MOBILE PROGRAMMING AND MULTIMEDIA SIMPLE (FOR REAL)

GABRIEL ROVESTI

1 TABLE OF CONTENTS

Disclaimer

The course is divided into *two parts*: Mobile Programming and Multimedia, hence the name. Mainly, this will be about frameworks, mobile design (first part) and multimedia formats (second part). There will be homeworks, explained better in chapters dedicated to them.

Quick explaining of exam here:

- you can do the project (also, you can do one united with WNMA course agreeing with both professors – follow my WNMA FAQ for more, on MEGA and GitHub)
- the project has not any requirement upfront: you decide, discuss with the professor and do your own thing as you please, so do not wait more than necessary
- alternatively, you can do a presentation about topics selected 2 days before the date
- homeworks will make you skip the oral questions if you completed them

Read the Moodle (and instructions I saved on MEGA for you) for more information.

This course is once again the typical one: interesting for the content, not so much in the final delivery, at least in my opinion. The professor is really good and competent, its contents are interesting, the course is quite interactive. But with one complete notes file, you are definitely organized upfront.

The following topics will not be treated (even though they are present in the Moodle automatically, hurray, just to give more confusion – even present in other notes to make you confuse even more):

- PhoneGap/Cordova
- Corona/Solar 2D
- Xamarin
- iOS Platform
- Android Platform
- Wearable Devices

About the seminars, only the Android one was made – as reference, I left it at the end of this file, but do not waste your time, while the iOS one was not made (anyway, same considerations hold).

The course contents are presented chronologically, so you can easily understand just by opening this file everything about timing and content (important, I know! – given the Moodle courses of this one are closed after a while every year – poor me, I like people being organized upfront!). Here you can find a complete explanation of homeworks, advice, tips, tricks and their solutions. So, a useful file for everything, basically what notes *should always be*.

Hope this can be useful. Feel free to reach me to give some feedback about the content. Also to thank me, it does not kill me that much.

2 Course Introduction

The course is based on two parts:

- *Mobile Programming*
    - this is the section that will change, given the nature of it
    - this year we will get to the detail of design of applications
- *Multimedia and Data*
    - this remains pretty much the same
    - this part is focused on understanding how to encode data
        - media can be particularly heavy on memory/battery/bandwidth usage
        - we want to save time in downloading and streaming the files
        - this of course because usage is very much mobile and not desktop anymore

Other general info:

- Slides will be gradually updated according to the need on the Moodle, which is free to access
    - There will be 4x4 (4 slides per page) and normal slides (1 per page)
- After 15 days, data will be collected in order to get account data of labs
- In case of need, an Apple PC is available in the library to be used to develop applications
- Recordings are available but only if attendance doesn't go down

Quality of the application is measuring "how much it does what it's supposed to do". While in Web is important the number of clicks to get to the data, here it's important how much data is asked to the user.

When creating an application, it's important also to drive metaphors towards a common goal, giving the everyday use inside what is done inside of a product/application. Consider the desktop, which seems not a metaphor, but it is, actually.

When we are developing an application, consider this is done for smartphone, which do not have OS, but RTOS (Real-Time Operating System): for example, when we receive phone calls, an applications stops, has to save save state, then we get back to what we were doing before.

The following is an example of compression; the difference is present but not very much:

To understand this, we will understand what we can or cannot perceive in light and image. This way, we can remove all the information we can't see, without losing quality. It's more important to know what the formulas do, instead of understanding for real the elaborations.

Representation and encoding revolves around:

- How human sight works
- Image properties: size, quality, transmission, visualization
- File formats
    - GIF
    - PNG
    - JPEG
    - JPEG2000
    - others

Also, it's important to understand how *sound* is made:

- Audio properties: fidelity, ì transmission, playback
- Standard file formats:
    - WAV
    - MP3
    - others
- MIDI
- Compression (lossy – with loss of some data)
    - understanding what to perceive and what not

But also images with audio - *video*:

- Human vision with motion pictures
- Digital and analogic video
- Video properties: quality, representation, transmission
- Standard formats:
    - H261
    - H263
    - MPEG family
    - DivX, Xvid

We also talk about *data compression*, in particular:

- Reasons behind data compression
    - Storage space, transmission time
- Continuous and not-continuous media compression
- Lossless and lossy compression
- Lossless encoding
    - Entropy encoding methods
    - Semantic compression
- Lossy encoding
    - Image compression: JPEG
    - Video compression: MPEG1-2
    - Audio compression: MP3

Prof. says slides are not enough for the examination; Moodle material is suggested and also textbooks are needed here.

Written by Gabriel R.

Fragmentation is present between platforms (Apple vs Android and all of its versions) and different values of settings according to devices and specific needs (e.g., brightness, virtual environment, singular devices with particular sensors/features, etc.).

This is the knowledge and skills targeted from the course:

- Mobile interface design
- Cross-platform development
- Emotional design
    - transform the user into a customer
- Wearable devices
- Market

The examination will require:

- A presentation of an argument with slides plus an oral examination

Or:

- A small group project with a final report plus a small oral examination
    - Develop an application
        - It can be however you want, there is not strict requirement
        - Remember you can do both WNMA and this project
            - This can be done talking with both professors
    - It should be made for all platforms (iOS and Android)
    - But it's not mandatory, can be only for one specific system
    - Should be sent 5 days before the examination
    - Subscribed on Uniweb and poll of what examination was chosen
- In-depth analysis of an argument with a presentation and a small oral examination
    - 10-page essay presented on the end of the course
    - The argument must be defined by the end of April
    - Possible dates: 4th – 7th June 2024
    - It's very important to choose topics and prepare/explain material seen during lectures

In any case, oral examination is made on two questions about all the class program and material. One can avoid these two oral questions if one attends in presence only the homeworks (exercises). These can be something like "solve a multimedia algorithm problem" or "understanding design flaws of something and explain it to the class". If all are delivered (or at most you miss one – but at least half or more of the half), then this is considered.

Exercises will be evaluated anyway even if wrong and can be given ½ points, then asked in oral examination, when given feedback of course. It seems they will be on Tuesday afternoon.

So, for you lazy people:

- you can do the project (also, you can do one united with WNMA course agreeing with both professors – follow my WNMA FAQ for more, on MEGA and GitHub)

- alternatively, you can do a presentation about topics selected 2 days before the date

- homeworks will make you skip the oral questions if you completed them (or did most of them)

3 0 - Introduction to Mobile Development

To create the right product, between web and mobile, we have to *study the user and understand his necessities and needs*. As we all know by now, smartphone market has been exploding since years and more and more users are active using a mobile device.

Some data we can give about them:

- Over 5 billion people are using smartphones, with 4/5 billions being active social media users/Internet users respectively
- There are at least 7 billion of mobile subscriptions worldwide
- China, India, and the United States are the countries with the highest number of smartphone users, but also Indonesia, Brazil, South America and Africa
- While desktop is mostly used inside USA, South America, Oceania
  - connections can be wireless easily not needing an infrastructure
- Operating systems are Android and iOS
  - who wants to create premium services or something that people will buy, the preferred choice is actually iOS
  - on Android we have a lot of fragmentation between features and various things
- Worldwide users are on smartphones and usage data is collected from developers themselves
  - this does not mean we have to forget desktop however
  - consider mobile devices are also tablets, not only smartphones
    - they weigh more, they cost more, we use them with two hands, and we sit down using them concentrating using it
    - this way the application must come with some way to handle the error situation, recovering from them
- There are differences between males and females
  - females use more mobile apps than desktop
- Smartphones beat TV for younger users (the younger, the more usage)
  - less gestures required, less fatigue, more content present
- There is a relationship daytime-device
  - low-to-middle use between morning and daytime for mobile
  - daytime to early evening for desktop
  - in evenings for tablets
- On average there are more than 2000 interactions with a smartphone on a day
  - consider the user can make a lot of errors because there are a lot of interruptions
    - good quality means good experience, even when errors happen
- Today we have smart* (smartphones, smart watches, smart homes, etc.)

Mobile phones are not considered anymore as a simple device to make calls but incorporate a lot of different features. All of these ones are provided by apps, in whatever form.

- Messages, calls
- Internet navigation
- Sensor data collection and usage (app for training, biking, running, etc.)
- Agenda
- Entertainment (games, music, video, reading, etc.)

There are different *false myths*:

- Mobile app development is not expensive
  - A bad app is worse than no app
- Mobile app development is easy

On the contrary:

- Mobile app development requires big teams
- Mobile app development is <u>not</u> like winning at the lottery

The first step to determine if it is better to develop a mobile layout of your own website or a mobile application is to understand the differences between the two:

- Diversified content → content will be personalized remembering user preferences
- Native interface vs. Company brand → more opportunities to better meet user needs
- Development time → depends on the needed for the goals to achieve
- User interaction (ex: push notification)
  - Using gestures instead of point and click
  - User experience improves
- Access (icon) vs bookmark → the icon remembers you to use the app
- Target (loyalty vs. reach) → ease of access/personalized experiences/convenience

The only data we need from users comes from payment information, all the rest is needed *because the developer asks for it*. The website is the best way to get information in a quick way, apart from push notifications. Also, icons are suitable to do that the best way, because it *remembers the user what to do*.

Remember also *mobile e-commerce* is going strong and has a greater market share each time.

- Usually, transactions are simultaneous and can happen on multiple devices at a time and also multiple apps at the same time, outperforming with apps mobile browsers
  - bringing an environment together (without having to put all data again)
- Committed retailers capture more transactions on mobile apps rather than browsers
- In both platforms, iOS devices capture the majority of these retailers' transactions
  - they will definitely spend more money

Other general statistics:

- There are millions of apps in the stores and a 25% of them are used only once
- A user usually spend 90% of the smartphone time using apps
- The 84% of the time is spent using 5 apps that change between users
  - which include, in this order, social apps, games, music, and video streaming
- It does not matter the number of downloads in the end, but the number of installations
- Study the user remember: screen time depends on different factors but also context

There is the *app vs mobile web*:

- A mobile application usually tends to encourage brand fidelity (icon on the desktop, notifications, etc.)
- A website with a mobile layout allows reaching the user in every situation, immediately

There are situations when it’s useful to create an app:

- A lot of graphics or calculations
    - Leverage device resources for better user experience
- Camera, sensors, or microphone usage
    - Audio/image processing, sensor data analysis, not feasible on web
- Gallery or contacts' access
    - Seamless user interaction
- Push notifications or background service
    - Data synchronization, location tracking, content downloading and convenience
- For games
- It is the only way to have access to the store → maximizing reach and revenue

The number of installed apps changes depending on the device (and its operating system). According to Nielsen, the best approach is to interview the users to understand if they would accept to install the new app on their device:

- Storage space
- Purpose
- Loyalty

The development of a mobile app requires several *resources*:

- Interface design
- Development
    - E.g., Which operating system?
- Maintenance

There are different *advantages in web apps*:

- They require a very low knowledge base, HTML is popular
- HTML5 now provides access to almost all smartphone features
- More straightforward “conversion” to different operating systems
- User *does not have to worry about the update* of the application
- It is not necessary to wait for application approval
    - Apple can require more than 2 weeks

Applications for mobile devices are different from desktop applications:

- Mobile operating systems are soft real-time operating systems:
    - An application can be suspended or terminated in every moment
    - The operating system manages context switch
- Only one application active
    - Not with iOS on iPad
- Limited space, it is not possible to open more windows at the same time
- Easy to install (or at least discourage less the use)
- Incredibly high number
    - To design and create an exciting app is extremely challenging
- Market fragmentation

Bibliography of this chapter: here and here.

Written by Gabriel R.

4 1 - Frameworks for Cross-platform Development

Once upon a time, whenever there was an idea for a new app, the main goal was to develop it for iOS.

- On iOS there are users that spend the most
- On Android there is the highest diffusion of users
    - Before 2010, there was still choice in OS other than these two
- To have a lot of income, it's important to develop for both platforms
- In Android, there is a lot of fragmentation between devices
    - layout needs to be flexible and suitable for many of them
    - so many different manufacturers
- iOS overall is pretty much well-updated and organized
- For different OSes different languages are needed
- It is necessary to develop different apps (all the same) for several devices
    - Creating one for each operating system *by hand* is quite expensive

There are *different variables* to consider, which are independent between devices:

- Operating system
- Programming language
- Development tools (IDE, simulators, etc.)
- API
- Sensors/equipment
- Screen size
- Computational capacity

The goal is: *develop once, adapt for all*.

- Do not follow the principle of "code forking" (e.g., if iOS do this, if Android do that)
- Cross-platform frameworks for mobile development reduce market fragmentation
    - Allows to reduce negative effects
    - "Write *once*, distribute *everywhere*"

There are different *main features*:

- Application developed on time, using only one programming language
    - or a set of languages → with one language/one environment
- The chosen framework allows the distribution of the application in several applications stores
    - so, there are several applications deployed
- The frameworks usually provide support for native API

There are so many frameworks one can choose from: jQuery Mobile, jQTouch, Sencha, Sproutcore, xui, appcelerator, PhoneGap, appMobi, QuickConnectFamily, Worklight, netbiscuits, dragonRAD, pyxismobile, kony, MoSync, bedrock, LiveCode, Unity, Unreal, Adobe.... the list will go on if you will.

We move the problem from *choosing the right platform to choose the right framework*.

- Most applications are developed with frameworks, not natively
    - because of versatility and convenience) – e.g., Uber, Pinterest, etc.
- It they are developed natively, this happens because of performances reasons

- This is incredibly time-saving and cost effective for a developer
    - it needs to create effectively just one application
    - instead of one dedicated to each and every platform

Other data:

- There are billions of dollars achievable with cross-platform tools
- Different devices spread across all countries and nations
    - The most between North America, Europe, Pacific Asia, Central/South America
- The most known frameworks for this specific development paradigm are Flutter (supported by Google), React Native (supported by Meta) and Xamarin (supported by Microsoft)
    - other ones just to quote: Ionic, Corona, Sencha, Unity, etc.)

There are different *pros and cons to cross-platform development*:

- Pros
    - Wide market reach
    - Single codebase
    - Faster and cheaper deployment
    - Reduced workload
    - Platform consistency
- Cons
    - Possibly slower performance
    - UX and UI discrepancies (create widgets suitable for the platform)

There are also *pros and cons for native development*:

- Pros
    - Usually, a native application offers a better user experience, a faster and more high-performance interaction
    - Non-native applications are limited by the expressivity of the used framework
        - E.g., available APIs
- Cons
    - Fragmentation = higher development costs
    - Problems with test
    - An Apple computer is always needed

Good question: *How to choose the best framework?* App development involves 4 steps:

1. Idea analysis
2. Interface design
3. App development
4. Store deployment

Consider in particular:

- Store deployment is necessary every time there is an update and for each platform
- Native development requires repeating steps 2-3-4 for each platform

Class discussion with Wooclap: *What are the features that influence the choice of a cross-platform mobile development?*

- Documentation and good references
- Third party support
- Development cycle time and maintenance
- Cost of framework and license prices
- Energy consumption requirements
- Good learning curve for the language
- Support for native look and feel
- Community and good support
- Compatibility between different platforms

4.1 RAJ AND TOLETY CLASSIFICATION

Frameworks' classification is still an open problem. Raj and Tolety classification (used in research and companies) define 4 different classes (paper is inside the *fundamental* material – aka need to study/read it, here):

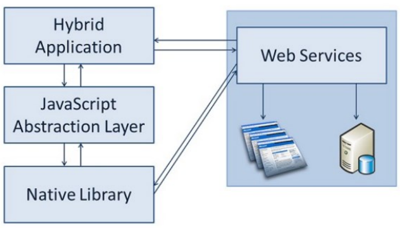
- Web Approach
  - *General features*
    - This is not a mobile app, instead it's a web application accessible via URL
    - Mainly developed using HTML, CSS, JS
    - It executes a web service to obtain a native version of an app
    - It does not require an installation, easier to update without manual intervention

  - Pros
    - Same interface (but not same experience) on all devices
    - No installation necessary
    - Easy update and maintenance-free
  - Cons
    - No store publishing
    - Network connection necessary
    - Difficult test
      - Cannot access mobile device hardware and software
      - Difficult to support different screen resolutions this way
    - Strongly connected to HTML5 support of the device
      - Widgets with native look and feel for applications
      - Less control over content rendering
      - Limited to leverage the gestures offered by the platforms
    - Non-native interfaces bring to low usability
    - More difficult to monetize

- This is the concept of Progressive Web App (PWA), which are web pages that behave like native applications
    - The term was coined by Steve Jobs in 2007, since apps using new functionalities like service workers and web app manifests needed to be categorized
- In particular:
    - They are developed using web technologies, therefore HTML5, CSS3, JavaScript
    - It works independently from the browser
        - using *progressive enhancement* (according to the device equipment)
        - the more features the browser provides, the more features provides the app
    - It works even offline, but with limited support
    - Can be installed without using the store (but in this case, they are a sort of link)
    - Like every web page, these apps adapt themselves to device size (responsive)
    - Secure (HTTPS) and indexed by search engines
    - Easy to update
    - Support push notifications
    - No need for stores to publish the app
        - but there is no payments management
        - and there is no control of what is published

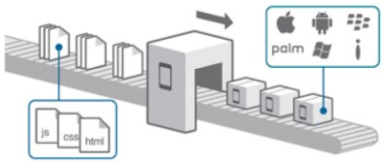
- Examples of PWAs: Sencha Touch, Angular, React (note: not React Native), jQuery Mobile

- <u>Hybrid Approach</u>



    - This is developed using web technologies and gets executed inside native container on the mobile device
    - Uses the browser engine of device to render/display HTML content full screen
    - Separate in layers
        - Hybrid application
        - JavaScript Abstraction Layer
            - Allows to expose device capabilities
        - Native Library
        - Web Services
    - Pros:
        - Store publishing available
        - Reusable UI
        - Usage of device components and powered by device computing capabilities
        - Usable for both server backend and standalone applications
    - Cons:
        - Need to be installed on devices
        - Lower performances compared to native apps
        - Has cross-communication vulnerabilities because of JS
        - UI do not follow native Look and Feel hence styling is required

- Examples: PhoneGap/Cordova
  - The project started in 2008 trying to solve these problems
    - Development of mobile applications using web technologies
    - Solve the problem of low support of mobile browsers to HTML5
    - Allow access to unique features of the device
  - Actual support to HTML5 of the mobile browsers and HTML5 evolution has partially solved these problems
  - In 2011 PhoneGap code was offered to Apache to continue the development
  - Apache Cordova is the engine below PhoneGap
    - like WebKit is the engine of several browsers

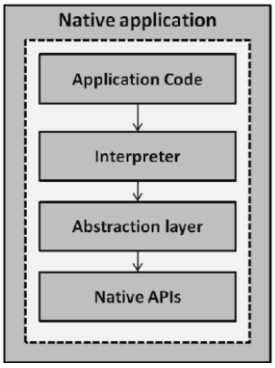


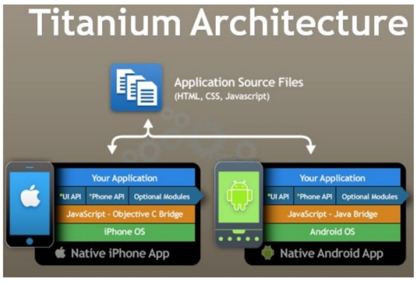
- Apache Cordova is a hybrid framework
  - Applications development works with HTML, CSS and JS, well known to web devs
  - It uses plugins to access hardware components of the smartphone (camera, GPS, etc.)
  - It provides tools for testing (emulators) and deployment of the final application
  - Figure on the right shows an example of *HelloWorld* in this framework

```
<body>
  <div class="app">
    <h1>PhoneGap</h1>
    <div id="deviceready" class="blink">
      <p class="event listening">Connecting to Device</p>
      <p class="event received">Device is Ready</p>
    </div>
  </div>
  <script type="text/javascript" src="cordova.js"></script>
  <script type="text/javascript" src="js/index.js"></script>
  <script type="text/javascript">
    app.initialize();
  </script>
</body></html>
```

- Other frameworks/tools allow app development using Cordova:
  - Monaca – Framework7 – NativeScript – Ionic Capacitor – Progressive Web Apps
  - Really easy to use for web developers to develop mobile applications
  - Quite easy to customize interface using CSS
- Cordova usually is not used stand-alone
  - but as a support framework for other frameworks

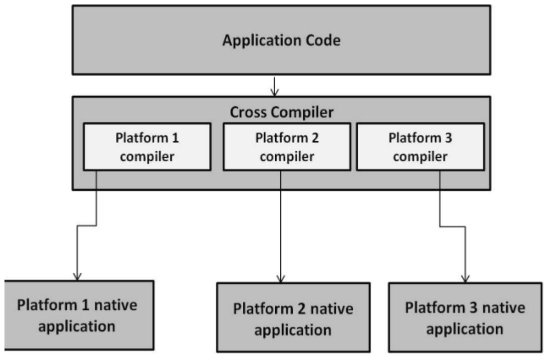
- <u>Interpreted Approach</u>
  - Application code is deployed to the mobile device and gets interpreted thereafter
  - Pros:
    - Native Look and Feel
    - Store publishing
    - APIs for device components
    - Close to native approach without coding in native language
  - Cons:
    - Really difficult to reuse the UI
    - Available features depend on the framework – need code forking here
    - The interpreter can have low performances with respect to native/cross-compiled
      - Because the application needs an interpreter
        - And we also need its code





- Examples
    - Titanium Architecture (now called Appcelerator Titanium)
    - C# runtime interpreter

- <u>Cross-compiled Approach</u>
    - The best one from the user point of view
    - Finished running the application, the compiler translates it into the native language
    - It strictly depends on the compiler and there can be errors when translating the app
    - Best for user performances



    - Pros:
        - Allows to use all the components available from native app
        - Reuse of the existing source code
            - by cross-compilation to another application run on different platforms
        - Native interface
        - Good performances
        - Store publishing
    - Cons:
        - Not reusable UI
            - specific to the platform
        - Overly complex apps can have problems during the building process
        - Mapping between source language and target language is difficult
            - so the cross-compiler supports a few platforms
            - and focuses on the common elements of those
        - Identifying/correcting cross compilation issues might be hard

- Examples:
    - Solar 2D
    - React Native → learning curve it's really important
        - because it's a branch from React, so it's quite large
        - but also use common tools like HTML, JS, CSS
    - Flutter
    - Particular case: Xamarin → interpreted for Apple/cross-compiled for Android

(For your info: course started at the end of February, second week of lessons, prof. says she will release one of the homework assignments next week. I like people being organized, so I tell you straight away).

In summary:

Approach	Programming language	Supported platforms	Pros	Cons	Example
Web	HTML, CSS, Javascript	Android, iOS, Windows, BlackBerry[a]	– Easy to update – No installation – Same UI over different devices	– No access to application store – Network delays – Expensive testing	jQuery mobile, Sencha Touch
Hybrid	HTML, CSS, Javascript	Android, iOS, Windows, Blackberry,	– Access to application store[b] – Support to most smartphone hardware	– No native UI – No native look and feel	PhoneGap
Interpreted	Javascript	Android, iOS, Blackberry	– Access to application store – Native look and feel	– Platform branching – Interpretation step	Titanium
Cross-Compiled	C#, C++, Javascript	Android, iOS, Symbian	– Native UI – Real native application	– UI non reusable – High code conversion complexity for complex applications	Mono, MoSync

[a] Support depends on the browser chosen by the user.
[b] Apple store usually tends to refuse applications developed with this approach [34].

4.2 El-Kassan Classification

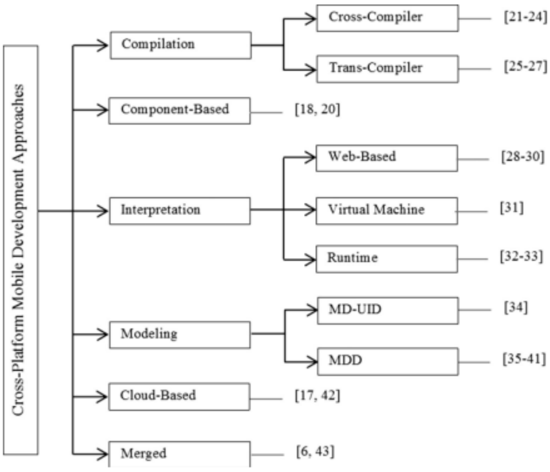
There are several methods for frameworks classification, some of them based on the development approach, others based on the result. El-Kassan et al. divide the apps into three categories (reference of this one here):

- Native app
  - Developed using native languages (for the final output at least)
  - they use the tools and programming languages provided for a certain mobile platform
  - What is used in between it's not important
  - Native look and feel and full access to devices features

- Web app
  - Website used through a mobile phone
  - No new technologies required in learning
  - Same for all mobile devices

- Hybrid app
  - Everything that falls in between
  - Developed using the web technologies like the web app
    - but it is rendered inside the native app using a web view control
  - Has the native app running but also the web engine (e.g., Webkit)
    - because has also other components running
    - other requirements may be precision with the user, interaction with the user, secondary factors, etc.
  - Device capabilities are exposed to the hybrid App through an abstraction layer
    - (JavaScript APIs)
  - The app can be downloaded from the store, size is small and it's on server
  - UI can be reused across different platforms, so lacks native look and feel

These are not the only ones however; consider the following figure from the paper above, detailing the other classifications:



Example:

- The game teaching fractions with pizza slices shown in the lesson using Solar2D
    - you don't need to know anything about the platform
    - but only know details on how the signing process work

In the compilation process, define the following ones:

- <u>Cross-Compiler</u>
    - Runs on one platform (host) but generates code for a different platform (target)
    - Used when the target architecture differs from the host architecture
    - Requires a toolchain configured for the target platform
    - Optimizes code for the target platform's hardware and operating system
    - Produces native apps and code can be reused over different platforms
        - keeping an eye on a few because mapping precisely is hard

- <u>Trans-Compiler</u>
    - Converts source code from one programming language to another
    - Often used for translating code between high-level languages or dialects
    - Doesn't necessarily involve different platforms or architectures
    - Focuses on language semantics and syntax translation
        - rather than platform compatibility
    - Mostly used to reuse legacy applications code natively

We can consider also:

- Component based approach
    - App development starts from different components that communicate together using interfaces, with each one having different functions
    - Each component has the same interface in every platform, but different implementations, so no need to know about them precisely
    - Usually, the output is a native application
- Pros:
    - It simplifies the development
        - assuming that there are several *off-the-shelf* components available
    - E.g., platformer games or games like Doodle Jump

- Modeling approach
    - With this approach the developer uses several abstract models to describe user interface and functions of the application
        - UI can be generated at execution/development time
        - Used for prototyping to evaluate apps usability in many devices/platforms
        - Helps focus on app functions rather than technical issues
    - Models are then translated into native source code

- Cloud-based approach
    - With this approach, all the app computations/app processing are done in the *cloud*
        - and the application receives user interaction, sends them to the cloud, and shows the result of the elaboration
    - Pros & Cons
        - Continuous network usage (high-speed network)
        - Lot of efforts in development
        - No need for specific hardware
- Examples: Stadia
    - sending logic to the server, creates the logic and sends back the streaming to the client
    - like YouTube, but not with limited commands, but the whole game itself

Again: *how to choose the right framework?* This is still an open research problem, but some steps have been taken. Compare the different cross-platform frameworks:

- *Idea*: write one application using different frameworks, deploy on 2 target platforms, and compare results
- 4 different approaches were considered: *Web, Hybrid, Interpreted*, and *Cross Compiled*
- In this case, four frameworks were considered:
    - jQuery Mobile
    - Titanium Appcelerator
    - PhoneGap
    - MoSync
- We have to understand how to ask questions to users or to different people each time
    - Research provides “standardized” data, we would need real data and experiences,
    - This also consider developers feelings and habits

An application like the following figure with a lot of interactions and graphics are Corona and Unity, but here was asked to the developers which other ones to use instead of those,

- This was used to help children with dyslexia
- Using the touch, the keyboard is used to select the correct letter – interaction has to be fast and in position
- Two situations
  - obstacles and letters with treasure to be found
  - fishes, background, animations changing
- When the correct path is selected, speed/difficult is increased
- In case of errors, speed/complexity decreases

The case study revolved in using 4 frameworks of different categories, then asked to evaluate the framework.

- Is this a good idea or a good result describing the judgement impartially?
- There are human factors to consider here: human psyche
- 4 times means more difficulty to make apps/algorithms each time differently
  - even for the same problem
  - more resources, more time
- The development here was to develop the app once with a framework and then be evaluated for that
- The good result would be with at least a hundred of users → very costly in testing
  - even with tens of users can be useful for smaller companies
  - big numbers remove the human factor (avoiding their personal preferences)

We analyze them (written here for the sake of completeness – you don't need to know them in detail of course, just to give you context to think/talk about/study):

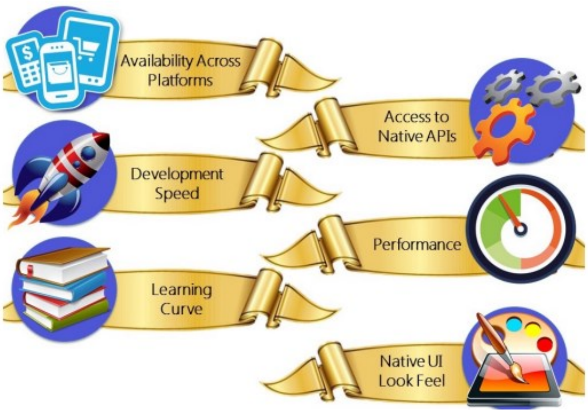
- jQuery & jQuery Mobile
  - Desktop and mobile applications were developed as a single one
  - Hardware access depends on HTML5 support of the browser (therefore not controllable by the developer)
  - Low initial knowledge (Javascript, HTML, CSS)
  - Low development complexity
  - Animation support
    - but complex animation performances rapidly decrease

- PhoneGap/Cordova
  - An application developed with Cordova is a web application plus the WebKit rendering engine
  - Allows access to several device sensors (accelerometer, compass, etc.)
  - Good performances with simple apps, but poor performances with complex apps
  - Development languages: HTML, CSS, and Javascript

- Titanium
  - Titanium allows maximizing reuse of pieces of code
  - Several APIs are available, especially for iOS and Android
  - Provides support for iOS (starting from version 5.0), Android (from 2.3.3), Window Phone, BlackBerry and Tizen (Samsung OS used inside Smart TVs)
  - Apps with native *look and feel*
  - Development language: JavaScript

- MoSync
  - Allows using different development languages: C, C++ and HTML+Javascript
  - Several APIs are available
    - but some of them are for obsolete operating systems versions
  - C++ development is a bit tricky
  - Native UI

There are *different selection criteria* for applications, combining different development approaches from the ones listed before. A framework is selected *not only for convenience*, but also *considering final results* – remember this:



We will make an *energy consumption analysis*:

- Energy consumption is a crucial element for application success
  - apps that drain the battery are rapidly uninstalled by users
  - almost as tedious as having widgets/buttons cut from the UI itself
- We considered the energy consumption of apps that acquire data from different sensors:
  - Accelerometer
  - Compass
  - Microphone
  - GPS
  - Camera
- Result compared between native apps and ones developed using cross-platform frameworks
  - Cross-compiled because of platforms
- How can we avoid interferences from internal (OS events) or external (user interaction) factors?

