perhaps) face-recognition systems work well enough for commercial applications, the same can not be said for more general objects. Labeling components of an image or analyzing the sounds in a waveform are still unsolved problems, and the subject of much research ([**?**], [**?**]). If a computer cannot identify a tiger in the grass, how can we search for a tiger? Multimedia IR systems of today make heavy use of human-generated words (or tags) [**?**], [**?**] and almost ignore the content features to generate an answer for the user.

Even if the objects in a picture are known, an image or an audio signal carries complex traits that allow subjective and emotional interpretations that are difficult for computers to reproduce. In speech, this non-semantic information is conveyed by the prosody of the signal. It's what makes the difference between, "don't stop" and "Don't! Stop!". But even in the best of conditions [**?**] this emotional information is hard to recognize, let alone convey to the user of an IR system. While some speech response systems do look for prosodic information

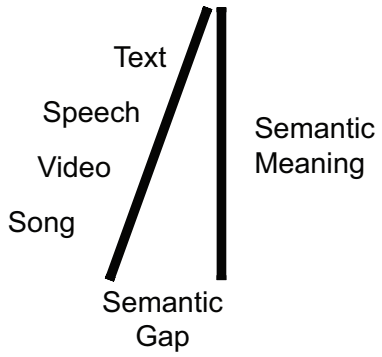

Figure 0.2 The semantic gap gets larger as we get further from words. **Add labels along top: signal and knowledge. The order should be: text, image, speech, video, music. Dulce thinks we should add audio as follows: text, image, speech, audio, video, music. Add arrows above the label "semantic gap" to point between the lines. Can it be done side-by-side with above? Or convert to sideways.**

to judge such emotional state of a customer [**Reference?**], most IR systems do not have any way to represent the emotional content.

Check bibtex so that it has the label for babyears match

0.2.3 Feature ambiguity

Another challenge with multimedia IR is caused by the ambiguity of the features. One example of this is known as the aperture problem and is illustrated in Figure 0.3. In this figure a bar is moving from the lower-left to the upper right, as it can be seen from the two snapshots at the top taken at time instants $T = 0$ and $T = 1$, but a simple motion detector only measures a portion of the image, known as the aperture. The aperture limits the decision (and computational effort) to a small portion of the image, as it can be seen from the two snapshots at the bottom of Figure 0.3. Unfortunately, when viewed through this aperture, the motion is perceived as completely horizontal. There is no texture in the bar that leads one to think there is diagonal motion, so the simplest explanation is to postulate horizontal motion only.

Check the correctness of this example, i.e. that the movement is diagonal but perceived as completely horizontal or the other way around. We had the example on the tutorial slides

Use Smaller Fonts. Reduce Size and CENTER the figure.

The aperture problem is caused by a lack of information within a window (the aperture). When the entire image is considered there is strong evidence, especially at the edges of the bar, that something more complicated is going on. But this top-down integration is still difficult for machine-algorithms to

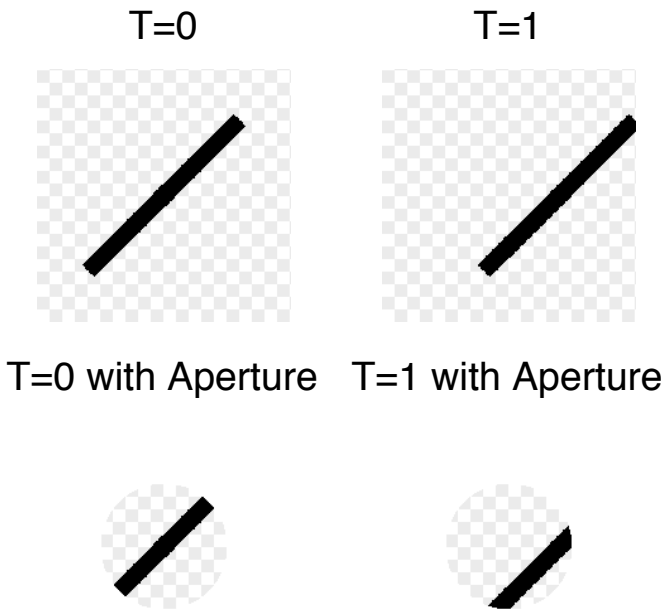

Figure 0.3 An illustration of the motion-aperture problem. In the top figure the bar is moving to the right. But when viewed through a small aperture, as is often done for efficiency reasons, the bar appears to be moving down and to the right. **Berthier: Reduce image size, make it compatible with the text-font size. Use 1x4 grid. Make checkerboard darker.**

perform. Speech recognition systems, to be discussed in Section 0.7.6, have a solution based on a language model. But this type of solution is not yet possible for the general image problem.

0.2.4 The Evolution of Multimedia IR

Multimedia IR has gone through at least three distinctive phases over the last 30 years: content-based, text-based, and emerging hybrid approaches.

For several decades much effort, both in academia and industry, was directed at understanding the content of an image so that computers could generate an index for it to enable searching. For many of the reasons described above, which all contribute to the semantic gap, these efforts were not successful. Companies such as IBM (with the QBIC system for images search), Virage and Musclefish (for sounds) endeavored to make a success of this approach but it was challenging to capture consumer's interest in such an early stage of the market.

In 2001, Google introduced an image-search engine that used the text surrounding an image on a Web page to improve results. This is a clever way to close the semantic gap, since the words on a page often describe the accompanying image. For this to work, the file name and the anchor text, i.e. the pieces of text more closely associated with a Web link, are the best information and should be given higher weights [**?**]. This information enables large-scale Web-search engines to automatically label images using the tags already generated by humans on the Web.

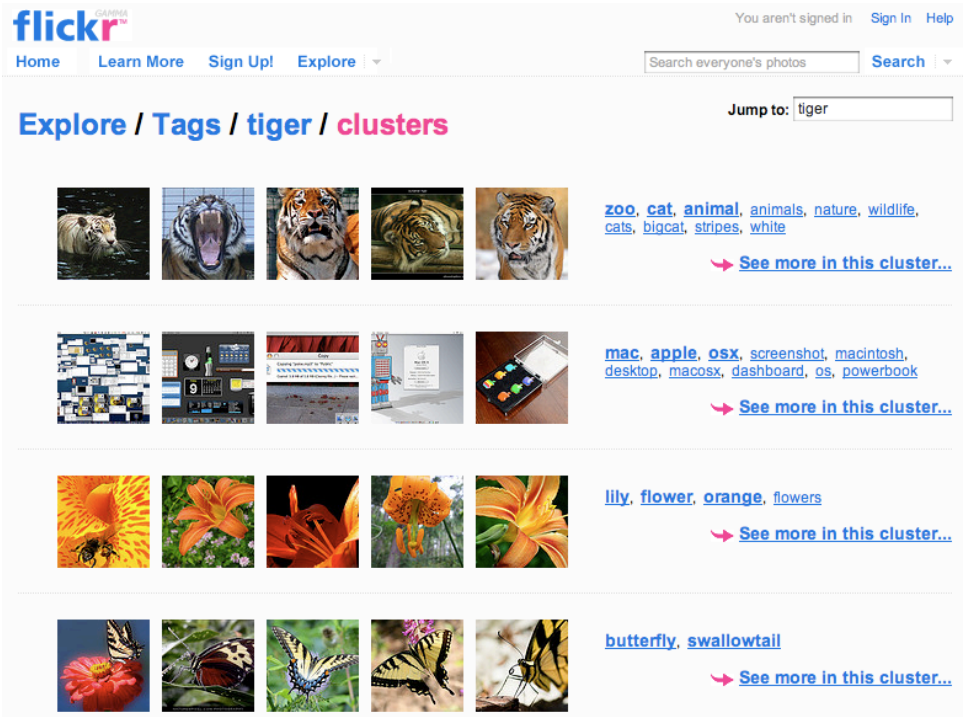

Figure 0.4 Images clustered by tags in Flickr^{TM}, showing images of tigers, computers, flowers and butterflies. **Recapture, using PDF and extra the portion we care about for higher resolution**

This text-based approach for searching Web images has been extended in two different directions. In the first, sites like Flickr encourage a community of people to upload photos and tag them with a few words that describe their contents. Search becomes then a matter of retrieving the images whose tags match the query and ranking these tags (see Chapter**??**). The photos and their tags can be clustered (Figure 0.4) but image relevance is still an issue. Flickr sorts images by date and a score that is called interestingness. However interestingness is often a measure of artistic content instead of a query-specific relevance score.

In the second, sites like ESP Game encourage people to label an image

through a competition [**?**]. Two users, selected at random, are shown an image for which they independently suggest labeling words. The competition comes about from the fact that each user should try to pick words that are in the mind of the opponent. Neither user sees the words the opponent is guessing until there is a match, i.e. one user types a word previously typed by the other. Since the image is the only thing the two users share in common, the common words used by both users are good tags for the image. By giving the same image to multiple pairs of people, we can ensure that the tags are real and reflects a consensus, and not just a fluke of the game. The ESP approach aims at annotating the whole image. An extension aimed at specific regions of the image is the game PeekABoom, which asks people to select a region on an image containing the key object of interest [**?**].

The content-based search approach, which is fully based on image features extracted automatically, and the text-based search approach, which is fully based on words suggested by humans, represent extremes in the multimedia IR world. Clearly, something in the middle is more appropriate. Face recognition, speech recognition, optical character recognition, and other supervised automatic techniques for content recognition and extraction indicate that computers can assist with reducing the semantic gap in multimedia IR. In the remainder of this chapter we explore some of these approaches.

0.3 Compression and MPEG Standards

Unlike text documents, one almost never sees an uncompressed multimedia object—the files are just too large. Most multimedia files are stored in a lossy format, based on special algorithms that remove redundant information that the human brain can not perceive. For example, the human eye can more readily perceive changes in intensity than it can perceive changes in color. We can send less information about the color changes and still the human eye sees the image in a form that is perceptually identical to the original image. Strictly speaking information is lost, but the *perceived* quality of the compressed content can be as high as that of the original content.

As we will describe in the following sections, perceptual coding is accomplished by converting the signal from the form in which it is recorded into a new format where it is easier to separate the perceptually important parts from the irrelevant or less necessary parts.

Compression is important in multimedia IR for two reasons. First, all of the source material is in a compressed form. Thus, we need to understand the limitations of the compressed files so we can build compatible algorithms. Second, we can use the compressed-domain features to reduce the time it takes to decompress an image and the time it takes to compute a feature vector for the same image. This feature vector is important because it is a mathematical description of some feature of the image that will be used in pattern recognition or in a decision making stage to follow. This is commonly done, for example, by

using the motion vectors (computed to match different portions of an image) as a proxy for a full-blown motion-estimation algorithm.

Five key procedures combine to make image and video compression so efficient and useful. These factors are color subsampling, removing spatial redundancy with the discrete cosine transform (DCT), entropy coding, motion compensation, and removing temporal redundancy. They constitute the fundamental steps that compose XXX state of the art image/video compression algorithm. We will not talk about audio compression, such as MP3, because it is not hard to decompress audio, since it is not that expensive to (re)calculate the desired feature vectors, and lossy and lossless principles used in image/video compression are similar to those for audio [**?**].

0.3.1 Intensity and Sampling

Color and intensity are the most basic elements of a picture (or movie frame.) Everything else in the image and thus multimedia IR builds on top of this data. **Berthier did not understand the last sentence**. By the time a picture becomes part of a multimedia IR system, the image has already been captured by an image-sensing device. As part of this process, the intensity of the light is sampled at discrete points as a function of time and space. If the image is sampled too coarsely, then information is lost. If it is sampled too finely then there is unnecessary (redundant) information captured, in which some compression rate is not optimized. **dulce: is this really what we want to say?**. Usually, we safely assume that the sampling is just right for the image content and the display parameters. However the intensity information is usually captured uniformly, no matter what the color, and as we will see, the human eye is not uniformly sensitive to all colors and changes. This presents the first compression opportunity.

0.3.2 Color

Color is a basic feature of an image, one which is perceived and distinguished by humans. However, the human eye is sensitive to only visible light, which is a small region of the electromagnetic spectrum. Visible wavelengths are in the range of 400 to 700 nanometers (nm). Each color corresponds to a narrow band in this range and the human eye can distinguish 400,000 colors. Values above 700 nm constitute infrared, FM radio, TV waves, while values under 400 nm correspond to ultraviolet and X-rays. None of these are perceivable by humans.

Humans generally perceive color with photosensors that are sensitive to three different bands of color. Because of this, the gamut of all colors are produced with red, green and blue (RGB) phosphors in graphics displays. A specific color in an image is produced with a particular intensity of these three colors, often on a scale from 0 (black) to 255 (bright).

While color is represented on a screen in terms of RGB intensities, this is not how the human visual system perceives it. Because of this, in practice, other color systems are used to represent the color information in a manner that is more consistent with how the way humans perceive color. One popular alternative color description scheme is the one known as hue, saturation and value (HSV). In this scheme, basic colors (red, green, purple) are encoded in the value of hue. Value (or brightness) is the overall intensity or energy of the light source. The amount of saturation determines whether the color is pink or a deep red—its vibrancy. This color representation is more perceptually relevant than a hardware-based scheme such as RGB.

dulce: we need to decide if we want to use math notation for RBG and the color schemes, or if we leave them as regular fonts

A related color system known as YCbCr is used as the basis of image (JPEG) and video systems (MPEG and DVD). Like HSV, the YCbCr system encodes the color with three values: a luminance or Y, a blue chroma signal C_b, and the red chroma value C_r. Given RGB values that are gamma corrected (a non-linear correction applied to intensity values so the perceptual intensity is more linear, which accounts for the non-linearities in the display system), the YCbCr values are given by the following three equations:

$$\begin{aligned} Y &= K_r * R + (1 - K_r - K_b) * G + K_b * B \\ C_b &= \frac{1}{2} * \frac{B - Y}{1 - K_b} \\ C_r &= \frac{1}{2} * \frac{R - Y}{1 - K_r} \end{aligned} \tag{0.1}$$

where the variables R, G, and B represent the intensities of red, green, and blue in the RGB scheme and K_r and K_b are constants given by $K_r = 0.299$ and $K_b = 0.114$.

Downsampling the color or chrominance information is an important step in image compression. Our eyes are much better at detecting changes in the luminance (intensity) than at detecting changes in the chrominance (color). Thus, after conversion of an image to the YCbCr scheme the Y signal is kept unaltered, while the C_b and C_r signals are each downsampled by a factor of 2 or 4,in both horizontal and vertical directions.

Figure 0.5 shows the effect of this downsampling on a color image and its three components. The original image looks "perfect" (even though it is rendered in black and white in this book). But looking closely, we notice that the C_b and C_r signals are both downsampled by a factor of 2 both horizontally and vertically. It's clear in the B&W pictures, but we can't see it in color. Ignoring all the processing still to come (removing spatial redudancy and using entropy coding), this downsampling alone reduces by 50% the number of bytes needed to encode the picture. This is because the 12 values needed to encode the RGB colors of an original 2x2 grid of points still needs four values for the intensity or Y values, but only 1 value for each of C_b and C_r, giving a total of 6

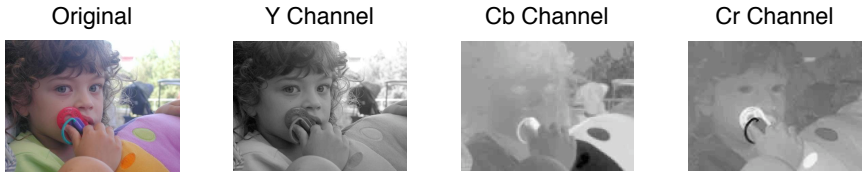

Figure 0.5 A compressed image, and its three YCbCr components. Note the compression artifacts, best seen by looking for the jagged diagonal lines in the Cb and Cr images. The image after compression, far-left image, looks "perfect" even though there are visible artifacts in the Cb and Cr subimages.

values instead of 12.

0.3.3 Lossy Compression

After conversion of the image to a perceptually relevant color space, like YC_bC_r, two kinds of compression are performed. First, a lossy stage throws away information that the human eye cannot perceive—the information is lost, but in a way that is imperceptible, or minimally perceptible, to humans. For instance, we have already described the benefit of downsampling the color information. Following this first lossy-compression stage, we perform a second lossless compression to remove statistical redundancies in the signal. At this stage we do not throw away any information because even a single-bit error at this stage will lead to a very perceptible image distortion. We introduce the lossy stage first, followed by a discussion of the lossless coding process in Section 0.3.4.

The sensitivity of the eye is often described in terms of its ability to perceive different frequencies. Beyond roughly 6 cycles † per visual degree,**what is visual degree? explain** our ability to perceive a pattern is quickly reduced. Thus, an effective way to compress an image is to sort the frequency content of the image and only keep the low-frequency changes.

Images are often described in terms of their spectral content. An image, or a portion of an image, is decomposed into spectral components using a discrete Fourier transform (DFT). The DFT represents an image in terms of a weighted sum of spatial sinusoids, such as those shown in Figure 0.6. Depending on the image (or fragment of an image), different spectral components will get different weights. It is important to note that a 256×256 image is transformed into 256×256 spectral weights, and then when the inverse DFT is performed, the original image is restored, within the bounds of floating-point accuracy.

check that the 256 x 256 looks OK.

Use smaller fonts in the next figure describing different spatial frequencies.

† A portion of an image goes from light to dark and back to light again in one cycle.

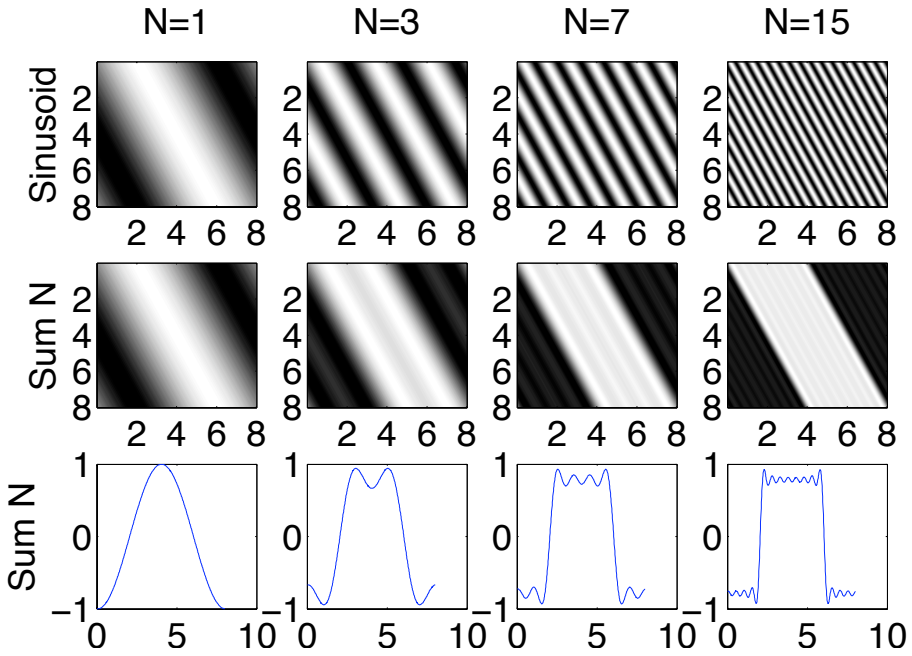

Figure 0.6 Several different spatial frequencies are shown and how they combine to represent arbitrary data. The top row shows simple sinusoids of 4 different spatial frequencies. The middle row shows the sum of the first N harmonics, showing better approximations to the image of a band with sharp edges. The bottom row shows a slice through the middle of the images in the middle row. **Berthier is not sure he understands the sentence right before this comment**. *The bottom row shows the intensity of the image along a horizontal line that crosses the center of the image.* This is the basis of spectral analysis.

Since our eyes are most sensitive to low spatial frequencies, it is practical to aim at transmitting the coefficients of these frequencies with higher precision. This spectral analysis is accomplished using the discrete cosine transform (DCT) and ensures that the most important frequencies are transferred with the highest fidelity. As part of a DCT-based image compression algorithm, the image is first partitioned into as many blocks of 8 x 8 pixels as are needed to fully cover the image.

Can you motivate the next paragraph in greater detail? Why 64 blocks? Where do horizontal and vertical frequencies come from? What do they mean physically?

The DCT represents each block of pixels with 64 different base functions, each representing a different pair of horizontal and spatial frequencies. These 64 base functions are shown in Figure 0.7. We can compute the DCT of a block, which yields 64 coefficients, one for each base function, and then compute the

DCT again and return to the original image with only inconsequential rounding errors. Nothing is lost [**?**]. **Give equations for DCT and define basis functions! Insert the equations**

the term CPB in the next figure is not defined.

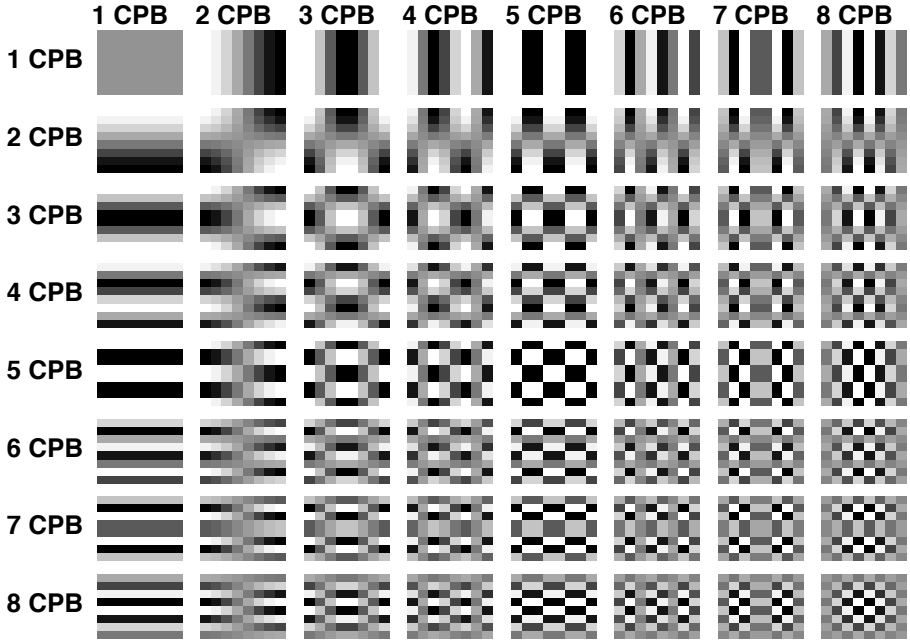

Figure 0.7 This figure shows the 64 basis functions that are used for an 8x8 DCT. Rows and columns are labeled with the number of DCT cycles per horizontal or vertical block. **no need to put the zig-zag pattern at the side of this figure, Dulce: are we not showing the zig-zag pattern at all?. May be we can supper-impose it in this figure with some arrows, it would not be too hard graphically but it would need some explanation.**

The DCT transforms an 8x8 block of pixels from the original image into a spectral domain that has two important properties. These properties are important because they tell us which portions of the signal can be thrown away, reducing the number of bits used to represent the image while keeping those with highest impact on the perception humans have of the image.

The most important energy in an 8x8 block is transferred into the upper-left-hand (low frequencies) corner of the DCT representation. For the majority of natural images, the low-frequency terms of the DCT concentrate most of the energy. Thus, there is little high-frequency energy associated with the image representations, which implies that we often do not need to retain their low values. That is, we gain compression efficiency by simply dropping the high-frequency (low-energy) terms. To compress the coefficients we traverse them in a

zig-zag pattern from upper left to bottom right, reserving most of the transmitted bits for the first coefficients.

dulce: in the last paragraph I would say, the coefficients are stored in a specific patterns, a zig-zag pattern, that enables compression.

Following, we notice that humans are more sensitive to the precise levels of the low-frequency components, than to the precise values of the high-frequency components. This offers an additional opportunity for compression by varying the amount of quantization for different points in the DCT block. That is, varying the number of bits used to represent the analog signal. Often, the intensity of each pixel of an image is quantized to one of 256 levels, but the same image can also be represented with 128 levels, with a corresponding reduction in the image quality (in effect, more noise is added to the image.) A range of quantization levels is shown in Figure 0.8.

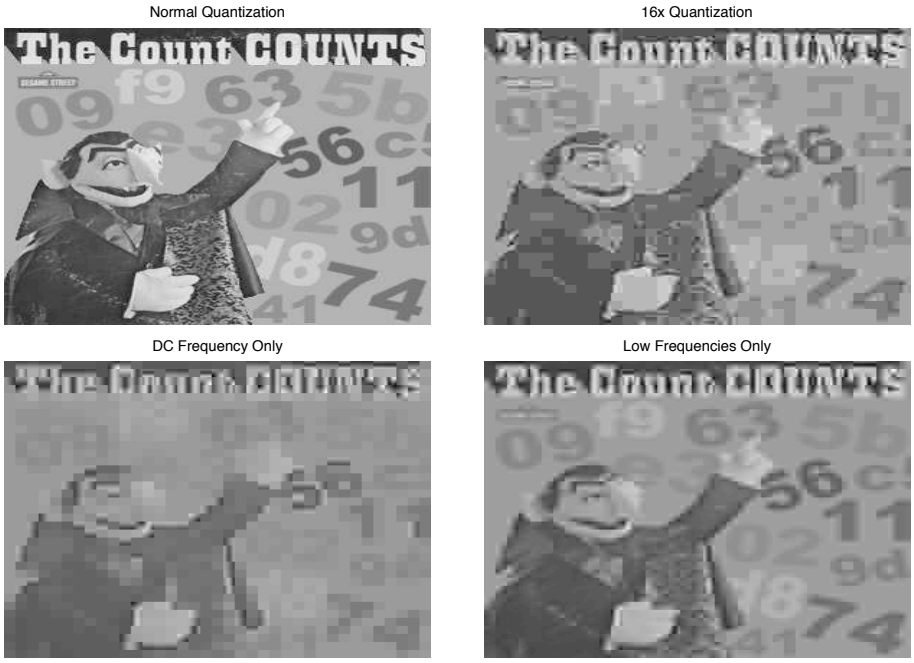

Figure 0.8 Images that were quantized with different quantization levels. Upper left: normal quantization. Upper right: all spatial frequencies were quantized using intervals that are 16 times bigger (4 fewer bits per frequency). Lower left: DC term transmitted, higher frequencies are dropped (note that the 8x8 blocks are now visible). Lower right: only DC and first frequency (a linear slope) are transmitted (note that the 8x8 blocks are no longer clearly visible)

Berthier asked what is the DC Term, so we might re-phrase it or clarify. Dulce suggests that this is a good point to indicate that the

lower left image is usually denoted to be a blocky image.

The DC (direct current, or average value) coefficient, often used in multimedia processing, represents the average value of the pixels in the 8x8 block. By assembling all the DC components of the compressed file we get a thumbnail, an image that is subsampled by 8 in both horizontal and vertical directions. We still need to undo the entropy compression, discussed in the next section, and undo the quantization, but we can use the raw DC coefficients without modification. Working with just the DC coefficient is one type of compressed-domain processing. Since the DC component is the most important one, it is transmitted with the highest fidelity. High-frequency components, even if they have energy left, are transmitted at a very coarse level of quantization, perhaps only a couple of bits per coefficient.

0.3.4 Lossless Compression

The lossy compression techniques described above are usually followed by lossless compression techniques—techniques that further compress the data but without introducing any errors in the representation. Once we have removed the perceptual redundancies in the signal, we aim at removing the statistical patterns in the numbers. Two common approaches for this are known as run-length encoding and entropy coding.

As part of the DCT processing we rearrange the pixels in the 8x8 DCT block in order of their importance. We throw out the coefficients that are closer to the lower right corner of the 8×8 block because they are high-frequency, thus hard to see, and low-energy, thus not important. But because nearby values, even after the DCT, tend to have similar values—they are correlated—we gain efficiency by only transmitting the changes. We then use run-length encoding (RLE) to more efficiently transmit these coefficients by transmitting a value and a count of how many times it is used. **Berthier thought one could transmit the values in a series of deltas**

A second stage of compression is known as entropy coding. Entropy coding uses the entropy (randomness) of the coefficients to design an optimal coding scheme. In the text domain, Lempel-Ziv is a common way to compress a string of characters in a way that it can be decompressed to extract the original text without errors. In image compression, Huffman coding is used to transmit each symbol with the fewest number of bits, which still allows for the exact reconstruction of the original run-length encoded string.

0.3.5 Temporal Redundancy

Removing redundancy is the key to compression. In image coding, we remove redundancy by using lossy compression techniques to throw out details of the image that are imperceptible to the human eye. Then we follow with two kinds of lossless coding, RLE and entropy coding, to efficiently transfer the remaining

information. In video compression we can do two additional and related types of redundancy removal: motion estimation and image prediction.

Motion estimation and image prediction are important in video because one frame of a video often looks very similar to the next. We can transmit just the changes between one frame and the next, which compose a delta image, and thus improve our ability to compress the video. Ideally we transmit only the first image in a scene, and then transmit delta images based on the prior reconstructed images. This implies that an intermediate image can only be reconstructed if we first decompress all preceding images. Further, this scheme is very sensitive to errors, which makes skipping through the video more difficult.

An alternative is provided by MPEG compression in which the delta images are computed in both forward and backward directions relative to fully-transmitted images known as I-frames. Frames that are predicted from preceding frames are known as P-frames, while images that are based on backward predictions, based on future images stored in a buffer, are known as B-frames. A typical MPEG video stream is transmitted as one I-frame followed by **Dulce: check the correct sequence** P-frames (and any B-frames), as we discuss in Section 0.3.7.

0.3.6 Motion Prediction

Given two images, we compress the first image using the techniques in Section 0.3.3 and then use this compressed image to predict the second image, transmitting only the differences between them. The simplest way to do this is to use a pixel in the first image to predict the exact same pixel in the second image. This works but is not optimal for video, since video frames change and objects move.

Thus video compression uses a more powerful technique known as motion compensation. In motion prediction, we predict the value of each 16x16 block of pixels in the new frame based on nearby blocks of pixels in the prior frame. This involves a search for the best match in the preceding frame. In other words, we look for a Δx and a Δy that minimize the difference function $E(\Delta x, \Delta y)$ between two consecutive frames I_1 and I_2, represented as follows:

$$E(\Delta x, \Delta y) = \sum_{x,y} (I_1[x + \Delta x, y + \Delta y] - I_2[x, y])^2. \quad (\mathbf{0.2})$$

As a result, each 16×16 block of the image that is highly (or sufficiently) similar to neighboring blocks is represented by a a predictive displacement function, which requires fewer bits.

Dulce: if possible I believe we should re-phrase the two paragraphs on this Motion Prediction Section. It is not clear what Δx and Δy are. That is, it is not clear that they are displacement vectors corresponding to the new location of the 16x 16 block. It is also not clear that the prediction is done based on color, the distance measure is based on color and this is why it does not reflect true motion. This

is a fundamental limitation of this type of motion prediction and ti is key to the limitations of all the MPEG streams when it comes to any inferences based on motion, and precisely the limitation of motion vectors in the MPEG stream. We do not have to write all that here but it should be covered somewhere in the chapter.

Frames in the stream are compressed either independently (i.e. standalone) or relatively to neighboring frames. Those compressed standalone are called I-frames and might occur every 30 frames in a video compressed with typical compression parameters.

Each 16×16 block of the image is compressed by analyzing the prediction error

$$I_e(x, y) = I_t(x, y) - I_i(x + \Delta x, y + \Delta y) \quad (\mathbf{0.3})$$

where I_t is the current frame in the video stream, I_i is the reference frame, i.e. the decompressed frame we are using as a reference, Δx and Δy are the motion prediction vectors for this macroblock, and I_e is 16×16 block image error. It is important that I_i is the *decompressed* image because that is all the receiver has access to, i.e. the receiver does not know what errors were made during the lossy compression. **Dulce: This is also the reason it is denoted the reference image, it is the image that both encoder and decoder can refer to without getting out of synch**. In the best case, the error image I_e is all zero and consequently only the motion prediction vectors need to be transmitted. Typically, the errors are non-zero but small leading to good compression. When the errors are large, it implies that the motion prediction vectors are not doing a good job predicting the values in the new frame. In this case, the system rules discards the motion prediction vectors and instead the entire frame is transmitted as an I frame. All of these ideas are part of the MPEG compression standard we discuss in Section 0.3.7.

The idea behind equation 0.2 and motion prediction is clear, but it hides four details. First, the summation is usually carried out over a macroblock, a 16×16 region of the image, yielding an estimate of which pixel values are best found in the other image. Thus, the summation for each offset costs 256 multiplications and additions. Second, in a brute-force implementation of this calculation the cost grows with the square of the maximum distance we consider. Searching a 2x2 window is easy (but not very likely to find a match), while searching a 32x32 window is expensive. Thus people often use intelligent search strategies to search a large range of possible motions, yet avoid calculations that are unlikely to bear fruit. Thirdly, the best motion-prediction vectors for any one macroblock are independent of any other block. The calculation looks for the best prediction of any one block, which often will be similar to object motion, but this calculation ignores the aperture problem we discussed in Section 0.2.3. An example of the motion prediction vectors between two images is shown in Figure 0.3. **Dulce: we show motion vectors, it is a fundamental concept. We have a figure in the tutorial and I can get the permission from James Normile**. Finally, the name motion-prediction for this data is slightly

misleading. The "motion" data does not represent how objects in the image sequence move, but instead represents a mathematical correlation that improves compression.

0.3.7 MPEG Standards

The MPEG (Motion-Pictures Experts Group) standard is used to compress much of the audio and video content available to multimedia IR systems. MPEG describes standard approaches for compressing, transmitting, annotating, and decompressing multimedia content using a combination of the lossy and lossless techniques described above.

MPEG was formed in 1998 under the direction of the International Standards Organization (ISO) and the International Electro-Technical Commission (IEC) with the purpose of standardizing digital compression of audio-visual material. To date, there are five fundamental standards defined by MPEG: MPEG-1, MPEG-2, MPEG-4, MPEG-7, and MPEG-21. MPEG-3 was created to target HDTV, however it was dropped as a standard because MPEG-2 reached a level of maturity that proved adequate for HDTV. Figure 0.9 shows the evolution of the MPEG standards, the applications driving the requirements and the technology that enabled such standards. Initially, for MPEG-1 and MPEG-2, the standards focused on pixel-based compression algorithms, and then, for MPEG-4, they moved to object-based compression algorithms.

Different applications pose different requirements. Broadband, cable and HDTV, which required high quality (and bit rates), drove the creation of MPEG-2. MPEG-4 addresses applications that require lower bandwidth, such as Internet streaming. MPEG-7 represents a radical change in the standards; it was no longer about the pixels but about their inherent semantics, described by associated metadata. For example, multimedia search engines no longer need to compare the content of the multimedia objects directly, but they can compare the metadata. Finally, MPEG-21 is about enabling digital rights management in an efficient and transparent manner.

MPEG-1

Published in 1992, MPEG-1 came about at a time when several industries (computer industry, consumer electronics, telecommunications, cable television, etc) were converging on digital video technology. The telecommunication industry had already developed H.261, a digital video compression standard for teleconferencing. However H.261 does not support interactivity or fast-forward/backward capabilities.

MPEG-1 addressed the need to fit digital video on a storage media such as Video CD and CD-ROM. To grasp the magnitude of the challenge, it is worth noting that MPEG-1 requirements specificied that the compression rate should be sufficient to store the video and (stereo) audio streams in the same storage

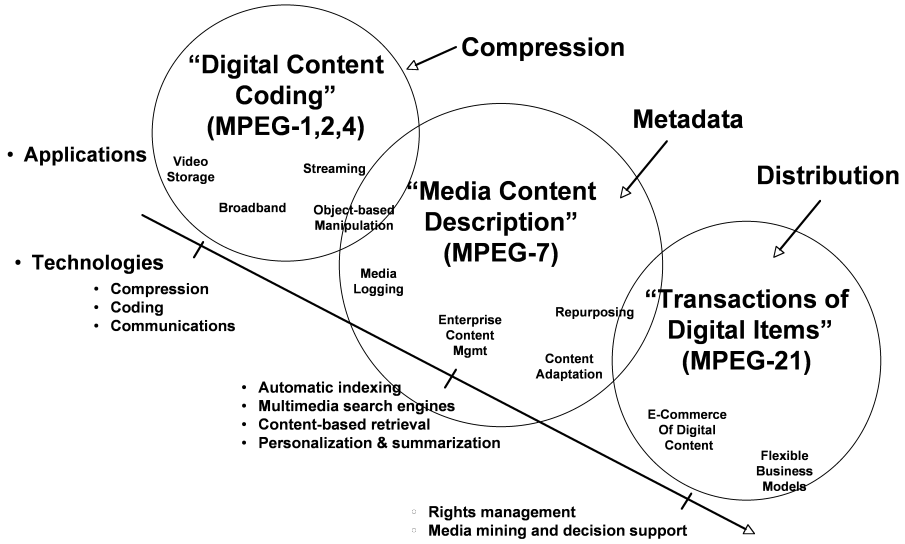

Figure 0.9 Evolution of the MPEG standards, and their driving applications.

space that, until then, was used exclusively for (CD-quality) audio. Moreover, the need of the gaming industry to increase interactivity in video games drove a requirement for random access. Consequently, precise timing control and synchronization of the audio visual bitstream was necessary. As a result, MPEG-1 was designed to achieve acceptable video quality (roughly equivalent to VHS recording technology) at 1.5M bits/s with a frame size of 352x240, 29.97 frames per second, and stereo audio at 192 bit/s. In short, it provides an efficient compression algorithm that could be decoded in real-time using hardware available at the time (1993).

MPEG standard compression dulce-dulce decompression algorithms are asymmetric, that is, content is encoded once and decoded frequently. The complexity of the encoder to the decoder is of roughly 3 to 1 **find reference**. Thus, early real-time applications implemented the encoder in hardware while the decoder could be done in software. A coder and decoder pair is denoted as a dulce-dulce *codec*. Details on the encoding procedures are outside the scope of the specification. Instead, a codec is compliant as long as it produces a valid bitstream for decompression.

An MPEG-1 bitstream is composed of a system layer and of a compression layer. The system part defines a multiplexing scheme for interleaving audio and video packets and timing information such that that they can be played together synchronously. The bitstream is composed of a sequence of packs. Each pack has a start code, a header, and one or more packets of data.

In this section we focus on the video layer. The video part of the standard describes compression for non-interlaced (i.e., progressively scanned or sequen-

tially scanned) video signals. In progressive video all the lines of a frame are sampled at the same point in time. Distinctively, in conventional television, a frame is typically composed of two interlaced fields from two points in time. The first field contains the odd numbered lines of a frame (where the top line is 1) and the second field contains the even numbered lines sampled a few seconds after the first field. TV uses interleaved video because it better encodes high-motion objects, at some loss in spatial resolution. Most video applications other than television use non-interlaced video.

The audio part of the standard describes compression for perceptual coding of audio signals. Perceptual coding identifies portions of the audio signal that are not perceptible and transmits them at lower resolution. The standard defines three "layers" of MPEG audio coding: MPEG-1 Audio Layer I (MP1), MPEG-1 Audio Layer II (MP2) and MPEG-1 Audio Layer III (MP3). These layers are simply three different ways of encoding a stream. They vary in complexity and each layer uses different structures for applying psychoacoustic properties for masking. Currently, the most popular audio format is precisely MPEG-1 Audio Layer III, more colloquially known as MP3.

The MPEG-1 standard was a pioneered in many ways. It was the first video coding standards to be format agnostic supporting NTSC, PAL, and SECAM. It was the first audio-visual standard that defined the receiver and not the transmitter. The first video codec developed entirely in software and to provide a reference implementation. **Dulce: we might point out that the reference implementation is key to provide adopters with a quick way to check their development**. MPEG-1 is the most popular codec of the MPEG family, so widely adopted that it is playable in most computers and DVD players. MPEG-1 and MPEG-2 encoding are only block-based, while MPEG-4 is the only MPEG standard that can use object-based encoding.

The basic and smallest building unit in MPEG is an 8×8 set of pixels called a *block*. An MPEG video stream is composed of five layers on top of a block, as follows: macroblock, slice, picture (i.e., frame), "Group of Pictures" (GOP) and video sequence.A macroblock is composed of several blocks and it can include a motion vector. The 16×16 pixel macroblock, depicted in 0.10 is the basic unit in a video frame and is used for motion compensation. As shown in that figure, a sequence of macroblocks form a slice, while one or more slices (preceded by a header) compose a picture (or frame), several pictures form a GOP, while one or more GOPs (preceded by a header) compose a video sequence.

MPEG uses two fundamental techniques for compression: intra-frame Discrete Cosine Transform (DCT) coding and motion-compensated inter-frame prediction as described in Sections 0.3.3 and 0.3.4. For intra-frame coding, a two-dimensional DCT transform is applied. The transform takes as input an 8×8 block of 8-bit pixels of the original picture and generates as output an 8×8 block of 11-bit DCT coefficients

MPEG uses information from neighboring areas to compress specific areas of a *frame*. **Berthier: the following sentence is vague, use a better description or provide a reference to motion vectors**. A motion vector captures the movement of the target area and makes prediction easier. Prediction

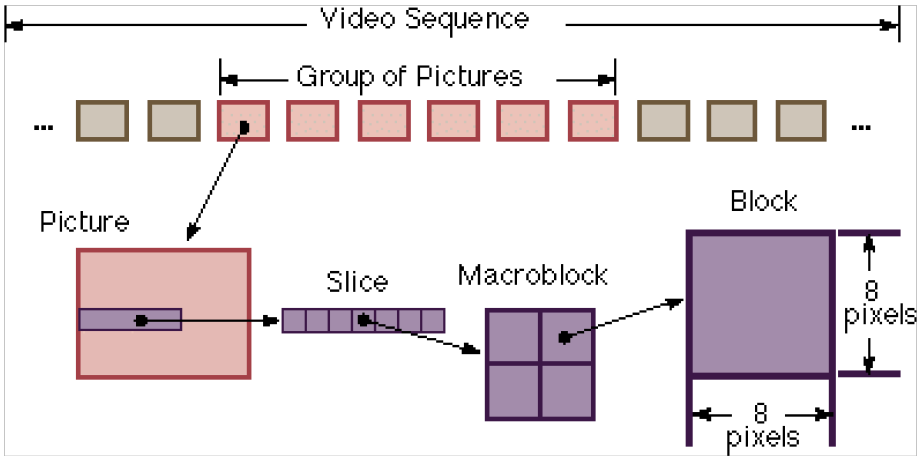

Figure 0.10 This figure shows how different layers of the MPEG standard are used to represent an entire video.

involves more than just looking at previous frames. In fact, three types of frames are used: I, P and B frames. I-frames are denoted "intra frames," while P-frames and B-frames are denoted "inter frames." I-frames do not reference any other frame. They are simply coded as a still image. Consequently, decoding can start at any I-frame. We say that I-frames provide anchors into the video stream since they constitute the entry points for random access, as well as synchronization point for error recovery. For example, in the case of extreme loss of packets in the video stream, instead of attempting to recover from such drastic errors, the approach usually taken is to skip the stream until the next I-frame is found. Such I-frame provides a fresh start from the error recovery point of view. **Dulce: do we need to explain intra vs inter? it is kind of obvious, i.e. intra means WITHIN a frame and inter means BETWEEN frames**.

The P-frames are compressed and then reconstructed using forward prediction. Their reconstruction requires either the previous I-frame or the previous P-frame. From one of these previous frames, along with the motion prediction vectors, we can calculate the new frame.