Several authors measured energy consumption of mobile applications:

- Thompson et al. proposed a model-driven approach (SPOT, System Power Optimization Tool – link here if you want) for energy consumption estimation before application development
    - using lighter languages/predicting power consumption on API calls/power consumption on sensor usage/assessing effects on power consumption protocols
- AppScope (Yoon et al. – link here if you want) is an Android application that estimates energy consumption of each hardware component

As a measurement system, Monsoon Power Monitor was used (with Monsoon PowerTool used to measure) and we can give the following features over the research:

- Provided data: energy consumption, average current and consumption, estimated battery duration, etc. – measuring the difference of power consumption between devices
- Analysis
    - Goal: energy consumption comparison of different hardware components during data collection, considering different platforms and different frameworks
    - Applications considered
        - Native Android application
        - Web application
        - Hybrid application developed with PhoneGap
        - Application developed with Titanium
        - Application developed with MoSync using C++
        - Application developed with MoSync using Javascript
    - Sensors depend on the API available with each framework

The *developed applications* were various:

- Elementary applications that collect data from different sensors at a given frequency, showing data with a simple interface
- It is necessary to define *a base level (or 0 level)* of energy consumption: device in standby, airplane mode, black screen with minimum white elements
- The base level depends on the device and its battery. Considered smartphones:
    - An iPhone 4 and an iPhone 5
    - A Samsung Galaxy Nexus and a Samsung Galaxy S5
- Basically, an Hello World example was tested on each framework
- It's quite impossible to have the same color on different screens
    - If they are not exactly the same, also the energy consumption is different
    - Colors used: black (pixel off)/white (pixel on)
        - Done to see the difference in updating

In testing, more precisely, when plotted and compared:

- Base energy consumption was higher on recent devices (e.g., iPhone 5)
    - native approach has almost 0 consumption
    - web-based approach has 2-4% more in energy consumption
    - cross-compiled approaches are similar to native ones
- Accelerometer energy consumption was higher on MoSync and Galaxy S5
    - JS compiler is similar to native one, C++ one requires a lots more of energy

    - this kind of data will be used to update data on the screen
- Compass energy consumption was higher on iPhone devices (4/5), but almost the same for all frameworks
    - activity which requires lots of data from sensors - very high frequency
    - that's why we want to leverage the native approach in development
- Orientation sensor one between Webkit browsers was higher (Opera/Safari)
    - it uses polling, waiting for continuous updates for data when available
- GPS consumption greater when native and also with different frameworks
    - again, higher on Apple devices

As results (table with numbers just to give context, given is an old research):

- Cross-platform frameworks determine higher energy consumption, even if the framework generates native code
- The most expensive task is the interface update
- Data acquisition frequency strongly influence energy consumption
- Not really cross-platform
    - Frameworks have different energy consumption depending on the operating system where they are running

	jQuery	PhoneGap	Titanium	MoSync
Licenses	5	5	4	5
APIs	2	2	4.5	3
Community	5	5	5	3
Tutorials	4	5	5	4
Complexity	4	4	5	2
IDE	-	-	5	4
Device support	5	4	4	2
Native UI	3	-	5	4.5
Learning curve	5	5	5	5

So, in conclusion:

- Results show that cross-platform frameworks consume more energy, hence determining lower performances and user acceptance
- Depending on the type of application, native development should be preferred:
    - Lower energy consumption
    - More APIs are available
- The results suggest that
    - Framework choice is critical
    - For a complex application, efficient frameworks are still missing
- Framework choice is crucial because it can influence user experience
    - Providing an ugly application is worse than not providing an application at all
    - Results show that, at the time of the experiment, Titanium seems to be the framework with better consumption

Some notes about *future development*:

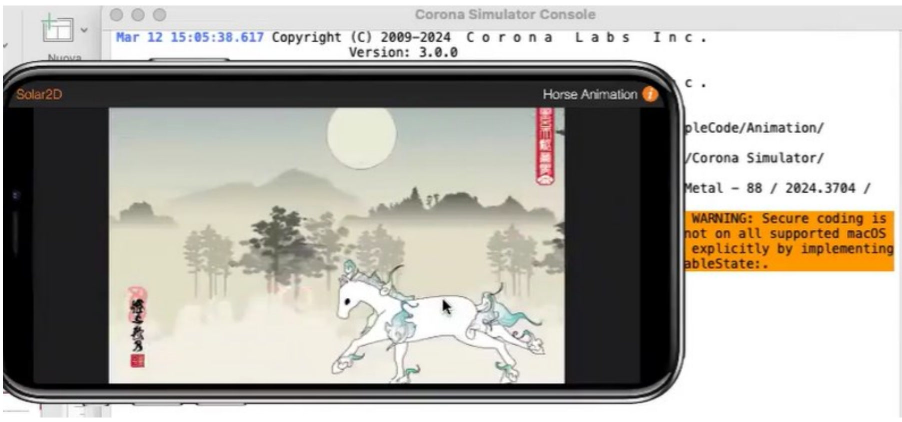
- An efficient cross-platform framework must provide an extremely efficient UI rendering
- Improvements of efficiency of rendering engines will provide improvements even for those frameworks that work with the web approach
- The cross-compiled approach seems to be the most promising, but it is not easy to implement
    - The development of these frameworks strongly depends on the development of a compiler able to produce an efficient code for the application
    - Development should be focused on the optimization of events handling and management

Titanium seems to be the framework with better consumption for the moment, but we should always consider that providing a lousy application is worse than not providing an application at all.

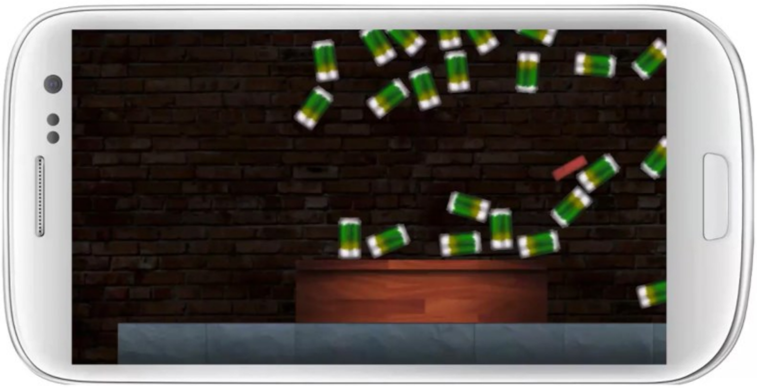
All references for chapters were papers, already quoted in different parts of this chapter.

As said in the disclaimer at the beginning of this file: Corona/PhoneGap/Xamarin won't be done. Instead, the teacher will show different examples about Corona framework:

- the first one about an endless runner with a running horse (animated GIF), with parallax scrolling of different backgrounds (here below)
- the second one about some physics simulation with soda cans thrown away (following page)
    - put images about cans/brick
    - the brick has a speed and angular velocity (called "body" inside this framework)
    - if two bodies collide, something happens
    - this allows for collision and some random mess inside a scenario
    - basically, the code define background/floor/stand and for each soda can, bodies are added to make interactions happen as physics
        - each with a bounce, friction and density
    - everything else is handled by the "Physics" library
- the third one about a guy collecting money (following page)



More complex example



Code – positioning

```
local background = display.newImage( "bricks.png", centerX, centerY, true )
local floor = display.newImage( "floor.png", 0, 280, true )
physics.addBody( floor, "static", { friction=0.5 } )
local stand = display.newImage( "stand.png", 170, 220 )
physics.addBody( stand, "static", { friction=0.5 } )

local cans = {}
for i = 1, 7 do
  for j = 1, 8 do
    cans[i] = display.newImage( "soda_can.png", 190 + (i*24), 220 - (j*40) )
    physics.addBody( cans[i], { density=0.2, friction=0.1, bounce=0.5 } )
  end
end
```

Code - bricks

```
local bricks = {}
local n = 0

local function throwBrick()
    n = n + 1
    bricks[n] = display.newImage( "brick.png", -20, 140 - (n*20) )
    physics.addBody( bricks[n], { density=3.0, friction=0.5,
                                  bounce=0.05 } )

    bricks[n].isBullet = true
    bricks[n].angularVelocity = 100
    bricks[n]:applyForce( 1200, 0, bricks[n].x, bricks[n].y )
end
```

4.3 Exercise 1 – Cross-platform development frameworks

Choose a framework for cross-platform development that was not discussed during the lessons (Cordova, Solar 2D, Flutter, React Native) – means not those ones – and classify it according to Raj and Tolety classification. Prepare a presentation of 5 minutes (max 5 slides) which briefly introduces:

- *the chosen framework*
- *why you choose it*
- *its classification*

The presentation must contain references to the used documentation. This exercise must be done in groups of two students and will be presented on the 19th of March 2024. The presentation must be uploaded before 15th March 2024 1.00 p.m.

<u>What I did</u>

- A Google Docs with a colleague choosing the framework
- A Google Slides presentation to easily work together file base on the Math UniPD department format, because it's simple
- 5 slides including the points above and a questions/bibliography idea

If you used Moodle at least once it's pretty obvious, BUT:

- I asked it to prof at the end of last lesson and she said that just one to insert slides on Moodle is sufficient, she also told me to write the name and student ID of the two group members in the section "notes or comment" during the uploading phase
- Hopefully upload it in PDF, it doesn't matter that much even in .pptx

Once you have uploaded, you receive a confirmation email.

- Rename the file with surnames of both components
- You will receive the evaluation after the last day of presentation via Moodle as "Competente" – you have the skill/"Non competente" – you don't have the skill
- The teacher will possibly ask you questions, but this will not be anything hard

Please, do not make the error of writing/telling her to "present one day instead of one another", because as she rightfully points out, this is part of the exam, so if you are missing it's like you don't wanna follow the course.

Things teacher pointed out:

- the fact no framework talked about flexible layout design
  - difficult because of lack from the frameworks themselves

4.4 Flutter Framework

(These slides were an in-depth analysis done by students as an alternative exam modality, that's why the format of slides is different from the others the prof. uses – basically, you have to communicate her directly you want to do this before April or something, then you can do this on your own)

Flutter is the youngest framework, making it still a fresh solution for developing cross-platform applications – first version in 2015, released in December 2018. More precisely:

- It is an SDK for mobile devices, developed by Google, for the development of native application for iOS and Android starting from a unique *codebase* (initial name was Sky)
- It also easily allows for Windows projects/web applications to be done with this one
- It uses a *cross-compiled* approach
    - *Trans-compiled* according to El-Kassan
        - meaning we will not obtain an APK which can run on the device
        - but rather a directory which we will need to open on XCode on iOS for building the application and sending it to the store
        - So, we can run the framework everywhere
            - but in order to create a iOS application
            - we will still need an Apple computer
    - When run, OS-specific folders will be created
        - you don't need to know the details of the program compiling
        - the only thing different is the icon, needs to be done for each OS manually
- Application written in the language Dart
    - allows for efficient execution of application
    - in particular for the web version
    - e.g., Telegram with good mobile/desktop counterpart, both interactive and good
    - React Native version was dropped because of this one
- Supported platforms: Android/iOS/IoT devices
- Some applications developed with these:
    - Google Ads/Greentea/Abbey Road Studios/Alibaba/reflectly

Its main characteristics:

- Fast development
- Expressive and flexible UI
    - A lot of effort was put to create good animations similar to the native ones
    - All done considering lots of fragmentation between Android devices
        - defining specific breakpoints and sizes
- Native performances
    - no studies on energy consumption of this one as of now

Fast development is done via:

- *Hot reload*, which allows to build and reload the code during runtime
    - without rebuilding the application again from scratch
    - useful in particular for UI design, e.g., small elements like labels
        - moving the elements in the simulator and seeing the result immediately
    - just with small lines of code and simple changes
- It's also *stateful*, remembering the interaction with the developer
    - redoing only the operation affected by the change and not anything else
    - also useful to test just some parts of a long, complex interaction of an application

About the *expressive and flexible UI*:

- There is a good, personalized user experience
    - thanks to the enormous amount of widgets available
        - material design (modern Android UI) and Cupertino (clean/minimal Apple UI)

Also, performances are as good as native apps:

- Given each element is considered a widget even when accessing sensors
- These ones incorporate all the main characteristics of different platforms
    - e.g., scrolling, icons, fonts

There are different pros and cons to this framework:

- Pros:
    - Free and open-source
    - Single codebase running virtually everywhere
    - Easy setup
    - Hot reload
    - Widgets
    - Native performances
    - Many plugins for IDE
    - Good documentation
- Cons:
    - Available only for mobile
    - Low number of libraries
    - Difficult to create animations
        - since it is a general purpose framework and not thought for games
    - Need to know Dart to develop for this one

Flutter gives *guidelines* to the developers in order to – very powerful but also very risky:

- Gain *control* of the whole system they are creating
    - which is a very powerful but also a risk in the wrong hands
    - maintaining predictability and consistency across all UI and interactions
- Create cross-compiled applications as efficient as possible (good *performances*)
    - responsiveness, smooth animations, efficient memory usage
    - impacting positively user satisfaction and preventing frustration

- Try to obtain *fidelity* from developers
  - thanks to all the quality of life improvements they focus on
  - matching UI and design specs

Another big focus is *accessibility* from this framework, bringing components doing this:

- in the mobile world there are no precise rules to realize these principles
  - this was not the first framework to do so – this is written inside of the docs
- this can be easily done defining *Semantics* and *role* inside of the code of each element
- there is native support for big fonts, screen reader support and different contrast options
  - considering mobile devices serve a lot of different purposes
    - other than calling and reading

A good option is the presence of a big *community*:

- different options, between GitHub, YT, Slack, Medium, Stack Overflow
- official website with Cookbook, Codelabs and tutorials

Let's talk about *Dart*, the programming language of Flutter running on a C++ engine (similar to C#):

- It is a programming language, object-oriented, used to develop web, server, desktop and mobile applications, developed by Google (its first name was Dash)
- Supports all the known data types
- Each variable points to an object and stores a reference
- Every Dart app is a *library*
  - It is possible to use libraries for code modularity
  - *Lazy loading* for libraries (loaded only when needed)
- Once a library is imported, only some parts can be used/imported (via *show/hide*)
- Flow control is as always
- Exceptions are not managed
- Classes can inherit from other classes but only once (single-inheritance)
  - Keywords: abstract, extends, implements, @override
- (Important here) Dart code can be compiled in different ways:
  - *Just-in-Time (JIT)*
    - to test performances, we have to use this phase (done when deployed or run)
    - feature which makes Flutter *cross-compiled*
    - it's only used in debug mode, never when app is built
  - *Ahead-of-Time (AOT)*
    - simply compiles the component when it's actually needed at runtime
    - translating source code into machine code before execution
    - allows to save time during development

Let's have a look under the hood:

Flutter architecture is based on levels, where the lower levels implement the simplest operations and provide the operations logic, while the upper ones compose the displaying and graphical part.

In particular, it is based on the following components:

- *Material and Cupertino*
  - implements widget Material (Android) and Cupertino (iOS) style
  - basically, according to OS style, they adapt
- *Widgets*
  - implements generic widgets, e.g., a button which can be pushed
- *Rendering*
  - simplify layout management
- *Animation*
  - tween
    - an object which moves from an initial to a final position with a speed)
  - physics-based animations
- *Painting, Gestures*
  - an area where to draw something
- *Foundation*
  - lower layers, which allows to create easier widgets
  - allows to create a lot of widgets similar to each other (e.g., network connection)
- *Dart:ui*
  - manage communications with the Flutter engine
  - taking for example a button and rendering the shape and main functioning

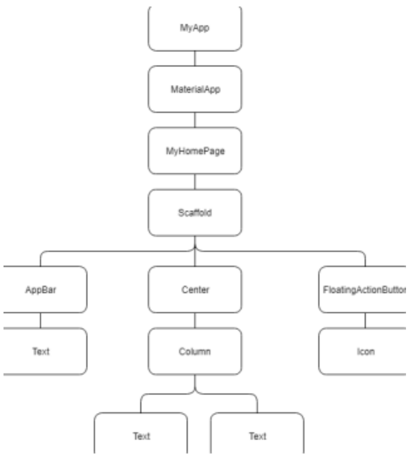
A *widget* has different features (in Flutter, everything with an interacting interface is a widget):

- Base components of the user interface
- Each widget is an unchangeable declaration of the user interface
- A widget can define:
  - A structural element (button, menu, ...)
  - A style element (font, ...)
  - An aspect of the layout (padding, ...)
- Define as hierarchy based on composition
- Allow to manage events

The widget building allows for a tree definition (a hierarchy basically), using build() as a method to create it. This is called at the root of this tree then activated on cascade to all other elements.

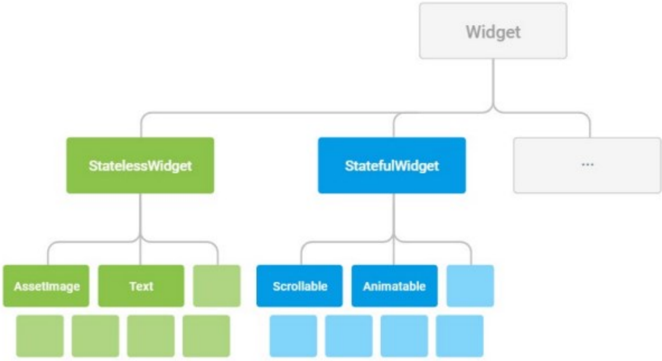
According to the following figure definition, everything is a widget and only using the Scaffold (top-level widget organizing app bar/body content/additional components) each one is positioned correctly.

The layout itself is a widget, organized as hierarchy.



Apart from the `Scaffold`, we will have the `HomePage`, indicating which page we are on, putting elements center/left/right to the current scene. When rendering, the build()method is not called on all elements (hot reload), since it's not necessary to rebuild all widgets.

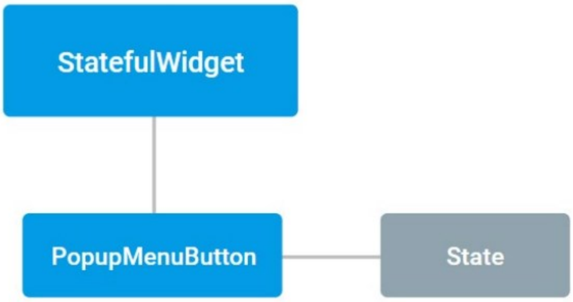
The widgets can be either stateful or stateless, considering this figure:



In particular:

- *Stateful* means having a mutable state, so the element can change or be updated via an interaction or an external event
    - this might be for structural/style/layout elements
    - allows for saving time and energy
- *Stateless* does not have a mutable state and remain constant overtime
    - these elements do not need to be rebuilt, because they cannot change

An example of a stateful widget might be the following, using as elements `createState()` (which creates for the first time the state of an object) and `setState()` (which updates a widget and notifies the application when the state is changed in order to call the reload):

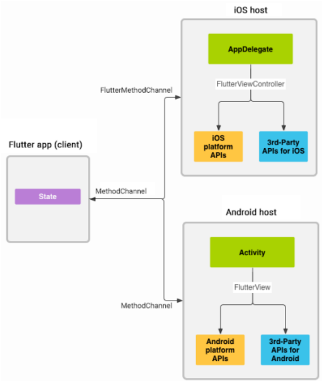


Flutter has a set of base widgets, the most used are:

- `Text`
- `Row`
- `Column`
- `Image`
- `RaisedButton` (it elevates when pressed upon – now replaced by ElevatedButton)
- `AppBar` (the top bar of an application)

Other features:

- To inspect all widgets available, one can use *Flutter Inspector*
- *Flutter Engine* is a runtime environment written in C++ which implements key libraries
    - It provides Dart runtime, Skia (2D graphics library), Platform channels
- *Platform channels* allow communication between Dart and specific code of each platform
    - Different channel types
        - `BinaryMessages` → low-level binary code to platform-specific cde
        - `MessageChannel` → bidirectional asynchronous communication
        - `MethodChannel` → calls a method and makes communication possible between specific APIs and IOS/Android OS
- *Code forking*, practice of writing platform-specific code branches to accommodate differences in features, APIs, or behaviors across different target platforms
    - To import features which are not present inside other platforms, calling different APIs
    - *E.g.,* Apple which hasn't got any vibration functionality inside iPads



- *Extensions*, via usage of Package or Firebase
    - Through Firebase it is possible to install packages for Flutter, in order to reduce the work needed for the developer
    - Provides a vast array of packages that offer ready-made solutions for common tasks, functionalities, and integrations

To develop Flutter applications we need:

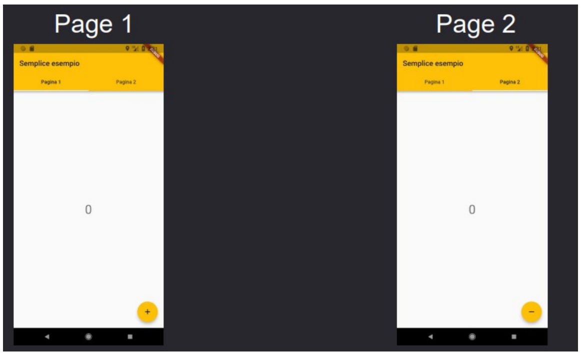
- Flutter SDK
- An editor or IDE, suggested ones are:
    - Android Studio
    - IntelliJ IDEA
    - Visual Studio Code
- For the proposed IDE there are Flutter plugins

It's not so hard to setup this framework:

- It is possible to install Flutter on Windows, macOS o Linux
- Installation process:
    - SDK installation
    - PATH variable modification
    - Command `flutter doctor` → Check for missing packages

With this simple example we will learn how to use the following components of the framework:

- Stateful widget
- Stateless widget
- Tabbed layout



The application has a tabbed layout with the following pages:

- Page 1: allows to increate a counter through button click
- Page 2: allows to decrease a counter through a but ton click

The following are the classes employed:

```
class MyApp extends StatelessWidget {...}
class FirstPage extends StatefulWidget {...}
class SecondPage extends StatefulWidget {...}
class _FirstPageState extends State<FirstPage> {...}
class _SecondPageState extends State<SecondPage> {...}
```

This goes for the first page:

```
class FirstPage extends StatefulWidget {
    FirstPage({Key key, this.title}) : super(key: key);
        final String title;
        @override
        _FirstPageState createState() =>
_FirstPageState();
}
```

Then, its state:

```
class _FirstPageState extends State<FirstPage> {
 int _counter1 = 0;
 void _incrementCounter() {
   setState(() {
     _counter1++;
   });
 }
```

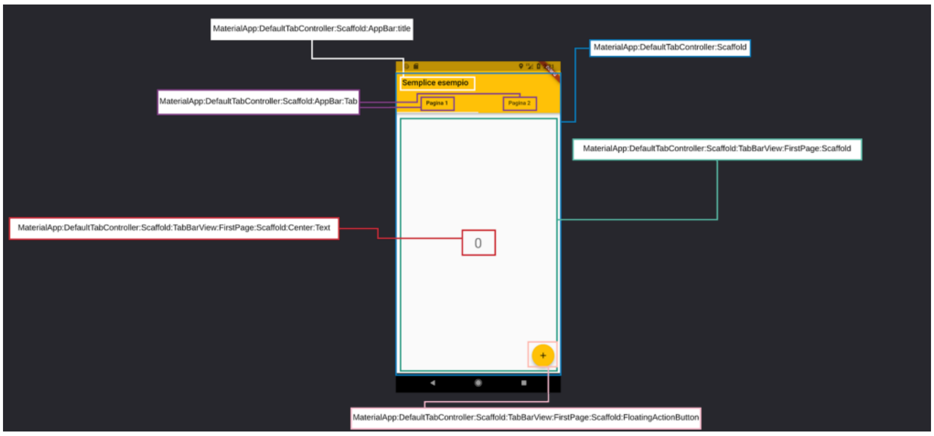
```
@override
 Widget build(BuildContext context) {
   return Scaffold(
     body: Center(
       child: Text(
             '$_counter1',
       ),
     ),
     floatingActionButton: FloatingActionButton(
           onPressed: _incrementCounter,
           tooltip: 'Increment',
           child: Icon(Icons.add),
         ),
       );
  }
}
```

Written by Gabriel R.

Completely, the application:

```
class MyApp extends StatelessWidget {
 // This widget is the root of your
application.
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       primarySwatch: Colors.amber;
     ),
     home: DefaultTabController(
         length: 2,
         child: Scaffold(
           appBar: AppBar(
             bottom: TabBar(
               tabs: [ Tab(text: "Page 1"),
                          Tab(text: "Page 2")]
             ),
             title: Text("Simple example"),
           ),
           body: TabBarView(
             children: [
               FirstPage(title: "First page"),
               SecondPage(title: "Second page")
             ]),)),);}}
```

Overall, the interface behaves as follows:



4.5 REACT NATIVE FRAMEWORK

(Again, these are student-based slides for the examination – also, as of now, slides on Moodle are not complete considering the ones in class – present inside "In-depth slides" folder inside MEGA)

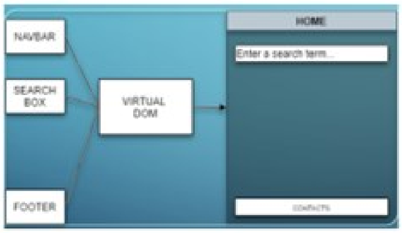
React Native is a cross-platform open-source framework that implements Facebook's ReactJS (*react.js*) library. Some features:

- It uses JavaScript, in particular in its JSX form
    - Languages used: JavaScript and XML
    - Easy to learn both for devs/web devs and not-so-technical people
- Everything inside this framework is a component (component-based) and it's made for creating UIs
- It can be used for developing Android and iOS mobile applications
- Like Flutter, it has a very large community and many docs/tutorials
    - supported by a very big company: Meta
    - along with the first one, the most used framework today
- Developing and debugging is very fast
    - just like Flutter which has hot reload
    - compiler understands changed components and recompiles only them
- Only pro over Flutter: easier to learn because of web technologies used
- Native look and feel
- Allows to create and use personalized components
    - they accept input data, called properties (props) and return React elements
- Possibility of integrating React Native with an already existing mobile application
- Develop once → deploy for all (without using native tools)
- It works with command-line tools

A bit of history:

- First released as an open-source project by Facebook in 2015
- Created because the use of HTML for the creation of Facebook app brought to very low performance and user experience
- They initially created a way to generate UI elements for iOS using a JS thread in background
    - then the framework was created starting from this prototype during an hackathon
    - user interaction was really slow
    - combined the pros of hybrid approach (web technologies – very low learning curve) with cross-compiled approach (good performances)

Some apps created with this: Facebook, Skype, WordPress, Discord, Tesla, Pinterest, NFL, Outlook, Office, Teams, PlayStation App, Xbox Game Pass, Puma.



A small code example is shown: simply a JS function defined with fields and placeholders to activate logic.

- As said, it's *component-based*:
    - Every different part of the application is a component
    - Every component is indipendent and reusable
    - They are JavaScript functions that accept input data (props) and return React elements

Written by Gabriel R.

React uses a JavaScript representation of the DOM, called *Virtual DOM* (*VDOM*):

- Starts from DOM (Document Object Model)
  - where each element is defined inside of a tree
  - the actual dom is based upon it
- Each and every time a component is updated
- React creates a new VDOM and compares it with the old one
  - in order to render only what has been modified
  - same principle as hot reload in Flutter

Each component can have properties (*props*) and a state:

- *Props* (properties) are used to configure a component when it renders and allow to customize it
- They are 'read only'
- They're used to influence components in a top-down approach (Parent to Child)

```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;
  }
}
```

There is a relationship between props and state:

- *State* is used to keep track of any component data that is expected to change over time
  - due to user action, network response, etc
- These can change overtime, props cannot
- States can be used to influence components in a bottom-up approach (Child to Parent)
  - passed from children to parents easily

```
class Square extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        value: null,
      };
    }

    render() {
      return (
        <button
          className="square"
          onClick={() => this.setState({value: 'X'})}
        >
          {this.state.value}
        </button>
      );
    }
  }
```

More in detail over states:

- To modify the state, it should be used `setState()`
- React may merge multiple calls to `setState()` into one but these updates may be asynchronous
- Another way to manage the state is to use the 'Hooks' (React 16.8)

```
// Wrong
this.state.greetings = 'Hello';

//Right
this.setState({greetings: 'Hello'})
```

```
const [value, setValue] = useState(null);

...

<button
    className="square"
    onClick={() => setValue('X')};
> ...
```

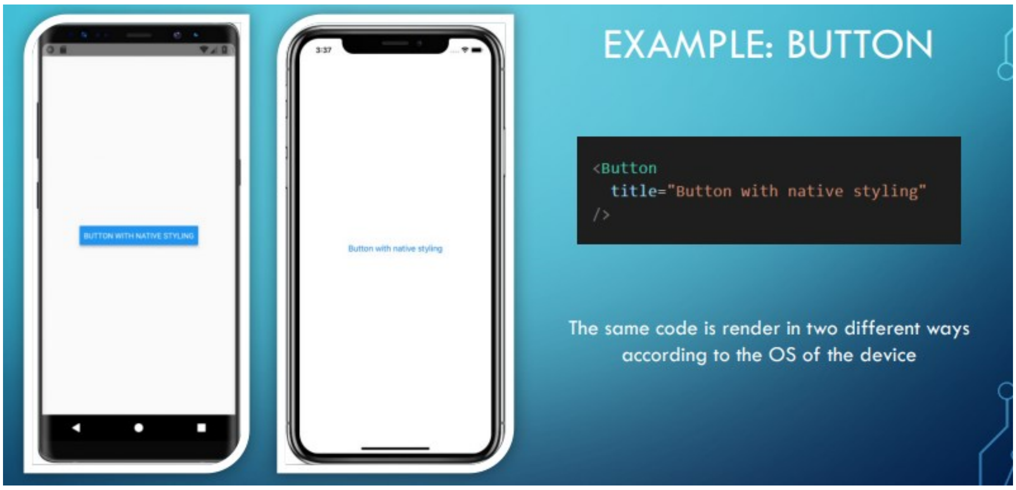
Redux library can be used to make the state update synchronous.

- To connect Redux to any application, we need to create a *reducer* and an *action*:
  - An *action* is an object (with a type and an optional payload) that represents the will to change the state
  - A *reducer* is a function that takes the previous state and an action as arguments and returns a new state

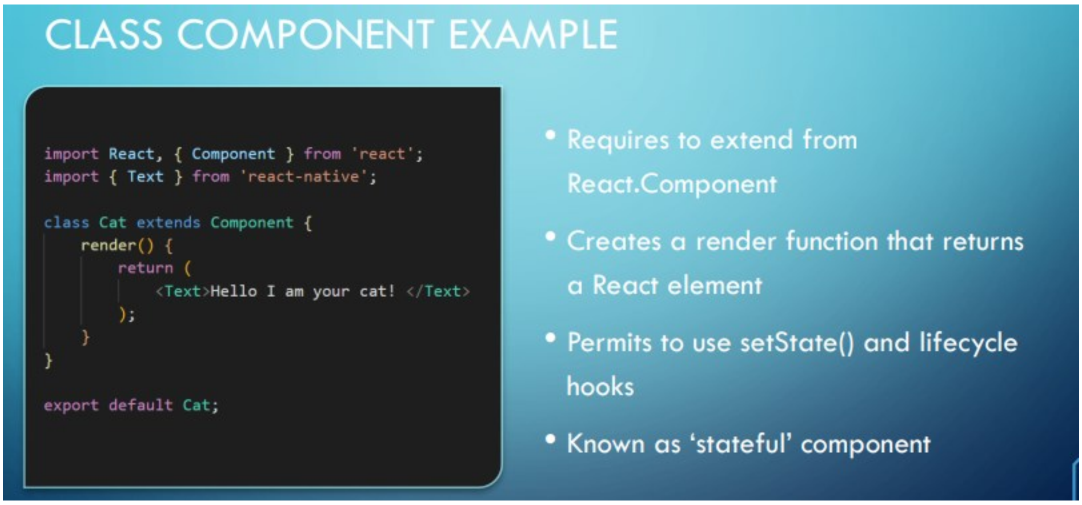
- Redux also introduces two functions:
  - `mapStateToProps`
    - used to make state accessible to the screens that implements it
  - `mapDispatchToProps`
    - to access to defined actions

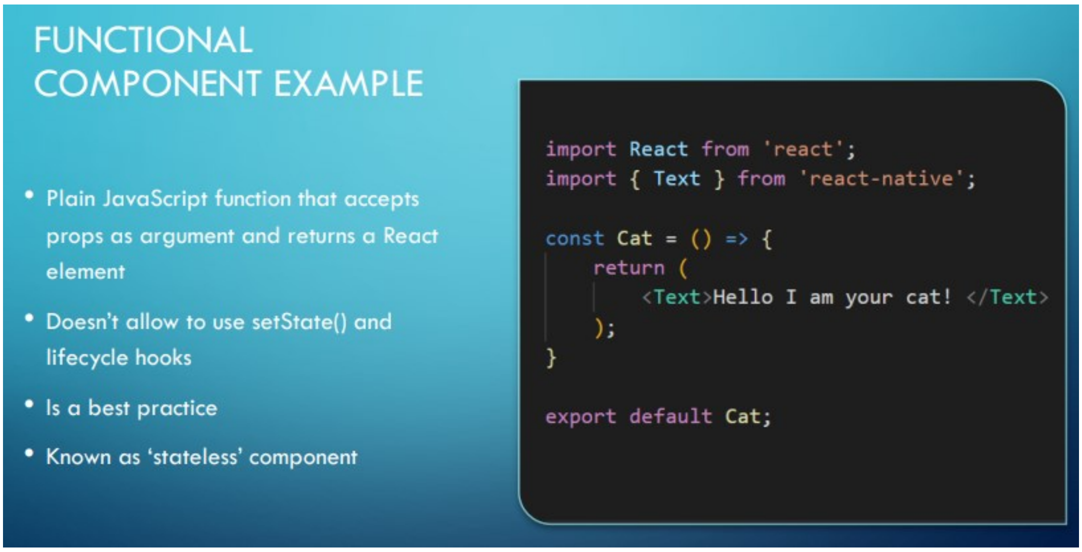
The following is a code example for a color-changing wheel app:

Another example for a button:



Just to show you, the following are examples of different components (not required to know):





4.6 Store Deployment

Once completed the development of the app, work is not finished, because one has to work to make the apps rank higher. There are two critical steps to take:

- Test
- Deployment

Other things to note:

- Both the stores, with different accuracy levels, test the applications
    - before adding them to their catalog
- Therefore, it is essential to *deeply test* the application before the deployment phase
- The deployment procedure asks for different screenshots of the application
    - both for the tablet and the smartphone versions
    - this phase should not be underestimated
        - as it may take some time
        - and several temporal constraints are not manageable by the developer

There are different management policies between stores:

- Apple Store has a lot more control and can take much more time to approve (up to 3 weeks)
    - They have the claim of applications having "*very* good quality"
        - God is in the details (e.g., widgets, elements, buttons, etc.)
        - app has to render perfectly on all possible resolutions
    - Lots of testing and debugging are done before putting the apps on the store
    - When you want to deploy an app to a device
        - a device has to be connected to the user
        - connect devices and plug it for one of the users
            - up to 100 devices/users
    - TestFlight: private store to buy assets, test apps and collect feedback
        - it can be used by devs, companies, etc.
    - Much more precision required but more manual control over testing
        - app working correctly on all possible products

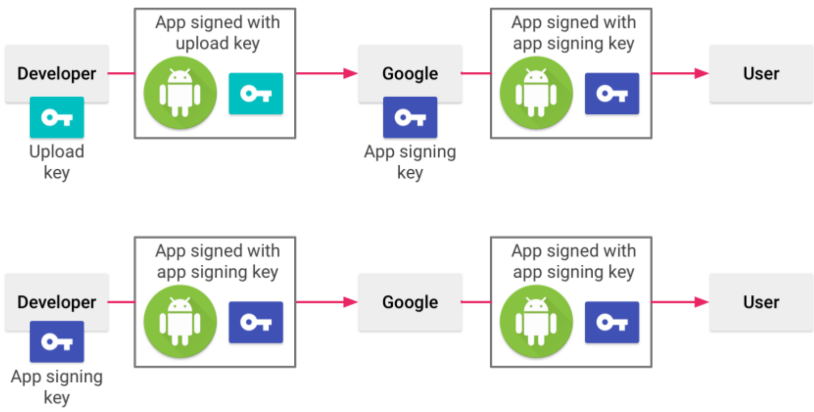
- Google Play Store is much more permissive
    - The same day or the day after can be approved or rejected
        - requires manual feedback from lots of users to remove
        - a reason for rejection can be also a high number of crashes/run time errors
        - otherwise, literally everything gets published
    - Much more choice when installing applications
        - `.apk` format to install them
            - not even a store required to install apps

4.6.1 Google Play Store

We start looking at stores more in depth, starting from the Google Play Store.