B-frames, or bidirectional frames, are unique because they use both forward and backward predictions. B-frames are reconstructed from the closest past I-frame or P-frame as well as the closest I-frame or P-frame in the future. MPEG prediction can be described from the point of view of the encoder or the decoder. Figure 0.11 shows a typical sequence of decoded frames, B B I BB P BB P BB P, where the solid arrows show forward prediction and the dotted arrows show backward prediction. This repeating sequence is called a GOP.

Berthier: Center figure and show dotted arrows for backward prediction. How are frames numbered? Check the order.

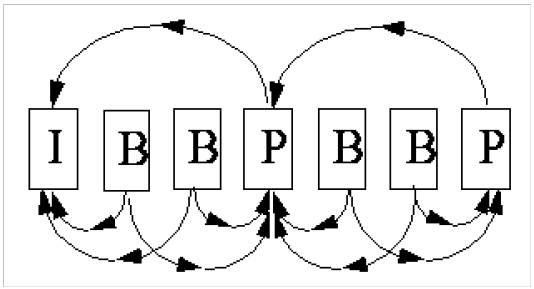

Figure 0.11 This figure shows how I-, B-, and P-Frames are used to represent a sequence of frames, providing efficient inter-frame prediction and random access.

The structure of a GOP supports random access into a video sequence. A GOP is characterized by the number of frames in the sequence and the spacing of P-frames. In theory, a GOP can be of any size. In practice, to enable immediate decoding (without reference frames) at intervals close to 0.4 sec, there is a limit (12 in the US) on the number of frames between two consecutive I-frames. The ratio of P to B frames can vary. For example, the first frame of a new shot is quite different from the reference frame, hence encoding it with a P-frame leads to poor prediction—instead an I-frame should be used. With compression, an I-frame is typically three times the size of a P-frame, and this in turn is 1.5 times the size of a B-frame.

Given the frame dependency, the coding/transmission order of frames must be different from the display/playback order. Otherwise, the decoder would have to suspend reconstruction of B-frames until the reference P or B frames arrives. The frame sequence shown in Figure 0.11 can be transmitted as I BB P BB P BB P BB, or equivalently 3 1 2 6 4 5 9 7 8 12 10 11, if we use frame numbers. The decoder needs three buffers, one for forward prediction, one for backward prediction, and one for the reconstructed image. Each block in a P-frame can be intra-coded or predicted. Analogously, each block in a B-frame can be intra-coded, of predicted (forward, backward or bidirectional).

MPEG-2

MPEG-2 was developed in response to applications like broadband and high-definition TV that require higher quality (and demand higher bandwidth) than that provided by MPEG-1. The MPEG-2 standard allows bit rates from about 3–15 Mbit/s for broadband and 15–30 Mbit/s for HDTV. MPEG-2 shares many of the video coding elements in MPEG-1. MPEG-2 decoders will also decode MPEG-1 bitstreams. Unlike MPEG-1, it provides multi-channel surround sound coding. Also, MPEG-2 provides a simple profile and a main profile. The simple

profile is targeted at applications that cannot afford a big delay, such as teleconferencing. In this case, the profile does not include any backward prediction; there are no B-frames. This means that there is little delay, because there is no need to reorder the frames for transmission.

MPEG-7

MPEG-7 is the first MPEG standard that describes the semantics of media. It encodes metadata about the content, not the content itself. In this regard, it can be seen as a content description standard. **Dulce: I wanted to mention the bits about the bits since this seems to be a famous phrase to describe MPEG-7, however we could mentioned it in the bibliographic notes**. MPEG-7 specifies a set of description schemes, descriptors, a language to specify such description schemes, and a procedure for coding the descriptor. The language is denoted Description Definition Language (DDL) and it is defined in XML. **Berthier indicated that the following paragraph is vague. Dulce believes that it is valuable to include an example of the description schemes to give people a flavor of what they are and how they have been used so far**. Examples included in the MPEG-7 describe standard features for video summarization and navigation. The technology to compute these descriptors are outside of the scope of the standards.

MPEG-21

Finally, MPEG-21 is an open framework for multimedia delivery and consumption. It addresses the challenges that multimedia producers face while describing the intellectual properties rights associated with a piece of multimedia. In MPEG-21, the fundamental unit of transaction and distribution is a Digital Item (DI), i.e. a combination of audio, images, video, and text metadata which captures the relationship among the components. MPEG-21 defines the technology to support transactions of Digital Items, so that users can interact with them efficiently and seamlessly. The Rights Expression Language is a standard to allow sharing digital rights information among the various players involved, from content providers to consumers.

0.4 Segmentation: Prior Processing of Multimedia Data

Video and audio signals are often difficult to work with because of their size and the lack of obvious boundaries. An hour of video at normal television resolution consumes 2 Gbytes, when compressed with MPEG-2. Even today these quantities of data are difficult to work with. But more importantly, there is often little information in these large multimedia signals to help one understand its structure. This has practical implications for instance, while searching a video,

the user should not be obliged to view the entire two-hour video to find the 10-second shot of his interest.

Thus, one of the first steps when inputing **dulce: berthier changed ingesting inputing but the literature denotes is a video ingest** a video into an IR system is to segment the video into manageable semantic units, i.e. shots. A shot corresponds to an uninterrupted sequence of frames captured by a camera. For instance, an interview-type video will have several shots, with the camera alternating between the shots of the interviewer and of the one being interviewed. A video is then composed of a number of scenes, where a scene is defined as a sequence of adjacent shots that are semantically coherent. In turn, related scenes are grouped into higher sematic units, which receive different names in the literature, such as segments, and stories [**?**]. This hierarchy of units in a film or a video is shown in Figure 0.12.

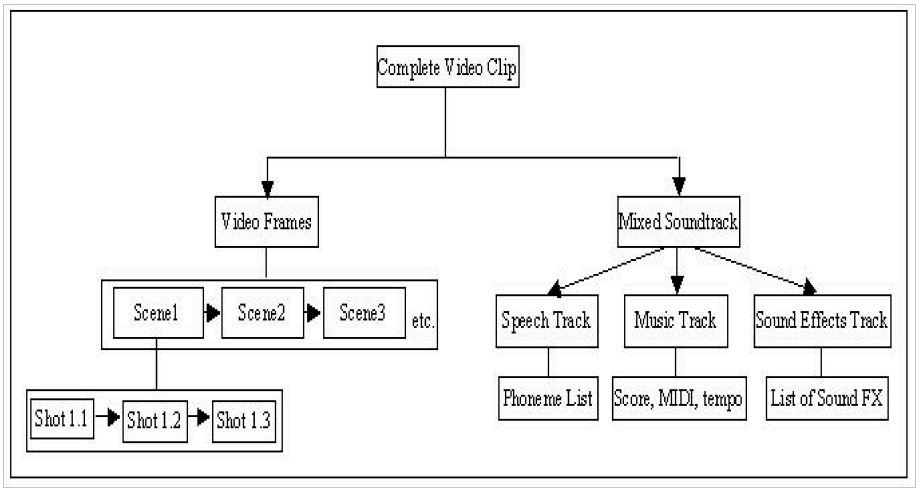

Figure 0.12 The hierarchy of objects in a film or video.

A video changes from one shot to the next with a transition. For example, a director cut is an abrupt transition, one that is often easy to detect because the entire image changes instantaneously. Other transitions like fades, dissolves, and wipes occur slowly over time. Noise in the signal, camera flashes, fancy transition effects and even fast moving objects make shot boundary detection challenging. In here, we consider three common types of scene breaks in this chapter: cuts, fades and dissolves.

Segmentation algorithms can be of various types or categories, such as, pixel-based, statistical differences, histogram-based, edge-based, DCT-based, and motion-based. We discuss some of them here. We also discuss audio segmenta-

tion based on speaker identification. However, before we proceed, let us illustrate with a simple example.

0.4.1 A Segmentation Example

Simple measures of the image, such as the pixel-by-pixel difference between two frames, do not provide a good foundation for segmenting video. To exemplify, when common changes happen in the video, such as a change in the camera viewpoint, all the pixels change thus causing a large pixel-by-pixel error. A different approach is provided by a shot-boundary detection algorithm that looks at summary statistics over the entire image to determine when there are major changes in the video. Early methods based on statistical differences divided the image in several regions to subsequently compare the differences in those regions, such as mean and standard deviation of gray levels between the regions. However, many false positives are generated by these types of methods.

One simple global statistic of value is a color histogram of the image. A histogram is computed by counting the number of pixels of each color in the image. Color is often represented as three 8-bit numbers, so the number of colors is too large to count directly. Instead, we broadly quantize each color dimension into perhaps 8 levels. This gives us 512 different color types ($8 \times 8 \times 8$ colors). Histograms are the most common method used for shot boundary detection.

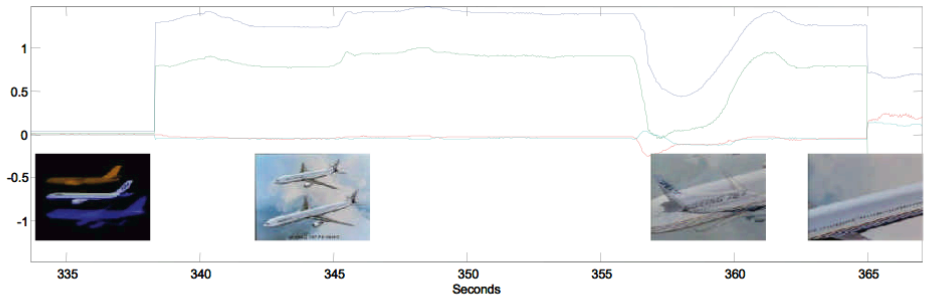

Figure 0.13 This figure shows the first four dimensions of a reduced-dimensionality color signal. It shows shot boundaries between 338 and 365 seconds; and a dissolve from 357 to 360 seconds. Sample frames from each section are also shown (from "21st Century Jet").

Berthier: Define reduce-dimensionality. Supperimposing two images does not show clearly in the figure hence we either use a better example or add enough of an explanation. It is hard to explain the tail that shows in the supper imposed frame

The color-histogram approach is illustrated in Figure 0.13 [**?**]. Here, a 512-bin color histogram was computed for each frame. This yields us a 512 dimensional signal over time, sampled 30 times a second. For illustrative purposes, we use the singular-value decomposition (SVD) algorithm to find the best low-rank approximation to the signal. We then show the first four resulting color signals—the dimensions from the Singular Value Decomposition (SVD) that contain most of the energy—as a function of time. **Berthier: check that SVD has already been defined**. We now have a simple global measure of changes in the video. As we observed Figure 0.13, we notice small changes in the signal during the middle of a shot, for example, due to motion of the object in the frame (at 345 seconds). But shot changes are clearly visible as a dramatic change in the color signal, for example at time 338. And near 357 seconds, we have a picture in which two related images are superimposed to provide a smooth transition between video clips. Such smooth transition is denoted a *dissolve*.

0.4.2 Segmentation Schemes

The color-histogram method works well because changes to the video caused by camera or object movement, or even lighting changes, cause the color histogram to change slowly—most of the image does not change, it is only rearranged. There are many different ways of calculating these global statistics and performance depends on how well a specific statistic captures abrupt changes and ignores normal motion in the video [**?**].

Another simple measure of an image is its overall mean luminance. This is calculated by throwing away the color information in the frame and averaging the overall brightness. Counting the number of edges in a frame provides a more sophisticated measure, as discussed later in this section.

Berthier says: define fades and dissolves! Malcolm: I do, don't I? At start of each paragraph. Do I need more? Berthier: fades has not been defined

Detecting fades is more difficult because it represents a change in the video over time—starting with a normal image, then the video linearly decays to black, and then the video grows into a new image. A simple fade detector looks for frames that have a constant color. Then the luminance before and after the fade will follow a linear curve that we can detect by calculating the first derivative of the mean frame luminance.

Dissolves are the hardest segment boundary to detect. In a dissolve, the image is slowly changed from one scene to another by cross-fading the pixels, for example performing linear interpolation on a pixel-by-pixel basis. One way to detect such changes is to measure the overall variance of the luminance of each image frame. Normally this measure will be quite high because the incoming and outgoing images contain changes in illumination that we can see. But during a dissolve, the two images are blended and the combination necessarily reduces the overall variance. Thus, we can detect a dissolve by looking for a dip, lasting several seconds, in the mean luminance variance, as illustrated in Figure 0.14

[**?**].

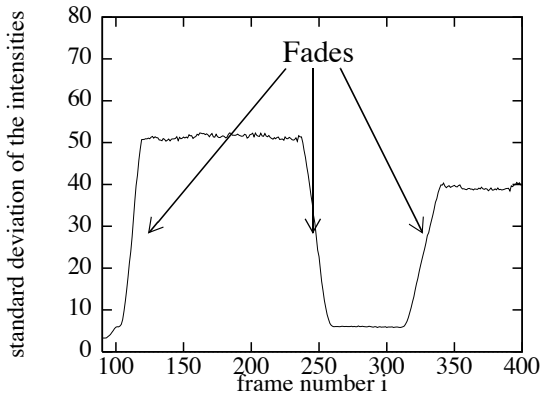

Figure 0.14 This figure shows the dip in the standard deviation of the image intensity during three fades.

A more robust approach to find dissolves is to build an explicit model as done by Covell [**?**]. In her analysis-by-synthesis approach, she notes that given any two points within a dissolve the intermediate points are simply a linear interpolation of the endpoints. This gives us a direct means to detect dissolves. We can sample pairs of frames in the video at intervals of 1 second, for example, and check to see if the intermediate frame is predicted by a linear interpolation of the endpoints. The prediction error gives us an estimate of how likely a dissolve is at this point. We can expand the region with the low-prediction error to find the beginning and the end of the dissolve.

Shot-boundary detection is challenging because false positives are so common. Video can change quite quickly—a car goes streaking through a shot—confusing simple segmentation algorithms. At the other extreme, gradual shot transitions can appear as slow and smooth changes in color and/or intensity, in which case they might not be detected as transitions. Yet, even with these limitations, shot-boundary performance is quite good, certainly good enough to get users to the right general area of the video they are interested in. These algorithms are straightforward and work in real time, using only one pass through the video. Even the compressed-domain signals can be used for segmentation.

0.4.3 Segmentation with Edges

A more robust measure of scene breaks is based on edge statistics. Edges are sharp discontinuities in the luminance of an image. They are interesting because they are robust to lighting changes and camera motion. Their presence or absence tells us a lot about what is happening in the video.

Zabih describes a system that combines motion estimation, edge detection, and then computes the edge-change fraction [**?**]. The basic idea is to look for edges that do not appear in the next image in the sequence (and vice versa). This is challenging because there is often camera motion that causes the edge in one frame to be at a different location in the next frame. Thus, we first register the two images to remove any global motion. Global image registration is done by finding the shift $\Delta x, \Delta y$ that maximizes the correlation between the current frame (I_1) and the next frame (I_2), given by

$$\sum_{x,y} f(I_1[x + \Delta x, y + \Delta y], I_2[x, y]). \quad (\mathbf{0.4})$$

Given the offset required to bring the two frames into alignment, we now have two images that are roughly aligned and, as a result, can find and match the edges.

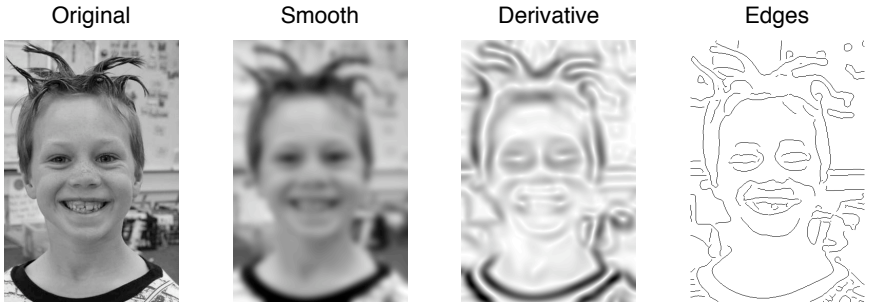

Figure 0.15 The process of computing edges using Canny's method. From left to right: the original image, the image after blurring with a Gaussian with standard deviation of 6 pixels, the magnitude of the spatial derivative of the smoothed image, and the location of the edge points after thresholding.

Berthier: use a,b,c,d, labels for original, smooth, derivative and edges

A Canny edge detector [**?**] finds important (for the purposes of this algorithm) points in an image. This process is illustrated in Figure 0.15. The image is first smoothed using a two-dimensional Gaussian function $G(x, y)$, as specified in Equation 0.5, to blur the image, remove noise, and set the smallest edge that we want to see. **Berthier: where is y? Define μ and σ.**

$$G_\sigma(x, y) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \quad (\mathbf{0.5})$$

where μ is the medium and σ is the standard deviation. The Gaussian function is convolved over the entire image, effectively averaging the image over small windows, to smooth out high-frequency edges and noise. This blurring operation

is written

$$I_s(x,y) = \int\int I(s,t)G(x-s,y-t)dsdt \qquad (\mathbf{0.6})$$

but it is often implemented using fast Fourier transform (FFT) methods, because they are much faster than the brute force equation above. The smoothed image is shown in Figure 0.15b. We then compute the magnitude of the spatial derivative at each point in the image. The result is seen in Figure 0.15c, in which we notice a sharp transition on either side of the black collar and a high-magnitude signal on either side of the collar. The final edge locations are determined using an adaptive threshold that picks the points of maximum derivative (Figure 0.15c) from the smoothed image. The locations of the edges are shown in Figure 0.15d.

Scene breaks of all kinds are found by counting edges that come and go between frames. For each edge location, we look for a corresponding edge in a small region of the other image. The fraction of edges that are found in one image and also in the other image provide a measure of image similarity. In a static or slowly moving image, most edges will not move far from one frame to the next. Only when the entire scene changes, either due to a cut or dissolve (everything changes) or due to a fade out (when eventually the edges are no longer visible), will this measure register a low similarity and thus a scene break. Edge-based detection can be more accurate than histogram-based detection, since it is less sensitive to motion and chromatic scaling.

0.4.4 Bayesian Segmentation

A segmentation boundary is characterized by a decision on the event that something has changed in the signal. A probabilistic way to make this decision is to build a model of the first portion of the signal, advance the model through the signal, and detect the point at which the model no longer fits or explains the data. This becomes a segmentation boundary. But this (one-sided) calculation is error-prone because a new point might not fit the model and we cannot be sure if this is because of noise, or because of a real change in the signal. Instead, we use a double-sided approach that compares models of the signal on both sides of a potential boundary. This test is implemented using the Bayesian information criteria (BIC). This approach is often used in speech segmentation, as we discuss in Section 0.7.4. The details are not important for now, but suffice to say that this signal is a 13-dimensional vector sampled at 100Hz.

A Gaussian mixture model (GMM, again see Section 0.7.6) models the data using a sum of independent Gaussian probability surfaces in 13-dimensions. Each dimension of the data is assumed to be independent of the others and the covariance matrix that characterizes each mixture component is diagonal. Thus each mixture has 26 parameters, 13 for the means and 13 for the variances.

BIC builds a model for a long segment of the signal, and then segments the signal into two smaller pieces and builds two different models. Two separate

models will always fit the data better than a single model, since there are more parameters available to fit the data. The data is different on either side of the boundary and thus two different models work best. A segmentation boundary is proposed whenever the two models, when balanced for their extra complexity, are better predictors of the data than the single model.

The BIC test accounts for the extra complexity, when comparing the one-model approach versus the two-models approach by including a penalty term for the extra modeling complexity. Given a model M_i and data D_i with $i = 1, ..., N$, BIC computes [**?**]

$$BIC(M_i) = \log P(D_1, D_2, ..., D_N|M_i) - \frac{1}{2d_i} logN \qquad (\mathbf{0.7})$$

where d_i is the number of independent variables in model M_i. The first term expresses the log likelihood that the model explains the data with high probability. Higher likelihoods are better. The second term penalizes models that are more complicated because they take more parameters to describe them. When using BIC to predict segmentation boundaries, the number d_i parameters changes by a factor of two, while the *log* of the number of datapoints only changes by a small amount. Thus, we are looking for places in the signal where two models produce a significantly better model, with higher likelihood, that outweighs the cost of the extra parameters in Equation 0.7.

We can find the optimal point to split the data by using fixed-sized windows, perhaps 10 seconds long, on either side of the proposed boundary and sliding the boundary along the signal looking for the biggest gain in the BIC test for a two-model explanation over a single model for the entire 20 seconds.

0.4.5 Hierarchical Segmentation

The techniques described above serve to find scene breaks by looking at small portions of the video. However, the local image information does not tell us anything about how the scenes are combined into shots or larger story elements. Researchers have suggested using color information to group shots, or even building models of the video background, but these are limited. **Berthier: what is this semantic content?**. A better way is to use the semantic content in the video as a key to the scene breaks. This is because the semantic content gives a more coherent sense of the direction the video is going. Except for a film like the 1997 film Titanic, where the first part of the film is bright and cheerful and the second half is dark and foreboding, the overall color of a movie does not tell much about the story.

In general, the words in a video script, and the story they tell, give information about much longer time scales than the color information. This is shown in Figure 0.16, which compares the coherence of the color and a low-dimensional estimate of the semantic information. The color information is represented using a low-dimensional SVD approximation as was done in Figure 0.13.