- It is created to facilitate the search for the name of an app and then the download immediately
  - this can also happen inside a normal search engine
  - the stores have search engines ad-hoc for them with customized algorithms
    - the information present here is retrieved via reverse engineering
    - done via uploading of fake apps in order to understand how it works
- In addition to a Google account, it is necessary to have a developer account on the Google Play Developer Console (reference here):
  - 25$ unlimited duration, unlimited number of apps, it allows different accounts with different roles
    - inside of the Console, one can see the app status, how they are going on the store and other information
    - inside Apple store, 99 dollars/euros are paid once every year
      - Apple logic! (sorry Apple fanboys :-D)
    - overall the costs are very low, in order to stimulate developers to publish applications
      - this does not avoid the absolute chaos the store is (sadly)
      - since Play Store does not run too many checks on what's published
  - Credit card is required to be available inside of the account
    - Normally, require this info only when needed (too much trust needed)
- For payment apps or apps with in-app purchases, it is necessary to have an account on the Google Payments Center
  - activated from the console under the section "Commercial Account"
  - and after "Configure the commercial account" with the required data
- One thing Google is pretty strict upon since last years is the change of policies
  - particularly regarding Android system updates and latest APIs compatibility
  - or even change to legal rules for applications
  - if such are not anymore compliant, they get automatically removed or unlisted

The *application signing* process is required in order to put an app inside the store. It is very simple (compared to Apple once again) but it's not possible to use debug signatures. It allows all APKs to be digitally signed with Google signing key and developer key together:



Written by Gabriel R.

To deploy the apk of an app in the Play Store it is not possible to use debug signatures:

- To generate a private key
  - it is possible to use the keytool command from a shell
- Some information required:
  - keystore password and key password, developer name and company name

```
keytool -genkey -v -keystore
mykeystore.keystore -alias aliasname -keyalg
RSA -validity 999999 -keysize 2048
```

mykeystore is the name of the app
aliasname is the name of the alias

There's a lot of *required information* (here to be gathered comprehensively, I won't remember either):

- APK of the application
- Name
- Description
- Screenshots
  - Providing a lot of screenshots = faster approval process in stores
    - Also more convincing to the users to download it
  - At least two for smartphones, min 320, max 3840, *.jpg* or .png $24b$ no more than 2:1
  - At least two for tablets
    - At least one screenshot for 7 inches tablet and one for 10 inches tablet
  - A good icon (*.png* $32b$, with alfa)
    - Nice layout, good description, some interesting factors
  - $512x512$
- Feature image for the presentation page
  - $1024x500$, *.jpg* or *.png*, $24b$
  - Really important for presentation purposes and to choose it over other apps
- Other images
  - Android TV, Wear OS, promotional images, trailers, screens for wearable devices, etc.
  - Not so mandatory in the end
- APK
- Categorization
  - Type and category
- Content classification
  - Necessary to determine the minimum age for the app
  - Requires compiling a questionnaire
  - One of the few checks Google does before putting the app publicly
    - Is it suitable for children?
    - Does it contain advertisements?
    - Does it contain in-app products?
- Contact email
- URL with privacy information
- List of compatible devices
- Prices and countries where it is available

There are other services which can be added:

- App signature can be managed using the console
- It is possible to add different app localizations
- It is possible to buy translation or optimization services
- The app can be distributed as an alpha or beta version, deciding if it can be tested by
    - A closed group (email invitation)
    - An open group (all the users of the Google Play Store)

Some notes about *prices and distribution*:

- Once an application is deployed as free, it is not possible to transform it into a payment app
    - Solution: ex-novo project (start from scratch)
- The price can be automatically converted into different currencies or can be manually defined
    - Both fixed and even require payment in other countries
    - The Freemium version (free base version, advanced paid functionalities) is a good solution to let the user evaluate the app before buying it
- It is possible to buy different app localizations, both translations and adaptations to different cultures and useful in app promotion
- Google keeps 30% of the net amount
- Only some hours are required before publishing an app
    - It needs to undergo tests but in quick time, a response will be there
    - A wrong classification and/or categorization can lead to the app removal

4.6.2 Apple Store

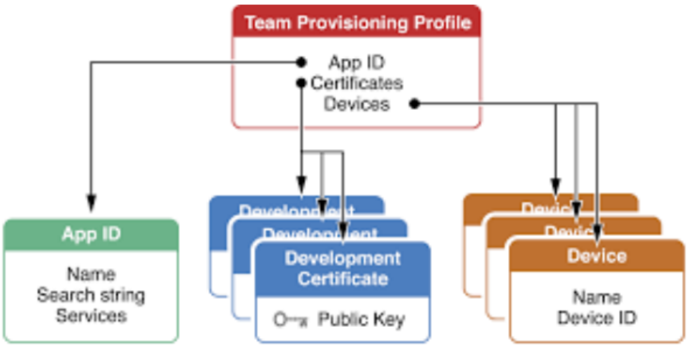
Now let's talk about the Apple App Store.

- Publishing an application on the Apple Store requires an Apple Developer Program account
    - Cost: 99$/year
        - Much higher than Play Store
        - In theory, done to discourage people from putting bad quality apps
        - In reality, it speaks to Apple status symbol: money money money
            - crafting a standard for more wealthy like people
    - Necessary for signing the application
    - Registration as a private user or as a company who publish the product
- Moreover, an App Store Connect account is necessary
- The entire process is much longer than the one for the Play Store
    - Because it follows standards
- Publication process requires a Mac with Xcode installed
    - It is not always necessary to directly use Xcode, but the program for building the app uses Xcode, only available for licensed platforms
        - There are workarounds on this
        - but the building process will use Xcode nevertheless
    - Necessary for high-resolution screenshot

The certificates needed to sign an Apple app are way more complex with respect to the Android ones, as you can see in the picture below.

Apple provides different certificates, all needed to sign the application, which are:

- *Developer*: is the developer signature of the application
- *Application*: linked to the application, defines the name
- *Device*: linked to a single device, used during the test phase for deployment of the app on devices linked to the signature account (developer who signed the application)



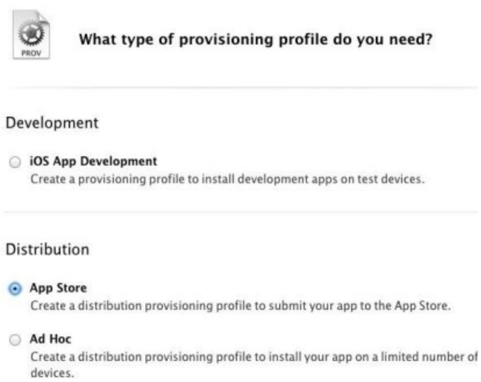
An application must be signed with a *Provisioning Profile*, which collects information from all the previous certificates (profile for distribution, collects info on app ID, developer ID and kind of device). This profile can be:

- *Development*: for the test phase
- *Distribution* (or *Production*): ready for the deployment on the store

A company account can be linked to different Developer, Application and Device certificates, and to different Provisioning Profiles.

The *certificates creation* requires an Apple Developer Program account.

- One goes here
  - and selects “Certificates, Identifiers & Profiles“
- Create before an App Id and after a Distribution Provisioning Profile
  - the name of the app is almost like an ID, do not change it; when you name it for a company, give it the company name
- Consider the bundle ID
  - it uniquely identifies an application in Apple's ecosystem



- Once created a Provisioning Profile
  - download it (on an Apple computer)
  - and add it into the keychain (double click on the certificate)
- If the developer certificate was not already installed on the used computer
  - to build the app, it is necessary to download it and add it to the keychain

The professor made an example of certifications creation (a complete step-by-step guide here):

- Bundle ID: it.unipd.math.example
  - cannot be changed, one can only add some capabilities
- Next step is selecting categories
- Select the account for the application
- Select the devices in which the app can be installed (provisioning profile)
- To deploy that, we need an App Store account and then use the device accordingly
- After 7 days, the app cannot be deployed again if one selects other organizations or things not related to the other ones
- To have a private store, one can use TestFlight, which one can access only having the link of it
  - Here, crashes reports will be automatically sent to the company

The teacher showed an example of app here, called "Pizza al Lancio" made by Primo Round.

Once created and installed all the certificates, it is possible to create a *.ipa* file.

- Xcode is the tool for native development on iOS
- In our case, we use the Corona/Solar 2D framework
  - that allows sending the app directly to the Apple Store without opening Xcode

Just like we did with Play Store, some *mandatory information* to publish an Apple app:

- If we want to publish a payment app, it is necessary to add bank coordinates under the section «Contractual, fiscal and banking information»
- Apple keeps 30% of the amount
- To publish a new app the initial information are:
  - Platforms of use
  - Name
  - Primary language
  - Packet ID (is the ID chosen when creating the certificate and the Application ID)
  - SKU (Stock Keeping Unit - a unique id, not visible to Apple, defined by you)
    - a carry-over from iTunes days, basically an inventory tracking ID
- Once created a record on iTunes/App Store Connect, other information are required
- This goes inside the panel "App information"
  - URL with privacy information
  - Subtitle (optional)
  - Category and classification: attention, very important
    - This way, a content is categorized, basically to let the app store algorithm what to display; like Play Store, wrong ones can lead to removal
    - It is possible to choose two categories and one or two subcategories
  - License agreement
  - Screenshots (high quality ones) and promotional videos
- It is possible to save draft and continue later, given the process is long and time-consuming

Under the panel "Prices and availability", an app can be set in price with the following information:

- Free
- For payment
- With in-app payments (freemium)

About the *screenshot* process:

- They are a critical part of the approving process
- The App Store requires to add at least one screenshot for each supported device, because resolution can deeply change
    - Screenshots must be taken for all platforms the app can run on
    - showing every part of the application
- Insert this information with two purposes:
    - Attract the user
    - Show how the app works on different devices, adapting to different resolutions
- *.png* and *.jpeg* images with precise dimensions
    - https://help.apple.com/app-store-connect/#/devd274dd925
- Xcode simulator can be used to create the screenshots (cmd+S) for devices not owned

Other information that can be added in a second moment are:

- Application description and keywords
    - Necessary to place the application in the ranking of the store, and especially necessary to have a good one
    - Keywords cannot be modified in a second moment
        - so be really careful about this decision
        - since they are used to find our product
    - Keywords should be put both inside name and description
- URL for support and customer care information
- The build (or binary)
- Icon (1024×1024 pixels)
- Copyright information
- Version number
- App review information:
    - Reference contact, particular requirements must be written in the notes
    - timing for app release
    - (immediately after verification or in a different moment decided by the developer)

Apple will *check* that the app:

- does not contain malware or unauthorized content
- has an high-quality standard of the interface and follows the indications
    - also, it's used for the specified category of users
- works more or less in general (and does it well)

Tests are made manually:

- in the past they required on average two weeks
- now, it actually takes one week (if everything goes well the first time)
    - otherwise, verification may take less, while runtime errors may take definitely more

After the publishing:

- App Store Connect allows to monitor the application
    - providing information about the number of downloads and sells
    - but also crashes (only if users provide consensus to send feedback)

4.6.3 After the publishing

The user should always be preserved and protected, making him choose our app instead of other ones, in order to have success. In order to increase ranking of an app, update it often.

- Given users are very demanding
    - there should be reasons to continuously use our app and reminding him to use ours
- For example, we can use push notifications
    - carefully, to avoid annoying the user
- A good app
    - gets installed through good positioning and publishing
    - gets opened the first time leaving a good impression
    - able to get back users an indefinite number of times

Work has just begun again *when the app is published*:

- Especially for Android applications, it is crucial to monitor app usage
    - Given Google is not exactly like Apple, so "as long as it works, they publish"
        - but then every burden is for devs
    - With "Abnormal stops" and "ANRs - Application Not Responding" it is possible to see malfunctions and with which devices, and error reports from the users
        - These are called Android vitals: stability/rendering/battery/loading issues
    - Crashes are listed only if the user accepts to send the report
    - New versions can solve these types of errors
        - possibly solving bugs and providing new functionalities

Consider the process of *app optimization* for the stores:

- The higher is the position of the app on the listing, the higher probability of new installations
- Stores are huge and to be downloaded, an application must be in the first positions of the ranking
    - App ranking is crucial
    - ASO - App Store Optimization
- Two different strategies:
    - *Onsite*
        - Similar to on-page optimization for search engines
        - It defines methods that modify the application page on the store
        - It considers name, description, icon, screenshot and all other info of the app
    - *Offsite*
        - Similar to off-page optimization
        - It works on factors that cannot be modified autonomously
            - Such as number of downloads and users' rating

Unluckily, differently from "normal" search engines:

- Both Google and Apple do not publish ranking criteria
- All the information we have is thanks to reverse engineering – shall I remind you it's not legal, but just for completeness, I'll pirate anything if I have the chance

Onsite criteria that influence ranking are:

- Keyword
  - both inside name and between the ones defined when app was inserted in the store
- Presence and type of icon
  - Also dev name is important, especially if you have many apps, as a way of "trust"
- Number and quality of screenshots
- Preview video
- Usage and evaluation stats

Offsite criteria that influence ranking are:

- Number of downloads
- Download evolution
- Number of installations (conversion rate)
- Number of uninstalls/removal
- Number and average of the evaluations
  - If good average = good, otherwise not so much of course
- Ratings evolution

Do not try to influence these values with dishonest methods to avoid being "banned".

- This can be paying people to review well the app
- If the store finds this, the app will be removed or banned for a period
  - Factors like no updates, same app and everything goes well can be a factor

Let's discuss now about ASO for the Play Store.

- Since there is no available information about ranking implementation, the starting point is the optimization of the Google engine
- Keywords:
  - Excellent if they are contained in the title but it must be shorter than 20-25 characters
  - Insert the brand name in the title only if it is well known, otherwise it is better to put keywords that describe the content
  - Very important is the app description (both in the short and long form)
  - The Play Store provides an "Auto Suggest" feature that shows the most used keywords
    - and can be a resource to choose the correct keywords
- Screenshots and videos:
  - Used to improve ranking, and increase the possibility of being chosen by the user, which considers more seriously pages with a lot of visual elements
  - Really important are colors usage and quality
  - The icon must be easily recognizable even within an interface full of elements if you want to be sure that the app will be used once installed

- Choosing the right category is very important
- Play Store gives high importance to the number and quality of external links that point to the page of the app
    - Good app promotion outside and good quality are the winning factors

Let's discuss now also about ASO for the App Store.

- As for the Play Store, keywords are very important, but in this case are inserted in a specific field with a maximum of 100 characters
    - If the app is multilanguage, even the keywords must be multilanguage
    - Separate keywords with commas without spaces
    - Do not add the category to the keywords, but choose it carefully
- The name of the app can be at most 50 characters (suggestion: maximum 23)
    - and must avoid terms that recall the app content
        - do not put keywords inside the name, it's not like Play Store
- The description is a maximum of 4000 chars, but is better to use less to improve readability
    - Apple officially declared to use the keywords only in the specific input field
    - but adding them in the description is not a bad idea
    - First sentences are the most important
- Even in this case, icon, visual material, number of downloads, and ratings are critical
    - Ratings are important even if not positive
    - Customer service helps improve ratings
    - Use good icons/feature graphics

References used in chapter and its slides:

- Android
    - https://developer.android.com/studio/publish/
    - https://developer.android.com/studio/publish/app-signing
- Apple
    - App Distribution guide
        - https://help.apple.com/xcode/mac/current/#/dev8b4250b57
    - App Store Connect Developer Guide
        - https://developer.apple.com/support/app-store-connect/

5 2 - Mobile Design

User Interface (UI) design for mobile applications must consider several aspects to differentiate them from desktop ones, this way creating good and likable applications. The main differences are:

- Device size and weight
- Computational capabilities
- Operating system
    - On mobile applications there is a RTOS – Real Time OS
    - This is a soft real-time, like answering calls or notifications to do something else
    - Hard real-time is for nuclear plants or real systems somehow
- Interactions
    - Touchscreen
        - Fingers are more natural, the mouse is a lot more precise
        - More natural interactions in this environment, considering screens and sensors
        - Widgets must be bigger on mobile
        - Hands can be dirty, and this may be a discouraging factor
    - Sensors (movements)
        - Accelerometer/GPS/etc.
    - Vocal I/O
    - More direct/natural interaction
        - This can be a pro but also a con: user has more expectations towards this one

Touchscreens are available in several different situations:

- ATM, machines for electronic selling
- Informative panels or totems (ex: museums)
- Tablet/smartphone
  Mobile computer

Consider in 2011 Apple sold more iPads in one year than all the other products in 20 years, but still there are many websites and mobile applications that do not optimize interactions with touchscreens, making this problem really actual in our times. Consider the following:

- Interfaces very, between smartphone and tablets
- Touch is used even in desktops
    - User with smartphone "has one eye and one finger"
    - Give some support for user and give him simple gestures
    - Easy way to recover from error, given the user is not fully focused on what he's doing
- Vertical interfaces do not work well, because they give "weight" to the user

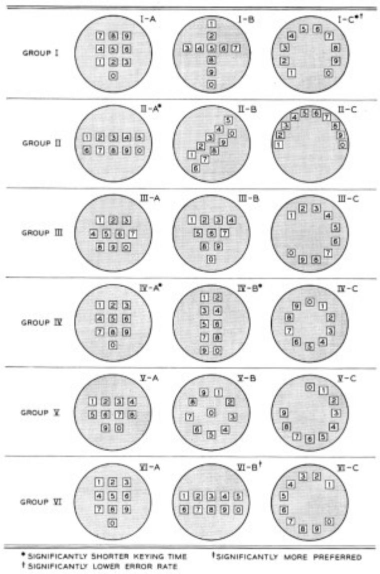
More generally, the market of tablets saw a big increment of sales over the years, which make possible to distinguish interfaces in:

- *Smartphones*: from 3 to 8 inches
    - use as interactions touch or gestures
- *Tablets*: from 9 to 12 inches
    - use as interactions touch or gestures

- *Laptops*: from 13 to 18 inches
    - nowadays can both use touch or gesture and cursor or keyboard
- *Desktop*: from 19 to 28 inches, use as interactions cursor or keyboard

5.1 How to design a good user interface?

- Interfaces are not just the same when moved from desktop to mobile
- Responsive interfaces are one step forward but not the answer
- Users do become "defective" with age, so UIs need to consider all factors to give a smooth and valuable experience – need to consider bigger widgets

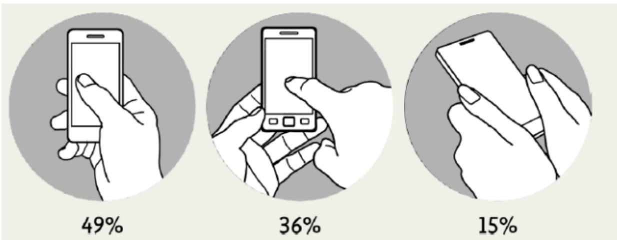


An old but effective story in design and testing of it:

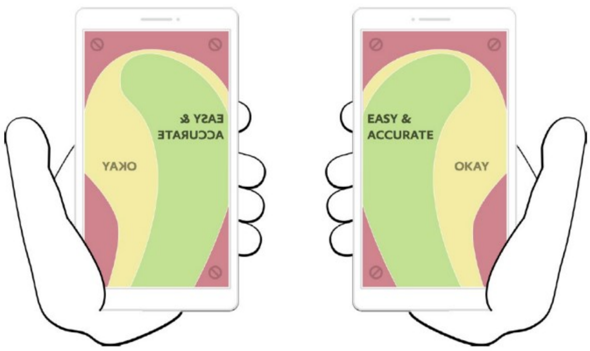
- In 1963 Bell's labs designed the new Touch Tone phone (here and here if you wanna read)
- They tested 16 different keyboards to find the best design, comparing them 3 by 3 with 16 users' groups
- Objective:
    - improve the following:
        - speed
        - accuracy
    - user preferences
        - create a good enough design
- This is still the way we use to test users' habits

The first consideration:

- is the *reachability* of different components of the interface depends on how we hold the device



Carefully choose the right position for widgets, both considering right-handed and left-handed users. This is the comfort zone for users, useful to know where to put the considerations.



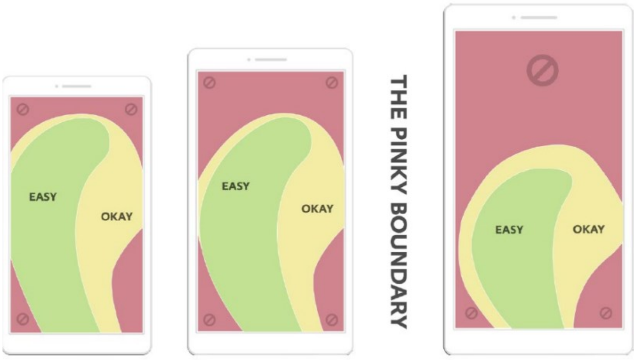
Exercise: try to guess interactions outside of the zone.

Written by Gabriel R.

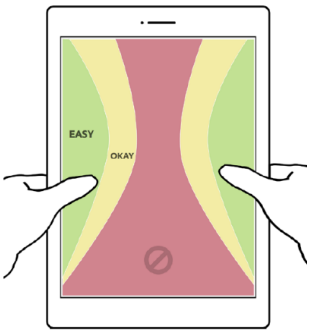
Solution: Actions like accessing status bar, use/close notifications or accessibility controls

- It's important to position buttons in areas where these can be easier to reach and where tapping can be more precise
- Where there is a double choice for the user between a primary and secondary one, it's important to position primary action on the right and the complementary one on the left
    - Also, primary action positioned above secondary one

Also, the *screen size* can be a problem, particularly because of the height. In case of a tablet, one needs to use both hands in order to use the device. That's why mobile devices should be used single handedly. The following figure specify how easy those zones are to reach.



With the increasing size of the screen, users tend to use the device with two hands for a better holding.



The *gorilla arm* problem is one which often arises: here, the user doesn't want to interact with vertical interfaces for long period of times, since its arms and shoulders tend to get tired after a while.

Quoting Steve Jobs: "Touch interfaces don't want to be vertical, it gives great demos, but after a short period of time, your arm wants to fall off."

- Here, it's important to make widgets big enough to always accompany the right action
- Vertical interfaces are useful in context of different interaction
    - e.g., Just Dance, Kinect (physical movement)
        - this allows you to clearly see what's going on
        - then interacting with hands

Increasing device dimension means increasing weight:

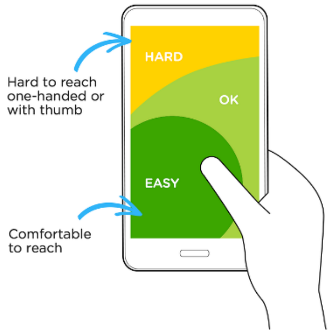
- 88% of tablets usage occurs while seating, against 19% of smartphones
- Tablets are used on a holding surface two out of three times

Big devices are used in a similar way to laptops but:

- Mouse is usually moved easily
    - whereas fingers are moved by the hand and require a higher effort
- It is essential to group controls together to avoid user's tiredness

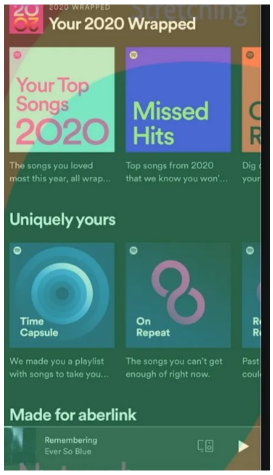
Written by Gabriel R.

It's important to identify the most frequently used controls and put them in the comfort zone (*thumb zone*), which allows to clearly see and use widgets.



- It is also very important what to put *outside* this zone
    - Controls for data modification to avoid unwanted edits or data loss
    - e.g., "logging out", "delete all", "delete account"

In classic website design the priority of the interface is given to interaction elements on the screen, like buttons clickable by a mouse.



- Touch interfaces change the interaction tool
    - the mouse cursor is tiny
    - but hands can hide part of the interface
- It's important to calculate encumbrance of hands and the space itself
    - Important data must remain visible
    - If necessary, hide the controls can be useful

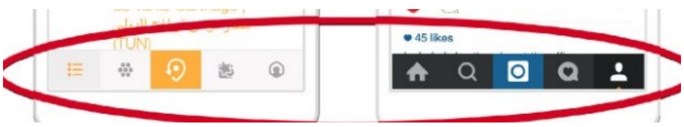
Consider how interactions are conveyed:

- Physical interfaces are made to convey a physical feedback
    - e.g., clicking and receiving the key press
- Touch interfaces are virtual, so you do not have buttons going down
    - give vibration or haptic feedback
- Give feedback to inputs
    - e.g., seeing for a brief time the key pressed after pressing it
    - provide an alternative for the covered content

As an operational guideline, consider the "*Content always on top*" rule:

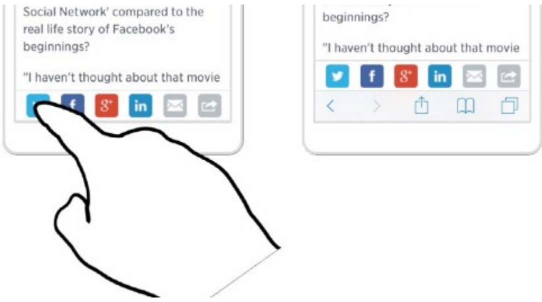
- This forces to leave the content in the center and move controls above, below, or on the sides
    - hand will not cover content
    - but in some cases, this can be a problem
- If an application requires to modify data frequently, then controls must be inside the comfort zone
    - also consider Android OS bottom bar
    - for iOS, you have the back swipe gesture
- Do not consider the normal user as a Computer Science user
    - so, never fall into the "we're all experts" bias

An exception to this rule can be made for apps like Instagram (figure below), where frequent data modifications may occur, in those case the controls must be inside the comfort zone. This is the case for most apps.



Written by Gabriel R.

You can clearly see the problem here; the thumb rule is important, so apps might go out of boundaries and not be placed correctly in widgets handling. This can cause app rejection in publication, particularly in contexts like Apple devices, where distribution is definitely strict, as saw before:



The difference can also be seen in a version of the Facebook app:

- On Android menu on top
- On iOS menu on bottom

Some *practical rules*:

- Android
    - Controls must be on the *upper* side of the screen
- iOS
    - Controls must be on the *lower* side of the screen
- Phablet (big height, in the middle between phone and tablet)
    - Controls must be on the *lower* side of the screen
    - It is possible to introduce a *floating trigger button* for frequent operations
        - figure shows it, like note apps or Musixmatch (app for song's lyrics)
        - you can also see, in the middle of it, a menu opened via a floating button
        - it does not scroll with content, but stays in its position
    - *Swipe* usage, especially for the tabbed layout

You can see the bar does not retrieve additional space covering the content, since it leaves content on top and floating trigger make easier to hide content and expand it only when needed:

About *tablet layout*:

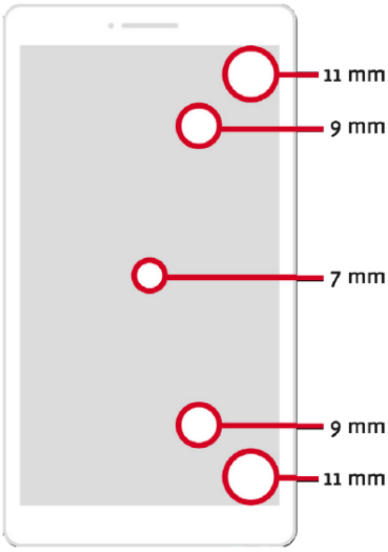
- With bigger screens, it is difficult to have a unique overview, but designer must consider that eyes move from top to bottom
    - Buttons for interaction must be at the top or on the sides of the device
    - Prefer corners at the top and not the center position
    - If an element controls the content
        - it must be below or on the side of the content, *never* above

- The bigger is the screen, the more precision and physical activity is required during interaction
    - It is crucial to reduce the number of interactions
    - Group together interaction elements

When changing medium, there can be a *different translation of known interactions*, like the one of hovering over an element with a mouse:

- With except to several proprietary solutions (e.g., the pen of the Surface or the Stylus pen by Samsung), the *hover* event is not available on touch interfaces
- Adopted Solution: the first tap is the hover event, the second is the real click
    - not important: the professor pronounces it as "huuver", but this is called "oover"
    - for the web, the CSS3 *hover* property allows more appropriate solutions
- So: if you really need the hover, use that
    - otherwise, effort increase

According to a Google study of 2013, 83% of websites provide interaction buttons too small to be used with fingers – so, desktop is very different from mobile once again.



- Consider human thumb has these sizes:
    - the minimum is 8 millimeters for a child
    - the maximum is 18 millimeters of an adult
- *Control size* needs to be measured precisely
    - It is a good idea to increase controls' dimension if an error requires more than 2 interactions, 5 seconds or a context switch to be corrected
    - The minimum size to use is 7 millimeters
        - which can be increased to 9 mm with big tablets
- Number of errors increase when sizes of buttons decrease

In figure, consider how *size and position* can *influence* usage and positions.

- Minimum size of the button depends on where the button is
- The dimension of the thumb is in millimeters, but this unit of measurement is not the best one for interfaces design
    - Table on the right details dimensions

Millimeters	Pixel	Em (16px)
7mm	44px	2.75em
9mm	57px	3.5625em
10mm	63 px	3.9375em
11mm	69px	4.3125em

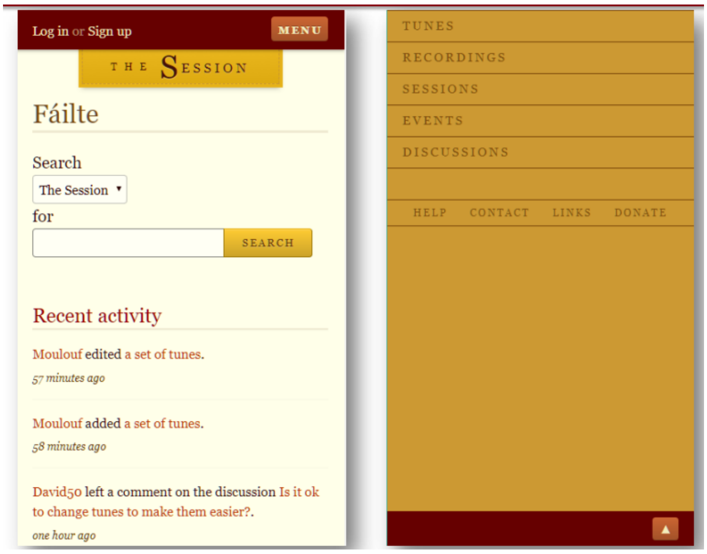
Do not crowd interfaces – less widgets, more content, more gestures. Even proximity between elements is a crucial element:

- If controllers are too close, they must be bigger to avoid errors
- If the elements are small, they must be far away to avoid errors
- Two buttons of 7 millimeters must be at least 2 millimeters away

One can use *just-in-time interfaces*, providing only what is necessary at that moment:

- The main operations must be available and selectable from a list (ex. menu, products list)
- The primary information must be easily available, with further details available with another interaction (*progressive disclosure*)
    - This approach allows clarity of the provided information
    - e.g., Google advanced search, file-sharing programs
- Try to foresee users' needs (e.g., context menu)
    - Usually order is: opening/sharing actions – inserting actions – mutating actions
    - Consider using safe zones
    - If possible, put icons and descriptions

Two UIs designs here:



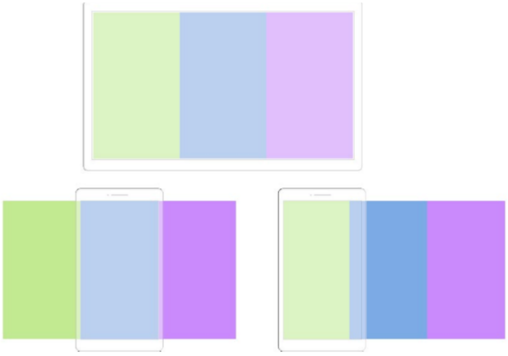
About *number of interactions*, with increasing bandwidth available and the availability of a local database, the number of taps is less critical:

- There aren't only taps
    - but also gestures! (normal or accessible-like ones, like Braille)
    - consider average sizes for gestures
- Distinguish between useless and quality taps
    - A *quality* tap is a tap that adds new information, completes a task, or simply adds a smile
    - The *garbage* taps are taps that could be eliminated with better interface design or substituted with gesture
- It is possible to add taps if they provide a better interface organization
- In case, teach gestures via tutorials

Written by Gabriel R.