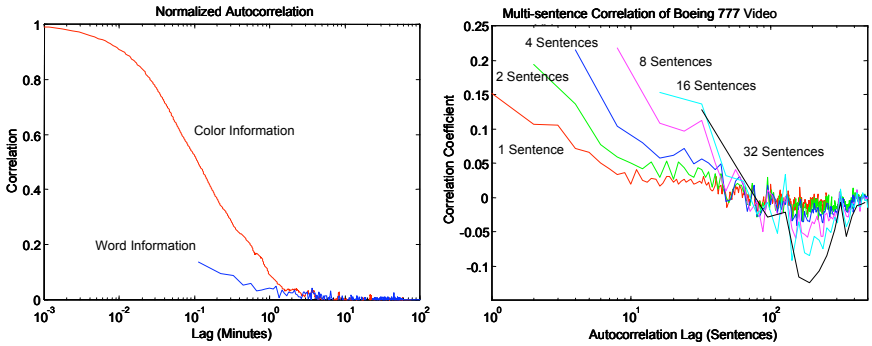

Figure 0.16 These two figures show the correlation for color and semantic data in a BBC video about the Boeing 777 airplane. On the left the autocorrelation is a function of minutes (the word information corresponds to a smoothing window of one sentence in this left graph). On the right the autocorrelation is a function of the number of sentences. [From XXX] **Berthier: complete the reference**

We look for changes in the semantic content of the video document by performing LSI analysis (see Chapter **??**) over different portions of the document, like the video script. The chapter you are now reading talks about compression, segmentation and retrieval. Each of these subsections contains different types of words and will be represented by different points of an LSI space. We want to characterize the distances over which topics change.

Berthier: The next paragraph needs proper motivation to fit into the text. May be use the terminology metadata instead of semantic content?

In principle, we do this by starting at every sentence and computing an n-sentence histogram, $h(i, n)$ where i and n vary from 1 to the number of sentences in the document. It doesn't make sense to compute word histograms over sections of text shorter than about 1 sentence because the histograms of so little text are noisy reflection of the underlying topic. Thus, chunks of N words at a time are extracted, the histogram is computed, and this vector is projected onto the LSI model giving us a low-dimensional vector representation of the semantic position of these N words [**?**]. The semantic vector moves as the sliding window shifts through the text. It moves slowly when the sentences contain largely the same material, and moves by large amounts when the topic changes. It is these topic changes that we want to note. The autocorrelation of these vectors gives us a measure of coherence, which is computed by evaluating the following integral

$$R_{xx}(\tau) = \int_{-\infty}^{\infty} x'(t)x'(t+\tau)d\tau \qquad (\mathbf{0.8})$$

where $x'(t)$ is the signal we are measuring with the mean value removed.

Figure 0.16 shows the autocorrelation or coherence for both color and se-

mantic information. The color coherence in the left of Figure 0.16 shows that there is little coherent color information after .1 minutes, or 6 seconds. This corresponds to the average shot length, suggesting that after a few seconds there is little we can tell about the structure of the video from the color information. On the other hand, as shown in Figure 0.16, the coherence of the semantic information depends on the length of the sliding window. For short windows, it is hard to get an accurate sense of the position in the semantic space. But when the sliding window covers more sentences, a better estimate is possible and the correlation goes up. After about 30 sentences, the coherence starts to fall back towards zero. A sentence takes perhaps 6 seconds, so we are seeing correlations in the semantic signal using this simple measure that last up to 3 minutes. The sliding window essentially acts as a rectangular smoothing window over the semantic data, much as the Gaussian window does when finding edges.

We can exploit the longer coherence of the semantic information to build a hierarchical segmentation of a video [**?**]. In practice, the binary indication of whether a sentence is included in the n-sentence histogram is replaced with a smooth weighting function, using wider and wider windows in a method known as scale-space filtering. The modified signal then becomes

$$s_\sigma(t) = \int s(\tau) g(\sigma, t-\tau) d\tau \qquad (\mathbf{0.9})$$

where $g(\sigma, t)$ is a Gaussian kernal with a variance of σ^2 and t is the index of the center sentence in the calculation. When σ is close to zero there is almost no change in the signal, but as σ grows the signal is smoothed by greater and greater amounts. We now look for edges, or points of maximum discontinuity as a function of both σ, the scale parameter, and t, the time.

The result of this analysis is shown in Figure 0.17. The figures on the left side show the results of processing a 30 minute news program. The top left panel shows the magnitude of the semantic change (as a function of time along the horizontal axis and smoothing on the vertical axis.) At small scales there are many small changes that are sharply defined. As we move to longer scales, the small changes are filtered out and we are left with just major topic changes (as seen at the top of Figure 0.17).

The local maximum, as a function of time, indicates the points of maximum semantic change at that scale. Because of the properties of the Gaussian smoothing window, given a maximum at one point we can track this maximum through smaller scales to find its location in the original (unfiltered) signal. Thus we can trace the signals shown in the middle–left of Figure 0.17 to find the exact point of the discontinuity, and then straighten the line to show a scale-space segmentation. The resulting segmentation captures the magnitude of the semantic change by the length of the line (how many scales does it exist over).

The right side of Figure 0.17 shows a comparison of the scale-space segmentation (lines on the left) to a hand-generated ground truth for this video (text on the right). Titles in all capitals represent major events in the story and match long lines in the scale-space segmentation. Two long lines in Figure 0.17

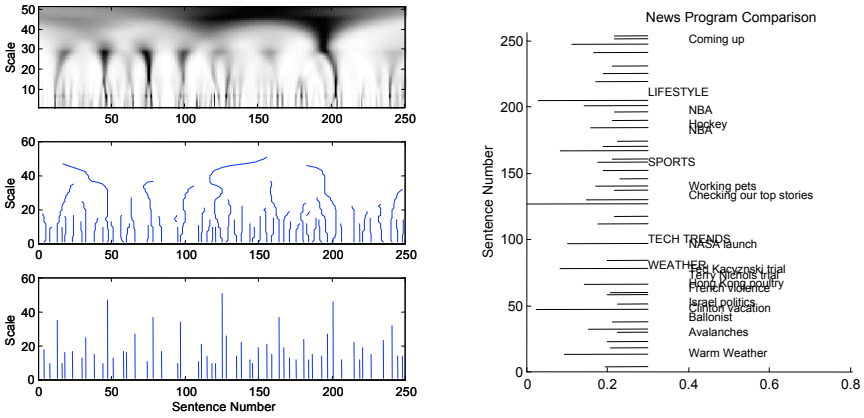

Figure 0.17 This figure shows the calculations for scale-space segmentation on the left, and results on the right. Along the left side, we see at the top: filtered semantic change as a function of position in the video (horizontal axis) and scale (vertical axis), middle: local maximum as a function of time, bottom: straightened lines indicating point of original segmentation.

do not have a corresponding major heading in the ground truth. The line just below 50 sentences is not matched by a major heading, but the new stories before 50 sentences are mostly related to weather, while those after the line represent more general news. Likewise, the line at the midpoint of the newscast is not marked in the newscast, but again represents a major topic change because the top-of-the-news is restated.

0.4.6 Segmentation Evaluation

Shot boundary detection is considered a relatively mature area of research. Numerous approaches can be found in the literature, not only for detection but also for classification of shot and transition effects. Several representative survey papers compare many of the existing approaches [**?**], [**?**] [**?**] .

Early shot boundary detection algorithms focused on abrupt transitions, i.e. cuts, and later on, attention was given to gradual transitions. Approaches that target a specific type of transition have been developed. However in practice, this implies that a general purpose algorithm might require several passes through the video. Some approaches, such as those using global statistics involved a threshold or a set of thresholds, that are set either manually or automatically. In practice, only automatic adaptive thresholds make sense, since trying to find a global threshold that works with all kinds of video content is futile. The challenge is to develop a single-pass algorithm that can robustly detect cuts and transitions in real-time with a good precision-recall compromise.

The first benchmark results were obtained in the NIST TREC Video Track

shot boundary detection task, in 2001 [**?**]. Before then, comparative studies were difficult due to the lack a large test data set.

Currently, the best video reference collection is the NIST TREC Video Data, also called TRECVID 2003. This collection comprises two sets. First, 120 hours (241 30-minute programs) of ABC World News Tonight and CNN Headline News recorded by the Linguistic Data Consortium in 1998. Second, 13 hours of C-SPAN programming (i.e. 30 programs, that are mostly 10- or 20-minute length) from 1998 to 2001. The C-SPAN programming includes various government committee meetings, discussions of public affairs, some lectures, news conferences, forums of various sorts, public hearings, etc.

The total TRECVID 2003 video set is about 104.5 gigabytes (GB) of MPEG-1 videos, where 51.6 GB are from the development set (62.2 hours including 3,390 minutes from ABC and CNN, 340 minutes from C-SPAN) and 52.9 GB are from the test set (64.3 hours including 3,510 minutes from ABC and CNN, 350 minutes from C-SPAN). This data set is now available to the public.

0.5 Content-Based Image Retrieval

In this section we discuss the classic multimedia IR task—content-based image retrieval. Some of the oldest work on multimedia was aimed at identifying and extracting features related to the contents of image, as well as designing measures of similarity based on the extracted features. While the results produced were not commercially successful, the techniques they pioneered are an important part of the state of the art in multimedia.

The Problem

> To illustrate with one search method developed by this early research, we consider the query-by-example (QBE) approach. In this method, the user supplies an image and the system finds other images that are similar to it. Such systems ignore the semantic information and, instead, use simple features of the image, such as color, texture, shapes and most recently salient points to compare images.

For all types of QBE the best metrics are based on feature invariances. To exemplify, an image that you take of your dog is subject to many different changes such as pose, camera focal length and focus, lighting and your own viewpoint, which all have dramatic effects on the data we want to analyze. Almost any motion will move the pixels that represent an object—thus direct pixel comparisons never work. Instead, a common solution is to look at a summary of a feature across an entire image. This is similar to the bag-of-words approach use in text IR, since both approaches use histograms. We will look at several different approaches in the sections that follow.

0.5.1 Color-Based Retrieval

Color is often treated as a global feature for image retrieval. That is, it does not depend on the resolution of the image, even though the location of colors is very relevant for object perception. For example, consider US postal stamps that include the American flag in a red–blue striped background. Two stamps might look very different but still have an almost identical overall color distribution.

We use color as a feature in a QBE system by looking at color histograms of a picture. The colors are quantized into one of N bins (we discuss color representations in Section XXX) **Berthier: Because we are moving Section on Compression and MPEG to the end** and then we count the number of pixels in each bin. The advantage of a histogram approach is that even large changes in the viewing angle or in the composition of a picture do not change the average proportions of each color. That is, a color histogram is a measure that is independent of view and image resolution. As a benefit, there is no need to perform foreground–background segmentation, and it doesn't matter how complicated the object is. The histogram is defined as

Define n in the next equation.

$$h_I(c_i) = n^2 Pr[p = c_i | p \in I] \quad (\mathbf{0.10})$$

where $Pr[p = c_i | p \in I]$ is the probability that a pixel in image I is equal to color c_i.

We can improve our ability to retrieve the right image by modifying the color histogram to also include the relative locations of each color in the picture [**?**]. That is, we are now interested in the spatial correlation of the colors. For this, we build a color histogram by counting pixels as a function of colors c_i, c_j, and of the distance r between the pixel with color c_i and the pixel with color c_j.

A simplification, known as autocorrelogram, just counts those pixel pairs whose pixels are of the same color c_i and that are separated by a distance of r pixels. This is written as

Berthier: something is wrong in the next equation because the equation involves two colors, not autocorrelation. C_i should be either just c or C_1 and C_2.

$$h^{(r)}_{c_i,c_j} = Pr[v(p_1) = c_1, v(p_2) = c_2 | r = d(p_1 - p_2)] \quad (\mathbf{0.11})$$

where $v(p)$ is the color-value of pixel p, and $d(p_1 - p_2)$ is the distance between these two pixels. An example that shows the difference between the correlogram of two different images is show in Figure 0.18 [**?**].

Reduce the gap between the paragraph and the figure.

Example Color-based Content Retrieval

Image retrieval with color histograms (or correlograms) is performed by precomputing (and storing) the histogram of each image. Upon arrival of a query image, we simply search for the images with the closest histograms. An example of such results is shown in Figure **XXXX**, based on work in the QBIC system [**?**].

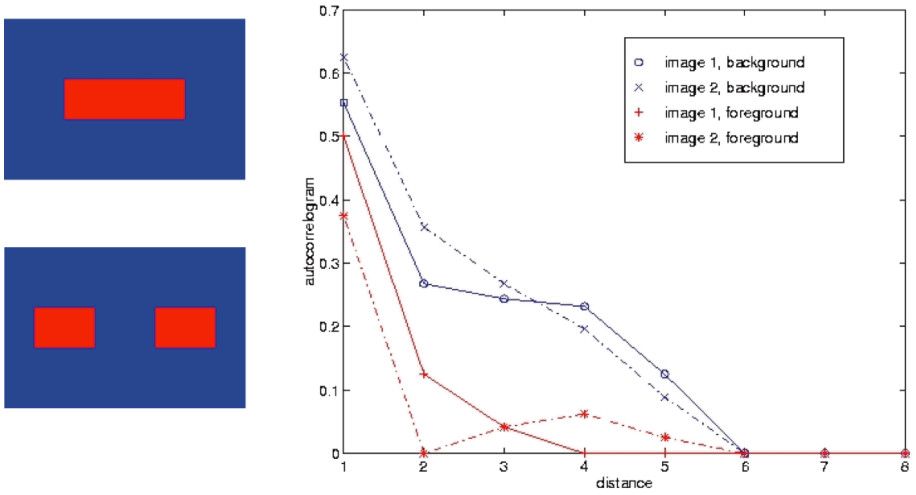

Figure 0.18 A comparison of color correlograms for two different images. The top two lines show the autocorrelogram for the background. The bottom two lines show the autocorrelogram for the foreground objects.

The biggest problem with retrieval based on color histograms is a perceptual property known as "color constancy." Human viewers have a dramatic ability to recognize the color of an object, almost without regard to the incident light. An apple looks red, whether we view it in daylight, under various forms of indoor light, or even at sunset when most of the incident light is red. Humans are good at perceiving the same colors, independently of the surrounding, but a color histogram is not so forgiving. In addition, the exact color of an object is often of little importance to its identity.

0.5.2 Texture

A second useful feature in image retrieval is texture. When we think of texture we think of the sensation we get when we touch the surface of an object, a fabric, food, a substance, or in general any tactile surface. For example, we say "the table has a rough texture, or a sandy texture or a smooth texture." The term texture is also used to describe visual patterns, that is, the structure of something in terms of size, shape, or the arrangement of its parts. We say "the wall has a busy pattern, a horizontal pattern, or a brick pattern, etc." Texture is

a property of the objects in the image—the texture of an orange is different from that of a grassy field or wood. Texture is an important component of human visual perception. Like color, it is a key feature to consider in image and video retrieval. However, unlike color, it is a property of regions of the image, not a property of a single point.

In the context of image processing, texture is a measure of the repetitive elements in the image. It is a perceptual phenomenon, easily detected by humans but challenging to describe mathematically. More precisely, texture measures characterize the repeating patterns of image intensity that are too fine to be distinguished as separate objects, and hence are dependent on scale. Most texture measures are invariant to intensity and orientation.

Texture can be represented with several metrics such as structural, statistical and spectral. There are numerous approaches for texture detection based on visual perception features, statistics features, filters such as Gabor wavelet features, etc. It is impossible to summarize all the research done on texture analysis, instead we will cover only a couple of approaches to convey to the reader the complexity of characterizing texture. For a detailed exposition of different approaches we refer the reader to other works [**?**] [**Mohan, XX**].

There are four broad categories of problems in texture analysis: texture classification, texture segmentation, texture synthesis and shape from texture.

- Texture classification refers to the problem of assigning a sample image to one of a set of known (labeled) texture classes of images.
- Texture segmentation consists of dividing the image into regions that have homogenous texture. Combining the difficulty of describing texture mathematically with the subjectivity of segmentation in general, texture segmentation poses quite a challenge.
- Texture synthesis is an area of research in computer graphics defined as the process of algorithmically constructing a large digital image from a small digital sample image by taking advantage of its structural content.
- Shape from texture consists of using the apparent texture distortions of a 2D image as a cue to determine the surface orientations in 3D scenes.

In here we are mainly concerned with the use of texture for image and video retrieval. Hence, we wish to find discriminative features, that allow for efficient representations and good quality similarity matching functions. With that in mind, we discuss four types of texture representations used in multimedia IR: co-occurrence, perceptual, spectral, and statistical. These methods have different properties in terms of their computational complexity and of the types of textures for which they provide an adequate representation. The first and the last ones are based on statistical representations, while the middle two (perceptual and spectral) are based on what is known of human vision.

Statistical approaches, which are good for capturing texture in natural images, were one of the earliest approaches used for texture classification. They look at the distribution of pixel intensities, or gray levels in an image. First-

order methods provide statistics on features that involve only one pixel, such as mean and variance. Second-order methods provide statistics on features that use two pixels that are in a particular relative position from each other. Given that texture is a property of a region, it is not surprising that second-order methods capture texture better. **Berthier: reduce the distance between this paragraph and the next section**.

Co-occurence texture measures

The simplest form of texture measure is built using a co-occurrence matrix, much like the color correlogram described in Section 0.5.1. The gray-level co-occurrence matrix (GLCM) measures the probability that two pixels at a specific distance and direction have the indicated values [**?**]. **Dulce: introduce that difference in location or distance between two pixels can be represented as vector**. Like the color correlograms $P_v(a, b)$ is the probability that two pixels separated by v, a vector, have the intensity values a and b. **Berthier: vector has not been defined, cannot follow**. This relationship is very detailed, containing $N_v N_p^2$ values, where N_v is the number of different directions and distances for which we calculate the GLCM and N_p is the number of intensity values. Since the number of directions and distances grows quickly, the pixel values are quantized so that only a few values are needed in the co-occurrence matrix.

The GLCM matrix summarizes information about the pattern of light and dark pixels at different directions and distances. Several different kinds of statistics are commonly used to summarize the information in a GLCM, as follows.
Energy: A measure of how bright the pixels are at this separation.

$$\sum_i \sum_j P_v^2(i, j) \tag{0.12}$$

where p, v, i, and j are defined as **define these variables**.
Entropy: A measure of the non-uniformity in the distribution of pixels at a direction and distance of v.

$$\sum_i \sum_j P_v(i, j) \log P_v(i, j) \tag{0.13}$$

where p, v, i, and j are defined as before.
Contrast: A measure of how different the two pixels are at the given separation.

$$\sum_i \sum_j (i - j)^2 P_v(i, j) \tag{0.14}$$

Homogeneity: A measure of the similarity of the pixels

$$\sum_i \sum_j \frac{P_v(i,j)}{1+|i-j|} \tag{0.15}$$

Example $N+1$: Coocurrence Texture-Based Content Retrieval

Berthier: Give an example with a figure to illustrate.

Perceptual Texture Measures

Another way to characterize texture is by using features that mimic the way people perceive texture. Tamura [**?**] proposed six numerical features to describe texture: coarseness (fine vs. coarse-grained), contrast, directionality (dominant directions), line-likeness (line-like vs. blob-like), regularity and roughness. He then went on to show in psychological studies that the first three features best capture the human perception of texture. These three most perceptually relevant texture features are described below. **Berthier: make sure that the spacing between this paragraph and the next are right**.

Contrast

Contrast is a first-order statistical feature. It captures the dynamic range of gray-levels, measured as a function of the variance σ and of the polarization of the distribution of black and white, measured as kurtosis α_4. Kurtosis is the fourth moment around the mean μ and the square of the variance σ^4 as follows: **check α vs. σ_4 in the equation below**.

$$\sigma_4 = \frac{\mu}{\sigma^4} \tag{0.16}$$

The contrast feature is defined as:

$$F_{cont} = \frac{\sigma}{(\alpha_4)^n} \tag{0.17}$$

Tamura found that setting $n = 1/4$ provides results that are closest to human judgments of similarity.

Coarseness

Coarseness ia a measure of granularity. It is directly related to the scale and periodicity of the image features. To compute coarseness we use moving averages with windows of varying sizes. The computation of coarseness can be summarized as follows. First, take averages at every point over windows of size 2^k pixels. The moving average over such window is given by

$$A_k(x,y) = \sum_{i=x-2^{k-1}}^{x-1+2^{k-1}} \sum_{j=y-2^{k-1}}^{y-1+2^{k-1}} I(i,j)/2^{2k} \tag{0.18}$$

where $A_k(x, y)$ is the moving average image at scale k and points (x, y), and $I(i, j)$ is the gray level at pixel (i, j). Second, for each point compute the differences between the moving averages in the horizontal and vertical direction as follows:

$$E_{k,h}(x, y) = |A_k(x + 2k - 1, y) - A_k(x - 2k - 1, y)| \quad \textbf{(0.19)}$$
$$E_{k,v}(x, y) = |A_k(x, y + 2k - 1) - A_k(x, y - 2k - 1)|$$

Third, at each point choose the size of k that maximizes the following:

$$\max_k(\max[E_{k,h}(x, y), E_{k,v}(x, y)]). \quad \textbf{(0.20)}$$

Finally, the coarseness is defined as the size of the window that produces the greatest contrast

$$S_{best}(x, y) = 2^k, \quad \textbf{(0.21)}$$

where k maximizes equation 0.20.

Directionality

Directionality is a measure of the prominence of a particular orientation in the GLCM. At each point in the image we look for an edge. **Berthier: edge has not been defined**. Following we characterize its orientation and the magnitude of the change. We then build a histogram of the directions we find in each region of the image. We compute the variance (second moment) of the histogram distribution to capture the sharpness of the directionality histogram.

Example $N + 2$: Perceptual Texture Based Content Retrieval

These three measures—coarseness, contrast and directionality—form a three-dimensional feature vector for an image, that we can use for content retrieval as follows:

Bethier: Give an example with a Figure to illustrate.

Consider deleting from the two approaches for texture that follow, i.e. spectral texture measure and statistical texture. If so, we need to fix the second paragraph on page 39. Alternatively we could move it to the bibliographic notes.

Spectral texture measure

A third approach to measure texture is based on Gabor filters, a common tool in computer models of human vision. A Gabor filter is a localized feature extractor in both frequency and space **Reference?**. The kernel of a Garbor function is the product of a Gaussian window multiplied by a complex exponential carrier

at frequency F **What is j in the next equation**.

$$g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-1/2(x^2/\sigma_x^2 + y^2/\sigma_y^2)} e^{2\pi jFx} \quad (\mathbf{0.22})$$

where $g(x,y)$ is the Gabor function at points x and y, and σ_x and σ_y are the width or variance of the Gaussian in their respective directions. **Berthier: previous sentence is vague**. We can then scale by m and rotate this function by θ using the following expression

$$g_{mn}(x,y) = a^{-m} g(x',y') \quad (\mathbf{0.23})$$

where $x' = a^{-m}(x\cos\theta + y\sin\theta)$, $y' = a^{-m}(-x\sin\theta + y\cos\theta)$, $a > 1$ and m, n are both integers to get a basic image wavelet. **Berthier: Has this been defined**.

We filter an image using a two-dimensional convolution as shown below to get the wavelet-transformed image

$$W_{m,n}(x,y) = \int I(x_1,y_1) g^*_{mn}(x - x_1, y - y_1) dx_1 dy_1 \quad (\mathbf{0.24})$$

where g^* is the complex conjugate of the Gabor filter g. Given the filter image, we compute the mean and the standard deviation of the transformed images, at each desired direction and scale, to give a measure of the spectral energy, and thus the texture.

Example $N+3$: Spectral Texture Based Content Retrieval

.

Statistical Texture

The methods above describe characteristics of texture but do not capture all the details necessary to completely synthesize a rich texture image. Texture in the real world is a combination of deterministic and random components. The deterministic component captures the repeating portions of the texture—the parallel lines or other regular structure. The random component represents the real-life variations that make our world so rich—the precise location of the blade of grass of the pebbles on the beach.

These two components of image texture form a useful measure for describing images. A herringbone **Berthier did not know this word, but it is a common English word, it comes from the bone of a herring, a fish** pattern has a deterministic component related to the spacing of the threads and a random component due to the color variations. The deterministic component of a grassy field will have a repeating pattern due to the width of a blade of grass.

A popular way to describe the statistics of texture is in terms of a Wold decomposition [**?**]. In a Wold decomposition, a data set is modeled in terms of an

auto-regressive moving average (ARMA) filter to describe the random variation in the object's intensity and a deterministic component that captures the regular structure. The ARMA filter describes the expected correlation between any two neighboring pixels. It filters white noise so that it has the correct spectral variations to match the image texture.

The deterministic and random components of an image are found using the Wold decomposition. Figure 0.19 shows how this decomposition works on a cloth texture. The original texture (a) is converted into the spectral domain using a Discrete Fourier Transform as shown in Figure 0.19b. The local peaks in this image represent the dominant frequencies and these points are shown in Figure 0.19c. When these spectral peaks are translated back into the image domain, we recover the very regular texture shown in Figure 0.19d. The random component of the texture is extracted by subtracting the regular component (0.19d) from the original image (0.19a) and then modeled with a low-dimensional ARMA filter. The ARMA filter describes the spatial structure of these random components. **reduce the space between paragraph and figure, plus center the figure**.

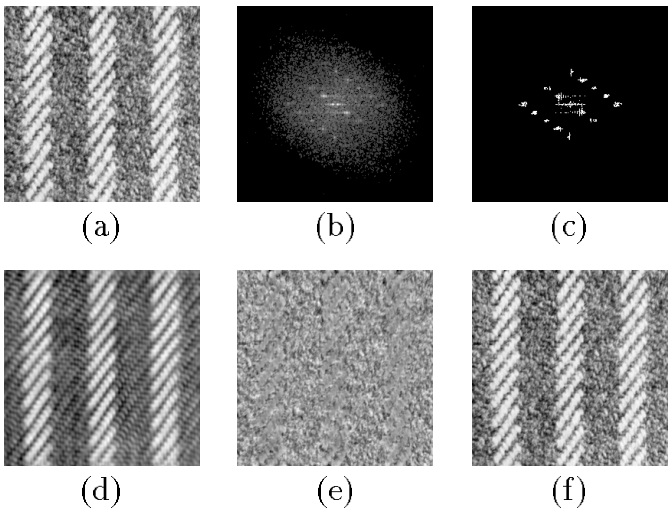

Figure 0.19 These images show the Wold texture decomposition. (a) the original herringbone texture, (b) DFT magnitude of the image, (c) the recognized discrete harmonic frequencies, (d) reconstructed harmonic components, (e) the random (noise) component after subtracting the harmonic components, (f) synthesized image formed by adding images (d) and (e). Images (a) and (f) should be identical.

0.5.3 Color Retrieval - INCORPORATE QBIC example in COLOR ABOVE, SAME as TEXTURE

We should not make this into a separate subsection. Instead, would use them as examples at the end of each subsection, as indicated above.

Berthier: the next section is hard to follow. However we all agreed that it is important material so we should improve it so that it is comprehensible for a beginner.

0.5.4 Salient Points

The algorithms for color and texture retrieval we described above are based on global histograms over the entire image. A more sophisticated approach builds a feature model, perhaps combining color and spatial frequency information, at "interesting" regions of an image only. We can then analyze an image looking for portions that are especially distinct (and thus salient to the perceptual system.) After detecting these points, at many different scales and over many different images, the local features at these points are clustered. We can then classify each salient point as belonging to one of these clusters. These then become "words" that can be counted.

Salient points [**?**] finds features in the image that are especially robust to blurring across a number of scales. These points are especially robust to changes in the lighting, position of the camera, and even the camera or the object's angle. This invariance is important because we want to find the same object regardless of these parameters of the imaging setup. Salient points tend to be corners or other very distinct places in the image. The entire algorithm consists of four steps: 1) find scale-space extrema, 2) select keypoints, 3) assess the most stable orientation, and 4) describe the local geometry or texture, as we now discuss.

Scale Space

Scale space is a way of talking about an image any many different levels of smoothness or blurring. At the largest scale we see only the building, while at a medium scale we see the windows, and at the smallest scale we see the stucco's texture. Different scales are analyzed by blurring the image with progressively larger Gaussian kernels.

Keypoints

At each scale, a difference of Gaussians (DOG) is computed by subtracting two images that are blurred with Gaussians at nearby scales (Equation 0.5).

$$D_\sigma(x,y) = [G_{k*\sigma}(x,y) - G_\sigma(x,y)] * I(x,y) \qquad (\mathbf{0.25})$$

where $*$ indicates convolution (see Equation 0.6). This is equivalent to filtering the image with a band-pass filter. This DOG operation can also be efficiently

calculated using rectangular windows [**?**]. Points in the image that are extremes, over both space and scale, are determined to be salient and thus worth of further study. These points are identified because the DOG output is larger than any other point in a 3x3x3 cube in image space and over scale. We can also fit a 3D quadratic function to this data to more accurately (to subpixel resolution) assess the exact location of the salient point. An example of the SIFT points for an image is shown in Figure 0.20.

Stable Orientation

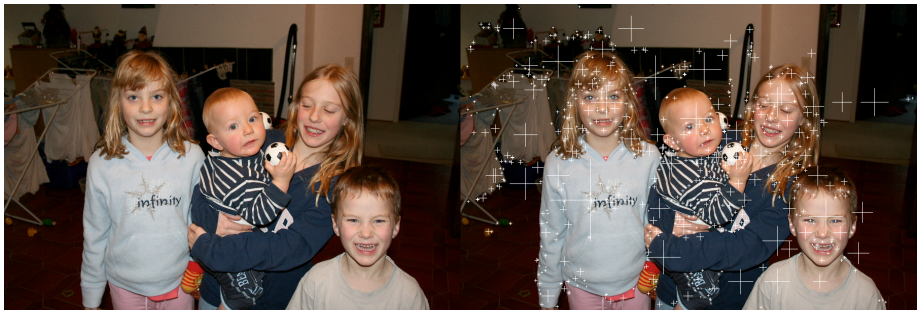

Figure 0.20 An image and its SIFT points. Image courtesy of Rainer Lienhart. **Make crosses bolder.**

Dulce: it is hard to see the cross points in the figure, may be we should manipulate the image so that the crosses stand out more easily and make the crosses bolder. Also it might help to add a sentence either in the figure or in the text indicating what to look for or pointing out what the image shows.

Given a salient point we want to characterize it so we can more easily compare it to points in other images. This is done in two steps. First, we find the dominant orientation by calculating the magnitude (m) and the direction (θ) of the local gradient at the points in space near the salient point. We form an orientation histogram of the derivatives (θ) by weighting each contribution by the product of the derivative's magnitude (m) and a circular Gaussian window centered at the location of the salient point. The largest direction, perhaps with one other direction of similar magnitude, determines the preferred orientation. The orientation stage is important because we want the descriptors to be orientation invariant—no matter how the camera is rotated we want to produce the same image descriptor.

Local Geometry on Texture

. Finally we describe the area around each salient point by computing the local orientation histogram over a 2x2 or a 4x4 set of blocks centered on each salient

point. These orientation histograms are computed relative to the dominant direction computed above. If we compute the histogram over 8 different directions in each of 4x4 blocks we have a feature vector at each point that is over 128 elements long. Clustering the resulting vector allows us to find typical features characteristic of different types of objects. This is shown in Figure 0.21 which shows two images and two different kinds of salient points [**?**].

dulce: check this reference on Srinivasan

Figure 0.21 These images show two different types of salient points in different environments. The upper images show typical salient points found in foliage and building scenes, respectively. The bottom figures show the same images with the opposite type of salient points. Note that very few angular (building) points are found in the foliage and visa versa.

Example M: Salient-Points-Based Content Retrieval

We calculate image similarity by summarizing the statistics of the salient points in an image. The image characteristics near each salient point are characterized with simple spectral filters, based on human perception, that sample the magnitude and orientation of the image change. This forms the basic "letters" of an alphabet that describe each point. These values are clustered, using an algorithm like k-means [**?**], to determine the "words" in the language. We then describe an image by how many times each word appears in the image. We can then use an algorithm like pLSA (probabilistic latent semantic analysis) to do image matching [[**?**]]. Figure 0.22 shows retrieval results, on an image similarity

task, in an LDA-compressed space [**Hoerster and reference to MIR book.**]

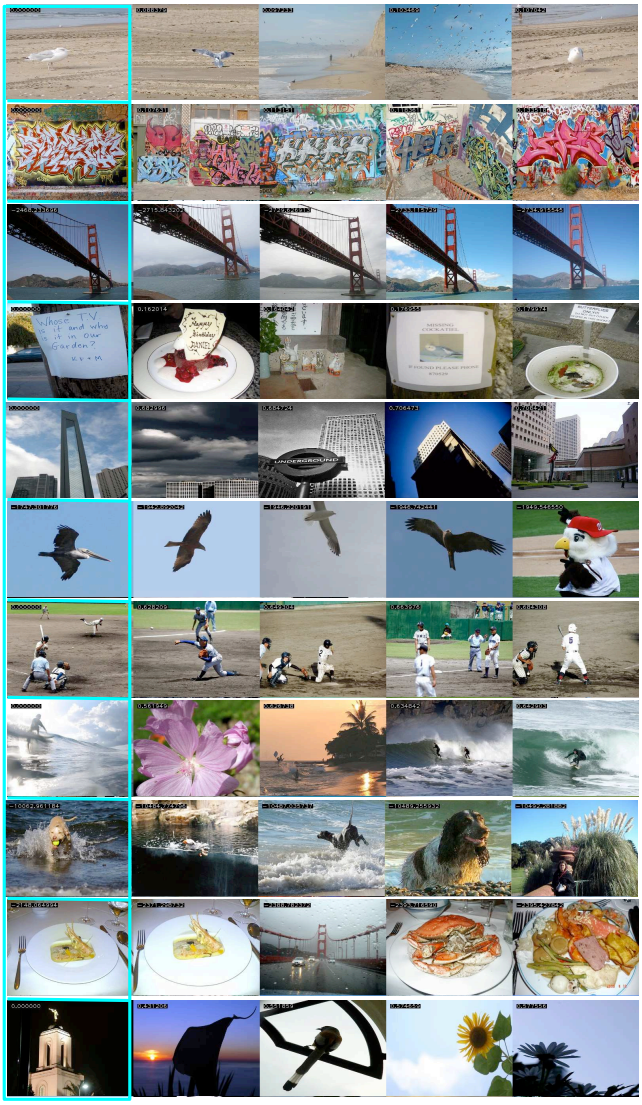

Figure 0.22 Retrieval results in an image-similarity task. The first image in each row shows the query.

0.6 Retrieving, Summarizing, and Browsing Video

Reduce space between figure and section title, as well as title and the beginning of the next paragraph

Searching and browsing are tightly coupled, although browsing receives less attention. In practice, we use an iterative search-and-browse cycle to identify the information we need **Shneiderman1998p523?**. Chapter **XXX** describes the strengths and limitations of these two fundamental paradigms of information retrieval as applied to textual data. This section describes the issues and possible solutions for multimedia data.

Presenting the results of a multimedia query is not as straight forward as it is in text retrieval. We need multimedia summaries to effectively show multimedia query results, making it easy for the user to efficiently navigate the media. In this section we describe the problems created when browsing media and we describe four types of solutions: abstracts, static summaries, dynamic summaries, and interactive summaries. **Dulce: revise this classification**

0.6.1 Static versus Temporal Browsing

Humans are amazingly efficient at scanning some types of displays for the information they need. The results of an image query—that is, a page with thumbnails of images corresponding to images that best match the query and possibly including textual labels—can be understood in a matter of seconds. Users can also quickly scan a static display and glean the information they need to know. Notice that this can also be a disadvantage since the user might not know where to start to best understand the display. For certain types of videos, one can use the fast-forward function to quickly grasp their visual content.

In contrast, we are not as efficient while browsing speech and audio data. It is challenging to define the audio unit equivalent to the thumbnail image due to the temporal nature of an audio signal. For starters, we can listen to one audio stream at a time. We can remove the silence, music and other non-speech sounds but this only helps a bit. A technique known as time-scale modification (TSM) can further modify the signal to speed up the audio signal by as much as a factor of four **fix this reference to Covell's Mach1**. This is done by modifying long sounds, such as the vowels in words, and relatively quiet portions of the speech so they play back much faster, yet retain the original sound for sounds that are very short, such as consonants. TSM preserves pitch, timbre and voice quality. While all these changes might reduce a 60 minute video to 10 minutes, this is still too long to be useful as part of a search-and-browse cycle.

In the following sections to follow we describe several approaches that allow users to more quickly understand multimedia content.

0.6.2 Video Abstracts

The problems of fast forwarding and rewinding have been noted in several places in the literature (Boreczky et al. 2000, p186; Yeo and Yeung 1997, p44; Christel et al. 1997, p21; Taniguchi et al. 1995; Elliott and Davenport 1994; Arman et al. 1994, p97). Users frequently have difficulty with identifying context. As a result, they tend to move backwards and forward, overshooting or undershooting the target. One approach for assisting the user with context is to provide in-context summaries.

User has not context and ends up moving backwards and forwards, overshooting and/or undershooting the target.

Interesting Taxonomy.. but I don't think we support it in this version of the text. Drop?. Dulce: What do you mean we do not support this taxonomy? the point is that there is not official taxonomy, every author makes its own, like you came up with the interactive summary one, that has rarely been used. I think it is important to note what have been the driving techniques for summarization. Perhaps we should not claim them as categories but as technical approaches towards summarization. Of course any opportunity that we provide the editors with a chance to drop a section they will agree, because we are tight in space.

Main categories of video summarizations: frame clustering, clustering and dimensionality reduction, leveraging domain knowledge, using close-captions, using speech transcripts.

A video summary (or video abstract) is a sequence of still or moving images (with or without audio) representing the content in such a way that it is easier for the user to inspect than the original video. Ideally, a summary is concise, coherent, and provides context and coverage. We say that a video summary exhibits coverage when it captures all the salient topics of the original video, which is useful for summarizing very long content such as a two-hour movie. However, this is generally too long for the search-and-browse cycle. While summaries can be generated manually or automatically, however we focus our discussion on technologies for automatic video abstraction.

Video summarization and browsing has been analyzed by many authors using different approaches. For example, while some authors focus on a specific granularity, i.e. summarizing a single video, others focus on a large video collection or a ranked list of videos, such as the results of a query. Some researchers further classify video summarization into two major categories: top-down and bottom-up **cite Aja and Rui's Book on video summarization**.

Dulce: I think we need to provide an example for MPEG-7, summarization seems to be the easiest of most appropriate, although I am not sure of where to introduce it. May be as bibliographical note where we mention the MPEG-7 Annotation tool that John Smith has. I think it is important

The MoCA (Movie Content Analysis) project defines a three-step procedure for creating a video abstract **dulce: fix the reference to Lienhart, Video**

Abstracdting. First, segment the video to understand the shots and scenes, to identify faces and dialog, and extra text from the titles. Second, select clips that best represent the movie. The idea here is to concentrate on the special events (explosions), primary actors, dialogs, and the title text. Third, assemble the clips by ordering them and chose the right transitions. The MoCA Project was one of the first systems to use several modalities for summarization by generating movie trailers i.e., short versions of a longer video intended to attract the viewer's attention. The authors describe their results as "similar" in quality to the hand-edited summaries.

Video Skims were introduced in the Informedia Project by Smith and Kanade [8]. Their rule-based approach combines speech recognition, image processing, and natural-language understanding techniques to detect events to include in the summary. A basic element of their interface design is the provision for alternate browsing schemes in response to the type of query, i.e. headlines, thumbnails, filmstrips, and skims. The headlines, thumbnails, and filmstrips are viewed statically while the skim is played back to communicate the content of the video. The filmstrip view reduces the need to view each video paragraph in its entirety by providing a storyboard for quick viewing.

Regarding evaluation, there is no universal definition of a set of evaluation metrics to determine the quality of a summary. In some cases the evaluation is subjective, for instance, conducting a user study to determine whether the user can successfully perform specific tasks while using a summary instead of the original video. The key point is that evaluating the quality of a summary depends on the questions you ask subjects. **Dulce: fix the references Taskiran, Amir, Ponceleon, Delp - Automated Video Summarization Using Speech Transcripts.**

Automatically generated summaries can be difficult to justify, especially for Hollywood footage. With movie budgets running millions of dollars, it is easy to justify tens of thousands of dollars for a human editor to create a trailer that gives the best reaction from the test audience. Thus video summarization is probably best applied to sporting events (generate a summary for different types of fans) and for home videos (where the budgets are lower.) Special algorithms have been developed, especially for sports video that take advantage of special characteristics of the domain. These summaries use crowd sounds to detect special events and scores, OCR to detect scores, information about a famous player, and built-in knowledge of typical angles, zooming and panning.

Summarization of home videos, summarization based on speech transcripts and SpeechSkimmer are worth mentioning

0.6.3 Static Summaries

For searching and browsing, succinct summaries are necessary because the user has to choose from many different examples. There are a wide range of approaches to video navigation, browsing and summarization. They exhibit various types of interfaces ranging from plain text to images, storyboards, sophisticated

storyboards, animations to derived video, or 3D representations. Let us first describe approaches that summarize the video in a static display—something that can be printed on paper.

The simplest summary of a document is a textual summary, i.e., its title. Analogously, the simplest visual summary of a video corresponds to a single still image, or *poster frame* **Dulce: fix the reference to Christel et al 1999-Informedia**. Next in complexity, a video is summarized with a subset of still frames called keyframes. These keyframes are automatically selected based on fixed time intervals or based on more sophisticated content-based video segmentation. These keyframes are ordered in time and arranged into a storyboard. In movie making, storyboards allow writers and directors to plan the action to be shot, describe the camera angles and provide a summary of the entire film. Traditional storyboards display thumbnails in chronological order. Storyboards vary according to the way such set of keyframes are displayed. A 1D storyboard, i.e. a row of thumbnails showing the chosen keyframes, is called a *filmstrips or keyframe lists*. A 2D-storyboard, composed of one or more pages of arrays of keyframes, is popular and is part of any decent video retrieval system.

Static summaries provide a compact alternative to a full video because they are assembled from static images. They were originally appealing because they are easily generated automatically, require no audio synchronization, and are printable. Static summaries are the first-step beyond VCR-like controls for speeding through content. Unfortunately, they are not suited for long videos, do not capture motion, and lack audio. They do not provide context, i.e. by looking at a one-page storyboard one can not determine the location of that page in the video.

Thumbnail images in traditional storyboards all have the same size. Researchers developed two-dimensional storyboards, where thumbnails have different sizes, motivated by the idea that the relative size can indicate the relative relevance of the keyframe. A Video Manga (Figure 0.23) was inspired by Mange, i.e. the Japanese word for "comic book," and represents one type of storyboard. In Manga, thumbnails of different size are packed in a visually pleasing form analogous to the style used in a comic book. Naturally, a special packing algorithm is used in order to efficiently and smoothly display thumbnails of different sizes.

Berthier: Illustrate mosaic with a figure. Dulce: we should introduce mosaic first and then PanoramaExcerpts. PanoramaExcerpts textbfDulce: fix reference goes beyond keyframes by creating a storyboard that has a combination of mosaicing and keyframes. It also requires a special packing algorithm to determine optimal layout. An image mosaic is created by overlapping similar images, perhaps by lining up the background, so a range of images capture over time is spread through space.