A big issue in mobile design is related to *long pages*.

- For example, a website designed for a desktop environment and not adapted to the mobile world could fit three times in the window of a smartphone
    - requiring horizontal scrolling to be visited



An easy fix to this could be the *carousels*, which should be used with particular attention, or, better, avoided:

- Loss of overall vision in favor of details
- If it is not clear the connection between different objects, users do not understand what comes before or after and lose interest
- A study shows that 84% of the clicks occur on the first page
- Instead of forcing the user to make several swipes for finding the information, it is better to ask for a single tap to open a page with more details

Correct usage is via *swiping*, considering Zalando/WordReference:

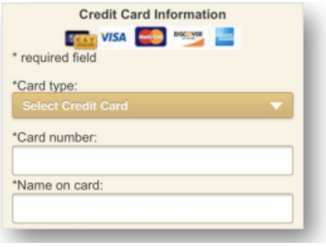
Carousels work very well in these cases:

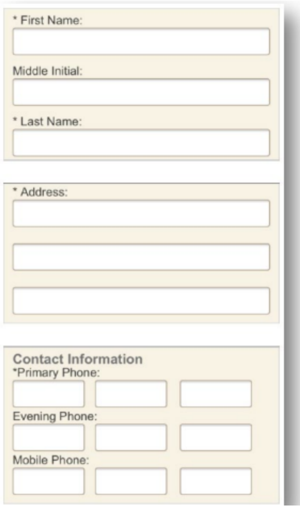
- Linear data: user knows what to expect (e.g., weather)
- Random browsing on interesting items for the user: pictures, slideshows
    - They work better if the user knows what to expect, for example, in a known context
        - All cases in which the user does not read to read
    - http://shouldiuseacarousel.com/

- To break up very long forms: in this case, they cannot advance automatically
  - In all other cases, do not expect the user to manually change slides
  - computational effort increases

Another problem in suitable mobile interfaces is the one of *long forms*:

- 21% of online shopping, which are not completed with the purchase, comes from the excessive complexity of forms necessary to complete all the steps
  - (1 over 5)
- Each field makes the difference: decreasing from 4 to 3 fields increases contacts of about 50%
- With touch interfaces there is not the tab control, hence each field requires one tap more and interrupts the flow
- Ask *only* for information you really need
  - e.g., do you really need the gender





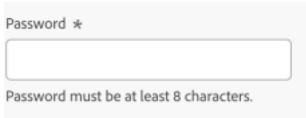
Generally speaking, the user is not fast when digiting, so if possible *avoid keyboard usage*:

- Provide the correct keyboard for a specific input
- Prefer a list of buttons to a menu if this one is short
- Avoid too long drop-down menu (a dataset is better)
- For numbers insertion, it is preferrable to show an average value with +/— buttons
  - if the real value won't be far away from the average (i.e. good for height, not for weight)

Some *guidelines for forms*:

- Don't forget the label
- Label should be above the input (because *content should be always on top*)
  - keyboard will go above the label anyway

- Instruction between the label and the input
  - put there, you probably cover what's under the field



- The following is an example under the input

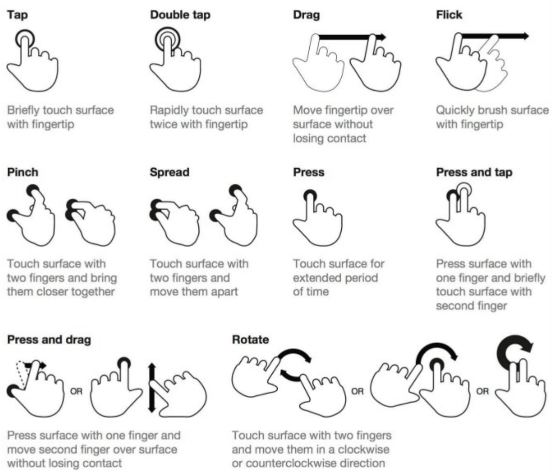
Not only we have smaller size of the screen, but a smaller size for input interaction: do not try to only port desktop interfaces to mobile.

Confirmation dialogs were introduced to let the user think about the answer, but today do not work anymore and slow down the user. It is better to use specific gestures, e.g., a *swipe* to answer calls or to unlock the device:

- It is sufficiently difficult to be only intentional
- It is sufficiently easy to be fast and avoids context lost
- Provides *undo* option
    - Most of the times, can happen by accident
    - Think about a phone call and a simple tap can make the call activate anyway
- To ask more attention to the user, increase gesture complexity

5.2 GESTURES

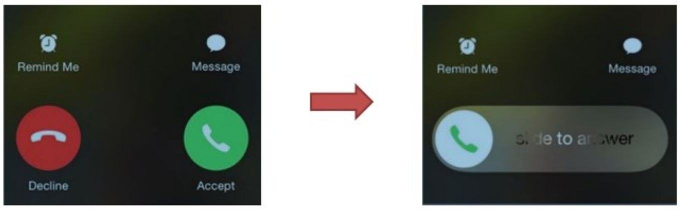
Gestures are introduced to make tasks easier for a user without being too complicated. We can distinguish many of them, e.g., considering the following figure which gathers them all:



So, in general we can describe:

- *Tap*: click for the "touch world". Interpreted as the hover event
- *Swipe*: frequently used for scrolling, view change, or show hidden panels
- *Long press*: used for context menu or detailed information
    - on Windows is equal to the right-click with the mouse
    - on Android opens the contextual action bar to select several entries from a list and fire events on all of them simultaneously
    - on iOS does not have a standard behavior, usually opens a contextual menu
    - keep in mind only expert users use it
        - or know it exists, given it's very non-standard and one may get it by accident
- *Long press + drag*: equals to drag&drop in desktop
- *Pinch/spread*: zoom in/out: semantic zoom uses pinch gesture as alternative of back button
- *Double tap*: zoom in/out, but can be used for other purposes
    - e.g., select an element and apply an action

Specific gestures are sufficiently “difficult” so that the action that is always intentional.

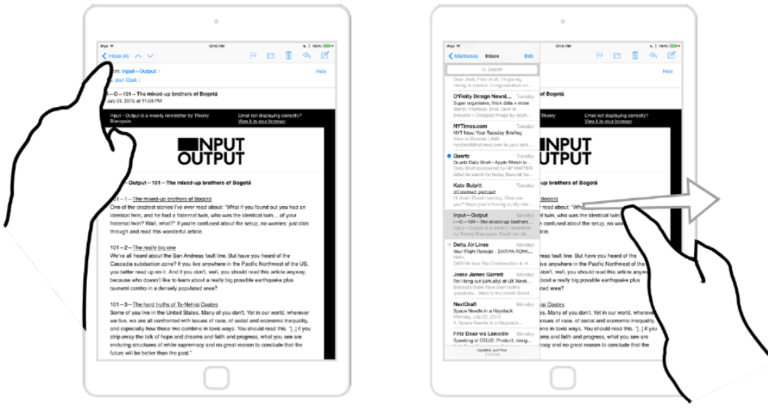


The “swipe” gesture from Apple is more error-safe, avoiding answering a call by accident: simple and coarse-grained enough (aka, requiring less precision). Gestures do not cover content.

5.2.1 Do we really need buttons?

- As you can see, using gestures basically we remove the need for that
- In mobile, give a way to interact with something without having to use a button directly
- Buttons also have the tendency to cover important content
    - meaning removing them could also lead to more usable space in the interface
- Buttons are generally less precise and confined to one specific action

In iPad’s Mail app shown in figure, the “Inbox” back button on the left requires a slight reach to open the message drawer, but you can also pull it open by swiping left to right anywhere on the screen, which lets you access the same content no matter where your hands sit. The entire screen becomes the control – no special trip to the back button required.



Moral is: *On larger screens like tablets, favor coarse gestures over fine-tuned pecking*.

The problem with gestures is the following: we miss the visual cues to actually do them. Consider the following examples:

- *Simon’s Cat* match-3 game
    - Gestures are intuitive and easy to understand how to control it
- *Earthworm Jim* platform mobile port
    - Controls covering the content and the action
    - Better suited for a gamepad

Gestures improve interface accessibility *because they tolerate less precision*. This is important in different situations:

- The elderly and children
- When the user cannot pay close attention to the interface
- Situation where it is necessary a fast interaction with no errors (games for instance)
- If the user knows the gestures, there is no need to watch the screen
- Good example in gesture usage and not buttons: Angry Birds
    - Pull the slingshot and simply release it in order to make the action work
- Consider Space Invaders: simple interface, using a spaceship to move
- The same principle was used inside the Boeing cockpit design
    - A lot of buttons needed and need for easy interaction
    - For the curious/crazy ones like me to read this stuff here
- When one decides to use a realistic interface, one has to follow it

Big gestures tend to become *reflexes*:

- Traditional interfaces are based on visual memory
- Touch interfaces use muscle memory

Consider *interaction with real objects*, buttons are used to control something not reachable (ex: fan)

- With touch interfaces, everything should be reachable
    - Before using them, always check if there is another way to manipulate the content
        - Consider the content as an existing physical object
        - Pay attention to external conventions (ex. salt and pepper, pictures gallery)
        - Follow the convention of the real world: *never cover the content*
        - People do not have to rely always on color or shape

The following are examples of wrong and right; which is the salt and pepper?

The shape might fool you, but the actual content is what matters the most. This is an interesting example and quite different from the others.

The *card* metaphor is frequently used and deeply understood by users:

- It is natural with applications dealing with business cards, plane tickets, coupons, etc.

Cards suggest several natural interactions:

- Flip the card
- Put them in a stack

Sometimes there is no direct correlation:

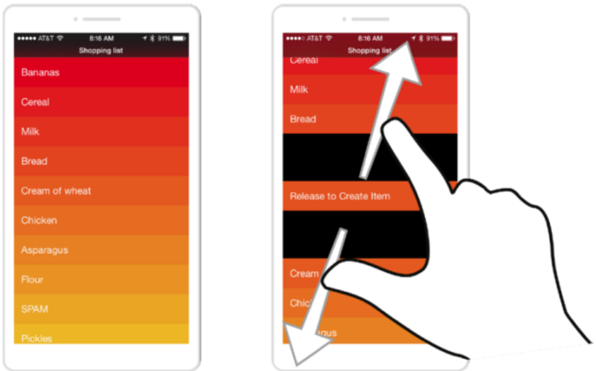
- In the real world, we do not flip the card clicking on a button
- Gestures are not always easy to find: help the user!
    - Make the interactions natural and flawless

5.2.2 How to help the user?

A good rule is to *get help from the real world*: if the gesture we want to use is the same as the real world, there is no need for instructions.

- Always necessary to follow physical constraints
- If we follow the real world, no explanations are necessary
    - Example: drawing apps
- Everything must follow conventions (do not betray user expectations)
    - Context is important
- Use single movements from well-known interaction tools

The following figure reflects a list of groceries app example:



In this case:

- To add a new item between two existing ones
    - it is necessary to drag away the upper and lower one
    - creating this way a new card
    - something like “inserting a book in the middle of a pile of books”
        - spread the others and insert yours into the pile
- It has to be taught to the user since it is not directly intuitive
    - but once learned it becomes very natural

Take *Paper* – an iOS drawing app which has gone to various technical lengths to recreate ink on paper.

- Often, though, you can borrow a familiar physical interaction without copying the entire original artifact
- You know how a knob or dial works:
    - Cranking clockwise means more or forward
    - Counter-clockwise means less or backward

- But if you want to use that *crank* motion, you don't have to add an actual knob to your interface. Instead, use the action as inspiration for a gestural equivalent
- The Paper app does this well:
  - Cranking two fingers counter-clockwise anywhere onscreen performs the undo action,
  - Traveling back in time to remove strokes from your drawing
  - Change your mind? Crank clockwise to redraw your strokes, as shown in figure
  - It's like "rewind – fast forward" with movies

Gestures should overall be easy to understand and to use by the user, but it is also possible to introduce complex gestures. These are also called <u>shortcuts</u> and are implemented when for an action the user need more than one touch.

- Even if it is important to use natural gestures, it is possible to introduce complex gestures that can be used as the keyboard shortcuts
  - Swipe to delete something instead of tap on the bin icon
- In this case, the design must prefer more natural but longer gestures
  - useful to use when multiple fingers are involved
  - but sometimes not supported by devices and it is not easy to learn
- It is possible to use gestures with more fingers
  - Display space
  - Multitouch support is not always optimal
    - Given they are more supported by more costly devices
  - More complex
  - Accessibility

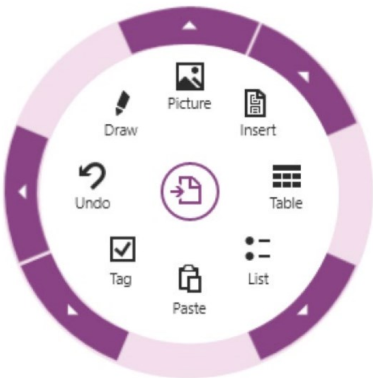
The *operating system* has the *priority*: gestures used by the OS cannot be used by the application.

Gestures are different depending on the OS:

- With Android always start from the sides
- With iOS can be completely inside the space dedicated to the app
- The *override* should be avoided
  - Means you don't override system gestures

To solve some problems related to gestures an idea may be introducing *circle menus*, which are frequently a good way to reduce conflicts between gestures.

- Easy to learn because of muscle memory, and fast to use
  - up to 8, effort is good
    - otherwise requires too much precision
    - for the smartphone, on tablet up to 18
  - in case, one can nest multiple menus of this kind
- Useful for primary navigation, context menu, tools, games



- Disadvantages:
    - Require more precision
    - Not scalable (on phones, 3 or 4 options)
    - Not easy to use the first time
    - Cannot change over time

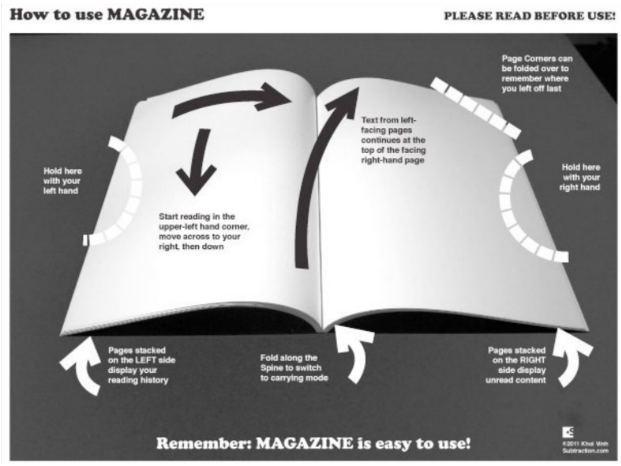
Above, the one used by Microsoft OneNote: each sector contains an item or a new menu.

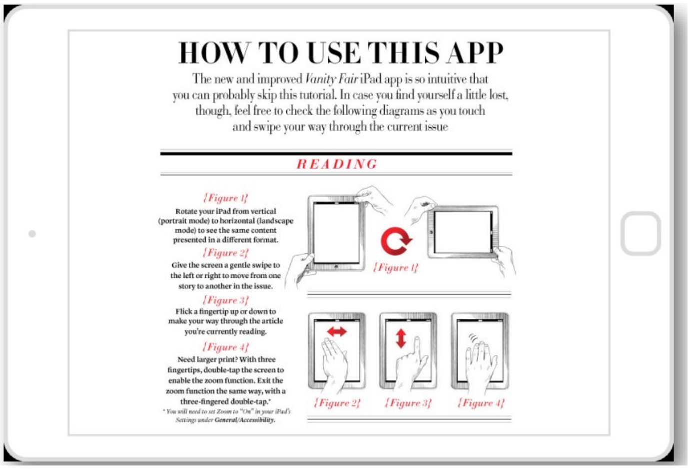
5.2.3 How to teach gestures

If we decide to stick with gestures, we need to ask ourselves how to teach them to the users, which bring problems to the overall usage of an application, depending on its context.

- Unlike buttons and widgets used with the mouse, gestures are almost always invisible
- Introduction of manuals is not effective (as well as impractical)
    - Who uses an application for the first time has a precise goal (he simply wants to do something), that usually is not reading the instructions
- Some gestures are trivial, but often after having discovered them
- Solution: just-in-time education
    - Practically, we do not send the information from the start, but we wait until it is needed

Below, some instances of visual manuals used in different contexts: physical world and mobile app.





Together with just-in-time education, it is also important to consider the idea of skeuomorphic design, which entails mimicking as many qualities as possible of real-life objects. It goes beyond shape and color, focusing instead on texture, lights and shadows, dimensions, and most importantly: use.

- It's no surprise that the term we use today comes from the Greek word “skeuos”, meaning vessel or container, and the English suffix -morphism, which means the state of being a shape, form or structure
- When it comes to UI design as we know it today, skeuomorphism had its boom with the advent of the first iPhone
- Steve Jobs believed that, to increase ease of use, UI elements should mimic real-life objects as much as possible

Consider that:

- The critical point is to choose the correct metaphor and not betray it
- The Skeuomorphic Design is not suited for gestures' teaching

The most famous example is the cycle bin as the trash icon in a personal computer. Others follow to make you clearly see what we mean by that (put by me to make you see instantly the concept):

Isaac Sanchez's synth plugin redesign.

Charlotte Doughty's music app gone retro.

5.2.4 Metaphors

The world of gestures is deeply linked to the one of <u>metaphors</u>. The use of those shapes the idea we have and how we want to communicate them.

- We use metaphors to be better understood
  - and the way we seek to be understood affects the way we think
- Interface design is full of metaphors:
  - they were the first tool designers used to teach how to use a computer
  - metaphors speed up and simplify the ways interface makes that argument

There are some *good reasons* to use metaphors:

- Metaphors bring the abstraction of software closer to life, making interface feel real
- A good design focuses on a special attribute of a feature
- Thinking in metaphors helps you thinking in systems
- The usage of metaphors is a great simplifier, both for concepts conveyed and usability

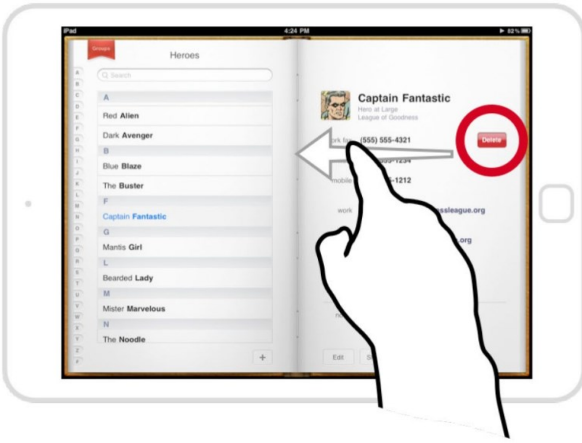
Metaphors use a *language* too, having different types of them.

- Consider the figure below: when asked to think about a table, each one of us thinks of a different table and view is personal and subjective of it
- Once you use metaphors, be sure this is the same way user interprets it

Design metaphors show up in different contexts and formats:

- *Visual metaphors* look like something in the real world
  - and help to understand how to interact with a tool
- *Linguistic metaphors* create a relationship between two ideas through words
  - often names given to features
- *Structural metaphors* help to understand how stuff is made/to simplify something complex
  - they are are used to rethink a design problem
  - sometimes they are not used in the app but during the design process
  - they help to avoid using too technical language

An example why skeuomorphic design is not a good idea to teach gestures to the users is given by the first version of Apple's contact book, represented in the following figure.



- In order to make it more familiar to the user
  - they tried to make it look like a real world contact book
- Firstly, the "Delete" button might be in the way of the gesture
  - and it might be better to move such an important button to a more appropriate space,
- Moreover, the user expects in this case that, if they turn the page, the contact page will change
- This is not the case, hence breaking the metaphor

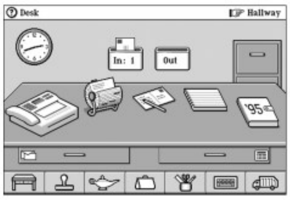
Another wrong example is the following:

- You have a book and a parchment
  - A parchment suggests me to scroll down
  - A book suggests me to swipe
- Using both leads the user to confusion

The biggest issue here is the fact that making interfaces way too realistic can be dangerous:

- Because usually the following equation stands: $Looks\ like \Rightarrow Acts\ like$
    - if this does not happen, the user will be confused
    - the inverse of this reasoning can stand
        - and it is why, nowadays, we see less and less skeuomorphic designs
- Differently, if an object $acts\ like$, it does not mean that it must $look\ like$
- Excessive realism can limit the possibilities
    - initial newspaper apps were not far away from simple PDF readers

From visual metaphors to linguistic metaphors:



- In Twitter case, we do not use birds, but use tweets
    - A concept existent only inside this app
        - meaning the “small exchange of messages” inside of this app
        - correctly describing the (once upon a time) small number of characters available to write anything
    - In this case, users create an identity specific to that app
        - this means increasing loyalty and forming a community
        - problem risen from the passing to Twitter to X
            - thank you Elon Musk once again!
- Desktop implementations
    - Macintosh “classic” one
    - a 3D desktop called BumpTop Mac
        - bringing a 2D interface to a 3D one
        - more here if you want to read about this one
- The universal “save icon” = floppy disk

5.2.5 Teaching

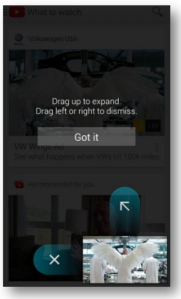
It is not true that if the user needs instructions, the designer has failed to design the interface:

- Better learning when doing
  - helps building muscle memory, reinforcing subconscious learning
- Teach gradually the gestures during the interaction

Videogames solve this problem very well since the user usually does not know what to do but learns while playing. We might see this one as a way to do “gamification”, actively reinforcing learning.

- *Coaching*
  - Telling someone what to do is not as effective as showing what to do
  - Coaching provides easy instructions to *learn by experience/by doing*
  - The key point is to understand if and when the skill has been learned, from that point instructions are a nuisance:
    - ask the user to make a gesture
    - ask several repetitions of the gesture, encouraging to do the same
  - It is possible to ask to repeat a gesture from 3 to a maximum of 5 times
    - depending on the difficulty of the concept – how much it’s abstract
- Mistakes made by users can actually be resources
  - Users’ mistakes help us understand when a gesture is *not* learned, so to understand if instructions are still necessary
  - An example is when a user stops during the interaction
    - since that could be a warning to toggle an animation
    - to show a complete gesture (e.g., the hand indicating what to do, ex. in games)
  - The best interfaces record when the user interacts, do not interact and the learning process, adapting hints and suggestions to the latter

A good example is the following:



☑ Give instructions for just a single element / task /step
☑ Very short text and simple instructions
☑ Ask the user to try

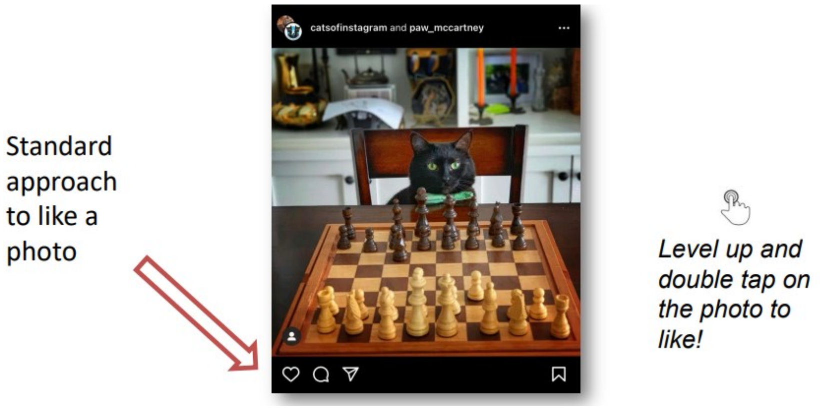
A wrong example is the following instead:

✗ Many features taught at the same time
✗ Too much text
✗ Discouraging

Written by Gabriel R.

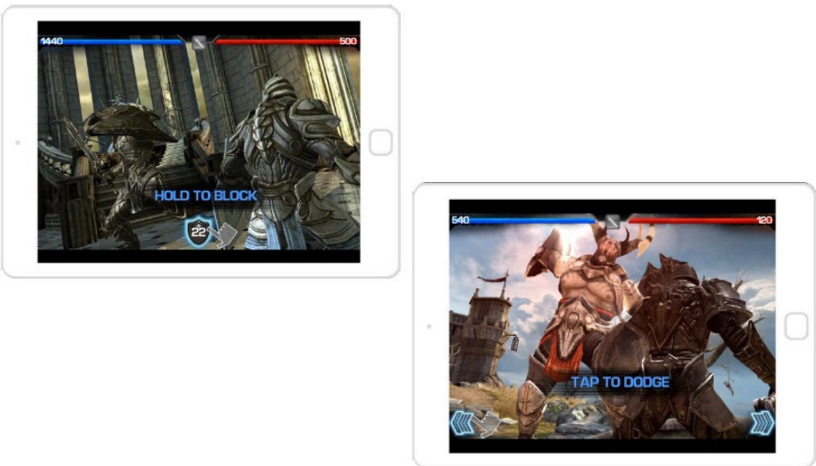
- *Leveling up*
    - Current guidelines for modern teaching suggest not to teach everything from the beginning, but to provide small-steps knowledge
    - This particular mechanism works in the same way:
        - Teaches only basic interactions at the beginning once the user needs them
        - Let the users use the complex gestures if they autonomously find them
        - Users are more motivated to learn something more complex when they need them
            - App must be organized in different levels of complexity
        - Provide the necessary time to learn

The following is a leveling up example on Instagram – user needs no explanation on the UI, for example on the “like” gesture. If he wants to do the same thing but faster, he can double tap on the image and obtain the same result (maybe, suggested after a number of usages – advanced gesture):

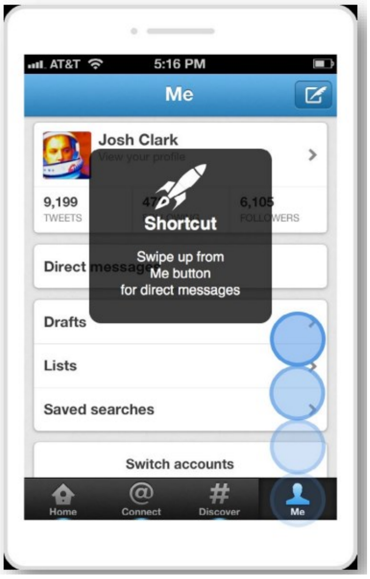


Different games provide this concept fairly well

- The *Infinity Blade* saga (figure) basically served this
    - Usage of natural gestures to parry and attack
    - Usage of more complicate gestures to make combos
    - When learning for the first time
        - time stops, so you have all the time to learn them
        - first holding, then blocking



- *Power-up*
  - Typical of videogames, power-ups provide facilitations to the user
  - In videogames, power-ups are gained, hence providing great satisfaction to the user
  - With mobile applications, power-ups are gestures that make interaction faster or easier
  - If users make the same gestures frequently, provide shortcuts to make their usage easier and faster overall
    - An example may be: once a user has tapped 10 times on a button to access a functionality
    - we can tell that there is a faster gesture to access it
    - providing an overall sense of satisfaction (right figure)
  - Teaching a new way to interact in an easier way provides the same satisfaction as with the power-up in a game
    - Think as a videogame designer and provide facilitating gestures as a reward
  - As for the likes for Instagram, do powering-up after a series of interactions



Concluding with the usual "hoping for a best future" fluff:

- Good touch interfaces design is still an open research problem
- There are no standards or precise guidelines, and these guidelines come from the experience of different designers and from what has been learned with errors
- Suggestion: experiment, fail, iterate, test, do stuff, break other stuff (a bit paraphrased, I know)

5.2.6 Not only touch! - Types of interactions

Mobile devices also allow other types of interactions besides touch:

- GPS can provide user's location
  - e.g., maps
- Accelerometer, compass, and gyroscope can recognize the movements of the user
- Other sensors can provide several data
  - e.g., luminosity sensor
    - can be used for accessibility purposes
- The camera can be used to collect data
  - e.g., translator, QR code reader, accessibility for blind people
- Fingerprints reader

The below example is used in the context of subways:

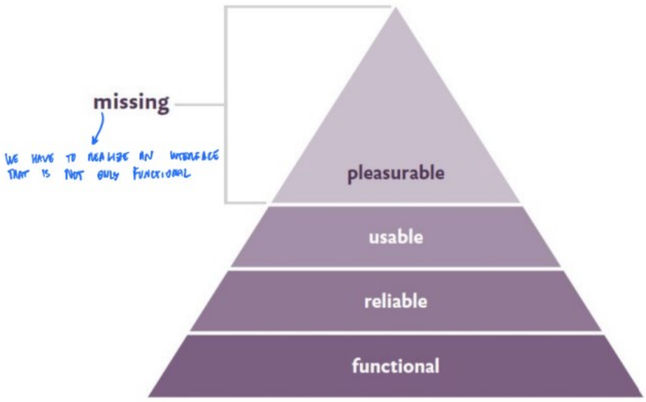
- ask to the app what line to take
- according to your position, the app puts with AR the arrows to make you go in a certain direction, reaching the specified one

5.3 Emotional Design

Emotional design is a new frontier of mobile UI design, since emotions are, at times, the best way to teach the user things about our application, conveying things even more naturally, touching the users' emotions.

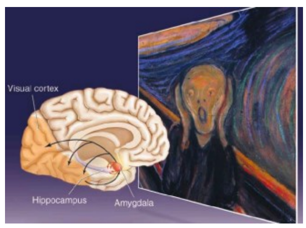
We all heard in some form or another about *Maslow's pyramid*, which suggests that human needs can be organized into a hierarchical structure, typically depicted as a pyramid:

This can be easily remapped upon users' needs, given for an app is not enough to be just functional, reliable or useful to be successful (courtesy of writing in below image by existing notes).



To describe in more detail the parts of the pyramid, we can describe applications as:

- *Functional*: if the user is able to complete the assigned task
    - "somehow" the user finds the way to make it work
- *Reliable*: if the system work, in this case every kind of failure is unwelcome
    - the user knows his data is secure and no data is stolen from him
- *Usable*: if it is easy for the user to learn how to use the system and its functionalities
    - usability is natural with not much effort
- *Pleasurable*: if the user experience in *pleasant* and the user is happy even unconsciously



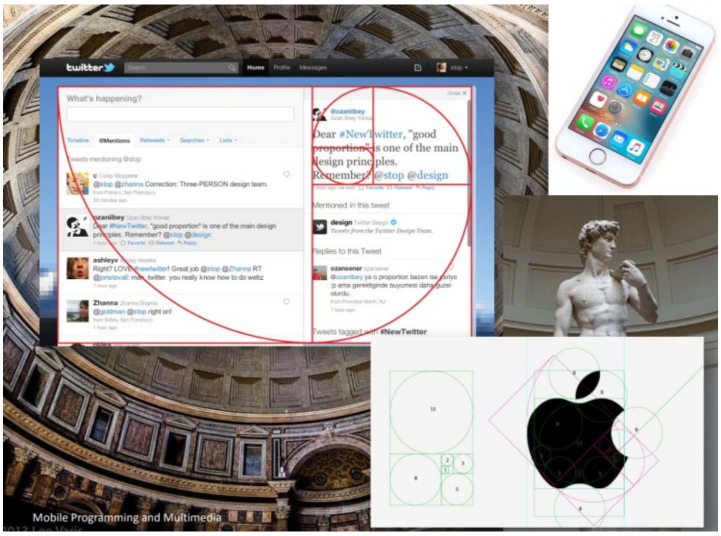
Amygdala is the oldest component of our brain:

- whose job is to answer to our needs such as hungry or thirsty
- this is the center of both memory and emotions
    - so, it would be nice to leverage it in the good way

Written by Gabriel R.

Emotions are essential for memory management because they are a sort of reminder, like using a post-it or a bookmark on a page of a book. This comes from using, once again, elements from the natural world, helping us build better applications (e.g. this lesson in which the teacher brings a rabbit – unusual thing and definitely remembered in the long-term memory – WIM memories, sigh, not sorry).