Salient Stills [Massey and Bender 96] allows the production of a composite image showing everything visible in an entire sequence. The algorithm computes the camera motion from frame to frame with a high degree of accuracy, running off-line. The resulting image is a composite of all the frames in the shot and most images look very different from any individual frame. The eye is drawn to

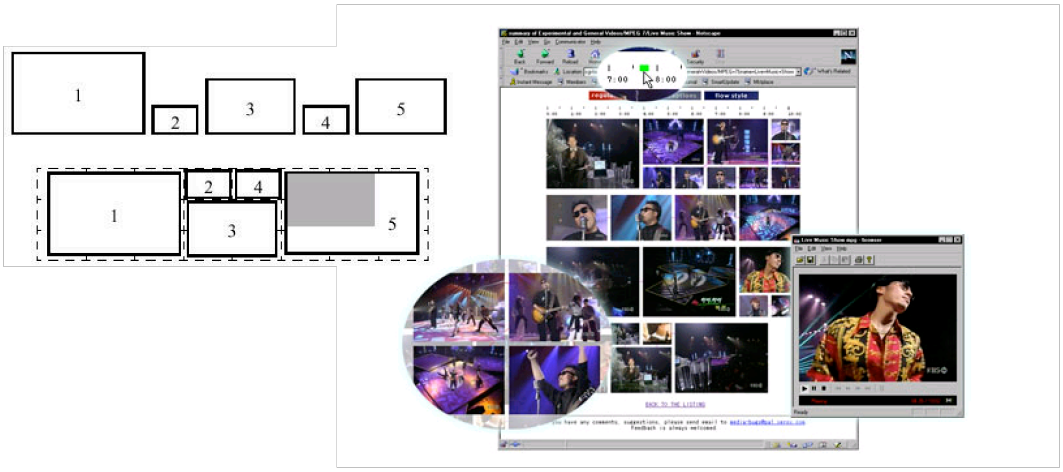

Figure 0.23 Manga is a pictorial summary of a video, named after a Japanese word for ?comic book?, where the size of the thumbnails reflect the importance of the associated key frame.

Figure 0.24 This example illustrates a storyboard that contains a mixture of mosaics and traditional keyframes using a packing algorithm to optimize the layout.

the larger images, allowing users to quickly grasp the proportions of the video devoted to each topic. But, for applications that require assessment of the exact

framing of the shot this approach is not acceptable.

Berthier: Illustrate Salient Stills and Scene Transition Graphs. Scene Transition Graphs **Dulce: fix the references to Yeo, Yeung - Classification, Simplification and Dynamic Visualization** attempt to capture the semantics of a video by displaying the underlying story structure as a graph. Think about a murder mystery. At the start it might consist of a dialog between the detective and the person that found the body. These alternating shots are a tightly coupled pair. Then the story retraces the victim's path by showing a long chain of shots. This story is shown as a graph by showing two coupled nodes, each node illustrated by the keyframe for that shot, followed by a long chain of shots to show the path of the story. This graph when rendered on a page quickly illustrates the overall structure of the story.

0.6.4 Dynamic Summaries

Computer displays allow movie summaries that change, giving designers a new way to summarize information. In addition, storyboards and slide shows are not suitable for those domains where most of the information is found in the audio track, like interviews, teleconferencing, education, and training. Instead we describe several alternatives in this section.

A *slide show* displays keyframes (in order) at a fixed rate. At fast rates they allow fast browsing of long videos overcoming the tedious task of looking at numerous large tables of keyframes. The use of a time bar and play controls is an improvement over storyboards. They are a true summary in the sense that their duration is significantly shorter than the original video and a small fraction of the size. This aspect makes them popular in scenarios where downloading speed is an issue.

A movie storyboard (MSB) is a slide show synchronized with the original audio track (same duration although not necessarily with the same image quality). At least one keyframe per shot is extracted and displayed during the entire duration of the shot. Extracting more than a single frame is advisable for long shots. MSB are specially suited for applications using low bit-rate connections.

MSBs are easy to generate automatically for most applications since lip synchronization is not an issue. For multimedia players that allow the user to manually step to the next keyframe, MSB are particularly interesting since each step advances one shot a time. In a classroom scenario, the static keyframes of the speaker's slides are shown at a much higher resolution than that of a low bitrate streaming video. Thus this approach is suited for talking heads and videos where the audio contains most of the content and where motion is not important. But this approach is not adequate to summarize high-motion videos, like a tennis match, a car chase, or anything where motion is relevant. The file size of an MSB is small than the original video but its length is determined by the length of the audio, hence they are not a summary.

0.6.5 Interactive Summaries

One of the earliest interfaces for video browsing was the Apple video magnifier [Mills et.al, 1992]. It provided a hierarchical view of an entire movie. Starting with a row of keyframes at a coarse level, every frame can be expanded into another row to provide the next level of detail. This approach is important because it allows users to drill down to see specific details of the video while still maintaining an overall view of the entire movie.

Even sophisticated storyboards do not work for both videos and collections of videos because do they feature compactness and versatility. Thus we need more sophisticated approaches to media visualization that let users understand how different types of media relate to each other.

One solution is movieDNA, a visualization for video, video collections and in general any linear data **Dulce: fix references to Ponceleon, Dieberger**. It is based on ContextLenses **Dulce: fix references to Dieberger, Russel**, a textual visualization motivated TileBars **dulce: fix reference by Hearst**. MovieDNA requires segmentation of the video either by a straight-forward approach (i.e. time-based) or by a more sophisticated content-based approach (i.e. shot-based). A movieDNA is a 2D image where each row graphically resembles a DNA fingerprint. Time (in one or more different videos) then flows down the image. Each pixel in the image says which feature is present in the video at that point in time. A feature can be the presense of a person, a topic, type of audio, or any other kind of metadata. A user can quickly see what is in the video, when it occurs, and by pointing to the segment of interest see the video. **Berthier: Figure to illustrate 0.25?**

Berthier: make sure that the text is readable, i.e. first flight, Alan Mulally, airline industry, etc., explain the seconds

Berthier: The Hierarchical movieDNA 6-hours figure is not referenced in the text, make fonts sharp and clear.

0.7 Audio and Music Retrieval

0.7.1 The Problem

Dulce: not sure what Berthier meant here

0.7.2 Audio Basics

Audio has many unique properties with respect to information retrieval. Most importantly, the information in the signal is not directly available from the audio waveform—there is a large semantic gap, just like video. Unlike video or images, information in an audio signal is not directly visible and thus one must analyze the audio signal to extract its meaning.

Audio is recorded as a waveform that measures the pressure change over time. To record this with the original fidelity and not lose any of the information

Figure 0.25 MovieDNA for a Boeing Training Video. **[Note: We might have to change this example if Boeing does not give us clearance. Malcolm, this example is just a mock-up, we can change it with another example if we want, and give meaning to the columns as we want. There is also boeing data if we want to make it a real example, but I do not think we need to because we do give the real example in the next figures**

that can be perceived by human listeners, the waveform must be measured as many as 44,100 times per second (44.1 kHz). This is a lot of data for a second of audio that consists of a single short sentence.

The sound we hear and use for information retrieval is a complicated signal. To illustrate, imagine a large lake containing a number of ships, boys throwing rocks, animals wading, and birds searching for food. ‡ Each object creates waves that propogate through the water. Connected to this lake are two small canals, and everything we know about the lake is infered by looking at the superposition of waves that enter these canals. Untangling this mish-mash of information is known as Auditory Scene Analysis (ASA) and is a hard research problem, one which is analogous to the foreground/background separation problem in computer vision.

The real world is often this complicated, but in practice we assume that our media will have only one sound source at a time. This greatly simplifies the

‡ Thanks to Al Bregman for this analogy

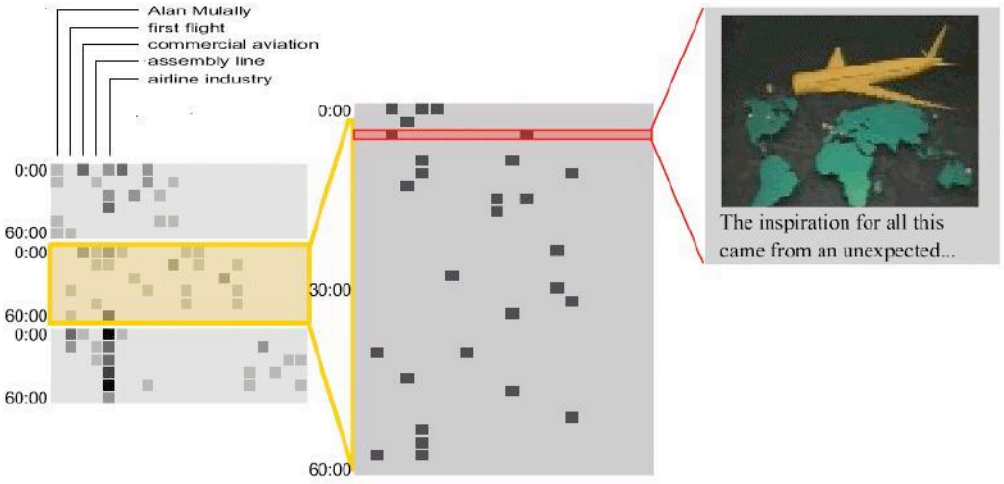

Figure 0.26 Hierachical MovieDNA.

analysis and extraction of information from auditory signals. In a sense, we work in a cartoon world where there is only one (sound) object and the background is invisible.

For information retrieval, it is important to realize that each object in a sound landscape has three primary dimensions: loudness, pitch and timbre. From an information-retrieval point of view, we can ignore the overall loudness of the signal—there is little information there. The pitch and the timbre carry different kinds of information. Pitch is defined as "that auditory attribute of sound according to which sounds can be ordered on a scale from low to high" [**?**]. It is where the musical information is, and some of the prosodic (emotional) information in speech. The pitch is often ignored in speech processing, except in tone languages where different pitch contours are used with the same mouth shapes to convey different words.

A separate dimension of sound is the timbre. It is defined as everything except for the loudness and pitch information. This includes the type of musical instrument as well as speech sounds that determine which words have been said. Thus, to understand the musical content in a signal, we often want to look at the pitch or frequencies in the sound; while to understand the words we look at the timbre. There are two different kinds of algorithms for identifying these two different kinds of information (see Section 0.7.4). But first it is instructive to visualize sound with a simple representation known as the spectrogram.

The spectrogram is a representation of sound that describes how the frequency, or spectral, content of a signal changes over time. It is analogous to a musical score, but contains more information, and represents information that can not be put on a score. Figure 0.28 shows sample spectrograms. The hori-

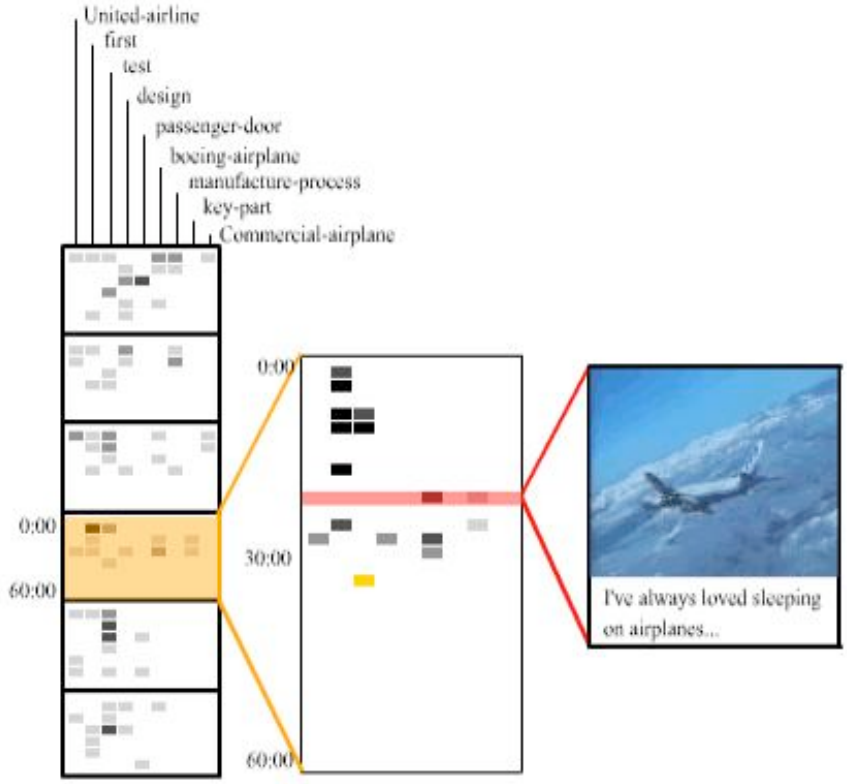

Figure 0.27 Hierarchical movieDNA for a 6-hour video collection. The columns represent automatically extracted n-grams from speech transcripts. The aggregated DNA on the left (top level) shows aggregated incidence matrices for each of the 6 videos. The full incidence matrix for each video, generated through phonetic matching of features with the speech transcript, is shown as a movieDNA on level 2.

zontal axis is time in seconds, the vertical axis is frequency, and the darkness of the image at each point represents how much energy is in the signal at that time and at that frequency.

The story in Figure 0.28 is simplest in panes a and b. The spectral content for the single-tone music is a direct representation of the information in the score. The extra lines in Figure 0.28b are overtones, or harmonics, of the fundamental frequencies. These extra harmonics are what gives sound its richness. But the story is more complicated with real sounds. Figure 0.28c shows the spectrogram of a piano recording. Each tone is accompanied by many overtones, or harmonics, that give the sound its richness. The pitch of the sound remains the same (see Section 0.7.3), but the timbre changes because a piano has many overtones. Figure 0.28d is the most realistic. Here a flute is played, and the pitch of the

notes is changed, a vibrato is added, changing the pitch periodically by a few percent, so that the sound has even more realism. The pitch is no longer precisely at the note frequency, but yet you st! ill hear an unambiguous melody. The spectrogram shows all this detail.

Now that we know how to visualize sounds, we discuss how to measure the pitch (music) and timbre (speech) information in the signal.

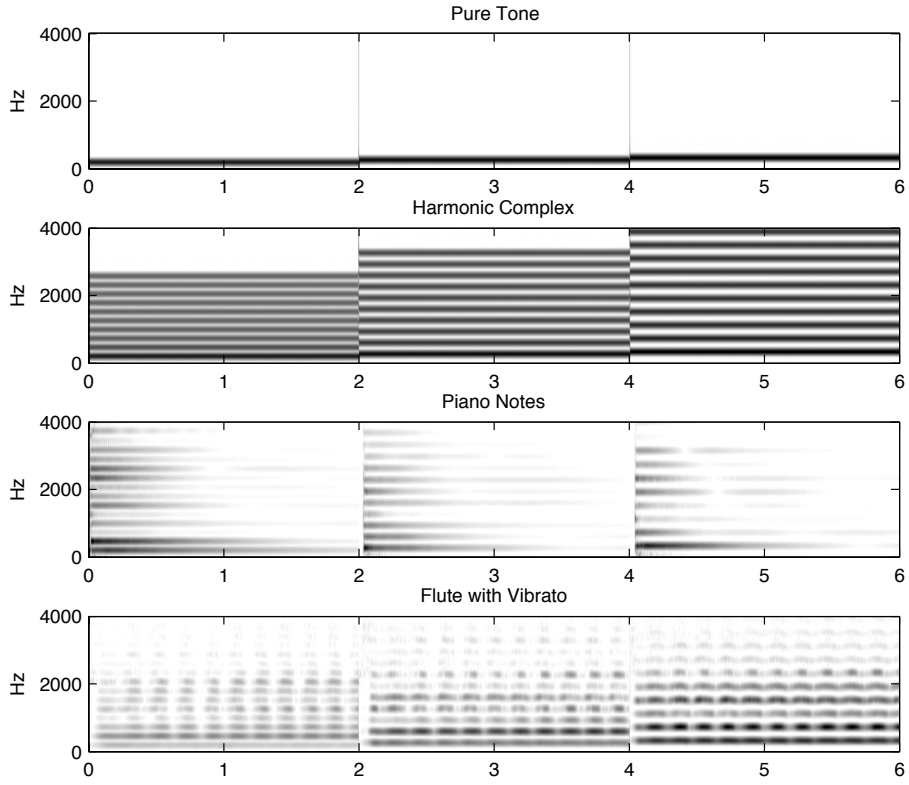

Figure 0.28 This figure shows four different (narrow-band) spectrograms for the notes C, E and G. The top image is for pure tones. The second image shows synthetic notes, each with 10 harmonics. The third image shows piano notes—note how the harmonics decay at different rates. The fourth spectrogram is for a flute, played with vibrato so the frequency of the harmonics varies systematically. Sounds courtesy of Kyogu Lee.

Berthier: add labels, a, b, c, and d to the figure.

0.7.3 Pitch and Notes

Pitch is an attribute of sound that describes the musical melody. It is used to form melodies, but the pitch is tied in a complicated way into the signal that

you hear. Pitch is defined differently by different people. Psychoacousticians define it based on what we perceive [**?**]. Speech people define it based on what the glottis in the throat is doing. Engineers define it based on the harmonicity of the signal. Here we will use the musical definition—we are most interested in what notes are played. For our purposes we can consider the pitch (or the note) to be defined as the lowest frequency in the harmonic complex.

Most musical IR depends on a representation known as the chromagram. This starts with the observation [**?**] that:

> Tone height describes the general increase in the pitch of a sound as its frequency increases. Chroma, on the other hand, is cyclic in nature with octave periodicity. Under this formulation, two tones separated by an integral number of octaves share the same value of chroma.

The chromagram is formed from the spectrogram by combining information from multiple octaves into a single 12-dimensional vector. If the base octave is from 65 to 123 Hz (C2–B3) then information from frequencies 65–131, 131–262, 262–423Hz and so on are combined to find the estimate of the 12 notes in the chromagram. This is equivalent to an octave, 12 keys, on a piano keyboard. This calculation is done 50-100 times per second, over short windows of the waveform. The resulting chromagram represents the notes (or chroma) of the music as a function of time. An example is shown in Figure XXX **Berthier: where is the figure**. The chromagram representation is most useful when we want to match melodic information in a song.

Figure 0.29 This figure shows the chromagram, as a function of time, for the 3 notes shown in Figure 0.28. Sounds courtesy of Kyogu Lee.

0.7.4 Timbre

Speech information is conveyed in the timbre of a sound. Most timbre analysis is done as part of speech recognition, and the most common representation is known as Mel-Frequency Cepstral Coefficients (MFCC) [**?**]. MFCC converts the *broad shape* of the spectrum into a low-dimensional vector. This process is illustrated in Figure **?? Berthier: fix this reference**. MFCC operates on each frame of the spectrogram, converting the detailed spectral information into a (usually) 13-dimensional vector that captures the broad shape of the spectrum. First the spectral channels are resampled and integrated to simulate a 40 cochlear filter. MFCC uses a logarithm to simulate the perception of loudness in the human ear—this is a form of loudness compression. Then a discrete-cosine transform (DCT) [**?**] is used to reduce the dimensionality. This last step is important for two reasons. First, by retaining the first 13 of the 40 DCT outputs, the

spectral information is smoothed, throwing away the pitch information that is contained in the harmonics of the sound and represented as fine horizontal lines in the spectrogram. Second, the DCT has the useful property that the output coefficients are uncorrelated. This means that simpler probabilistic models can be used—the covariance of the probability distribution is nearly diagonal, so it can be modeled with a diagonal-covariance Gaussian density—vastly simplifying any machine learning steps.

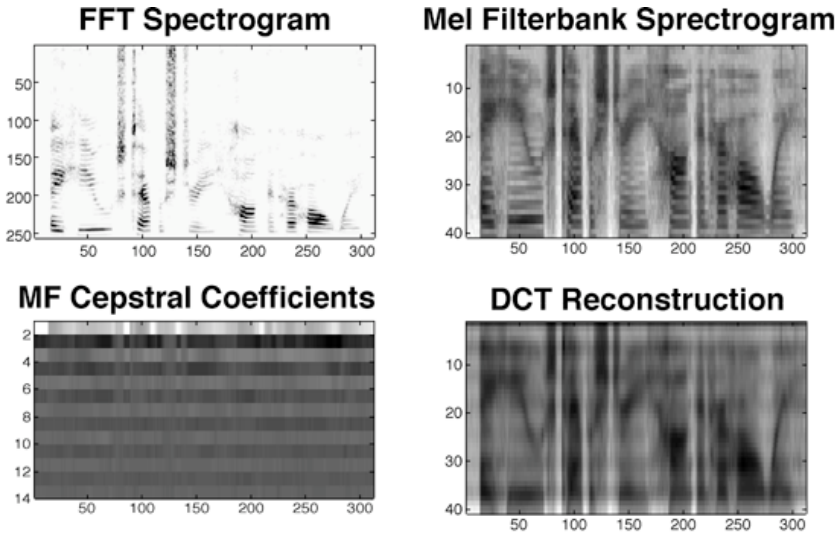

Figure 0.30 This figure shows the processing steps needed to compute the MFCC of the speech signal "a huge tapestry hung in her hallway." The upper-left panel shows the original spectrogram. The upper-right panel shows the output of the 40-channel filterbank. The lower-left panel shows the resulting cepstral coefficients. The lower-right panel shows a reconstruction of the original filterbank spectrogram from the MFCC representation in the lower-left. The reconstruction shows the information that is lost in this transformation (largely the pitch harmonics seen in the upper panels).

There are other approaches to capture the timbral information in a signal. Notably, linear-predictive coefficients [**?**] were often used in speech recognition, but the ultimate goal is to give a broad, low-dimensional representation of the overall shape. This information is of crucial importance to speech recognition, all speech-recognition systems use something akin to MFCC as an input feature. But MFCC has also been used to capture information about musical genre [**?**] and for speaker-identification (see Section 0.7.7).

0.7.5 Fingerprinting

Audio fingerprinting is a commercially successful audio IR task. In fingerprinting, we use a small snippet of sound to query a large database and look for an exact match. This process is complicated because the query is often corrupted—in a common case, a cell phone captures a snippet of sound from a noisy bar. To illustrate, assume that the user query is a portion of a commercial song, one that the user does not remember in full. The task of the IR system is to identify which of 2 million (commercially available) songs contain the query. This is called *fingerprinting*, a task that is becoming more important as we collect large databases of content in which we want to look for duplicates, or prevent the inclusion of illegal content.

The key approach is to look for peaks in the spectral-temporal distribution, the spectrogram, and encode the most salient portions of the audio signal in a way that is robust to common abuses suffered by audio signals (such as loud background noise, inexpensive microphones on a cell phone, and compression algorithms that are optimized for voice and not music.) The location of a peak is stable even when noise is added. Its location might move a bit, but it takes a large amount of noise to generate a new peak. The constellation of peaks is a fingerprint that identifies a section of the audio piece.

Given a robust representation of the sound, for example a constellation of spectral peaks, we need to quickly find the fingerprint in our database. This can be handled through either clever data structures [**?**] or via a form of hashing. Locality-sensitive hashing (LSH) is a common algorithm for finding duplicate Web pages. Normally in a hash, strings that are close together are widely distributed so that there are few hash-bucket collisions. In LSH we use a number of functions that takes a vector in a large-dimensional space and projects it to a one-dimensional line of buckets. We do this often enough, from many different directions, and then vectors that show up in the same buckets as our query vector are very close matches [**?**]. Another approach uses a neural net to learn a semi-supervised mapping that puts similar sounds in similar hash buckets [**?**].

0.7.6 Speech Recognition

Much multimedia information retrieval is based on recognizing the words contained on the audio track. But automatic speech recognition (ASR) on multimedia signals is an especially difficult task because the acoustic environment is difficult. ASR works well when two conditions are maintained: (a) the acoustic environment is constrained so the microphone only hears the single voice and there is no background noise or music and (b) the task is well defined so a language model can constrain the number of words that need to be recognized at any point in time. Unfortunately, speech from multimedia signals usually violates these conditions, whether it is media produced for entertainment or more casual recording in the home or for surveillance. These adverse conditions limit the ability of an automatic speech recognition algorithm to perform dictation-

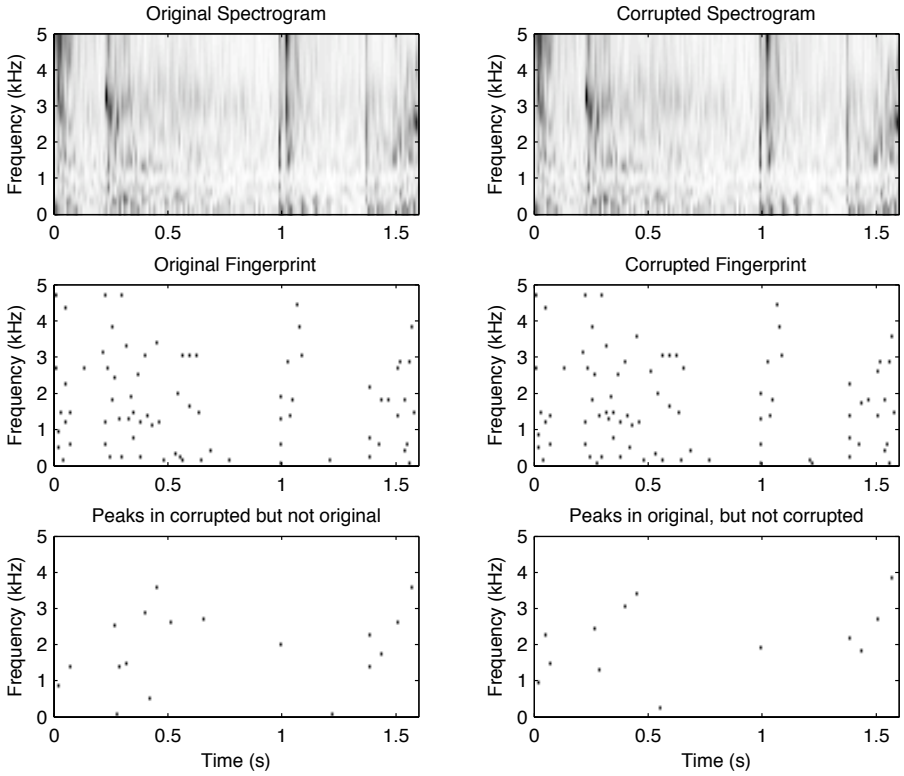

Figure 0.31 This figure demonstrates the robustness of fingerprinting using Madonna's song Borderline. The top figures shows the spectrograms (defined in Section 0.7.2) of the original and a copy of the song corrupted with noise. The middle row shows the locations of the spectral peaks. At this level of noise about 15–20% of the peaks are different between the original and the corrupted.

quality ASR. Instead, other approach must be used, such as keyword spotting and phonetic recognition. But first, let us describe the basic ASR architecture.

Speech recognition is generally performed using a hidden-Markov model (HMM) to find the sequence of word models that best explain the recorded data. This string of models is key to speech recognition and includes information about what are the legal phoneme sequences (the language model or LM) and what sounds are most likely given each phone (the acoustic model, or AM). All this information is tied together with a single probabilistic framework that trades off acoustic errors versus language errors. For each possible set of phonemes, the HMM gives an estimate of the probability that this sequence of phonemes (and thus words) sounds like what was heard. A simple HMM showing these two models is shown in Figure 0.32.

Speech is a complex and unpredictable signal. The audio signal is first

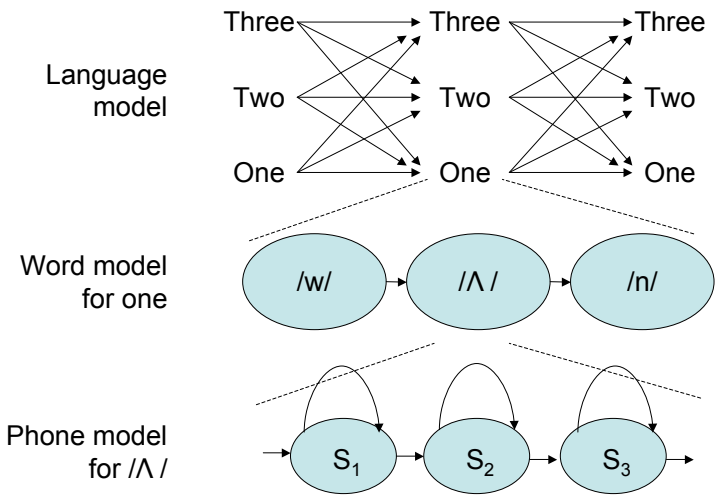

Figure 0.32 This figure shows a portion of an HMM model that recognizes a string of digits. The language model describes the valid words, the three digits in this case. Each word model describes the phonemes used to model that word. The phone model, a three-state sequence with self loops, describes how each of these monophones is pronounced. Each state of the phone model models the probability of seeing any particular Mel-Frequency Cesptral Coefficients (MFCC) vector with a Gaussian MIxture Model (GMM).

encoded as a sequence of Mel-Frequency Cesptral Coefficients (MFCC) vectors (see Section 0.7.4), sampled at 100Hz. Often the 13-D vector is further enhanced with the first and second derivatives of the MFCC representation. Each phoneme **correct spelling is phoneme** must be recognized from these 39-D vectors.

An HMM models a speech signal as a sequence of static states—the signal is assumed to be constant and when it changes the HMM moves to a new state. Each state models a portion of a speech signal with a probabilistic density function. This density function models the likelihood that each point in this (39-D) space will be heard when a person says the given phoneme. To handle the dynamics of speech, each phoneme model is often built with three to five states.

There are many ways to pronounce the phoneme /a/ in the word cat. This problem is handled in an HMM by using a Gaussian mixture model (GMM) for each phoneme. A GMM is a probability density modeled with a small number

(mixture) of Gaussian bumps in, this case, a 39-D space. The basic form of this mutidimensional Gaussian model is

What are N and T in the equation below? T should be the transpose but it has a weird font

$$G(x,\mu,\Sigma) = \frac{1}{(2\pi)^{N/2}\left|\Sigma\right|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \quad (\mathbf{0.26})$$

where now x is the vector data point, μ is the location of the Gaussian mean, and Σ is a matrix that describes the Gaussian covariance. We create a mixture of these Gaussians by adding a number of them, each component representing the probability of a different portion of the acoustic space

$$GMM(x,\{\mu\},\{\Sigma\}) = \sum_i A_i G_i(x,\mu_i,\Sigma_i) \quad (\mathbf{0.27})$$

where G_i is a single multidimensional Gaussian as in equation 0.26 and A_i is a weighting coefficient. Generally, because of the diagonal covariance of the MFCC representation (and because the alternative requires too much data) these covariance matrices are diagonal.

Further complicating matters, the /a/ in "cat" can sound very different from the /a/ in "bat." This is because the sounds before and after a phoneme change the acoustics of the intermediate phoneme. This problem is handled with context-dependent phonemes. If there is enough data, so the probabilities can be accurately estimated, more detailed models are trained so there is a different HMM model for each context the phoneme appears in. Thus, there will be a different triphone model (a phone model that is dependent on the phoneme before and the phoneme after) for the phone sequence in "cat" versus "bat."

A language model makes the speech recognition system work. The language model constrains the number of possible words, greatly reducing the chance of mistakes. At its simplest, the language model says that the only words that are allowed are the ten digits, or yes and no, or a valid business name. The complexity of a language is known as the perplexity, a measure of how many words are possible, on average, at each point in the sentence. An english phone-number recognizer has 11 words in its vocabulary (the digits 0 through 9 and "oh"). A typical large-vocabulary speech-recognition system might have a perplexity of 60. Speech recognition works as well as it does, even over lousy communication channels such as a cell phone, because the recognizer is highly constrained by the language model.

Speech recognition using HMMs works by finding a sequence of states (and thus phones and words) that best explain the available data. For example, the vowel in "one" might be mispronounced—there might be background noise—so the phone model with the highest likelihood might be the phone /i/ as in "win." But the word and language models do not allow this word so the HMM automatically chooses the best phoneme, /Λ/, that fits the constraints. This tradeoff between acoustic and language models is accomplished using a Viterbi

search **Fix this reference to Huang**.

Multimedia signals are often much less constrained; each video might be about a different topic. Thus, two different speech-specific approaches are used: keyword spotting and phonetic recognition **(footnote uses a weird symbol)** §.

Keyword spotting works because keywords are often acoustically distinct. Each word contains a lot of information, and the presence of one of these words is both easy to spot and highly informative. Users often think about their information needs in terms of keywords so there is a good match between the technology and the user. But this means that the speech-recognition developer should not think about ASR performance only in terms of the total word accuracy. Many common words are going to be ignored.

The second approach is to perform retrieval at the phoneme level. The sound of these two sentences, "it's hard to recognize speech" and "it's hard to wreck a nice beach," are very similar. Using a phonetic matching algorithm it is easier to note the similarity of two similar sounding phonemes–there isn't much in common between the letters of "recognize speech" and "wreck a nice beach" but we still want to recognize this sentence without looking.

A key to phoneme recognition is thinking about errors and confusability in terms of the underlying sounds. Using conventional IR techniques the words "bat" and "bet" are completely different. But phonetically the /a/ and the /i/ in these two words are very easy to confuse. Amir [**?**] groups the similar sounds into groups known as metaphones. Groups include vowels such as: AA, AE, AH, AO, AW, AX, AXR, AY, EH, ER, and consonants such as: TH, F. Within one group there is no penalty for a substitution.

It is important to note, however, that phonetic recognition is not a panacea. HMMs choose words based on both acoustic evidence and language constraints. By performing phonetic recognition one is throwing away the constraints provided by the language model. This is a good thing to do, when, for example, one might be looking for words that are out of vocabulary, and thus not even part of the language model.

0.7.7 Speaker identification

Speaker identification is a popular source of information in multimedia—we want to know who is speaking regardless of the words they are saying. This is useful when we are analyzing news footage so we can identify the anchors, or home videos when we want to understand who is present.

There are two common approaches: speaker-dependent speech recognition and GMMs density estimation. In an ideal world, an ASR system tuned for each possible speaker is the most accurate answer. This speaker-dependent system has unique models for the way each speaker pronounces each word. The output

§ Other techniques common in text-based IR such as query expansion (see Section **??**) are also useful

distributions for each state capture the difference in timbre between my /a/ and your /a/, and the transition probabilities capture some of the timing-dependent differences between speakers. But collecting this much data from each possible speaker is very difficult.

Instead, a more general model is used. A single (large) GMM models all the sounds produced by a speaker [**?**]. If the speaker says "y'all" a lot there will be a bump in the probability distribution around the sounds associated with "y'all". More likely, the GMM will capture the way you say /a/. Speaker identification uses a GMM, exactly like the GMM used in speech recognition (equation 0.27). But instead of perhaps 10 components in a HMM mixture, a speaker identification system might use a GMM with 2,000 components. The large number of components is necessary since the GMM is not trying to recognize individual words—all phonemes are possible at any time. Still the models are not specific, so speaker identification using GMMs often needs more than 10 seconds of speech to make a reliable decision.

Speaker identification can be used to help segment a multimedia signal based on who is talking. This is normally done using BIC as described above in Section 0.4.4.

0.7.8 Spoken Document Retrieval

Berthier says: add Spoken-Document Retrieval here, just a couple of paragraphs, define what is it. Mention it here but give reference on the bibliographic notes, instead

0.8 Multimodal Fusion: Combining it All

This section is about multimedia fusion, i.e. how to combine different kinds of information to make a better decision on the multimedia retrieval task. Multimedia differs from text because there are often different kinds of data to be mined from the same signal.

We discuss two distinct kinds of fusion: recognizing one domain based on information from the other, and using both domains for simultaneous recognition. In the first form, we solve the multimedia information-retrieval problem by building a joint probability model that fuses one kind of a multimedia signal to a textual or semantic model. This type of model is used, for example, to connect audio to text and faces to names. In a second kind of fusion model, we use the different kinds of information available in different modes of a multimedia signal to help us better understand the signal. The best example of this kind of fusion is audio–visual speech recognition—improving speech recognition by reading the lips in a video. We will treat each approach in turn.

First we illustrate techniques used by three different systems that link multimedia signals to text. In the first, faces in an image are connected to

names found in the image's caption. In the second, words are connected to an unstructured image. Finally, in the third example, a changing audio signal is mapped into the words that best describe the semantic concept.

0.8.1 Naming Faces - Fusing facial images and text

The Web, and especially news articles, are filled with pictures and their captions. These images represent a gold mine of labeled data, a natural source of correlation towards helping to identify which name goes with which face. Berg and her colleagues solve this problem in three stages [**?**]. They first use a technique based on principle-components analysis (PCA) to find the faces in the image. Second, they use simple named-entity detectors to look in the caption for proper names. At this point they have many facial images that could be labeled with any of the names listed in the captions. Finally, they cluster all the facial images that are labeled with each name, looking for a consistent set of image PCA vectors that cluster and best describe each name in all the captions. Let us first describe how to find faces.

Faces come in a multitude of styles and poses. Yet, throughout this range of images there are large common features. EigenFaces, which is based on finding an optimal subspace using principle components analysis (PCA), is an important tool in recognizing these common features in faces [**?**]. In EigenFaces all the (training) images of faces are aligned so the eyes and other features of the face are always in the same spot in a standard-sized image (say $N \times M$). The image brightness is then read out of the image, often in lexicographic order, and then is a single vector of size $NM \times 1$. Each facial image forms one point in this very high-dimensional space. Our task is to discriminate the portion of this space that corresponds to faces from that portion which does not.

PCA is used to reduce the dimensionality of this decision space to a manageable size. PCA uses a singular-value decomposition (SVD) to find the dimensions in this high-dimensional image space that correspond to faces. After arranging all the image vectors into a single matrix X, the SVD computes the matrix decomposition $X = U\Sigma V^T$. U and V are both matrices of vectors representing the row and column spaces, respectively, of the original matrix. The matrix Σ is diagonal, and its values are arranged from largest to smallest. If we take the largest k values, setting the rest to zero, then the resulting matrix $X_k = U\Sigma_k V^T$ is the best rank-k approximation to the original matrix X. This is important for EigenFaces because the top-k dimensions of the original face matrix X capture the primary dimensions in which images of faces vary. These top-k dimensions, each column reordered to put it in normal image format, are shown ! in Figure 0.33. To see if an image corresponds to a face, we project the image onto the EigenFace dimensions and measure the amount of energy in those dimensions. True faces will have a lot of energy in the EigenFace dimensions, and very little energy in the other (vector) directions. This, along with a rectification step to put each image into a canonical format, is Berg's step to fuse facial images and names.

center the figure with eigenfaces.

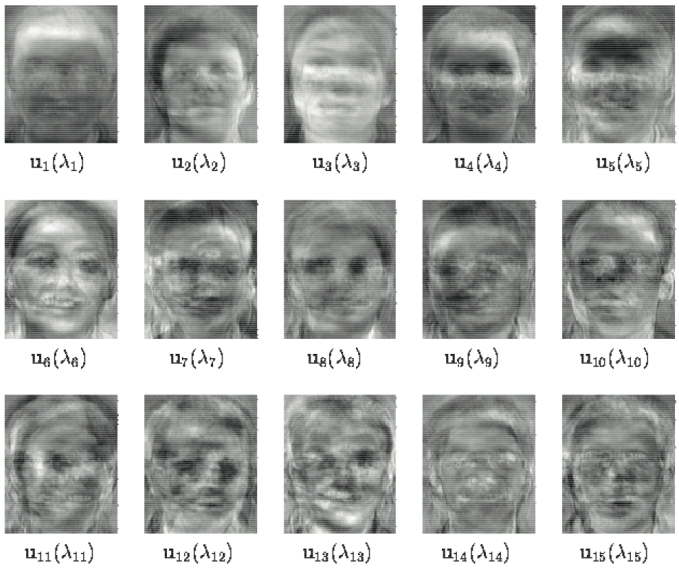

Figure 0.33 This figure shows the first 15 eigenvectors of an EigenFace analysis.

A named-entity detector, using techniques described elsewhere in this book [**REF?**], extracts common names from the captions associated with each image. Typically, there are proper names that do not correspond to a single face, such as an organization, and there are faces in the image that do not have a name listed in the caption. The task of the final step is to establish the correspondence between proper names and images.

The correspondence problem is solved using a combination of clustering and expectation–maximization. At this point we have a bunch of data that could represent "George Bush" because they are faces that appear in the caption of images that contain "George Bush," but they could be somebody else. Berg builds a probabilistic model that divides up the EigenFace space **(add reference to Berg)**. The expectation–maximization (EM) algorithm is used do this assignment. First she estimates a probabilistic model that connects EigenFace space to each potential name. Then, using either maximum-likelihood or an average estimate, each face image is assigned to a name (or null). The algorithm is repeated until the name-image assignments converge. Berg gets approximately 78% accuracy on an identification task over 1000 images on the Web.

0.8.2 Naming images - Fusing images and words

A more general approach to fusing images and words is possible using a generalized language model. Now any number of words are used to describe portions

of an image. We want to find the correspondence so we can build a model of what different objects look like. Barnard does this task by posing the problem as a machine-translation task [**?**]. Like translating from one language to another, he connects image features to words. This is done using hierarchical image clustering, and then each cluster is labeled with a set of words.

The first task when analyzing these images is to identify different regions of an image that correspond to different objects in the image—allowing to separate the tiger from the grass from the sky. One way to do this is to use normalized cuts [**?**]. In normalized cuts, a graph that connects each pixel to each other pixel is built. The weight of the edge is a function of how similar the two pixels are. More generally, it is necessary to compute a feature that describes a larger region of the image, and how spatially separated the two pixels are in the original image. The idea is to group the nodes (pixels) by making judicious cuts in the graph, each cut separating groups of nodes that are quite distinct (because their edges have high values) from each other. This problem can be formulated as an SVD. Sample results are shown in Figure 0.34.

Figure 0.34 This figure shows an image segmentation performed by normalized cuts. From left to right: original image, detected edges, five objects found by normalized cuts.

Each object in the image is modeled as a separate region according to normalized cuts by a feature vector that includes its size, its position, the color, oriented energy, and several simple shape features. The objects, using this feature vector, are then hierarchically grouped. This puts images that contain the same kind of object together. The clustering is done hierarchically because cats, for example, share many features with other animals. The connection between words and image features can be done using the EM procedure described above. In the end, ti generates a joint word–image probability model that fuses the two sources of data, words and image features, given by

$$P(w,b) = \sum_{l} P(w|l)P(b|l)P(l) \qquad (\mathbf{0.28})$$

where $P(w,b)$ is the joint probability, l is a node in the hierarchical graph, $P(w|l)$ is a model of word probabilities (i.e. a multinomial model over the words), $P(b|l)$

is the probability of image features (i.e. Gaussian mixture model with diagonal covariances over the image features for each blob), and $P(l)$ is the probability of each node in the graph. Given an image, we can query the coupled hierarchical model and estimate the words that are most likely. Or conversely, we can find the image features that best correspond to any word.

0.8.3 Naming audio — Fusing audio and words

Slaney studied an analogous approach but aimed at connecting audio and words [**?**]. This problem is simpler in some ways, since each sound file is assumed to contain just one sound and therefore no segmentation is needed. But it is also more complicated since representing the sounds, which are as long as several minutes and are constantly changing, and their descriptions, which are fragments of sentences and not keywords, is not easy. In this system, sounds from two different sound-effects libraries were linked with their textual description (e.g. "Horses: One horse approaches at walk, on rough track").

Sounds in this system are represented as points in what is called an anchor space. Sound is always changing, and it doesn't make sense to segment a sound over short periods (the silence between horse footsteps is part of the sound). Instead, each sound is represented by its distance from an ensemble of sound models. Each of these model sounds is called an anchor, and we form a vector by listing the distance from the query sound to each of the anchor models. These distances are computed using Gaussian mixture models (GMM), much like the models of a speaker in speaker-identification (see Section 0.7.7). We can build clusters in this space by noting the position of each sound (a vector of distances to anchor models) and clustering sounds that are close together in this space.

Conversely, the words in each caption are modeled using a multinomial model. We can also cluster words in this space, connecting words together that show up in more than one document. Again this is a hierarchical model.

Finally, words and sounds are connected together by building explicit links between each node in the two trees. A node in the hierarchical cluster of sounds is connected with a multinomial model to all the words that are probable with the associated sounds. A node in the semantic space is connected to a GMM that carves out a section of the acoustic space (as represented by the anchor-model distance vectors). The mathematical model for these connections is shown in Figure 0.35.