Studies have shown that people are attracted to elements positioned in the *golden ratio*, seen in figure as used by Twitter dashboard and Apple logo.



This is often applied to create aesthetically pleasing compositions that are perceived as harmonious and balanced, creating proportions balanced to different design elements (because it comes from nature).

Another natural element used by Apple was the patent of a status led that followed the human breath, an interaction that is not easy to notice but help internally relax the user.

The Volkswagen Maggiolino helps us introduce another topic in emotional design: its enormous success show that users are attracted to objects that are humanized. Same principle applies to Saetta McQueen from the *Cars* movies.

<u>Humanization</u>, something really similar to a human, is typically liked by users.

- Another good example for this phenomenon are the MM's
    - designing candies like humans improved the sails of the product,
- Also, using really big faces is a factor in the strategy of marketing
    - new-born babies tend to have bigger heads than adults in proportion to the rest of the body, which is an unconscious information we own as human beings

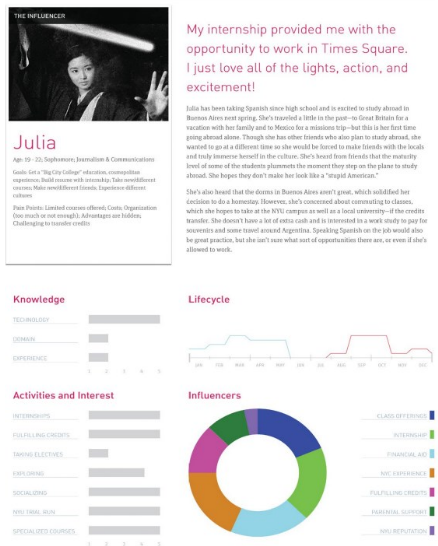
Written by Gabriel R.

- This, linked with the facts that humans protect the human children, and say yes to them, tend create a positive effect in users

A good technique is to imagine the brand as a person with a personality, and imagine how this person could speak, answer, act, etc. This is conveyed via *mascots* usage, like the one from MailChimp.

- In an application this can be used by making the mascot draw from a random pool of audio messages in a database
  - and execute them while the user is interacting with the application
  - creating an engagement
- Some customers may understand this trick though
  - making in reality pretty hard to apply mascots to applications
- Some others might fall for it
  - e.g. the customer quoted in lesson which shared with others how much of a good service MailChimp is given it complemented hair for the new haircut
- Other example might be Segugio.it
  - an Italian site to search for the cheapest insurance in Italy
  - empathizing with people, given the dog is man's best friend

5.3.1 If your app were a person, who would it be?



Firstly, before creating such a figure, we must create a *user archetype*:

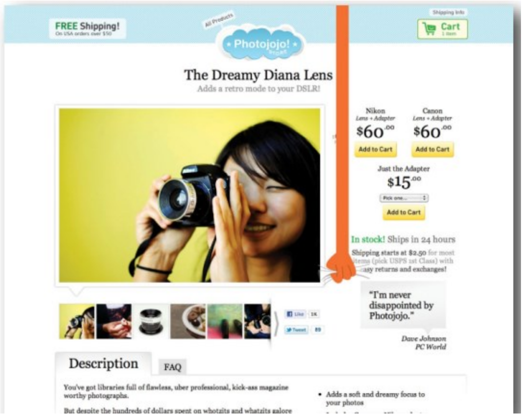
- Principles of user-centered design
  - we have to collect information about the brand image that we want to show
  - creating a blueprint of the behaviors and key traits
    - which also include the personality, and the language used by it
- This can be used on a mascot to get the user attention

More specifically, the personal data to collect is the following:

- Name and brand image
- Personality
- Language used
- Visual lexicon (font, type of icons, colors, etc.)
- How can I get his/her attention? Which emotions is the user more sensitive to?

The following is an example on emotions usage: basically, the website tells to "not pull" the lever on the side. Such control is useful to get more details on a single product. Of course, our natural curiosity makes us want to know more and one clicks (placing the hand you see here, revealing the content):

In same site, when you put items into the cart, it changes color (green) and becomes smiling (above you can see a bit).



5.3.2 Which emotions can we use?

There are no rules, but we can use *different emotions* depending on the context, given how powerful they are. Generally speaking, the most effective are:

- Surprise
  - Unexpected event making something positive
- Pleasure
- Preview
  - E.g., a movie trailer, used to set some expectations and create interest
- Status/Exclusivity
  - E.g., some kind of alpha/pre-access, bringing interest to the product
- Rewards
  - People are monkeys, so "give banana to monkey" always works, even when doing simple things – sorry, I like to be clear – designed to work on your memory, though

Remember: never force the user to change! Another good example of emotions is the following:

- This comes from Nike, which allows the selling of a particular set of sneakers only to subscribed members, or starts the selling at a certain time, creating exclusivity and preview

This is, though, a risky strategy:

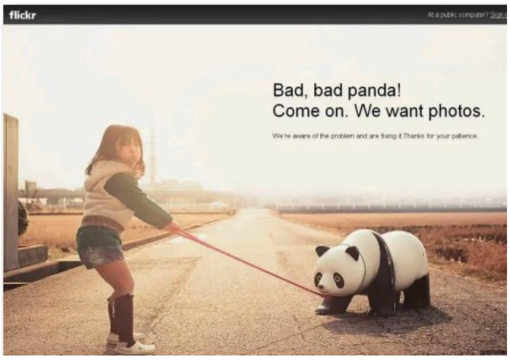
- since a person will decide considering pros and cons
- but, when these are not possible to measure, will just go out on instinct
  - and the major obstacles for this sensation are laziness and skepticism

A correct design will help mitigates these problems, but also the use of games (think of Duolingo and its annoying gamification technique – united with memes, I'd say, adding "humanity" to the whole context of the app "you all love and use") and incentives can help the user prefer our option.

For all of these strategies to work we need to check multiple things about the emotions we are using.

- Is the persona created for the brand correct?
  - While using a bank app, we do not want have mascots surprised at how poor we are
- Is our product too similar to other competitors?
  - This was the problem of Google+, which was way too similar to Facebook
- Are users' needs satisfied?
  - e.g., in social networks, if we are able to convince multiple users to stay in my application (or move to mine)
  - since the aim of social networks is to connect with friend and celebrities
- Is the language correct?
  - It is not always possible to use informal language in most applications
  - but generally when we have a disruption we should tell the truth directly (the 404 error)
- Is my application still usable, enjoyable and reliable?
  - After trying to invest into emotions to make the application more usable, are we still respecting all the other, fundamental, services?

Consider the following 404 page by Flickr – site for sharing images and photos of high quality usually, back in the day owned by Yahoo – you wouldn't want to see this from a bank:



It is not always possible to use informal language, but generally it is a good idea in case of disruptions. Always tell the truth!

The right-side is a good example always from Flickr: servers got flooded and badly damaged. They held basically a context depicting how to fill these images for the disservice given because of this.

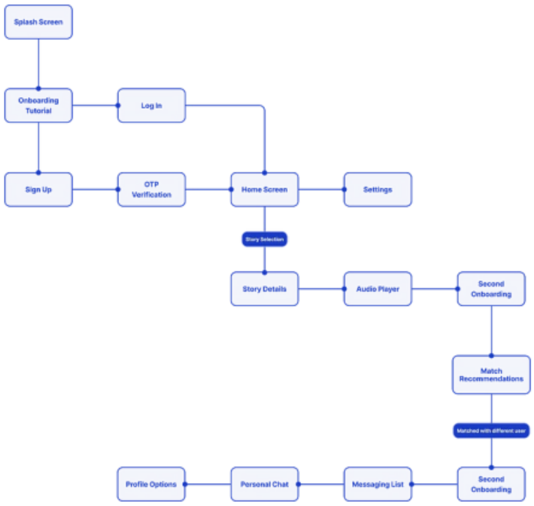
5.4 PUTTING ALL TOGETHER: THE APP AS AN OVERALL NARRATIVE

When you are thinking about how it might show up an app, remember that narrative is all about perspective. A story brings ideas to life by relating them to other ideas, thus finding relationships.

- Relationships can be:
  - *Sequence*: which idea comes first, and what comes late
  - *Theme*: how ideas are understood when looked at as a group
- If you consider the design of an app as a *storytelling*, this is a good way to discover *interactions*

A story can help to find some design elements:

- The app's information architect
- The repetition of words or themes across the app → keep the language consistent (same concept = same name)
- The user's journey
- The stories visual patterns tell



To create your app story, you need to find *pieces* and visualizing a concept beyond a single interface or flow and imaging all the other places where the concept may appear:

- Examine nouns (→ objects) and verbs (→ user's actions)
- Consistency in the descriptive content
- Call to action
- Test different ways of expressing an idea
- Count how many step the user must do

Exercise: Describe Twitter flow as you were giving directions to someone who was lost:

- On the first screen, you can see all the tweets that have been sent lately. One of them can catch you eye
- You click on that tweet. It is an article. Now you can see that tweets with all the replying tweets
- You can also see where you can reply. Click on a reply
- Now you can see that reply with all the replying tweets
- Those replies can have their own replies…

Solution:

- Imagine you're standing on a busy street, and you see a crowd of people gathered around a bulletin board. That's the first screen, where you can see all the latest tweets that have been posted. Look closely, and one of those tweets might catch your eye.

- When you see an interesting tweet, it's like walking up to that bulletin board and reading the message. Click on that tweet, and you'll be able to see the full article or content it's referring to. But that's not all – you'll also see all the replies and comments people have made about that tweet, just like a conversation happening around that message on the bulletin board.

Written by Gabriel R.

- Now, let's say you want to join in the conversation. Look for the spot where you can leave your own reply. It's like stepping up to the bulletin board and pinning your own comment to the thread. Click on the "reply" button, and you'll be able to add your thoughts or reactions

- But wait, there's more! Just like how a conversation can branch off into different directions, each reply on Twitter can have its own set of replies. When you click on a particular reply, you'll see that reply along with all the responses it has received, like zooming in on a specific conversation thread within the larger discussion

- And the rabbit hole goes deeper – those replies to the reply can have their own replies, and so on. It's like following a winding path of conversations, where each turn leads you to a new cluster of people adding their voices to the mix

- So, whether you're just observing the buzz on the main street or diving deep into the side alleys of discussion, Twitter's flow is all about navigating the ever-evolving conversations happening around the world, one tweet at a time

5.4.1 Have you finished? The icon!

The icon is really important because it is what users will see in the screen of their smartphones. It is the *most seen* element of the app.

- It is like a *post-it* to remember to use the app
    - Never use the stock icon of a framework when you deliver it to the prof, please!
- Good icons usually:
    - are built on existing brand iconography
    - contain the functionality of the application in a single image
    - show the name to strengthen the identity

Some examples which I think you know pretty well:

References used throughout the whole chapter (particularly suggested the first link):

- Steven Hoober, How do users really hold mobile device?
    - https://www.uxmatters.com/mt/archives/2013/02/how- do-users-really-hold-mobile-devices.php
- Josh Clark, Designing for Touch, A Book Apart, 2015
- Aarron Walter, Designing for Emotion, A Book Apart, 2011
- Erik Runyon, Carousel Interaction Stats
    - https://erikrunyon.com/2013/01/carousel-stats/
- Callahan, Hopkins, Weiser, Shneiderman, An Empirical Comparison of Pie vs. Linear Menus
    - https://dl.acm.org/doi/pdf/10.1145/57167.57182

Written by Gabriel R.

- iOS Human Interface Guidelines
    - https://developer.apple.com/ios/human-interface-guidelines/overview/themes/
- Material Design (Android)
    - https://material.io/guidelines/material-design/introduction.html
- Windows Design
    - https://developer.microsoft.com/en-us/windows/apps/design

5.5 Exercise 2 – Mobile Design

Create groups of two students and create a presentation with 2/3 slides with:

- *example of positive design of a mobile application (correct usage of gestures, proper metaphor, correct use of instructions, just-in-time content, ...)*
- *example of a wrong design*

Provide at least one screenshot for each of the two examples and a brief description of the main points and why the design is correct or incorrect. For the wrong example, provide a possible solution to resolve the problem.

The presentation will be held during the lesson on 2nd or 3rd May (max 7-8 minutes for each group).

Submission must be the same for each group member, so please write the other student's name in the first line. The name of the file must contain both the students' names.

What I did

Use as suggestions slied 85-86 of "Mobile Design" set. You can also use a single app, not mandatorily different apps and even screens not related to the specific situation you want to show and the suggestion (of course) is starting from your usually used apps.

Most people were criticized because they didn't criticize Mobile Design, but more the UI itself; what you want to criticize is mainly the UI, because of the UX missing pieces.

5.6 Exercise 3 – Design an app in laboratory

Some info about this one: held approximately at half of April in the laboratories (we were never told about this exercise upfront, so that's for you to be prepared instantly).

Basically, what she asks is to create the design of an app. This has to be done in pairs, so decide one to make the group with, then do this exercise.

- We will have an hour to create the app design
- This one was "The exercise asked to design an app for interaction with scale"

Some hints:

- Remember the design guidelines we have seen
- Take inspiration from other apps (not so much!)
- Who is your audience? Create a persona
- You don't have to develop it so use your creative talent!

Written by Gabriel R.

On Moodle, this one is called "Exercise in lab".

Remember a few things:

- You have one hour, you can't do anything particularly complicated
- You can use Figma, Balsamiq (in Windows with license), Canva, whatever

You can send your assignment within the full hour, so no worries. After, you will enter a Zoom sharing your work and discussing with others. You are not forced to participate, there will be time only for a few people.

Main things to remember:

- specify the goal of the application
- understand who the person is doing this app
- think about the ethical concern of the app created
- think about how colors are used overall and gestures

Other things to note:

- the teacher will just make some people present for the sake of time, not everybody
- in the next few days, the evaluation will be uploaded

I write everything down for you to be organized upfront, given she basically told us the same week we had to do this without considering telling us what programs to use and how to use them: squeezing them in an hour without saying anything is not good. This way, you can probably do a better job than what I did.

My advice on this:

I got "Not approved", given a comment "The exercise asked to design an app for interaction with scale". Carefully read the assignment, pay attention! We basically said: "Why is everybody doing an app interacting with scale?" Like wake up me, it was the exercise theme!

- Prepare yourself adequately, refining some design tools I listed and already having a ready idea to show. But definitely, carefully read the assignment (slides shown only for a minute, so be careful)
- In the next few days, when you will receive the evaluation, you will be ready enough thanks to this file

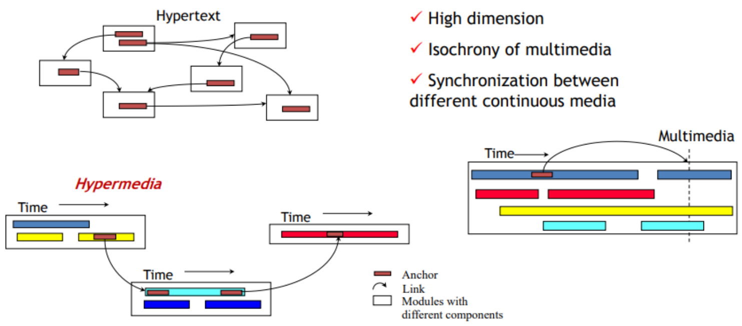
Write the names of both components inside the form, so you do not need to send an email doing this.

6 3 - MULTIMEDIA DATA ENCODING

(The part can find some references and useful material in the reference book (particularly for compression algorithms) "Fundamentals of multimedia" by Ze-Nian Li)

The way we perceive the world is a combination of media and different representations, combining hypertext (traditional Web structure with non-sequential links) and multimedia (integration of different media types like audio, video, images, and animations).

- We then talk about hypermedia, combining both hypertext and multimedia
- Compression is not a choice, but is a need to represent multimedia data in a good way, to preserve quality, synchronizing different continuous media
  - Data would be so huge without it (*high dimension*)
- There's the need of synchronization of different continuous media (video, audio, animations, and so on)
  - *Isochrony* of multimedia to link different continuous media streams
  - E.g., when we need to send a video, it has to be encoded at exactly a specific number of frames, otherwise it will not be fluid enough
  - There is no problem of synchronization anymore
    - it depends strictly on the type of media represented
    - given the HCI (Human-Computer Interaction)



A *multimedia system* is a system for storage, integration, and management of heterogeneous complex information, to represent the real world through multiple modalities, making information usage more efficient. This implies making space for multiple and different information modalities, like:

- text, graphics
- recorded or synthesized audio
- animations, fixed or motion images

It's not actually easy to search inside, for example, video data:

- You have to look for a particular actor?
- You can use metadata for instance
- But definitely search is not like normal data
- This would need analysis techniques, computationally complicated most of times

Consider the following: "Media are humans' extensions: technologies and products giving our senses new possibilities to receive information" – McLuhan.

What is intended with media?

In the last years, the meaning of “media” has changed following the evolution of technology:

- In the visual communication domain, it represents the different means of communication
    - print, radio, television, cinema, etc.
- Non computer-based multimedia presentations have different sources of media
    - e.g., artistic performances
- With the first computer applications, the mean of communication was only one
- The term “medium” specified the domain used to represent the information: text, image, animation, sound
- The increase of different informative channels in the modern multimedia system brings back to the original terminology

The innovations introduced by multimedia are not limited to the technological aspects of the problem:

- Complex interaction between users and the informative system
    - How can we highlight a link in a video or audio file?
- Information humanization
- Temporal dimension
    - Human beings prefer permanent objects, that we can keep in our hands, how can we use properly audio and video
- Inappropriateness of permanent copies

The production, organization, and distribution of information requires the development of new competencies:

- methodology: behavior models of information access
- technology: development and support tools, standards
- dramatization: emotional interpretation of information
- psychology: efficacy, privacy, control

This is not without *some problems* however:

- Higher expressivity means higher complexity
- The learning curve and technical management can penalize product usage
- Higher ease of use can increase products with low internal quality
- Increasing flexibility and variability of exploration can be energy consuming and confusing
- The efficacy of the system is strictly connected to who provides the information

Consider the problem of information representation:

- In the past, different multimedia encyclopedias existed
    - basically, Wikipedia in CDs
- It was good but never successful, given it was very complex for the user
- So, having a good thing is not enough, it’s important to represent it in an easy way
- In general, between all media forms, things change
    - Web/Internet/Network systems/Distributed systems
    - what are the real differences?

As soon as the complexity if a specific multimedia format increases, we must do a choice both for compression and representation.

- But also, too much flexibility and variability of exploration can be energy-consuming and confusion
- This has to be something which is natural for the user, given they will be able to understand the data lost and the “true” data

We have to make compromises, so we will get a compression which loses data – considered “lossy”.

- There is the problem of permanent copies of data
- You have to find a format in which to represent data without wasting energy and losing data or adding useless data
- For example, representing a movie and save it with a poster; this was not taken inside of the video
- Can be also with a song; you don’t have “the way” to restart from there again
- The problem of something which does not evolve in time, but represents data, basically
- So: huge data and you need more data to have it compressed

Multimedia technologies are widely independent from the Web:

- Information *representation*
- Information *compression*
- Information *transmission*

Nowadays, still, we will apply multimedia strictly to a Web scenario, meaning we need to clarify the differences between WWW and the Internet.

- The first is a distributed application, which uses the Internet as a technological infrastructure. Internet brings to the table services, protocols, remote storage systems, and so on
- While WWW provides the user interface, the applications and the environment for distributed applications
- Multimedia systems are related to both worlds:
  - YouTube, for example, provides an interface for videos
  - but also is a storage system, meaning these boundaries are not always so well defined

Web applications are not the most critical, and do not require the highest resources:

- E.g., video on demand
- E.g., multichannel communication
- Web evolution is imposing new goals and so new quality standards

With multimedia, we have to concentrate on the problem of reproducing huge amounts of data

- When there is data sent between servers and clients, there is a noise added inside frames to avoid packet loss and congestion
- There are single transmissions completely different from one to the other

6.1 Media Classification

The classification of topics is the following:

- Media classification
    - Based on content
        - text, image, audio, video
        - text is the simplest case
    - Based on media dynamic
        - diffusion and media usage
        - protocols and network technologies
- Media properties
- Information compression
- Images
    - Overview, GIF, JPEG, PNG
- Audio
    - Overview, psychoacoustics, MP3, MIDI
- Video
    - Overview, MPEG, H.261, H.263, DivX, XviD

The media classification will be the following:

- Based on *dynamism*
    - Static media
        - Information does not change over time
        - E.g. images, text
    - Dynamic media
        - Information changes over time
        - E.g., audio, video
    - Temporized media
        - Information changes over time subject to temporal constraints
        - E.g., webinar, live streaming, video seen only in a particular moment
- Based on *type*
    - Image
        - Encoding, compression, quality
        - Pictorial (pixel) vs. geometric (vector graphics)
    - Audio
        - Encoding, compression, quality
        - Voice vs. music vs. generic
    - Video
        - Encoding, compression, quality
        - Animation vs. video clip

size vs. time vs. quality

Multimedia has different encoding goals:

- Reduce size of a persistent data representation
- Optimize transmission time over reproduction time for continuous data
    - when compressing, can take time
    - when decompressing, be careful to take as less time as possible
        - so the user does not wait a lot when doing playback of data
- Fast decoding for data encoded on conventional and not conventional architectures
- Quality control of decoded data
- Provide support to replace missing data

6.1.1 Media Properties

Considering a dynamic media classification, we start from *static media* features.

- User reads information with his/her own timing
- Data transmission does not impose temporal deadlines (but with reasonable limits...)
- Persistent information
- Users can read the information several times
    - e.g., text, images
- Data size is not a problem
    - Higher size = higher time
    - compression / decompression
- Information diffusion and usage takes place in separate moments
    - download / display
    - save on local storage
    - caching

For *dynamic media*:

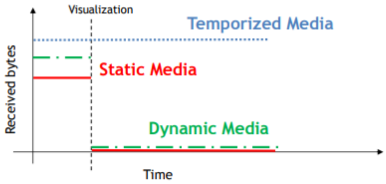
- User must follow information evolution in real-time
- Diffusion can be delayed, but information meaning can change (e.g., games)
- Information is not persistent (sometimes can be reproduced several times)
    - e.g., animated presentations, YouTube, video files
- Information size is frequently big, and should be used within a reasonable, predefined time
- Information diffusion and usage can happen at the same time
    - download vs. streaming
    - proprietary solutions
    - user interaction control
- Like before, we can download and consume later the information
    - but more commonly we are in a streaming scenario, where proprietary solutions are usually preferred

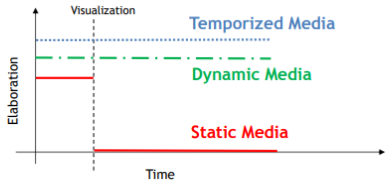
About *temporized media*:

- User must access the information following the timing schedule of the information
- Diffusion must follow media *isochronous timing*
    - it must be reproduced at a defined fixed frequency to be understood
- Information is not persistent

- The data cannot be used twice
    - E.g., audio and real time video
- Information size is big and proportional to reproduction time
- Data diffusion rhythm must be controlled
    - Real-time
    - Streaming
    - Bandwidth

The first image highlights what happens when we use the network bandwidth and the CPU: static media uses the network when it is downloaded and the same is for dynamic media (transferred to the client side, then it's playback). For temporized media, the network is played during the whole media reproduction. The second discusses the complexity of running such files: static/dynamic once downloaded, it's actually it, while streaming files/Zoom calls use the CPU for the whole playback.





6.1.2 Data Compression

About *data compression*:

- Multimedia data size is too big to efficiently transmit and store data without using data compression
    - CD audio quality (2 channels, $16b$ at $44.1\ kHz$): $172\ KB/s$
    - VHS video ($382 * 288 * 16b$ at 25 frames/s): $5.2\ MB/s$
    - NTSC video ($640 * 480 * 24b$ at 30 frame/s): $26.4\ MB/s$
    - Full HD video ($1080 * 1920 * 24b$ at 25 frame/s): $148\ MB/s$
- The main goal of data compression algorithms is to reduce storage hence reducing transmission time
    - Decompression is necessary
    - Data compression can be in real-time or deferred time
    - Decompression needs to be in real-time
    - Not necessary to save all the information contained in multimedia data
- Storage reduction can be achieved through:
    - The reduction of the number of bits necessary to encode the information
        - *entropic* compression
    - The reduction of the information to memorize or transmit
        - *differential* compression, semantic compression
            - saving only some information so to use less space

- Compression can or cannot preserve the original data
    - Compression without information loss (*lossless*, reversible)
        - based on coding redundancy of information
    - Compression with information loss (*lossy*, irreversible)
        - based on perception redundancy of the data

Compression happens through three *phases*:

- *Transformation*: information data (ex: pixel of an image) are transformed in a domain that requires (can require) less bits to represent data values
- *Quantization*: obtained values are (can be) grouped in a smaller number of classes and with a more uniform distribution.
    - This step is not reversible
    - This is the phase that actually reduces the data during the compression
        - meaning it is not present in the lossless compression
        - while lossy follows all the three steps
- *Encoding*: information is encoded (represented) based on the new classes and values, together with the encoding table

Image, audio and video, no matter what, all require an encoding and compression, which will influence the final quality of the object.

- We will study encoding algorithms, which have different advantages in term of size of the compressed object, the time required to compress it, and the quality obtained.
- There will always be a trade-off, meaning usually if we are able to have a low encoding weight we will have lower quality

Specifically, here listed the different lossless compression techniques:

- Run-length encoding (*RLE*)
    - Encodes sequences of the same value using a lower number of bits
- *Huffman* coding
    - Assigns a low number of bits to the more probable values using a fixed coding table
- Lempel-Ziv-Welch (*LZW*) Compression
    - Dynamically builds a coding table with a variable number of bits associated to fixed size sequences of values
- *Differential* coding
    - Each data is defined as the difference to the previous one (with linear or non-linear resolution)

Then, here are listed the different lossy compression techniques:

- JPEG Compression (for *images*)
    - Applies to the images a transform to the frequency domain (Discrete Cosine Transform) to suppress irrelevant details
        - reducing the number of bits necessary to encode the image
    - Possible lossless compression with predictive techniques
- MPEG Compression (for *videos*)
    - Encodes some of the frames as the difference to the expected values calculated with an interpolation

- MP3 Compression (for *audio*)
  - Based on psychoacoustics characteristics of human hearing, to suppress useless information

6.1.3 Run-Length Encoding (RLE)

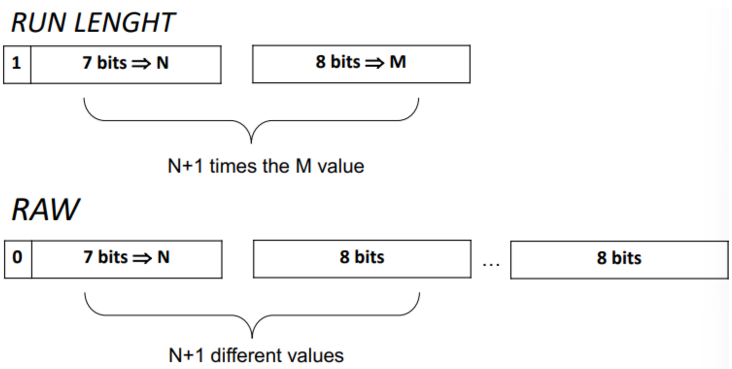
Instead of assuming a memoryless source, Run Length Encoding (RLE) exploits memory present in the information source:

- When data contains a lot of consecutive repetitive values, there is a lot of redundancy that can be canceled (e.g., a good example of this is a block of the same color in an image)
  - Long sequences of the same value can be substituted with a repetition symbol, the value, and the number of repetition
  - Compression is efficient for sequences of at least three equal values
  - Suitable for images with little details (ex: a uniform background)
  - Implemented in the bit-map BMP format
- In the RLE, the first packet has a bit set to 1, in 45 order to show that that is the RLE packet, followed by the number of times the character is repeated. An example of this can be:
  - *BGFDDDDDDDDDIJUPPPPPHYTGBUYYYYINNNNNNGHHHHHHHKPPPPPPP*
  - *BGF@9DIJU@5PHYTGBU@4YI@6NG@7HK@7P*
- According to ASCII, we can save on the number of characters using less bytes in memorization
- Using the "@" symbol, after there is the repetition of the letter
- But it strictly depends on the context
  - If these were email addresses, this can't work
  - It works only with a very limited domain, and I can find the symbol to use the repetition

The above description is the one-dimensional run-length coding algorithm. A two-dimensional variant of it is generally used to code bilevel images, using the coded information in the previous row of the image to code the run in the current row.

- We have a more general solution here, more used: two bytes, with the first byte used to write the number of repetitions or the number of characters that are following, the second one is the actual content.

For the first byte, we have two different kinds of byte: the run-length byte, beginning with the \ while the raw byte begins with 0. If the characters are not repeated there is 0 and considers the number of different characters between each other (e.g., if I have 10 chars, I add then 10 representing 0 for the raw byte and then 10 to represent the different chars – content not repeated).



Coming back to our example (report below from slides), it means we will have a byte with number 1, then in 7 bytes I will have the number 3, then 3 bytes (one for B, one for G and one for F). Then there are 9 repetitions of D, in this case, I have the first byte with number 1, then 7 bits in binary representation of 9 and then 1 byte with the letter B.

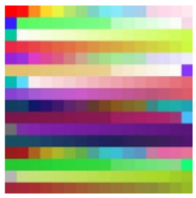
- BGFDDDDDDDDDIJUPPPPPHYTGBUYYYYINNNNNNGHHHHHHHKPPPPPPP
- BGF@9DIJU@5PHYTGBU@4YI@5NG@7HK@7P

This means the repetition needed two bytes: one for the number of repetitions and one for the actual letter content. The compression will add a byte every 128 byte; if there are non-repeated chars, the compression will increase the size. The repetition adds 2 bytes basically, so having 3 chars, because 1 is not repetition, the other uses the same number of bytes and for 3 repetitions, the compression can be considered efficient.

We list the different pros and cons:

- Pros
  - Suitable for Artificial images or few details
- Cons
  - Not suitable for
    - Text
      - since there is no language that repeats 3 letters
    - Images with several shades of color or high level of contrast

Enlarged image of a 16 x 16 pixels file, with 256 unique colors.
This file, saved in BMP not compressed, is 822 byte.
Saved instead in BMP format, using LRE algorithm, is 1400 byte, 1,7 times more the original size.



6.1.4 Entropy and Shannon-Fano Algorithm

The entropy of a source of information (e.g., a file, a streaming, etc.) is the measure of the *quantity of information* in the bits flow that has to be compressed = number of bits (theoretical minimum on average – giving this formula will give us a rational number, which needs to be approximated) necessary to encode an information source with alphabet $S = \{s_1, s_2, \dots s_n\}$:

$$H(S) = \eta = \sum_i p_i \log_2\left(\frac{1}{p_i}\right)$$

- p_i is the probability of the i-th element
- $\log_2\left(\frac{1}{p_i}\right)$ is the minimum number of bits necessary to encode the i-th value so that it can be recognized between other elements – also called self-information

Based on the probability of finding a specific value inside a sequence, we can create an entropy encoding, which encodes a value with a different number of bits based on its frequency (which means more frequent symbols will use less bits to be identified).

- Consider as example:
  - an image with uniform distribution of greyscale $p_i = \frac{1}{256}$
  - hence 8 bits are necessary and sufficient to encode each gray tone

From Shannon's Theorem, for lossless compression, the lower bound of the average code length (hence the compression rate) is the entropy of the information source:

$$H(S) \leq L$$

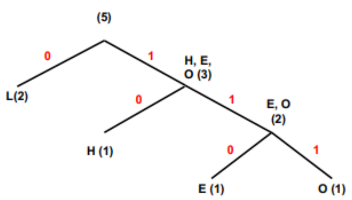
This means we cannot go below the number of bits of the original file, since otherwise the compression would not be lossless anymore. So, lossless can be done always, but this is not efficient.

Since the entropy indicates the information content in an information source S, it leads to a family of coding methods commonly known as entropy coding methods. As described earlier, variable-length coding (VLC) is one of the best-known such methods. Here, three algorithms will be described.

First, we can consider Shannon-Fano algorithm, which creates a binary tree with a top-down method. Each node is a symbol or a group of symbol:

- Each node is a symbol or a group of symbols
- It sorts nodes based on the number of occurrences
  - since we assume that we will need a different number of bits to represent 'E' and 'Z' in an alphabet like the English one, or even the "A" for Italian language
- Then recursively divides the set of nodes into two parts
  - each containing, approximatively, the same number of occurrences
  - until each part has only one symbol

In this example, we have the encoding of the word "Hello", having for each character the number of occurrences and bit encoding. We start from the root and the list is recursively divided into two sets (they are called sets in the book, but see them more of ordered lists) with same number of occurrences:



H	E	L	O
1	1	2	1
10	110	0	111

N. of Bit = 10 (theorical 9.6)

Then, to calculate the total number of bits occupied: H appears once and occupies 2 bits, E appears once and occupies 3 bits, L appears 2 times and occupies 2 bits and lastly O appears once and occupies 3 bits. Hence, summing $2 + 3 + 2 + 3 = 10$ bits.

It should be pointed out that the outcome of the Shannon–Fano algorithm is not necessarily unique since it can easily lead to different trees.

The Shannon–Fano algorithm delivers satisfactory coding results for data compression and given the chars are always at the leaves, these can be easily defined as prefixes of other chars. But this was soon outperformed by the following one.

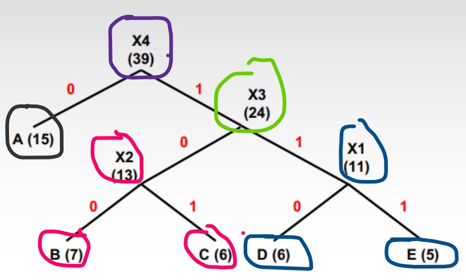
6.1.5 Huffman Code

First presented in 1952, Huffman encoding attracted an overwhelming amount of research and has been adopted in many commercial applications, such as fax machines, JPEG, and MPEG. In contradistinction to Shannon–Fano, which is top-down, the encoding steps of the Huffman algorithm are described in the following *bottom-up* manner:

- 1. Initialization: put all symbols on the list sorted according to their frequency counts.
- 2. Repeat until the list has only one symbol left
    - (a) From the list, pick two symbols with the lowest frequency counts. Form a Huffman subtree that has these two symbols as child nodes and create a parent node for them.
    - (b) Assign the sum of the children's frequency counts to the parent and insert it into the list, such that the order is maintained
    - (c) Delete the children from the list
- 3. Assign a codeword for each leaf based on the path from the root

For the following example:

- It creates a binary tree with a bottom-up method
    - Each node of the tree is a symbol or a group of symbols
    - At each step, takes the two nodes with the lowest probability, creates a subtree with probability equals to the sum of the probabilities of each node
    - Labels right branches with bit 1, left branches with bit 0
    - The code of each symbol is the label of the path from the root of the tree



simbolo	#	$\log_2(1/p_i)$	codice	# di bit
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15

N. di bit = 87 (teorico 85.29)

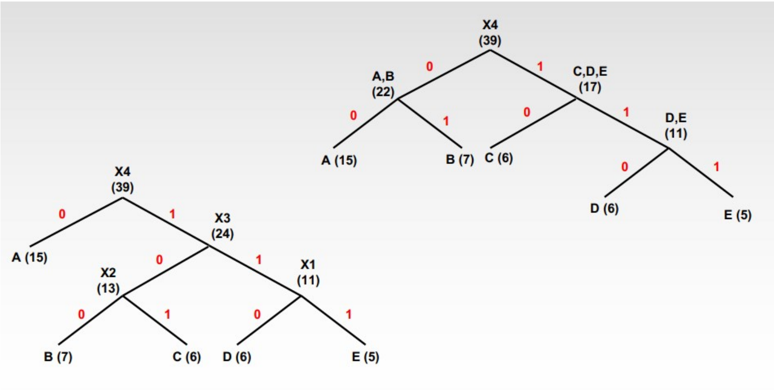
In the example we have 39 characters, taking the lowest-occurrence node (D) and (E), having the root as the sum of the two nodes. Again, we do the same with B and C forming a new tree this way.

- The more the tree is unbalanced, the more we have associated characters appearing more in our file, using less bits
- The algorithm itself is optimal, representing with the same number of bits given by the entropy representation
    - Huffmann is optimal, while Shannon-Fano is not
- The chars appearing more have less bits, while the least frequent have more bits

Other things:

- 111 is E, given we go down from the root, 110 is D, etc.
- then, we multiply in number of bits the number of occurrences
    - e.g., $B \rightarrow 21\ is\ 7 * (100) = 7 * 3\ = 21$, C is $6\ *\ 3\ =\ 18$, etc.
- hence, we get 89 as occupation

As you can see, the final result is really similar to the Shannon-Fano one (sometimes the same results, sometimes different). What you see in the figure can be described as follows:



N. of bit = 89 (87 Huffman, theorical 85.29)

What we can say is that in general Huffman is optimal, providing good efficiency over the source symbol probabilities, while the tree is more unbalanced for Huffman compared to Shannon-Fano, given is built incrementally from the leaves.

In general, Shannon-Fano and Huffman coding will always be similar in size. However, Huffman coding will always at least equal the efficiency of the Shannon-Fano method, and thus has become the preferred coding method of its type

Other considerations:

- Since the codes are built from the leaves we will not have codes that are prefixes to other codes, implying it is possible to decompress from any point in the file
- The drawback of these methods are the representation of the table, if we try to use it for all the possible colors in an image we will have 60000000 rows, which is not manageable

6.1.6 LZW Compression and Decoding

The Lempel-Ziv-Welch (LZW) algorithm employs an adaptive, dictionary-based compression technique. It dynamically builds the vocabulary of the symbols using fixed codes for sequences with variable length, meaning we can have a code representing two or three symbols at once. It was covered with copyright up to 2003, now it's free.

Some useful considerations:

- LZW is optimal as algorithm (while Huffmann is optimal for entropy-coding algorithms) and performs better than the previous algorithm
  - because they belong to different classes and the code belongs to a string of information, not to single chars
- LZW is not efficient from the first bits, while Shannon-Fano and Huffman are immediately efficient
  - Initially, ZLW reads the bits but does not encode them
  - Since it does not put the dictionary in the file, the decoder needs to read the bytes in clear at least once, then create the dictionary
- It works in real files and becomes efficient when file size increases

Specifically, we can give more details:

- Unlike variable-length coding, in which the lengths of the codewords are different, LZW uses fixed-length codewords to represent variable-length strings of symbols/characters that commonly occur together, such as words in English text
- As in the other adaptive compression techniques, the LZW encoder and decoder builds up the same dictionary dynamically while receiving the data—the encoder and the decoder both develop the same dictionary
- Since a single code can now represent more than one symbol/character, data compression is realized

In the table, we have as second column the input of the file, the output is the result of the encoding, and the last columns are the dictionary. Given the file does not memorize the dictionary, the receiver needs to receive it in white, so this is not encoded the first time then it creates the dictionary.

The *encoding* can be represented by the following algorithm:

$w = NULL;$

$while\ (not\ EOF)$

$\{\ read\ a\ character\ k$

$if\ wk\ exists\ in\ the\ dictionary$

$\quad w\ =\ wk;$

$else$

$\quad \{\ add\ wk\ to\ the\ dictionary;$

$\quad output\ the\ code\ for\ w;$

$w = k;$

}

} $output\ the\ code\ for\ w;$

Let’s start in the algorithm and read the first character:

w	k	output	code	symbol
NULL	a			

The key point is $if\ wk\ exists\ in\ the\ dictionary$ concatenate it, otherwise add the word never seen in the dictionary. This principle is used by color palettes. For characters, we consider their ASCII code. The first time a character is read, of course there is inside the ASCII code. So, $k = null$ and $w = a$ we consider.

Now, we have ab not present in the ASCII code, so we take the previous and concatenate the new character as follows, giving as output the code of a, giving $w = a$:

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab

There is then bc which is not present in the dictionary, then giving in output $w = b$ as code. The decoder can read a, can read b and the first time there is a concatenation not seen, you can use the previous entry (256) adding yours (257).

You can see the decoder can read the same character, but you can also see the encoding is not efficient, it becomes so after 100 bytes. This works for “real” files, so big ones, which can be used for “zip” algorithms:

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc

Then we read c and there is cd not present dictionary, putting the new code cd in the dictionary and giving as output c:

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc
c	d	c	258	cd

Same thing for d, we put into the dictionary da and we give as output d:

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc
c	d	c	258	cd
d	a	d	259	da

The second time ab appears and it already exists, and nothing is given as output.

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc
c	d	c	258	cd
d	a	d	259	da
a	b			

Same thing for abc, which is not present and is put as entry in the dictionary and we give as output 256, which is exactly ab, which means "begin to encode", so given it was already encoded, give as output its encoding (so, 256 exactly).

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc
c	d	c	258	cd
d	a	d	259	da
a	b			
ab	c	256	260	abc

There is the combination ca not present in the dictionary, then add it and output c.

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc
c	d	c	258	cd
d	a	d	259	da
a	b			
ab	c	256	260	abc
c	a	c	261	ca

So, again:

- Huffman will be efficient no matter the size
- LZW begins to become so when the file size increases

You can see here on the right the full result of this operation. You can see it will output the entries with their encoding as soon as you encoded them, and you meet them again.

Every time it meets a new string, it will replace them with entries, then encode them the second time.

Usually, LZW is more efficient than Huffmann code, which is optimal just for entropy coding. In this case, LZW compresses a variable number of characters and no single symbols.

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	c	b	257	bc
c	d	c	258	cd
d	a	d	259	da
a	b			
ab	c	256	260	abc
c	a	c	261	ca
a	b			
ab	c			
abc	c	260	262	abcc
c	a			
ca	b	261	263	cab
b	c			
bc	e	257	264	bce
e	a	e	265	ea
a	b			
ab	c			
abc	f	260	266	abcf
f	EOF	f		

The *decompression* recreates the vocabulary while expanding the text, which is given back as an output

- We read a character and see whether we have an entry for it in the dictionary:
  - if we do, we output it, we add w with the first character of this entry to the dictionary, and we set the new w as the entry
- The size of the table is dependent on the number of bytes we are using, meaning if the dictionary is full we remove an entry and restart, usually the first one

Decompression recreates the vocabulary while expanding the text:

$read\ a\ character\ k;$

$output\ k;$

$w = k;$

$while\ (\ read\ a\ symbol\ k\ )$

$\{\ entry = dictionary\ entry\ for\ k;$

$output\ entry;$

$add\ w\ +\ entry[0]\ to\ dictionary;$

$w\ =\ entry;$

$\}$

Let's see how the *decoding* works, seeing if we are able to obtain the original string this way:

w	k	output	code	symbol
	a	a		

In the above example, we read the character a and it's simply given as output. As before, this is present inside the ASCII table.

Again, we have a and we read the symbol b, this is searched in the dictionary, and it is found (so, output b) and then output the concatenation of a plus the concatenation of first character of entry found in the dictionary. So, we put ab and the first entry not empty, which is 256.

w	k	output	code	symbol
	a	a		
a	b	b	256	ab

We then read the ASCII code for c, then found inside the dictionary and then associate the code to the first that it's free (so, 257).

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	c	c	257	bc

Same exact thing for c now with same reasoning:

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	c	c	257	bc
c	d	d	258	cd

Then, we have d, w have $k = 256$ and we give as output the encoding found in the dictionary, which was if you remember ab. Then, we give as output d + the first character found in the string, so da.

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	c	c	257	bc
c	d	d	258	cd
d	256	ab	259	da

The complete decoding is the following:

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	c	c	257	bc
c	d	d	258	cd
d	256	ab	259	da
ab	c	c	260	abc
c	260	abc	261	ca
abc	261	ca	262	abcc
ca	257	bc	263	cab
bc	e	e	264	bce
e	260	abc	265	ea
abc	f	f	266	abcf
f	EOF			

There is a particular case: it always works because we can be sure there is always the entry for the dictionary, putting it inside and then encode it. This is almost always true, but let's see a particular case of this.

Consider the string $ababbabcabbabbax$ applying the same algorithm as before:

```
w = NULL;
while (not EOF)
  { read a character k
    if wk exists in the dictionary
      w = wk;
    else
     { add wk to the dictionary;
       output the code for w;
       w = k;
     }
  } output the code for w;
```

w	k	output	code	symbol
NULL	a			
a	b	a	256	ab
b	a	b	257	ba
a	b			
ab	b	256	258	abb
b	a			
ba	b	257	259	bab
b	c	b	260	bc
c	a	c	261	ca
a	b			
ab	b			
abb	a	258	262	abba
a	b			
ab	b			
abb	a			
abba	x	262	263	abbax
x	EOF	X		

In words, the decoding failure occurs because the algorithm assumes that the dictionary entries are built correctly during encoding, but in the given example, there is an inconsistency in how the dictionary is constructed during decoding.

Consider the *decoding* phase, which retrieves the string: ab 256 257 bc 258 262 x

This uses the following algorithm:

$read\ a\ character\ k;$

$output\ k;$

$w = k;$

$while\ (\ read\ a\ symbol\ k\ )$

$\{\ entry = dictionary\ entry\ for\ k;$

$output\ entry;$

$add\ w + entry[0]\ to\ dictionary;$

$w = entry;$

$\}$

We consider also this one step by step: we take the first character of previous step and give it as output.

w	k	output	code	symbol
	a	a		

Same thing for b, ab not present in dictionary, give as output b and create in the dictionary ba, which is the concatenation of 256, found as a in the dictionary and then b the newfound string:

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	256	ab	257	ba

Next input is 257 and concatenate the ab string with the first character of the previous encoding 257 which is b, so we have abb and giving the new code 258:

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	256	ab	257	ba
ab	257	ba	258	abb

We have then ba, we have b, give it as output and create bab with code 259:

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	256	ab	257	ba
ab	257	ba	258	abb
ba	b	b	259	bab

Same thing for 260 and then 258 is in the dictionary so it is found within, it's abb. Take the first char of it (a) and concatenate with c, so it will be ca (two steps in one here).

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	256	ab	257	ba
ab	257	ba	258	abb
ba	b	b	259	bab
b	c	c	260	bc
c	258	abb	261	ca

Now, we receive as input 262, but the entry is not present yet in the dictionary and so there is a failure here.

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	256	ab	257	ba
ab	257	ba	258	abb
ba	b	b	259	bab
b	c	c	260	bc
c	258	abb	261	ca
abb	262	??	??	??

More formally, "???" indicates that the decoder has encountered a difficulty: no dictionary entry exists for the last input code, 10.

- A closer examination reveals that code 10 was most recently created at the encoder side, formed by a concatenation of Character, String, Character
    - In this case, the character is A, and string is BB - that is, $A + BB + A$
    - Meanwhile, the sequence of the output symbols from the encoder are A, BB, A, BB, A.
- This example illustrates that whenever the sequence of symbols to be coded is Character, String, Character, String, Character, and so on
    - the encoder will create a new code to represent Character + String + Character and use it right away
    - before the decoder has had a chance to create it!
- Fortunately, this is the only case in which the above simple LZW decompression algorithm will fail. Also, when this occurs, the variable s = Character + String

A modified version of the algorithm can handle this exceptional case by checking whether the input code has been defined in the decoder's dictionary

- If not, it will simply assume that the code represents the symbols $s + s[0]$; that is, Character + String + Character
- Implementation requires some practical limit for the dictionary size - for example, a maximum of 4,096 entries for GIF

- Nevertheless, this still yields a 12-bit or 11-bit code length for LZW codes, which is longer than the word length for the original data - 8-bit for ASCII

So, in the end, Character + String + character concatenation is the reason of the failure of the decoder $(a + bb + a + \dots)$. This is the only case in which the decoder fails, and we add the *exception handling* as follows.

$read\ a\ character\ k;$

$output\ k;$

$w = k;$

$while\ (\ read\ a\ symbol\ k\ )\ \{$

$\quad entry = dictionary\ entry\ for\ k;$

$\quad /*\ Exception\ Handling\ */$

$\quad if\ (entry == null)$

$\qquad entry = w + w[0];$

$\quad output\ entry;$

$\quad add\ w + entry[0]\ to\ dictionary;$

$\quad w = entry;$

$\}$

If you are not able to find the entry in the dictionary, it means you are in the situation of "Character + String + Character". When we have 262, we have abn and the first character of w and concatenate it $(abba)$, giving as output w (so, $abba$ itself).

w	k	output	code	symbol
	a	a		
a	b	b	256	ab
b	256	ab	257	ba
ab	257	ba	258	abb
ba	b	b	259	bab
b	c	c	260	bc
c	258	abb	261	ca
abb	262	abba	262	abba
abba	x	x	263	abbax
x	EOF			

Everything will go well anyway in the end.

6.2 Exercise 4 : Lossless Compression

This drops after the end of the previous set of slides "mmdata" – "Multimedia Data", same day or the day after. This is not a collaborative homework, but an individual one.

Encode the string:

abcabcabcabcfffffffffff000000000000fffffffffffffffffffffff

using:

1. *the LZW algorithm and*
2. *choose an algorithm between Shannon-Fano and Huffman.*

Compare the two results in terms of compression ratio.

General tips

- Do not try to program it in a language on in one another
- The entropy formula is not mandatory and does not give the entropy of the file, but of the general output – minimum number of bits of each character (and the actual bits of the single character)
    - The output is then multiplied by the number of characters (here, 60)

Let's dive in into the actual homework (at least how I did it)

Let's first start considering the LZW algorithm on the string given in input. Here, the table is created to give an overview representation.

w	k	Output	Code	Symbol
$NULL$	a			
a	b	a	256	ab
b	c	b	257	bc
c	a	c	258	ca
a	b			
ab	c	256	259	abc
c	a			
ca	b	258	260	cab
b	c			
bc	a	257	261	bca
a	b			
ab	c			
abc	f	259	262	$abcf$
f	f	f	263	ff
f	f			
ff	f	263	264	fff
f	f			
ff	f			

fff	f	264	265	$ffff$
f	f			
ff	f			
fff	f			
$ffff$	f	265	266	$fffff$
f	0	263	267	$f0$
0	0	0	268	00
0	0			
00	0	268	269	000
0	0			
00	0			
000	0	269	270	0000
0	0			
00	0			
000	0			
0000	0	270	271	00000
0	0			
0	f	268	272	$0f$
f	f			
ff	f			
fff	f			
$ffff$	f			
$fffff$	f	266	273	$ffffff$
f	f			
ff	f			
fff	f			
$ffff$	f			
$fffff$	f			
$ffffff$	f	273	274	$fffffff$
f	f			
ff	f			
fff	f			
$ffff$	f			
$fffff$	f			
$ffffff$	f			
$fffffff$	f	274	275	$ffffffff$
f	f			
ff	f			
fff	f			
$ffff$	f			
$fffff$	f			
$ffffff$	EOF	273		

The algorithm is applied step by step based on the algorithm code taken from the slides. Basically, we encode each character starting from the ASCII table (0-255) and see if the word exists inside the dictionary; if not, the concatenation is then added inside of the dictionary and the code for the specific word is given in output, then adding the code for the considered string. This way, the vocabulary is dynamically built, encoding the variable-length strings each time.

So, the encoded sequence is:
$a\ b\ c\ 256\ 258\ 257\ 259\ f\ 263\ 264\ 265\ 263\ 0\ 268\ 269\ 270\ 268\ 266\ 273\ 274\ 273$

The original size is given by the number of bits of the whole original encoding multiplied by the number of bits given the representation, so $60 * 8 = 480\ b$.

We are using two bytes, given the number of bits occupied is 274, so we have the number of characters occupied by the encoding (21). Given the encoding is multiplied by the number of bytes occupied ($2\ B = 16\ b$), we would have $21 * 16 = 336\ b$ for the total encoded size.

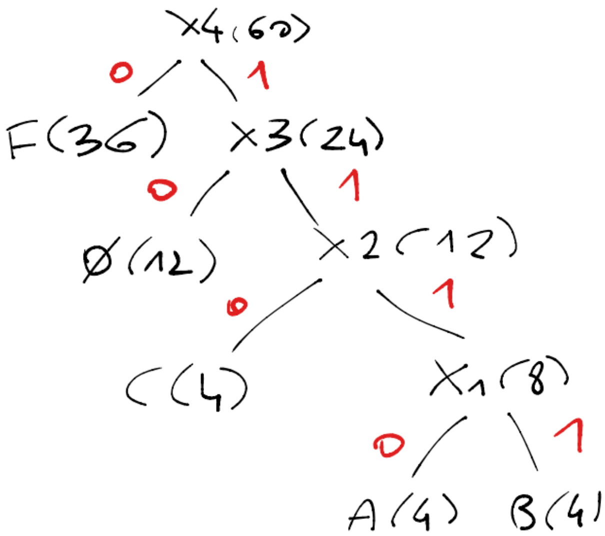
Quoting the formula of data compression ratio present here:

$$Compression\ Ratio = \frac{Uncompressed\ Size}{Compressed\ Size} = \frac{480}{336} = 1.43$$

In this section, the Huffman algorithm is chosen and applied, with the following table describing each symbol, occurrences and the encoding, obtained looking at the tree obtained (0 for left children, 1 for the right children), hence considering the total number of those.

Symbol	*N. of occurrences (#)*	$\log_2(\frac{1}{p_i})$	*Code*	*N. of bits*
A	4	$\log_2(\frac{1}{15})$	1110	16
B	4	$\log_2(\frac{1}{15})$	1111	16
C	4	$\log_2\left(\frac{1}{15}\right)$	110	12
F	36	$\log_2(\frac{3}{5})$	0	36
0	12	$\log_2\left(\frac{1}{5}\right)$	10	24

The corresponding tree is represented here representing the encoding is shown here:



The algorithm is bottom-up, so we start from the lowest-occurrences nodes, in this case A, B, forming a new node as sum. Given the tree would be unbalanced, the character C is then summed subsequently, forming a sum node of 12. Continuing this way, we sum all occurrences of the nodes, reaching the root of 60.

The algorithm gives in output as encoded string:

- Original: abcabcabcabcffffffffffff000000000000ffffffffffffffffffffffff
- Encoded: 1110 1111 110 1110 1111 110 1110 1111 110 1110 1111 11 0 0 0 0 0 0 0 0 0 0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

We then compute how many bits are occupied, considering this is computed multiplying the number of occurrences with how many bits the single code occupies:

$$4 * 4 + 4 * 4 + 4 * 3 + 36 * 1 + 12 * 2 = 104$$

The occupation is given from the number of bits needed by the ratio (8 bits) multiplied by the different chars found inside the encoding (5) multiplying by 2 to represent the column of the encoded sequence (as much as the number of symbols). Combining all of this we get $(8 * 5) * 2 = 80\ b$. This is then summed with the result of the encoding, specifically $80 + 104 = 184\ b$ for the encoded size.

To calculate the effect of the entropy, we use the formula $H(S) = \mu = \sum_i p_i \log_2\left(\frac{1}{p_i}\right) = 1.69$, obtaining as theoretical occupation $H(S)$ multiplied by the number of characters (60), $1.69 * 60 = 101.27$.

The original uncompressed ratio is given by the total number of original bits multiplied by 8 bits, so $60 * 8 = 480$ (so, number of bits occupied multiplied by 8 bits).

Quoting the formula of data compression ratio present here:

$$Compression\ Ratio = \frac{Uncompressed\ Size}{Compressed\ Size} = \frac{480}{184} = 2.61$$

The LZW algorithm achieved a compression ratio of 1.43, compressing the original data from 488 bits to 336 bits.

Written by Gabriel R.

The Huffman algorithm performed better, achieving a compression ratio of 2.61, compressing the original data from 480 bits to 104 bits (plus an additional 80 bits for the encoding table). So, in this specific case, it provided significantly better compression.

This is due to the Huffman algorithm's ability to assign shorter codes to more frequent symbols, effectively exploiting the skewed symbol distribution in the input data. This is based on construction of a code tree based on symbol frequencies, which can be computationally expensive for large input data.

LZW started becoming more efficient after more appearances of the same characters and their combinations. This, on the other hand, does not require any prior knowledge of symbol frequencies and can adapt to the input data dynamically, making it more suitable for scenarios where the input data is not known in advance or has varying symbol distributions.

As a matter of fact, recall LZW does not need to memorize the table, reducing the overhead needed in saving it but building it dynamically during the decompression phase. So, in the end, Huffman always obtains the optimum in terms of compression for each string length, while LZW needs data of at least 100 kb to obtain efficient results comparable to Huffman's.

(After the correction, the teacher will prepare slides and show them in class, basically in ½ weeks from the exercise presentation).

Main errors:

- not putting inside the Huffman encoding the table itself

The encoding *she* wants for Huffman:

- $5 * 2 = 10$ code's length
- $4 + 4 + 3 + 2 + 1 = 14$ bits codes (univocal chars)
- 104 as the encoding itself
- $5 * 8 = 40$ for the ASCII chars
- Huffman: $10 + 14 + 104 + 40 = 168\ b$

LZW: $21 \times 2 = 42\ B = 336\ b\ (all\ occupation) - 21 \times 9 = 189\ b\ (if\ you\ consider\ 9\ bits)$

Another thing:

- Huffman construct the tree and adds progressively chars to the left
- The derivation is hence considered rightmost
  - which means the tree is developed going towards the right
- Is that an error? Nope, just told ya

6.3 IMAGES

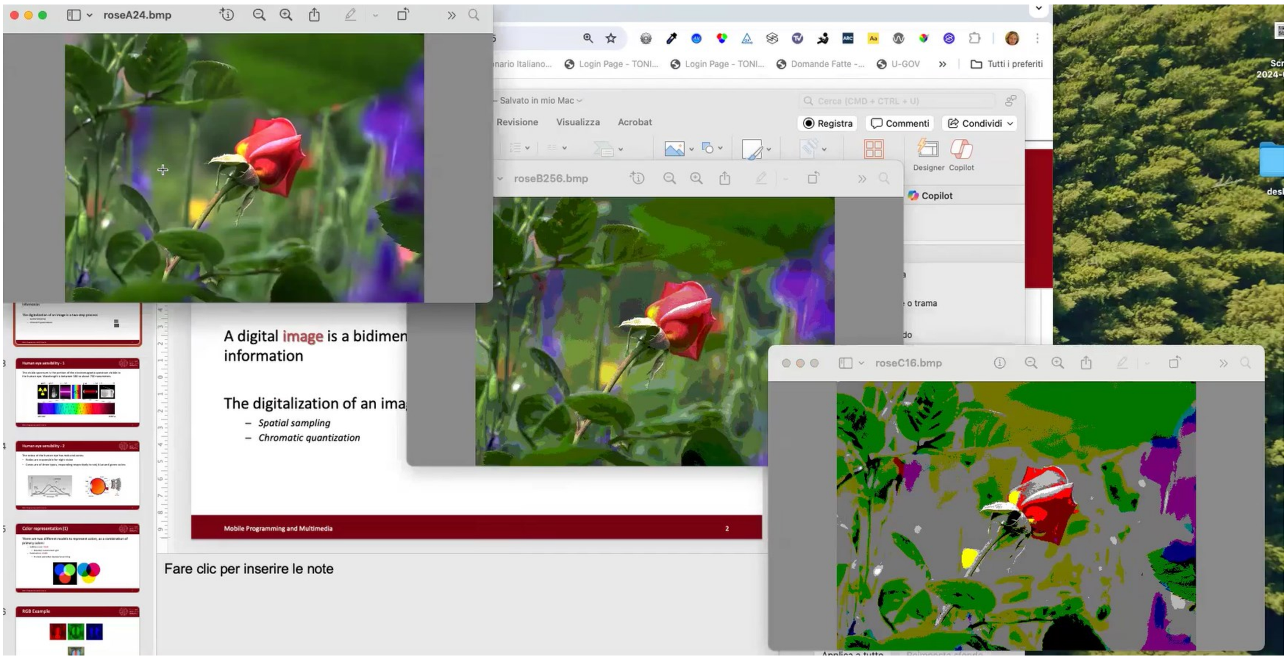
In order to create good compression algorithms, we first have to understand the way we perceive images. Let's try to give definitions more formally:

- An <u>image</u> is an area with a defined colors distribution
- A <u>digital image</u> is a bidimensional matrix, and each point has a chromatic information
    - It is made of pixels (=picture element)
- The digitalization of an image is a two-step process:
    - *Spatial sampling*
        - Here, we define the resolution and the number of pixels
        - More formally, convert the continuous spatial domain of an image into a discrete grid of pixels, representing single points inside the image
        - The resolution itself is the number of pixels for horizontal/vertical directions representing "how closely lines can be resolved in an image"
    - *Chromatic quantization*
        - Here, we focus on the colors
        - More formally, the process of converting the continuous range of colors in an analog image into a finite set of discrete color values
        - Basically, colors are represented via "bins", so specific ranges, according to specified bits per channel

In the below example (spatial sampling), the dimension of squares is so low you can see the pixels themselves. What changes in infact the dimension of the squares; where you see the pixels blurred you have bigger squares, the others have much smaller pixels.

Below then we have the color quantization; same image but <u>true color</u>, which is when we use 24 bits for each pixel (so, pixel x 24) to represent the color (first example in high-left) – so, up to 16.7 million of distinct colors. Here, each pixel has different RGB values, and 256 intensity levels are represented through 8 bits for each channel, hence getting $256 * 256 * 256$, giving the result above.

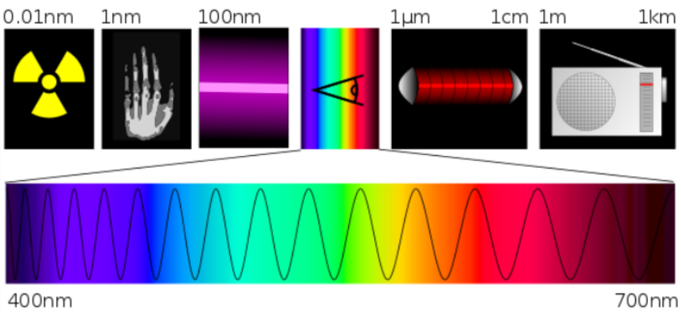
The second example (middle one) has 256 colors, given we have 8 bits to represent the colors. The last example (on the right) uses 16 colors, and you can see the result is not of good quality. Keep in mind this is not compression, but only how we decide to represent pixels:



If we use 24 bits we can represent 60000000 colors, which is a true color representation, but we may not need all of them.

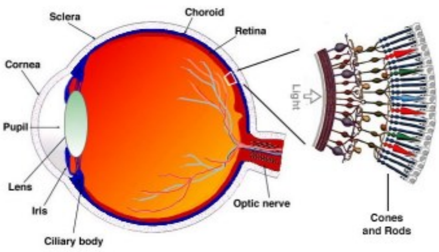
The visible spectrum is the portion of the electromagnetic spectrum visible to the human eye.

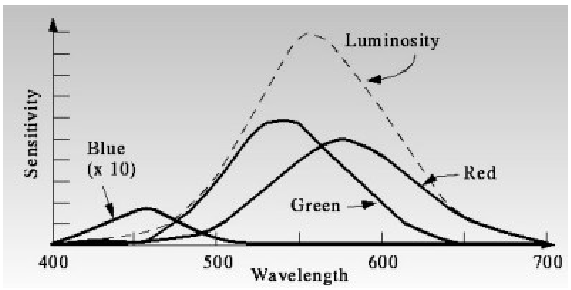
Wavelength is between 380 to about 750 nanometers, of course not visible to our eye. Depending on the length we measure how we will be able to sense those.



The retina of the human eye has rods and cones, perceiving the light with those.

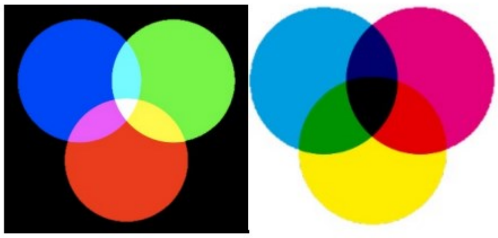
- Rodes are responsible for night vision
- Cones are of three types, responding respectively to red, blue and green colors
    - Night makes us not able to see the light because of this
    - Our eye is more sensible to the green rather than to the red and blue





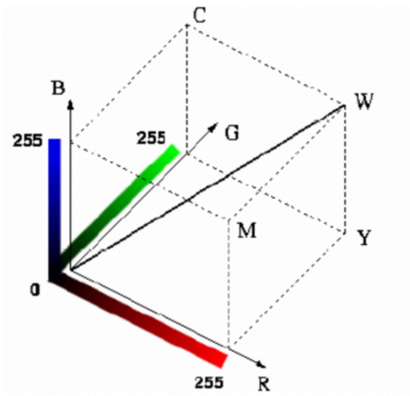
There are two different models to represent colors, as a combination of primary colors:

- Additive color: *RGB*
  - Monitor, transmitted light
  - The one used for monitors, where the presence of a combination describe the color
- Subtractive color: *CYMK*
  - Printers and other devices for printing
  - Here, the absence of a component define the color
  - The color perceived is the light human is not able to observe



In both models, combining the three primary colors we can obtain other colors, like cyan, magenta, yellow, or black.

This happens because our perception is influenced by the color vectors, which are represented via different dimensions. All colors are shades of black and white.



R = red
G = green
B = blue

C = Cyan
M = magenta
Y = yellow
K = black

The following is a decomposition of the three main colors of RGB which, by themselves, are not that expressive. The combination of their shades is what gives the perception of a certain tone:

Another way to conceive colors is through a color space based on psychophysical characteristics:, which try to rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant by humans:

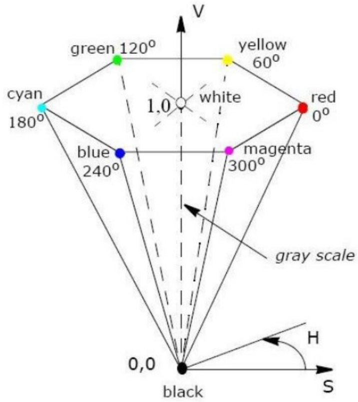
- <u>HSV</u>: hue, saturation and intensity (value) (same as luminosity)
  - it's also called HLB where B is for brightness
  - the *hue* is "the degree to which we perceive a different color", so "how strong a certain color appears" – generally, the color
    - it is described starting from the red, going through the yellow, then blue
    - and back to the red, in a circle like shape

- the *saturation* is the "colorfulness of an area judged in proportion to its brightness"
  - so basically, how dark is perceived an area judged to the light
- the *intensity* is how much light appears or stands out when perceived – how much light the color contains passing all the shades of grey
  - more luminosity when the shade of grey is near to white
  - contrary, near to black, less lightful
- both saturation and intensity are described on a grey scale, and are hence a percentual value
  - It is based on the perception of the humans of shades and gradients of color

When you choose the color, you are picking the color of the left; when you choose saturation, you are moving towards the center of the hexagon shape, moving near to white.