Use smaller fonts in the figure showing probability on semantic-audio retrieval

Given a new sound, we can find its distance to each anchor model, find the best-fitting node in the hierarchical audio model, and then read out the most likely words from the associated multinomial model. Or given a set of words, which form a point in the semantic space, we can find the most appropriate semantic cluster. Then, given the position of each sound in anchor space, we can test how likely the sound is given the desired portion of audio space.

Hard to follow

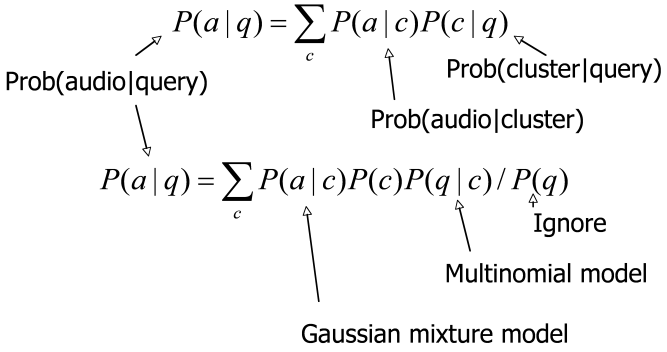

Figure 0.35 This figure details the math involved in the semantic–audio retrieval. The second equation follows from the application of Bayes Rule to the first equation.

0.8.4 Combining audio and video—AV speech recognition

A different kind of fusion combines two different types of multimedia information to produce a single semantic signal.

Audio–visual speech recognition (AVSR) combines acoustic information, much like a conventional speech recognizer, with visual information of the talking person's face to produce more accurate speech recognition results [**?**]. This is possible because many sounds, even in great acoustic conditions, are much easier to distinguish with visual information. And in noisy situations, where the acoustic features are misleading, the visual information is completely unaffected, providing a strong clue for speech recognition in what is otherwise a hopeless acoustic scene.

We represent the visual evidence with pixel-based or shape-based features. In pixel-based features, the image pixels, often transformed much like EigenFaces (Section 0.8.1), form a feature vector. Shape-based features represent a higher-order form of knowledge, based on finding the location of facial features such as lip positions and jaw outline. Either kind of feature, or a combination of them, can be used as input to a AVSR system.

Figure 0.36 shows two different approaches to the AVSR problem. In the first case, known as early fusion or feature combination, the acoustic and visual features, after appropriate normalization and resampling, are combined to produce a single feature vector for recognition. In the second case, known as late fusion or decision combination, the two streams of features are interpreted and analyzed separately, each stream providing one decision about the words present, and these are combined to form a final estimate. The acoustic features used in either form of AVSR are the same as conventional ASR and and are described in Section 0.7.6.

Berthier: center AVSR Figure

In the simplest form, the acoustic and visual features are simply concatenated and provided as input to a conventional recognizer. Usually, there are

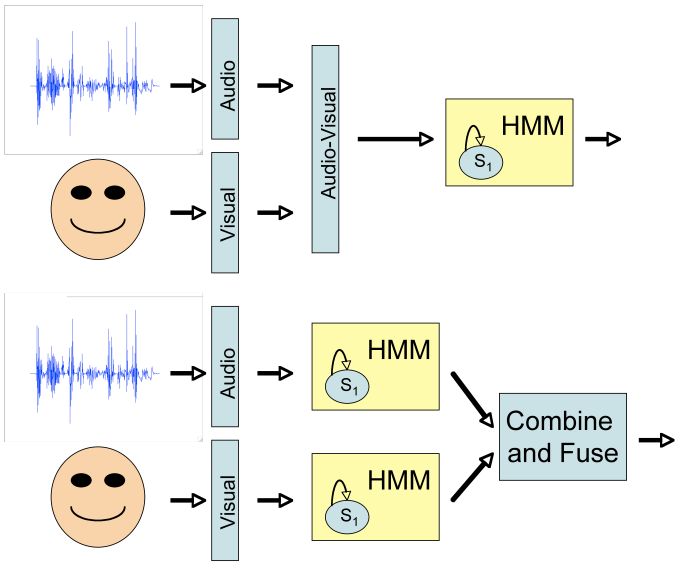

Figure 0.36 Two different approaches for audio–visual speech recognition (AVSR). Early fusion is shown on top, where the audio and visual information is combined early into a joint vector. The bottom diagram shows late fusion, where a decision is made for each modality (using separate Hidden Markov Models) and then combined.

many stages of feature manipulation to select the best dimensions [**?**] and rotate the features so each dimension is made independent [**?**]. Concatenating acoustic and visual features is known as early fusion, since the information is combined at an early stage before a decision is made.

HMMs (see Section 0.7.6) describe a signal in terms of a sequence of relatively static states, each state producing a characteristic set of output features. In the acoustic world, these states are known as phonemes and conventional speech recognizers might use various phonemes for English. The visual counterpart is a viseme—a characteristic visual pattern that represents one position of the lips—and about 13 different visemes describe the visual distinct patterns that can be seen during English speech.

In late fusion, we use separate recognizers to make decisions about the acoustic and visual information, one that recognizes phonemes from the acoustic information and the other that recognizes visemes from the lip data. The two decisions are then fused to decide which word is present.

Typical results are shown in Figure 0.37 for a range of audio, video and noise conditions (measured in terms of the signal-to-noise ratio) [**?**]. The word-error rate (WER) is relatively high when using just the visual information and is constant with respect to the level of the noise. The audio-only recognition results are good when there is no noise and quickly decline as more noise is added. Finally, the AVSR results are best, especially at high-noise levels. The

difference between the acoustic and AVSR results is the amount of noise tolerance added by the visual signal. In this case it was taken as 10 dB.

Berthier: Use a smaller front for the early vs. late and audio vs. Hybrid Fusion. Not sure why Berthier circle the vertical axis marks of 50 and 70.

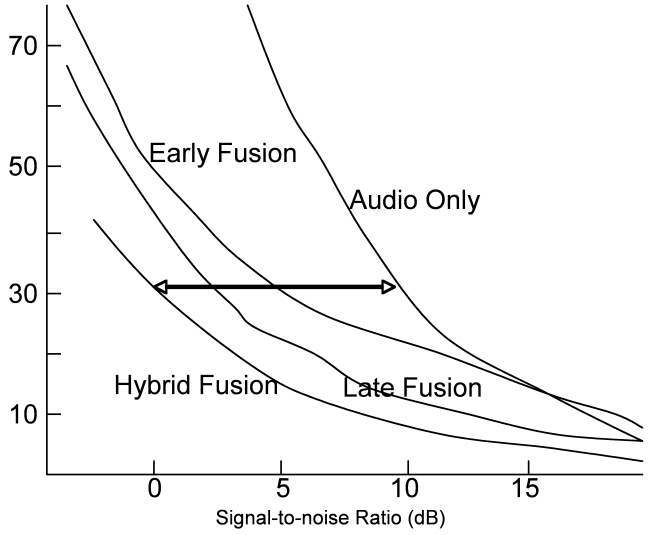

Figure 0.37 Typical results for audio and audio–visual speech recognition, showing a tolerance of 10dB in the signal-to-noise ratio (SNR), when visual information is used.

Interestingly, AVSR works better with late fusion instead of early fusion. It seems logical that a recognizer should do better with an early fusion approach. It has all the information it needs to understand the correlations and other oddities of both data. But in practice, early fusion has not worked well in AVSR compared to late fusion [**Malcolm fix this reference**]. One hypothesis is that the joint probability model, necessary when doing early fusion, is too complicated for a single recognizer to learn. This might be because the model can not capture the nuances of a joint distribution, or because there is not enough data. In either event, late fusion, or a hybrid approach that combines early and late decisions, works better.

Dulce: what is the format for capitalizing subsections?

0.8.5 Combining Audio and Video—-multimedia recognition

Finally, the more general audio–visual recognition problem can also be solved using the AVSR technology described above.

A multimedia system from IBM [**?**] labels different portions of a video with high-level concepts such as "segment contains video of person X," outdoor scenes,

fire, cityscape, and airplane. Here the visual features are motion, wavelet texture, color correlograms, and color moments. In addition, optical character recognition (OCR) is very important since many news videos have text that unambiguously identifies the subject of the video. Large-vocabulary speech recognition, perhaps with phonetic recognition, provides the transcript of the video. They use late fusion in this system.

In their tests, measuring the ability of their systems to label person X as "Madeleine Albright" the best audio-only model had a precision of 30%, the best visual model (face recognition) had a precision of 29%, and the best combined system had a precision of 47%. These numbers demonstrate the improvements possible with joint audio–visual models, but also suggest that much work remains to be done to improve the overall system performance.

0.9 Final Remarks

ACM SIGMM grand challenge is described by Rowe and Jain [XX] as "make capturing, storing , finding and using digital media an everyday occurrence in our computing environtment."

0.9.1 Open Problems

0.9.2 Challenges

0.9.3 Why is this important

0.10 Major Suggestion: New Ordering of Sections

0.10.1 Introduction

0.10.2 The Challenges

0.10.3 Content-based Image Retrieval

0.10.4 Audio and Music Retrieval

0.10.5 Retrieving, Summarization, and Browsing video

0.10.6 Fusion Models: Combining it All

0.10.7 Segmentation: Prior Processing of Multimedia Data

0.10.8 Compression and MPEG Standards

0.10.9 Final Remarks

0.10.10 Bibliographic Notes

0.11 Major Comments

Make sure we use the term multimodal since it is widely accepted. Would be nice to have a figure describing a high-level MM IR software architecture with references to each section.

0.12 Bibiliographic Notes - TO BE COMPLETED

Berthier: Fix and finish these. Add references to spoken document retrieval.

0.12.1 MPEG Standards

Berthier says: Write it down.. about the next three paragraphs. Different communities and disciplines have addressed multimedia retrieval over the years. We group them in three general categories. First, Digital Libraries and text retrieval have a long history. Specific conferences like SIGIR host research on text retrieval and contribute to text material and benchmarks, like TREC, SDR and Topic Detection. TREC is X, originally covering only text material and gradually expanding images and more recently to video retrieval.

0.12.2 Segmentation

Special algorithms have been developed for the compressed domain, although some researchers argue that the distinction between approaches for compressed and uncompressed domain is artificial [**?**].

Secondly, multimedia retrieval born in an academic environments, such as the pioneering work on Informedia at Carnegie Mellon and research conducted in Columbia University. **[Talk about related Areas signal processing, computer vision, speech recognition, pattern and image recognition, Optical Character Recognition, natural language, audio analysis?]**

Thirdly, the Web, where media search engines and new Web applications like Flickr, Google Video and Yahoo Video.

[We point out these three groups of active communities and converging in the sense that they are more aware of the interdisciplinary work being done. For example, using TREC-Video.]

The book "Video Mining" contains many useful articles on the subject of mining and understanding large collections of video [**?**]. Hauptmann [**?**] provides a comprehensive overview of the techniques, both interactive and non-interctive, that are successful in the TRECVID evaluation.

0.12.3 Taken from the MPEG-7 Section

MPEG-7 has been referred to as *the bits about the bits.*

The choice of MPEG-7 as the standard to follow MPEG-4 came about as a result of several discussions. MPEG members considered several numbers, including 8 due to its binary appeal. Finally, the lucky number 7 was chosen since the new work was so different from what has been done before.

Shot boundary detection is a mature area of research. Boreczky and Rowe 1996, Brunelli et al. 1006 and Leinhart 2001 are representative survey papers that compare many of the existing approaches. The first benchmark results were obtained in NIST TREC VIdeo Trak shot boundary detection task, in 2001 [Over and Taban 2001, Smeaton and Over 2002]

0.12.4 State-based Segmentation

All of these methods are very susceptible to the vagaries of real-life video. Camera flashes change the entire image to white for a single frame—this looks like a cut. People close to the camera might pass in front of the lens—this looks like a dissolve because the interruption is short and unfocussed. Fundamentally, cameras are noisy devices, especially at low light levels, so the image measures are not robust. There are many ways to deal with these problems and they often involve algorithms that consider longer time periods. This might be as simple as verifying that scene cuts happen over more than one frame. But more robust techniques often use a hidden Markov model (HMM) to model the likely time

course of each type of scene break. As we will discuss later (Section 0.7.6) an HMM is an excellent way to enforce a constraint on how fast the state of the video can change.

The IBM shot boundary-detection algorithm at TRECVIDEO 03 is a single-pass algorithm based on a finite state machine , processing one frame at a time of uncompressed video. It uses color histograms, localized edge intensity histograms an small grey-level thumbnails comparisons to evaluate the difference between pair of frames at 1, 3, 5, 7, 9, and 13 frames apart. The color histograms used are 512-bins in a three-dimensional (R,G, B) color space. Adaptive thresholds are calculated using rank filtering on pairs differences statistics in a windows of several frames around the processed frame. Frames at different time-differences use different thresholds. Instead of comparing only consecutive frames, or comparing frames at a fixed distance, this approach compares multiple frames at multiple distances. At each frame, a state transition is made from the current state to the next, based on the multi-frame comparison and additional rules. Sample states are In-Shot, In-Cut, Dissolve, After-Cut, Fade-Out, etc.

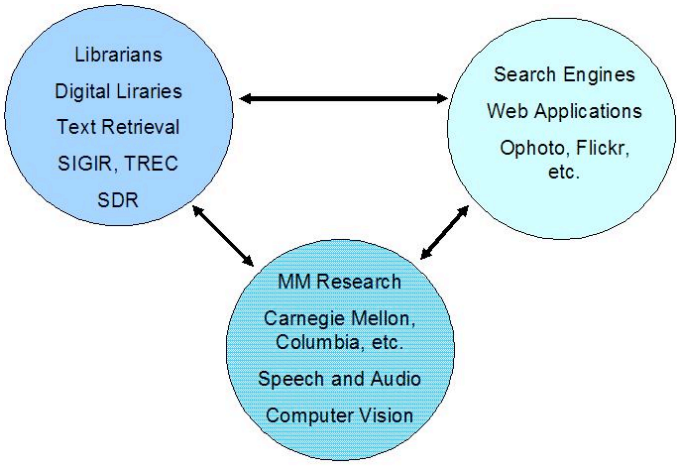

Figure 0.38 Multimedia Convergence of text retrieval, digital libraries, the Web and disciplines related to multimedia [**Change Ofoto to Snapfish.**]

References

[AGG+05] S. Axelrod, V. Goel, R. A. Gopinath, P.A. Olsen, and K. Visweswariah. Subspace constrained gaussian mixture models for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 13(6):1144 – 1160, November 2005.

[ASE03] Arnon Amir, Savitha Srinivasan, and Alon Efrat. Search the audio, browse the videoÑA generic paradigm for video collections. *EURASIP Journal on Applied Signal Processing*, 2003(2):209–222, 2003. doi:10.1155/S111086570321012X.

[BBE+07] Tamara L. Berg, Alexander C. Berg, Jaety Edwards, Michael Maire, Ryan White, Yee-Whye Teh, Erik Learned-Miller, and D.A. Forsyth. Names and faces. *Submitted for publication*, 2007.

[BC07] Shumeet Baluja and Michele Covell. Learning 'forgiving' hash functions: Algorithms and large-scale tests. In *International Joint Conference on AI*, January 2007.

[BDF+02] K. Barnard, P. Duygulu, D. Forsyth, N. de Freitas, D. Blei, and M. Jordan. Matching words and pictures. *Special Issue on Text and Images, Journal of Machine Learning Research*, 2002.

[Bli93] James F. Blinn. What's that deal with the DCT? *Computer Graphics and Applications*, 13:78–83, July 1993.

[BMM96] R. Brunelli, O. Mich, and C.M. Modena. A survey on video indexing. Technical Report 9612-06, IRST, 1996.

[BR96] J.S. Boreczky and L.A. Rowe. Comparison of video shot boundary detection techniques. In *IS&T SPIE Symposium on Electronic Imaging*, volume 2670, page 170179, San Jose, 1996.

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. Surf: Speeded up robust features. In Ales Leonardis, Horst Bischof, and Axel Pinz, editors, *ECCV (1)*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2006.

[BW01] M.A. Bartsch and G.H. Wakefield. To catch a chorus: Using chroma-based representations for audio thumbnailing. In *2001 IEEE Work-*

shop on the Applications of Signal Processing to Audio and Acoustics, pages 15–18, Octobter 2001.

[CA02] Michele Covell and Subutai Ahmad. Analysis-by-synthesis dissolve detection. In *Proceedings of IEEE International Conference on Image Processing*, Rochester, NY, September 2002.

[Can86] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[CBKH02] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. In *In International Workshop on Multimedia Signal Processing*, December 2002.

[CCS+04] Tatiana Almeida Souza Coelho, Pável Pereira Calado, Lamarque Vieira Souza, Berthier Ribeiro-Neto, and Richard Muntz. Image retrieval using multiple evidence ranking. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):408–417, 2004.

[CE01] Martin Cooke and Dan Ellis. The auditory organization of speech and other sources in listeners and computational models. *Speech Communications*, pages 141–177, 2001.

[CMM+01] Noel E. O Connor, S. Marlow, N. Murphy, A.F. Smeaton, P. Browne, S. Deasy, H. Lee, and K. McDonald. Fischlar: An on-line system for indexing and broadcasting television content. *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASP*, 3:1633 – 1636, 2001.

[CS07] Michael Casey and Malcolm Slaney. Fast recognition of remixed music audio. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2007.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[DKM+06] Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Visualizing tags over time. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 193–202, New York, NY, USA, 2006. ACM Press.

[DM80] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28:357–366, August 1980.

[FSN+95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, 1995.

[HAH00] Xuedong Huan, Alex Acero, and Hsiao-Wuen Hon. *Spoken language processing*. Prentice Hall PTR, 2000.

[HC04] Alexander G. Hauptmann and Michael G. Christel. Successful approaches in the trec video retrieval evaluations. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 668–675, New York, NY, USA, 2004. ACM Press.

[HKM+97] J. Huang, S. Kumar, M. Mitra, W. Zhu, and R. Zabih. Image indexing using color correlograms. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 762–768, 1997.

[Hou97] A. J. M Houtsma. Pitch and timbre: Definition, meaning and use. *Journal of New Music Research*, 26:104–115, 1997.

[HR05] P Howarth and S Rger. Robust texture features for still-image retrieval. *IEE Proceedings on Vision, Image and Signal*, 152(6):868–874, 2005.

[HSD73] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybertinetics*, 3(6):610–621, 1973.

[JJS93] N. Jayant, J. Johnston, and R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10), October 1993.

[Lei01] Rainer Leinhart. Reliable transition detection in videos: a survery and practitionerś guide. *International Journal of Image and Graphics*, 1(3):469486, 2001.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LP96] F. Liu and R.W. Picard. Periodicity, directionality, and randomness: Wold features for image modeling and retrieval. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(7):722–733, July 1996.

[LSDJ06] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006.

[MH91] R. Meddis and M. J. Hewitt. Virtual pitch and phase sensitivity of a computer model of the auditory periphery. i: Pitch identification, and ii: Phase sensitivity. *J. Acoust. Soc. Am.*, 89:2866 2894, 1991.

[NS04] Milind R. Naphade and John R. Smith. On the detection of semantic concepts at trecvid. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 660–667, New York, NY, USA, 2004. ACM Press.

[PNGG03] G. Potamianos, C. Neti, G. Gravier, and A. Garg. Automatic recognition of audio-visual speech: Recent progress and challenges. *Proceedings of the IEEE*, 91(9), September 2003.

[RDD03] Azriel Rosenfeld, David Doermann, and Daniel DeMenthon. *Video Mining*. Kluwer Academic Publishers, 2003.

[RQD00] D.A. Reynolds, T.F. Quatieri, and R.B. Dunn. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10(1-3):19–41, 2000.

[Sla02] M. Slaney. Mixtures of probability experts for audio retrieval and indexing. In *Proc. 2002 IEEE International Conference on Multimedia and Expo*, volume 1, page 345348, 2002.

[SM98] Malcolm Slaney and Gerald McRoberts. Babyears: A recognition system for affective vocalizations. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 985–988, 1998.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[SO02] Alan Smeaton and P. Over. The trec 2002 video track report. In *11th Text Retrieval Conference (TREC-11)*, November 2002.

[SPK01] Malcolm Slaney, Dulce Ponceleon, and James Kaufman. Multimedia edges: Finding hierarchy in all dimensions. In *Proceedings of 9th ACM International Conference*, October 2001.

[SS07] S. H. Srinivasan and Malcolm Slaney. A bipartite graph model for associating images and text. In *IJCAI-2007 Workshop on Multimodal Information Retrieval*, 2007.

[TDV00] Ba Tu Truong, Chitra Dorai, and Svetha Venkatesh. New enhancements to cut, fade, and dissolve detection processes in video segmentation. In *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, pages 219–227, New York, NY, USA, 2000. ACM Press.

[TEC01] George Tzanetakis, Georg Essl, and Perry Cook. Automatic musical genre classification of audio signals. In *"Proceedings International Symposium for Audio Information Retrieval (ISMIR)"*, Princeton, NJ, October 2001.

[TMY78] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Texture features corresponding to visual perception. *IEEE Transactions on System, Man and Cybernatic*, 6, 1978.

[Tur01] M. Turk. A random walk through eigenspace. *IEICE Transactions on Information and Systems*, Vol. E84-D(12):1586–1595, December 2001.

[vAD04] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM Press.

[vALB06] Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM Press.

[ZH00] Bowen Zhou and J. H. L. Hansen. Unsupervised audio stream segmentation and clustering via the bayesian information criterion. In *ICSLP-2000: International Conference on Spoken Language Processing*, pages 714–717, Beijing, China, October 2000.

[ZMM95] Ramin Zabih, Justin Miller, and Kevin Mai. A feature-based algorithm for detecting and classifying scene breaks. In *MULTIMEDIA '95: Proceedings of the third ACM international conference on Multimedia*, pages 189–200, New York, NY, USA, 1995. ACM Press.