Basically, saturation can be defined as the quantity of white to be added to the color. This is used in the creation or editing of an image.

- YUV, YIQ, YCbCr, separate information about luminosity (luminance) from color information (chrominance) – some reference here
  - they have this name according to the components in the Cartesian plane, specifically the vectorial ones
  - specifically, for the curious ones of you:
    - Y'UV is the separation used in PAL
    - Y'PbPr is the separation used in component video
    - Y'CbCr is any digital encoding of Y'PbPr suited for video and image compression and transmission formats such as MPEG and JPEG
    - Y'IQ is the format used in NTSC television

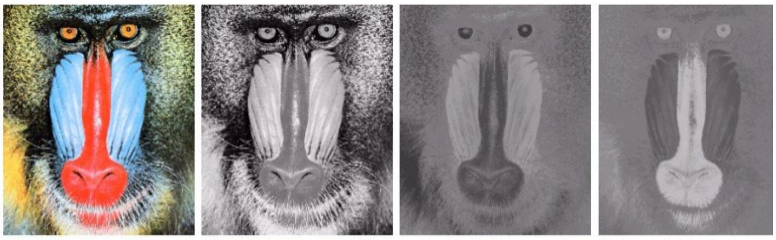


Considering the right figure: the color space is a pyramid with an hexagonal base, while the angle can decide the color itself. The distance from the center of the basis to the perimeter is the saturation. This color model is not used to represent images but to artificially create them, meant to interact with the user.

Formally, a *channel* in this context is the grayscale image of the same size as a color image, made of just one of these primary colors. You can see how the image is broken down between the CMYK or RGB color models. Black indicates the absence of color, while white indicates high presence of luminosity, with all components with maximum value.

The following is a YUV example, considering a color is described as a Y' component (luminance) and two chrominance components U and V. Y is how much can we perceive light (=luminosity / light by environment).

Rodes give luminance instead, which is a measure of how luminous intensity can possibly pass in directions and reflecting on surfaces seen from different angles of view.



Original *Y component* *U component* *V component*

In TV formats, color information (U and V) was added separately via a subcarrier so that a black-and-white receiver would still be able to receive and display a color picture transmission in the receiver's native black-and-white format, with no need for extra transmission bandwidth.

- Infact, this was invented for this specific reason, to add the UV signal to the luma component already existing
- The U and V signals tell the television to shift the color of a certain spot without altering its brightness; the higher these values, the saturated the spot gets

A theoretical model, non-used for practice, can help us see the full gradient of colors that we can perceive, and this is the CIE XYZ color space, which was created by the International Commission on Illumination (CIE).

- This is used in *tristimulus* system
    - a system for visually matching a color under standardized conditions
    - against the three primary colors—red, green, and blue
    - this is made in order to describe any human color sensation
- The values depend on spectrum properties of the cones
- This model represents the whole human color vision

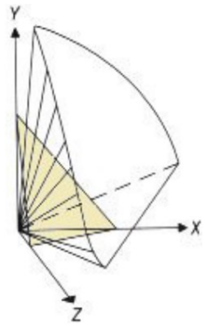
With symbols this can be represented as:

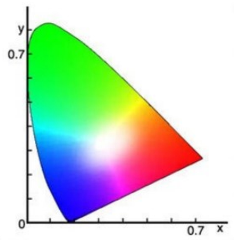
$$x = \frac{X}{(X+Y+Z)}$$

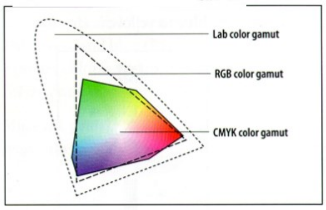
$$y = \frac{Y}{(X+Y+Z)}$$

$$z = \frac{Z}{(X+Y+Z)}$$

$$x + y + z = 1$$







The horseshoe curve represents pure colors. The *line of purples* connects the extreme points of the shapes and represents colors with does not correspond to a wavelength, hence it cannot be perceived.

At the center of the diagram, there are some reference colors (Illuminating, E, A, B, C, D).

This is used for example to illuminate and calibrate the monitors, as a reference model – such is also the standard model to illuminate TVs in Europe or America.

After the analysis of possible different models to represent colors, we go back to the properties that a digital image has:

- Color depth
  - Monochromatic images and grayscale images (1 bit, 8 bit)
  - Indexed colors: usually 8 bits/256 colors chosen between ??? (palette, table of colors or CLUT, Color Look-Up Table)
    - to index colors, 8 bits to represent colors
    - plus additional bits, 3 for instance, to add the table
      - needed to effectively represent colors
  - True colors: $24\ b/16.7M$ colors, $48b$ (High Definition), $24 + 8\ b$ (alpha channel)
    - very high resolution can be achieving using 24 channels per bits, not the converse
    - the *alpha channel* is an extra channel is added to add levels of transparency/opacity to each pixel
- Resolution
  - Limited to the more common sizes of the screens for Web applications
  - Images can be encoded with multiple resolutions
  - Miniature for fast preview

6.3.1 BitMap – BMP

BitMap is the MS Windows native format for images and:

- It can represent images via a palette (*colormap*) at $1, 4$ or $8b$ per pixel
    - but also with natural colors using $24b$ per pixel
- Not compressed (basically, raw files)
- Bitmap (also called raster) is formed from rows of different colored pixels
- Compressed using RLE (Run-Length encoding) algorithm

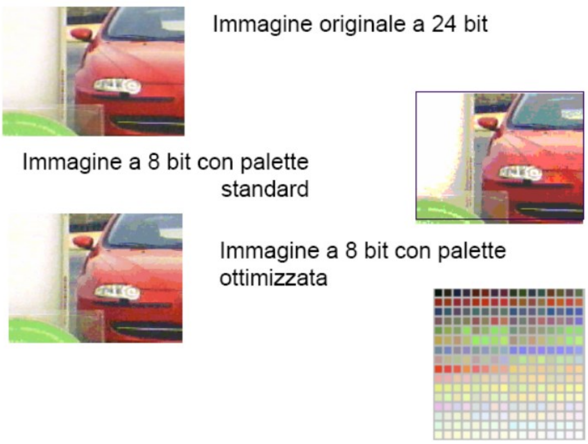
It's particularly suited for artificial images or images with several sequences of equal pixels and it's not suitable for photographic images (e.g., war images). Considering the palette:

- It is possible to use a standard *palette* with a subset of possible colors, dividing the RGB cube by a standard number for each channel
- With 256 colors, with 6 steps we get 216 equally spaced colors. Other colors are freely chosen
- *Optimized Palette*: built from the histogram, it best represents colors of the picture

The pro in using this strategy is that, in the communications between the same OSs we won't need to share the palette, since it is native, but different OSs will also have different palettes. KISS – Keep It Simple and Stupid: a good example of palette is the original color system from Windows '95 Paint.

If you look at definitions, you can see the main goal of this format is to give a device-independent bitmap, which can be used everywhere with no problems.

Here there is a palette comparison, giving how we actually save the palette inside the file – this simple thing determines how good the image is – $24b$ is true color and max depth of colors, $8b = 256$ color and the other uses the optimized palette, to reduce the number of colors use, but it can appear easily similar to the original:



As we said, BMP does not compress the original image, and is a lossless compression format, but this is not really feasible with the real-world requirements, and so talking about the storage space:

- Memory occupation depends on color depth and image size
    - Grayscale $160 \times 120 = 18.75\ KB$
    - 256 indexed colors $640 \times 480 = 300\ KB$
    - *true color* $1024 \times 768 = 2.25\ MB$ (images not compressed)
- Compression strongly reduces size maintaining quality
    - 1024×768 JPEG $\cong 300\ KB$

About the transfer time features:

- Transfer time for not compressed images is usually not acceptable even with fast network connections
    - 2,25 MB with ADSL 640 $Kb \cong 29\ s$ (recent data ADSL, ah?)
- Compressed JPEG image requires less time
    - $300\ KB$ with ADSL $640Kb \cong 4\ s$
    - Decompression time is irrelevant with the actual workstation
- Intervals of time needed are even shorter if the image is only visualized on the screen
    - Quality requirements are usually lower than print requirements

This brought the study of multimedia to create image formats, independent from the platform and the producer, and which do not require a license (except GIF in the past) :

- *GIF*: $8\ b$, animation, transparency
- *JPEG*: true color, quality, lossy compression
- *PNG*: lossless compression, alpha channel, extensibility

6.3.2 GIF – Graphics Interchange Format

GIF is the first standard for image transmission over networks and it does not require a license. Some general features:

- Developed by Unisys and by the provider Compuserve
- 256 indexed colors (containing up to 8 bits per pixel)
- Compression algorithm: LZW (patent Unisys, expired in 2004)
- Other properties: interlacing, transparency, animations
- Simple images with few different colors (artificial images)
- Lossless compression, fast decoding
- Animated images are usually engaging and make messages more visible (advertisement)
- Transparency allows better and easier integration with the background (ex. formulas)
- It also allows to save more images one after the other

6.3.3 PNG – Portable Network Graphics

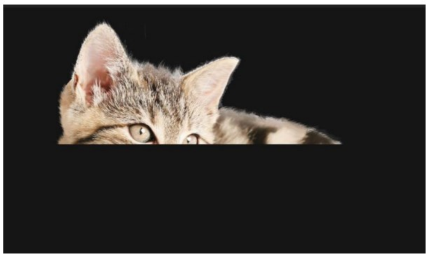
PNG is an extensible format (can be both read as PNG but also as “ping” but also “PNG’s not GIF”, used because originally GIF was patented, then expired in 2004, so same to use one or the other), with lossless compression (using LZW 77), portable, for colorful images. Other general features:

- No patents and licenses (to substitute GIF)
- Color depth is variable from 1 to 16 bit on color lookup table
    - From 3 bits to 48 bits in terms of color
    - Data into saved into pixel levels of 8x8 block, starting from all the first level of each block (1), then all the second (2), and so on
- Supports indexed colors, grayscale images, and *true color* images
- Transparency effects using an optional *alpha channel*
- Implements an efficient interlacing method for images transmitted through the network
- PNG can also be good for photographic images, given its lossless

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

In order to make easier transmissions through the network it uses an interlacing method, basically sending and receiving all the ones, then all the two, etc.

There is the concept of *interlacing*, which is a technique for doubling the perceived frame rate of a video display without consuming extra bandwidth.

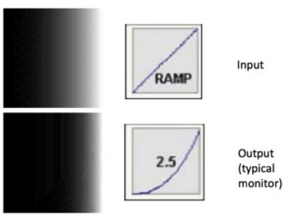


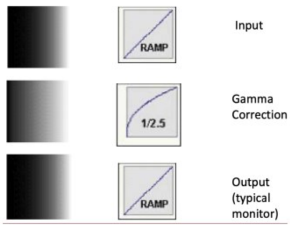
Here you can see a visual example of the concept. Without interlacing we have not the previous figure, but more of the right one here.

The interlacing gives the whole idea of the image sooner, getting other data too via network.

Gamma is an important but rarely understood characteristic of virtually all digital imaging systems. It defines the relationship between a pixel's numerical value and its actual luminance.

- Without gamma, shades captured by digital cameras wouldn't appear as they did to our eyes (on a standard monitor)
- It's also referred to as gamma correction
    - in the real world, input images tend to have less shades of black with respect to the usual monitors
    - to help this, a gamma correction function is applied to an image, in order to help contrast a bit the black shades of the monitor, as it can be seen below





You can see an example of the above concepts here:

Other features for this format to conclude:

- It has the same uses as the GIF format without requiring a license
- Equal or better quality than the JPEG format, but with a lower compression
- Allows a higher control of the different effects of the image through the alpha channel

Written by Gabriel R.

- It is extensible to insert additional information or proprietary information in a standard way
- Promoted by W3C

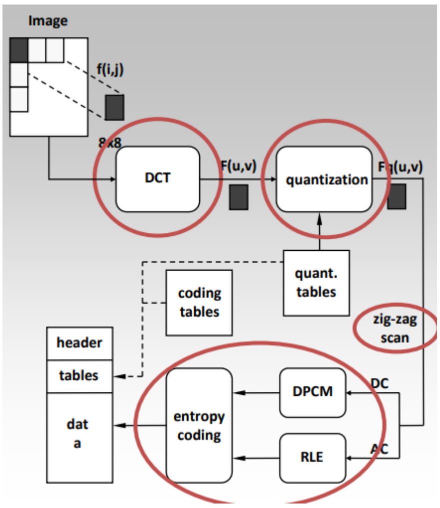
6.3.4 JPEG - Joint Photographic Experts Group

JPEG is is an image compression standard developed by the group of the same name for photographic images compression created in 1992. It overcomes the limitation of the entropic compression, using the redundancy of visual perception (uses lossy compression). Here, the smallest details of an image can be suppressed without losing useful information.

You can still the details with this 300 KB image:

6.3.4.1 JPEG Compression Algorithm

The JPEG compression algorithm for the *encoding* works as follows (figure on the right explains the sum of the JPG compression) – basically a subsampling by 4 of the chroma subchannels:



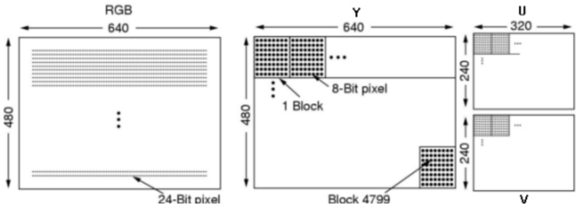
1. Image *preparation*, switching from RGB to YUV color model (or similar)
    a. to black or white with same level of detail
    b. subsampling of chrominance reduces the file size without losing data for human eye
    c. three tables are generated for luminance (Y), chrominance (U/V) in its components
    d. at least half of the data is lost here
2. For each 8×8 block, a *DCT* (Discrete Cosine Transform) is applied
    a. generating an $8x8$ matrix where the element $(0,0)$ is the dominant color
    b. while the other values rapidly tend to 0 (and there are all the colors which deviate from the dominant one)
    c. Also this step loses data, given this is reversible
        i. theoretically, but not always, given the numbers used and their nature
3. *Quantization*: each coefficient is divided by a weight defined in a quantization table – given we have frequency information, this way useless details (not perceived by the human eye) are removed
    a. Quoting the book: This step is the main source for loss in JPEG compression
4. The value $(0,0)$ is substituted with the difference with the same value of the adjacent matrix. The purpose is to obtain low values
5. Matrix *Linearization*: the obtained matrix is covered diagonally, and not by columns or rows, to get adjacent values equal to zeros (we need to apply zero-encoding in order to compact them)

6. *Compression*: RLE and Huffman encoding to the resulting list of values without wasting data, to compress as much as possible

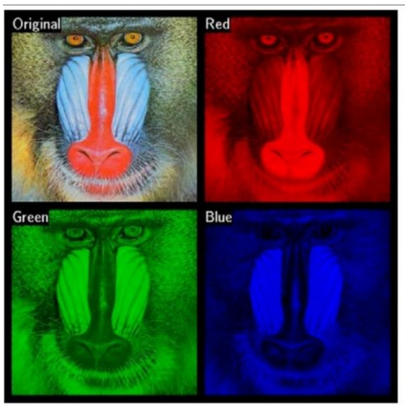
This process loses information in the preparation (about the colors), during quantization (the image becomes blocky), and some during the DCT (since computers use something similar, but not actual real numbers).

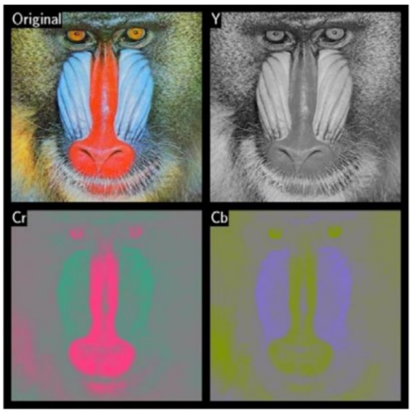
If we lose information during the preparation process, why do we sw*itch color representation* then?

- Human being perception is more precise for luminosity than for colors
    - The representation of luminance information must be more accurate than the chrominance information
    - Different resolutions for the two components
- Color coding YUV
    - Y (luminance) is a combination of the RGB components
        - It describes the amount of light that passes through, is emitted from, or is reflected from a particular area, and falls within a given solid angle
    - U ad V (chrominance) are encoded as differences to the reference colors:
        - $U = B - Y, V = R - Y$
        - Signal used in video systems to convey the color information of the picture, separately from the accompanying luma signal (or Y')
    - U and V are undersampled
        - Here, it refers to the reduction in the sampling rate of the chrominance components (U and V) compared to the luminance component (Y) in the YUV color space



Below, we can see examples of color space transformation of RGB (left) and YCbCr on a baboon:





Consider now the undersampling, starting from the leftmost up to the rightmost, where the blurriness increases downgrading the sampling of 2, 4 and 8 times:


Then, the chrominance, which increases the undersampling 2, 4 and 8 times.


On these, we might say the following (for you to have context):

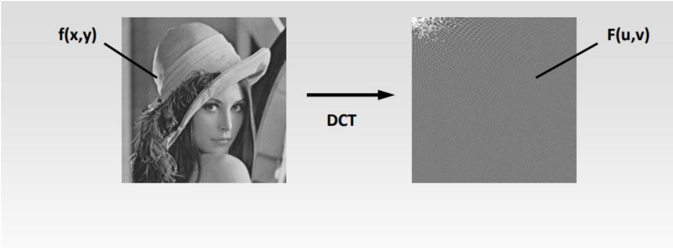
- $2x$ Undersampling (4:2:2):
    - Chrominance sampled at half resolution horizontally (Cb and Cr)
    - Slight reduction in colorfulness, often imperceptible
    - Most viewers won't notice significant differences
- $4x$ Undersampling (4:2:0):
    - Both horizontal and vertical chrominance resolution halved
    - More noticeable reduction in color detail
    - Still acceptable for most purposes
- $8x$ Undersampling (4:1:1):
    - Chrominance sampled at one-eighth the resolution of luma
    - Significant color artifacts, blocky appearance
    - Visible color bleeding and loss of fine details

6.3.4.2 Discrete Cosine Transform (DCT)

For each $8x8$ block, the Discrete Cosine Transform (DCT) is applied to break up the input signal into its different components – it's able to perform decorrelation of the input signal in a data-independent manner. Each coefficient of the DCT defines the weight of the relative frequency inside the image, meaning the biggest one will be our dominant color.

- The DCT transforms a finite sequence of data points (such as an image or a signal) from the spatial domain (where data points are arranged in space) to the frequency domain (where data points are represented by their frequency components)

- It helps separate an image or signal into different frequency components, making it easier to analyze and compress



$$F(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N}\sum_{y=0}^{N} f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)u}{2N}\right]$$

This function is theoretically reversible, but since we are losing data due to the fact we are not able to represent it, this is not precise in a real scenario.

- The element $(0,0)$ contains the predominant color (the one we perceive better), while the other elements a variation of the color – this component is also called DC, while the other elements are called AC, which represent various levels of frequency
- Each of the obtained 8x8 matrices follow this rule, meaning the closer to $(0,0)$ we are able to see fewer details, which increase while we stray away
- The left figure represents in the spatial domain the image as it is, then it's represented in terms of frequency components. Quantization means reducing the precision of lesser important coefficients, then encoded, resulting in significant data reduction
- The right grid shows different patterns, capturing different aspects
  - Horizontal lines capture the variations in intensity or color along the horizontal axis. Lower
    - Horizontal frequencies (near the top) represent slow changes
      - (e.g., gradual gradients)
    - Higher horizontal frequencies (near the bottom) represent rapid changes
      - (e.g., edges and fine details)
  - Similar to horizontal frequencies, lower vertical frequencies (near the left) represent slow changes, while higher vertical frequencies (near the right) represent rapid changes

Consider how the image is divided into $8x8$ blocks and, using DCT:

- This is applied to each block image $f(i,j)$, with output being the DCT coefficients $F(u,v)$ for each block
- The choice of a small block size in JPEG is a compromise reached by the committee: a number larger than 8 would have made accuracy at low frequencies better but using 8 makes the computation very fast
- This is why JPEG images look choppy ("blocky") when the user specifies a high compression ratio – we can see these blocks

After the DCT, the image is converted into a signal, so we can perceive the "details" (degree of variation). Specifically:

- the upper diagonal (upper part of the image) focuses on the colors
- the lower diagonal (lower part of image) focuses on the changes

6.3.4.3 Quantization

In this case, after the DCT, there is the Quantization, which aims at reducing the total number of bits needed for a compressed image. It consists of simply dividing each entry in the frequency space block by an integer, then rounding, this way reducing the precision of the integer.

- The DCT is invertible, hence the original image can be rebuilt
  - DCT coefficient quantization causes the degradation of the restored image
- These DCT coefficients are then quantized
  - which means they are divided by a quantization matrix
  - and rounded to the nearest integer value

The left part of the image shows the original DCT coefficients before quantization. The middle part shows the quantized coefficients after dividing the DCT coefficients by the quantization table values shown on the right. The JPEG algorithm implements quantization using a separate quantization matrix, an example of which is shown below. You see how this works: $\frac{150}{150} = 1, \frac{80}{80} = \frac{1,40}{20} = 2$, etc.

DCT Coefficients

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantized coefficients

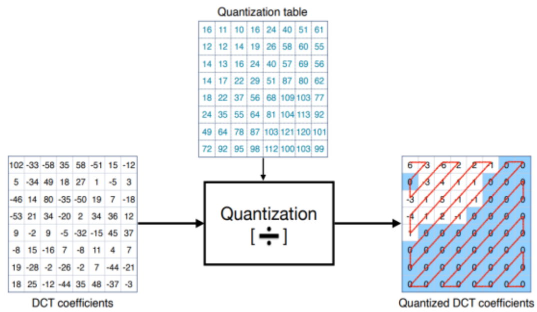
150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization table

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

For every element position in the DCT matrix, a corresponding value in the quantization matrix gives a quantum value indicating what the step size is going to be for that element.

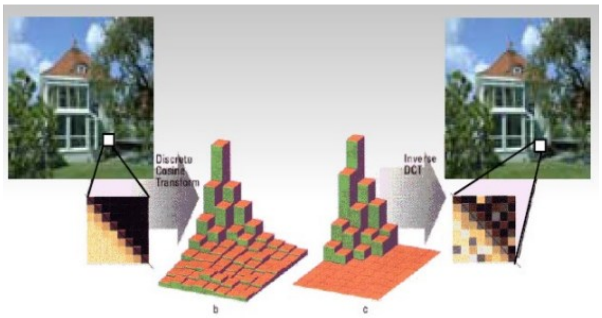
The coefficients that are most significant to the compressed rendition of the image (those closer to the upper left corner) are encoded with a small step size, while coefficients that are less important (those closer to the lower right corner) are encoded with larger step sizes.



This can also be seen here:

- It shows two copies of the same image along with graphical representations depicting the effects of different quantization levels.
- The graphical pyramids in the middle represent the quantized DCT coefficients
- The pyramid on the left (b) has a more complex, multi-colored structure, indicating finer quantization with more non-zero coefficients retained
- This results in higher quality, but less compression for the image on the left.

- In contrast, the pyramid on the right is simpler, with larger solid blocks, representing more aggressive quantization where more high-frequency coefficients are quantized to zero.



Because of quantization, we lose at least 50% of the size (and of data). What can we use to encode sequences of 0? RLE (in a specific version called "*zero-encoding*"). Infact, the goal is putting all the 0s together. The algorithm able to quantize the matrix uses a zig-zag scanning (reading diagonally).

JPEG, infact, may ask you when saving a file the level of quality you want to save it in: bigger numbers means bigger quality but of course lesser compression

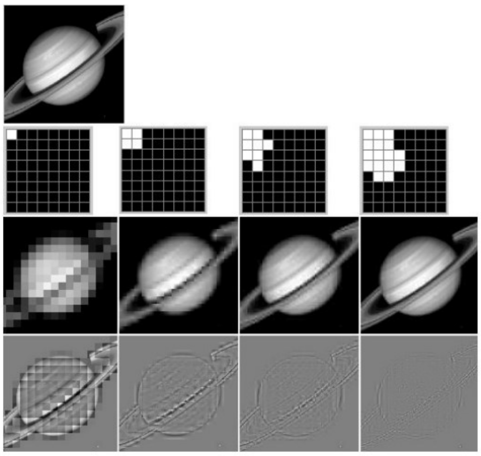
- The bigger number of numbers going to 0, the less quality there is
- The higher the coefficients of the quantization table, the more data is lost
  - but the higher the data to memorize inside the file
- The choice of the quantization table depends on the quality asked to the user when saving the specific file: hence, this will apply a good/bad compression depending on those

Another interesting note to summarize:

- To compress: $DCT\ coefficients\ *\ Quantized\ Coefficients$
- To decompress: $Quantized\ Coefficients\ *\ Quantization\ Table$ then IDCT (Inverse DCT) is applied to get the pixels colors back

When opening and saving the files multiple times, we lose many data because of DCT, given it's not fully reversible, so much data is lost there!

We see this example:



This image illustrates the effects of different quantization levels on image quality and compression artifacts using the example of an image of the planet Saturn. Here, only the values with zero get saved, having the image saved in only one color (so, black).

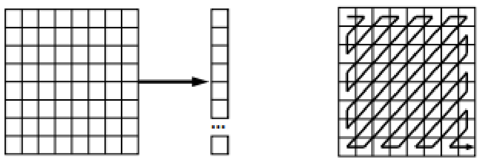
- The top row shows the original image of Saturn, followed by four versions of the same image quantized at different levels (always 8×8). The middle row displays the quantized DCT coefficient blocks corresponding to each quantized image version.
- From left to right, the quantization level becomes more aggressive (so high values), meaning more high-frequency DCT coefficients are quantized to zero
    - As evident from the increasing number of solid black blocks (representing zero coefficients) in the middle row
    - Only those are saved
- Using the first four of the upper part of the matrix, we could see more
    - And the use of 8 components would make the situation even better
    - Then 16 make the situation almost undistinguishable from the original image
- The bottom row shows the graphical representation of the quantized DCT coefficients, where the height of the grid represents the coefficient values
    - As quantization becomes more aggressive from left to right
    - The grid patterns become flatter
        - Indicating more high-frequency information is being discarded

So basically, when JPEG fails, it creates blocks.

6.3.4.4 Linearization and Compression

Going even further in the JPG compression we see the Linearization and Compression. The main idea here is that, after the quantization, we have lots of zeros in the matrix, which can be summed up with RLE encoding.

- This is due to the same idea as before: highest frequencies represent details of the image that can be suppressed without losing noticeable information
    - The 8x8 matrix with the DCT coefficients is linearized with a zig-zag scan
    - Coefficients of the highest frequencies are negligible or null
    - It is possible to use RLE compression techniques
- For each row, we save the first DC of the block
- On the next row, we save the difference from the current block and the previous one
- This way, we will require less block to represent
- The quantized DCT coefficients are then entropy encoded
    - to be formal, we would have to distinguish between RLE on AC Coefficients and DPCM (Differential Pulse Code Modulation) on DC Coefficients
    - but generally, Huffman coding is then used for both types

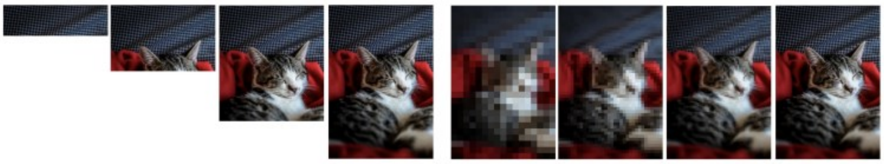


The JPEG compression has four different encoding methods:

- *Sequential* coding: the default JPEG mode, in which each image is encoded with one single scan from top to bottom left to right
- *Lossless coding*: uses predictive techniques, not the DCT

  - Each point is represented as the difference to the expected value
  - based on adjacent points
- *Progressive* coding:
  - it delivers low quality versions of the image quickly, followed by higher quality passes, and has become widely supported in web browsers.
  - it allows to show the image with low quality at the beginning, and progressively with increasing quality – first scans only few bits, then after each scan, image quality is gradually enhanced

About JPEG progressive, consider this generic example:



Progressive JPEG can be developed with one of the following algorithm:

- *Spectral* selection - Encodes image by adding frequency components progressively – takes advantage of the spectral (spatial frequency spectrum) and the higher are the AC components, the more detail they give:
  Scan 1: encoding of DC elements + few AC (example: AC1, AC2)
  Scan 2: encoding of some of the remaining AC (example: AC3, AC4, AC5)
  ...
  Scan k: encoding of the last AC (example: AC61, AC62, AC63)

- *Consecutive* approximations - Encodes image by refining bit precision progressively - Instead of gradually encoding spectral bands, all DCT coefficients are encoded simultaneously, but with their most significant bits (MSBs) first:
  Scan 1: encoding of few more significant bits (MSB), for example bit $7, 6, 5$, and 4
  Scan 2: encoding of some of the remaining bits, for example bit 3
  ...
  Scan k: encoding of the least significant bit (LSB), the bit 0

Continuing with the encodings:

- *Hierarchical coding*: the image is undersampled and JPEG coded, and then the differences between the rebuild image and the original one is coded
  - The encoded image at the lowest resolution is basically a compressed low-pass-filtered image, whereas the images at successively higher resolutions provide additional detail

It contains more versions of the image, and it works this way:

- Resolution reduction of the image by a factor 2 (f_2) for two times (f_4)
- f_4 compression using one of the other JPEG methods
- Compression of the differences between f_4 and f_2 coding using one of the other JPEG methods
- Compression of the differences between f_2 coding
  - and the original image using one of the other JPEG methods

Lastly, some conclusions:

- Photographic images full of colors and shades
- Precise representation of minor details is not essential
  - An image with grids of colors/high contrast, JPEG works worse with that
- Loading and visualization can be progressive
- Possibility to regulate the quality of the image for visualization or printing
- Most used compression standard – very fast in compression/very slow in decompression

We see some comparisons between the different image formats seen until now (these examples can be found in the “Didactic Material” section inside of the “Images” folder).

The “Strelitzia” images follow as examples, first comparing the .jpeg (in full quality (100%) - left) and the .gif (right) format. The program used to show this one is inside of the “Didactic Material” folder, since this is Jpegdata, which was created ad-hoc for the exam the teacher said some years ago.


GIFs are a lossless image format:

- meaning they don't lose any data during compression
- making them suitable for simple images with a limited color palette
  - like animated graphics or icons
- however, they are not ideal for complex photographic images

JPEGs, on the other hand, are a lossy compression format designed for photographic images.

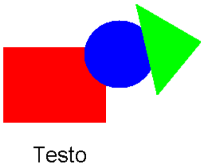
- They achieve smaller file sizes by selectively discarding some image data during compression
  - which can result in a loss of quality, particularly at higher compression levels

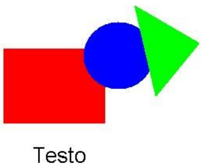
The following are examples of different compression levels of JPG, from 50%/5%/1%, which you can see get blockier and more blurred with each version. As soon as the variation is saved, we do not choose between different shades of the same color but encodings of the specific blocks/portions of the images. Blocks lose colors because of very high quantization table values:



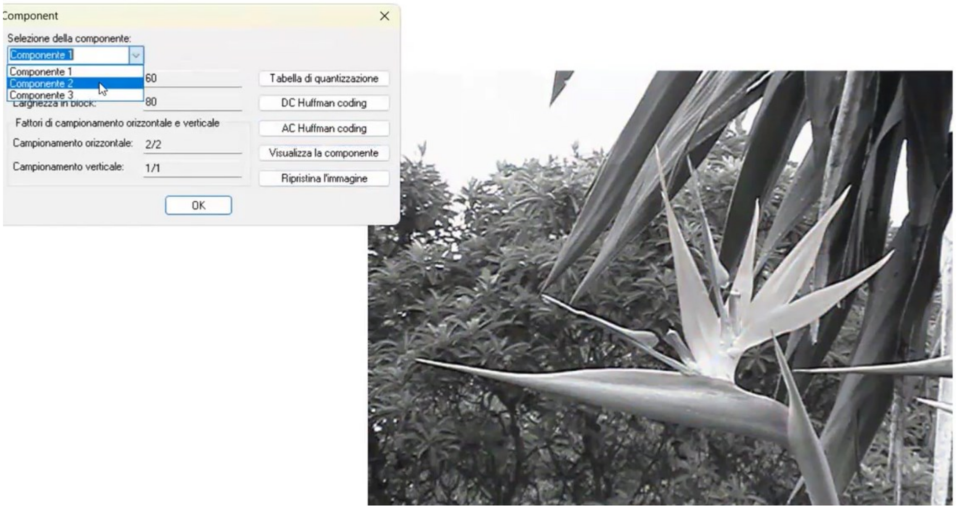
Another example is the following, comparing GIF (left) and JPEG (right) of a yellow flower (first 247 KB vs. 63 KB of storage occupied):

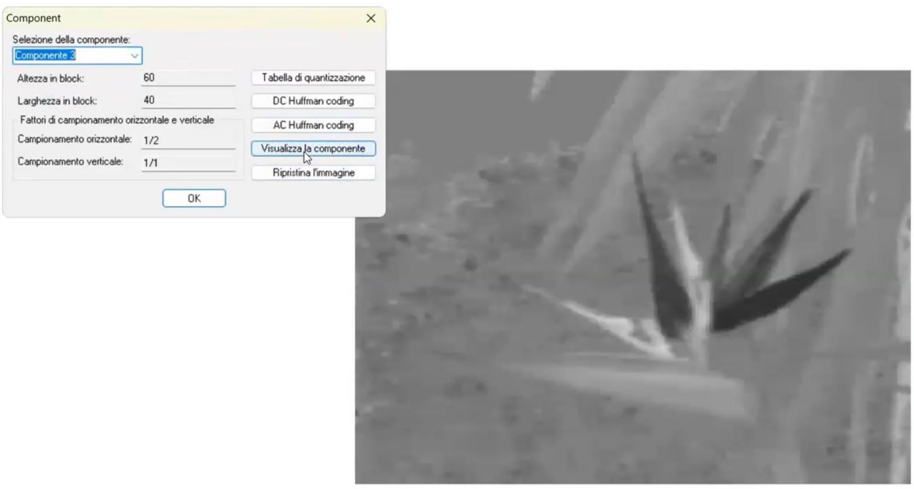

JPEG gives shapes and details (photographic images = works best) while artificial images it's better to use/have GIF. In this case, BMP was used (left) vs JPEG (right), and you can see JPEG is a bit blockier.



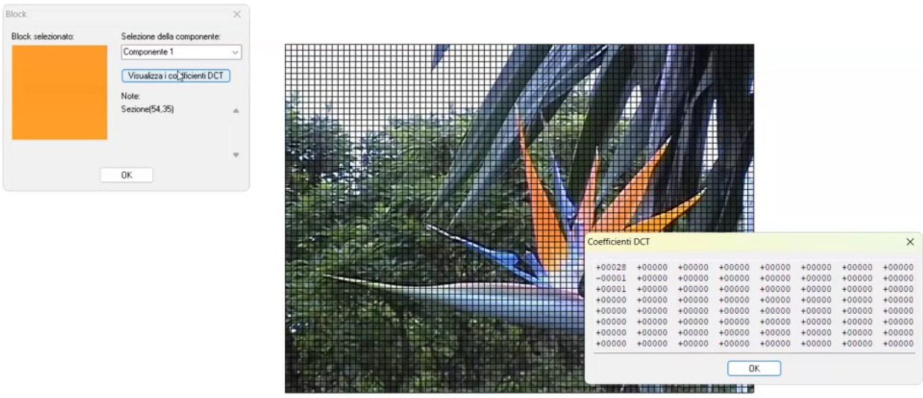


Using JPEG data, we can decompose the luminance component (first figure) from the chrominance component (right).





Let's consider now a component with only one color and the result after the DCT; given it's the dominant component, we save a lot of space, moving from the domain of the color to the domain of the difference of the variation.

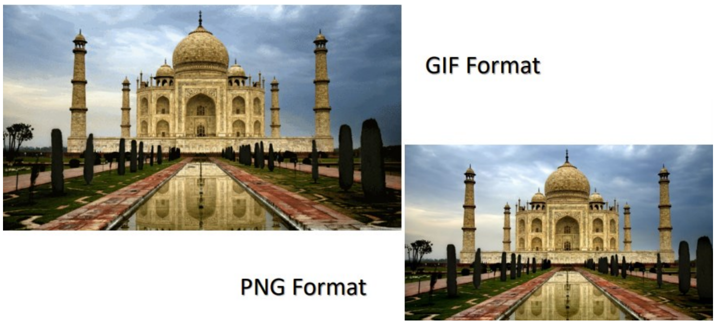


In this other case, DCT allows to have information with very few components with different levels of variations, changing between vertically and horizontally. So, many times I don't need the quantization step, because this reduces processing overall, reducing all the values different from zero.

So, to summarize with all the overall features:

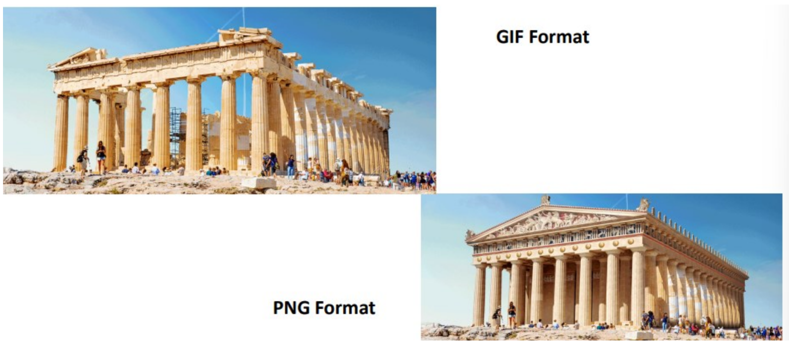
- Color
  - GIF (lossless
    - it supports only 256 colors (8-bit color depth)
    - some shades of colors are lost or pixels having random colors (like example above)
    - which can result in color banding and reduced color fidelity
      - especially in images with a wide range of colors
    - So, if the image has very few colors, use GIF

  - PNG
    - supports a wide range of colors
      - and maintains high image quality without loss of data
      - as it uses lossless compression

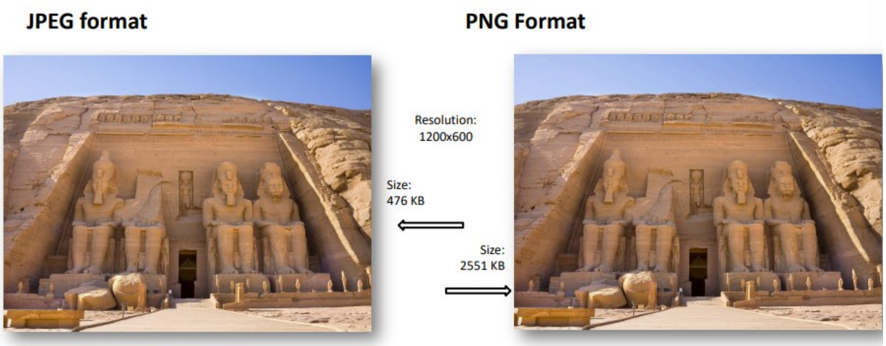


- Animations
  - PNG
    - shows a more accurate and richer color representation
    - it would also occupy more space on the disk
    - makes it unsuitable for storing or transferring high-resolution digital photographs but a great choice for images with text, logos and shapes with sharp edges

  - GIF
    - shows noticeable color limitations due to its restricted color palette
    - use this in case of animations and support to transparency
    - it can support animations and multiple frames



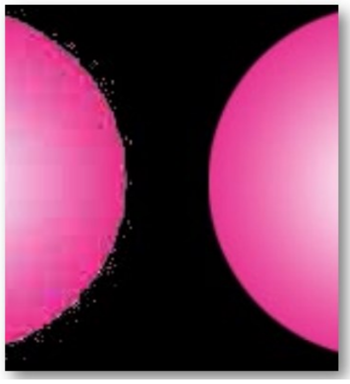
- Size
  - JPEG = lossy, much less space required
  - PNG = lossless, much more space required



So, in general:

- Prefer GIF for animated images
- Prefer JPEG for lossless compression and lots of colors/shades
    - The quality perceived depends also on the device used
    - For the printing, it might be better to use PNG

JPEG has its limitations:



On the left, there is an image of a cat that showcases compression artifacts and visible color banding, which are common issues with JPEG compression, especially at lower quality settings. It might happen that some pixels might have the colors of other regions – PNG/GIF do not have this problems. Also, when it degrades, it might create blocks – something we do not perceive as good.

6.3.5 JPEG 2000

JPEG itself is not suited for images that cannot afford to lose any information, like medical images. These reasons brought the creation of JPEG 2000, which is the last compression standard for image distribution over Web and smartphones, aiming to provide not only a better rate-distortion tradeoff/improved subjective image quality but also additional functionalities JPEG lacks.

- Someone thinks this format was completely replaced by .webp
    - The quality is quite the same and JPEG/JPEG2000 is even faster in decoding
        - especially for mobile devices
- Its main feature is the fact that is more resilient to lost data, but it is not thought to directly substitute JPEG, rather to have a gradual transition (without plugins)
- The main reason is not to create the 8x8 blocks JPEG creates when coding the image
    - In reality: it found its niche into digital cinema
        - It was adopted by DCI (Digital Cinema Initiative) and all movies are made with that
    - According to rumors, JPEG2000 was never adopted by Microsoft into Internet Explorer
        - Overall quality vs compression benefits were very small

Strongly oriented to transmission: an image of 34,6 KB at a transmission speed of 19,2 Kbps requires 14,4 s for download. JPEG 2000 makes the image visible in 1,2 s

With the exception of the web browser Safari, JPEG 2000 is pretty difficult to implement, since it requires a specific CPU configuration, and a plugin in the used application.

We list some of its features here:

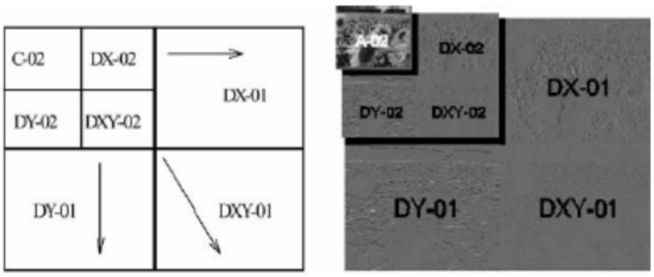
- Lower bitrate compression compared to JPEG
- It supports different compression modalities and color spaces
- Regions of Interest (ROI) encoding
    - certain parts of the image can be encoded with better quality than the rest of the image
    - which means using more bits for those
- Optimized for natural imager
    - and does not perform well on computer-generated imagery
- It supports data transmission in disturbed environments
- It uses a Discrete Wavelet Transform (DWT)
    - that provides an encoding that supports multiresolution without data redundancy
    - with both lossy and lossless encoding
    - allows you to decompose an image into different frequency components or subbands
        - those describe the time evolution of said signal
        - which can then be processed or compressed independently
- In addition, JPEG2000 is able to handle up to 256 channels of information,
    - whereas the current JPEG standard is able to handle only three color channels
    - such huge quantities of data are routinely produced in satellite imagery
- It supports images bigger than $64K \times 64K$ pixels (> 4GB)
- It supports for Watermarking (Copyright)
- Allows two different compression systems:
    - DCT: only for compatibility with JPEG
    - Wavelet: allows all new features

We see some examples here, in which we want to preserve data, since it can't be lost, and we want to always perceive all of the data. JPEG2000 works well for both satellite images and medical images, but natural images JPEG works best, but simple images/artificial images (high contrast) are not so good for JPEG. Before JPEG2000, the only solution possible was PNG.

Test images used for JPEG/JPEG2000 performance evaluation (a): Natural images. (b): Computer generated images. (c): Medical images.

Going deeper for the Discrete Wavelet Transform – used to move from the domain of color to the domain of frequency, applying it to the whole image.

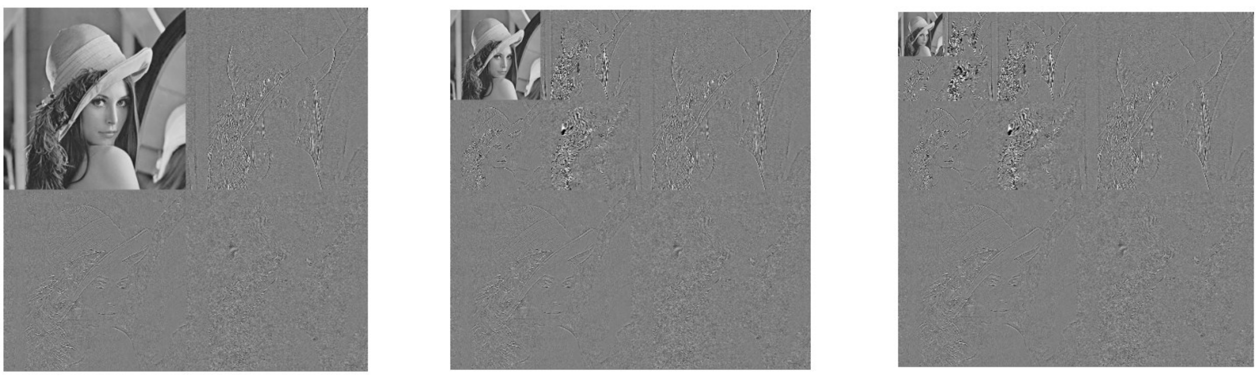
- Each color component of the image is elaborated independently, producing 4 regions (*tiles*)
    - such that each tile dimension is half of the original image
- It is applied to the whole image each time
    - so we won't see block in the decomposition process
    - we will only see some blur in the image, but nothing disruptive
- The same procedure is repeated several times
- On the first quadrant, there are the low frequencies; on the others, the high ones



As seen in the figure, the tile in the top left is the one with the lowest frequencies, the three that round it contain the details about the horizontal, vertical and diagonal sides of the image, like shown by the arrows. The image is subsamples by a factor of 4 (DWT on $\frac{1}{4}$, DWT on $\frac{1}{16}$ and so on).

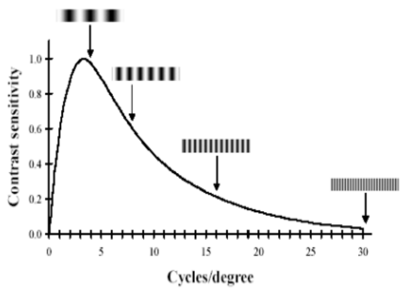
- In the first level of decomposition, the original image (DX-02) is divided into four subbands: DX-01 (low frequencies), and DY-01 (vertical detail), DXY-01 (diagonal detail), and DXY-02 (high frequencies)
- The low-frequency subband (DX-01 – vertical detail) can then be further decomposed in the next level, as shown in the bottom-right figure. This recursive decomposition process can be repeated multiple times, each time focusing on the low-frequency subband from the previous
- This multi-resolution analysis allows for efficient compression, as the high-frequency subbands often contain less visually important information and can be compressed more aggressively without significant impact on the perceived image quality
- The low-frequency subbands, on the other hand, contain more crucial information and are preserved with higher fidelity

The original image is the leftmost, the middle one has the wavelet decomposition applied – lower frequency areas are more visible while high frequency are still present (woman's hair/texture of the concrete). Each time the woman gets smaller to show the high-frequency compared to the low-frequency vision. Where there is "discrepancy", it's information presence there.



The Quantization in JPEG 2000 works on the base of the sensibility of the human eye over contrast variation. It is possible to draw a diagram of the human eye sensibility over contrast variation. Again, this is the step where the most data is lost. DWT is used here.

The image describes our degree of sensitivity according to the level of detail of the image (lower value on top left corner, higher values on top right corner)

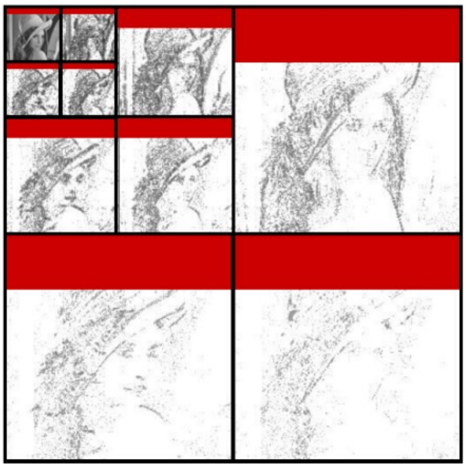


The top-left square shows the lowest resolution or most compressed version of the image, with only basic shapes and outlines visible. As we move from left to right and top to bottom, the quality and resolution of the image gradually increase. More details become visible, such as the woman's facial features, hair, and the texture of the background.

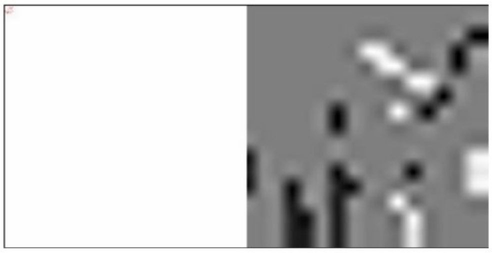
This progression of increasing resolution demonstrates the concept of random access in image compression.

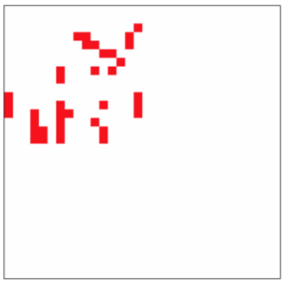
Random access allows users to decode or access specific parts of the compressed image data without needing to decode the entire image (but only a part of the image).

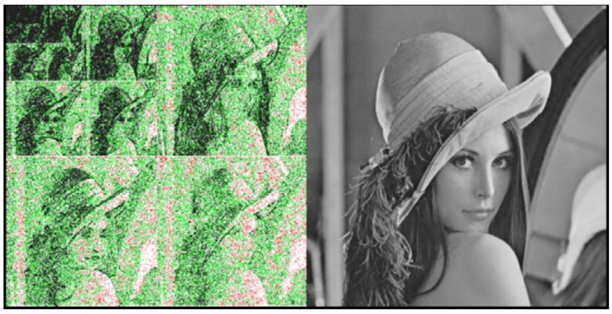
This is particularly useful for large images or when only a portion of the image needs to be displayed or processed.



Again as before, we need the quantization table inside the JPEG 2000, in order to decode it. The final result of these processes is a better decoding phase, where even few details allow to show images less blocky with respect to the original JPEG. Each time information (also called *plane*) is added.







It’s very difficult to distinguish between the different planes.

Comparing the visual quality of JPEG and JPEG2000:

- The JPEG images show more noticeable compression artifacts and distortion, especially around the woman's eyes, hat, and hair. The artifacts manifest as blocky, pixelated regions that degrade the overall sharpness and detail
- In contrast, the JPEG2000 images appear smoother and retain more detail. The woman's facial features, hat texture, and hair strands are noticeably clearer and less distorted compared to the JPEG versions. JPEG2000 seems to introduce less visible compression noise

The <u>ROI: Region of Interest</u> is a special technique that isolates an important image area and encodes it with higher quality than the rest of the image (usually the background).

The method is called *MAXSHIFT* and is based on shifting coefficients related to the *ROI* to the highest bitplanes. On the right we see an example of ROI – left one is without MAXSHIFT, right one is with the method.

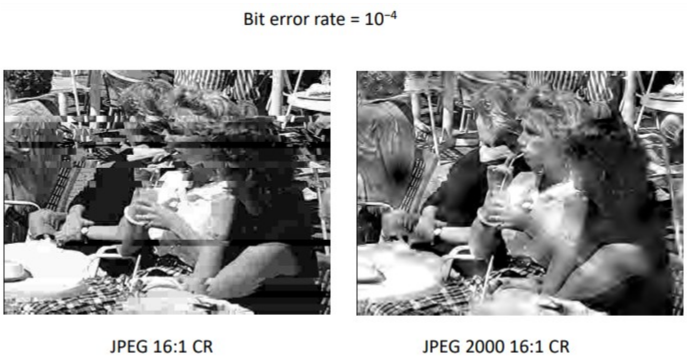
The image demonstrates the Region of Interest (ROI) feature in JPEG2000 compression using an example photo of a woman wearing a hat.

- In the left image, the entire photo is compressed uniformly. While the woman's face is the main subject, it receives the same compression treatment as the less important background areas
- However, in the right image, JPEG2000's ROI capability is applied. The woman's face is designated as the ROI and is encoded at a higher quality level compared to the rest of the image. This ROI is shifted to the higher bitplanes using the MAXSHIFT method, prioritizing its visual fidelity

The example here compares the error tolerance between the two formats. Basically, the difference is saved from the first bits recursively (given it’s easier to save) – if we lose the DC value, the difference represents the variability compared to the variance, in JPEG is completely lost, like you see below.

In the first image, at this error rate, the JPEG image shows significant distortion and loss of detail. Consider the white stripe there, left by JPEG encoding, because we lost the DC component related to the block and color is not present there.

In contrast, the JPEG2000 image remains largely intact apart from speckling artifacts om the far right of the image. The image is shady but safe from clear errors.



The JPEG image has degraded further to the point of being unrecognizable. The blocky distortions have compromised some meaningful information.

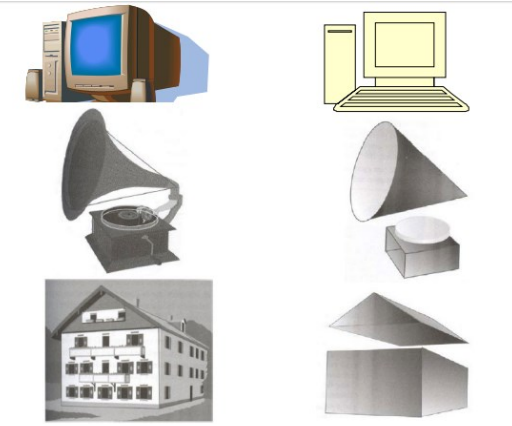
While the JPEG2000 image shows more pronounced speckling compared to the 10^{-5} BER case, the overall scene and details are still apparent and comprehensible, despite the bit errors.

Written by Gabriel R.

6.3.6 Vector graphics

Up to now, we talked about *pictorial images*, which are represented as matrixes of points (*pixels*); at each point, a color is associated.

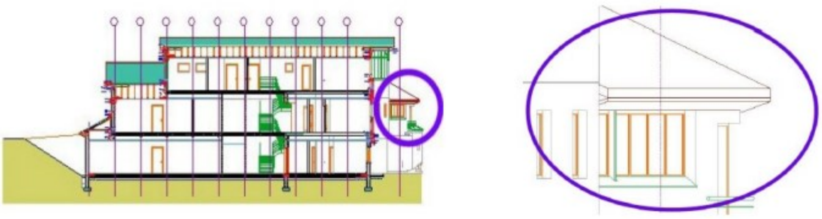
- Vector graphics are described using *geometrical formulas* (lines, curves, polygons...)
  - that define *shapes*, *color fills*, and *positioning* using only a mathematical point-of-view
- A great advantage in this case is that they are really easy to compress being only formulas
  - we can even use a lossless compression like LZW
- Pictorial images need to be acquired
  - artificial images are the result of elaboration from the calculator



There are characteristics of vector graphics:

- High precision for detailed draws
- Quality is independent of pixel
- Compact dimension
- Details enlargement without quality loss (*scalability*)
- Easy to manipulate

When a vector graphic is scaled, the mathematical equations are recalculated to adjust the size of the primitives while maintaining their proportions and relationships. Next figure shows just that.



Lastly, describing some features:

- For graphics and paper advertisement because easily editable and scalable
- For urban and construction design and for industrial design (3D graphics)
- Typography (Typeface description)
- Animation (videogames)

The following are a bunch of JPEG2000 test images (probably as summary, were not treated in lesson, but to archive data, I put them inside of notes anyway):

Dear Pam,

I was delighted to hear from you last week. Patti and I had a wonderful time during our week-long summer vacation. The weather was excellent, and the food was absolutely exquisite. I hope that we can repeat this next year and that you will join us too.

We came back with a lot of fantastic memories, which we would like to share with you through some snapshots that we took.

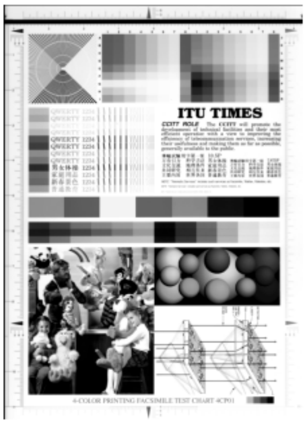
Summer of 1994.

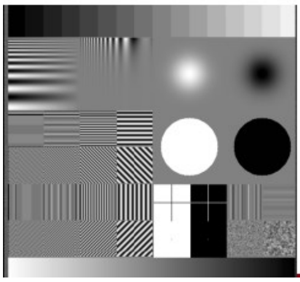
I Love New York.

Our favorite is this picture of us aboard the "Top Hat", which I have pasted into this letter using some really neat advanced digital imaging technology on my home computer. We will ship the rest to you or a CD-ROM soon. Wishing you the best.

Love,

Susan



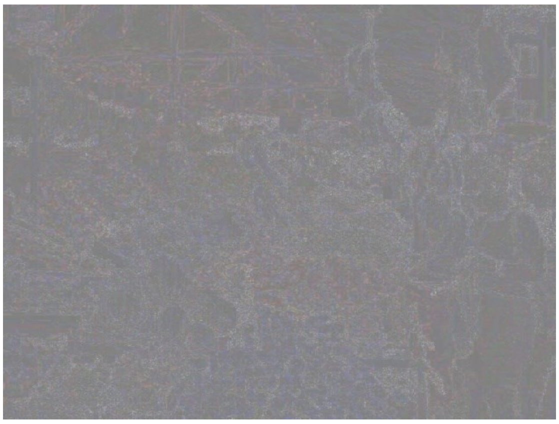


Once again, we refer to images present in the “Images” folder of Moodle. Here, we consider the Market gif. There are some pixels, but overall colors are pretty well displayed.

Now, we have the same picture, but this time in .jpg format. The perceived quality is higher.

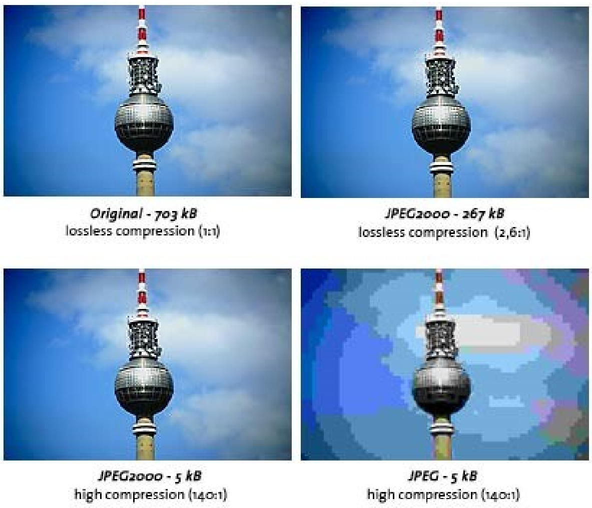
When you decrease the quality in this format (in this case, up to 10%), we definitely see the blocks.

In the same sets of images, considering the differences between JPEG/JPEG2000 (sets of images which start with Diff):

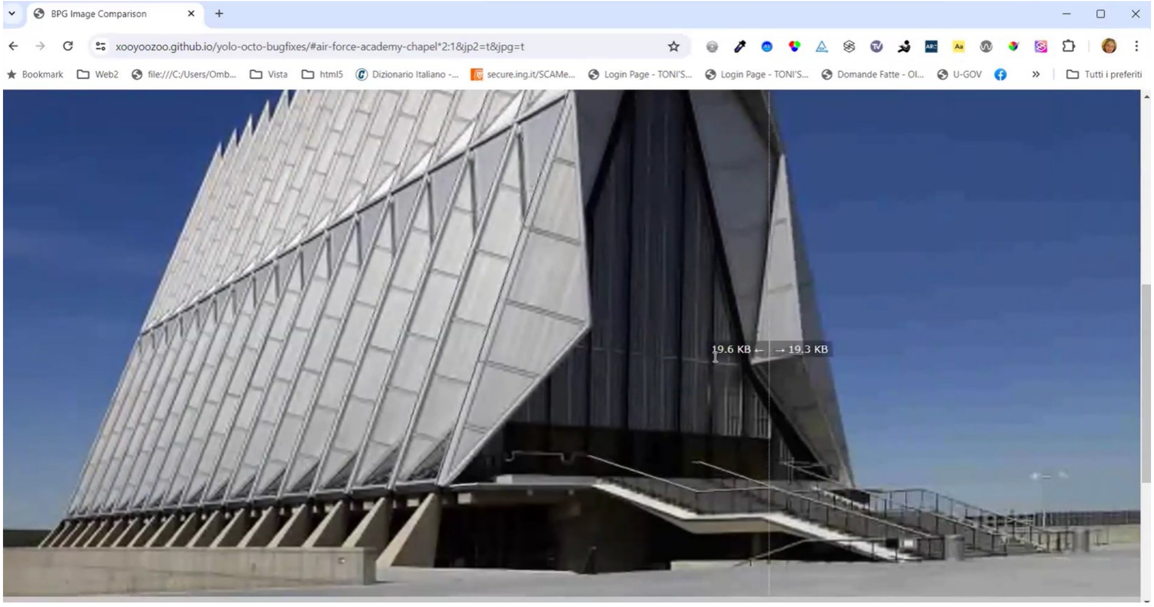


When you try to decrease quality inside of JPEG2000, you do not see the blocks, given the DWT is applied to the whole image:

Again the example of JPEG2000, considering how compression works compared in both JPEG/JPEG2000. The latter is definitely more complex for the encoding, given this drains battery and resources.



Again, another comparison between JPEG2000 and JPEG, in which JPEG shows blocks.



Some material linked at the end of this set of slides:

- Official site
  - http://www.jpeg.org/
- Additional material
  - http://www.di.uniba.it/~laura/JPEG2000/JPEG2000.htm

6.4 EXERCISES 5 (OPTIONAL ONE) – IMAGES

(Note: this exercise is available via 4 fixed questions on Wooclap, which are to be done asynchronously and then commented on the next class. Because I'm a good person and I like people to be organized upfront, here are the questions for you. Consider the quiz on Wooclap was closed up until the next day and she just told it at random without any kind of Moodle announcement)

Important: She says this counts as a homework which can possibly give you points and make you recover other homeworks not passed. So, I suggest you complete it.

1. *Why isn't CMY suitable for printers?*

Observations:

CMYK is subtractive model, given it absorbs the light and the element is black. Instead, RGB adds color because it projects to the maximum level of luminosity all of the colors (so, red to the max level, green to the max level, etc.).

Using this model to get a black surface we have to prevent it from reflecting all the primaries (red, green, and blue), so we have to color it cyan, magenta, and yellow at maximum intensity. Using the maximum amount of ink of all the three components allows to obtain black in printing.

Theoretically mixing the primary colors C,M,Y should result in black. In reality, however, this does not happen because of impurities in the ink pigments. For this reason it is also necessary to introduce the component K.

Written by Gabriel R.

Answer (Claude.ai):

CMY is not suitable for printers because the impurities in the cyan, magenta, and yellow ink pigments prevent them from producing a true, deep black when mixed together. Instead, the result is often a muddy, dark brown color (also, it's too costly to actually do that). The addition of a separate black ink (K) in the CMYK color model overcomes this limitation and allows for the production of high-quality black and grayscale prints. The other colors do not absorb light very well.

Actual answer:

All of the colors represent a different level of luminosity. In order to actually obtain black, we should represent all of the color components to their maximum level of luminosity. Because of impurities, we do not actually encode them completely, hence obtaining a muddy brown color (other colors do not absorb light very well). Hence, the addition of a separate black ink (K) in the CMYK color model overcomes this limitation

<u>Answer</u>

It's quite impossible to have a good/perfect black with CMY, so putting the exact drop of color of each one it's very hard. Since we want to use black, we need it as one of the colors. The second reason is that it costs a lot.

2. *Why do YUV, YIV and YCbCr color models use blue and red chrominance and not green chrominance?*

Observations:

Cb and Cr respectively represent the blue-difference and red-difference chroma signals as you noted. Chromaticity represents the perceptual deviation from the neutral, i.e. grayscale, so the greater the deviation, the higher the intensity of chroma. An image with low chroma is less colorful, and conversely. And so, Cb and Cr are one representation of changes in blue and red "colorfulness", respectively.

They have a channel for luminance and two channels for chrominance. They decided to use red and blue, to avoid wasting one channel. One was used only for black/white TV (Y), which is what we perceive as luminosity by rodes. The information about green is very similar to the one of luminance and half of this information is exactly.

The YUV, YIQ, and YCbCr color models use blue and red chrominance channels because they are designed to separate luminance (brightness) information from chrominance (color) information.

Answer (Claude.ai):

The YUV, YIQ, and YCbCr color models use blue and red chrominance channels (Cb and Cr) instead of a green chrominance channel because the luminance channel (Y) already captures most of the green information.

The visual system is most sensitive to green light, which contributes significantly to the perceived brightness of an image. By designing the luminance channel to closely correspond to the green channel, these color models can efficiently represent brightness information without the need for a separate green chrominance channel.

This approach reduces redundancy and allows for more efficient compression and transmission of video signals while prioritizing the preservation of brightness information to which the human visual system is most sensitive.

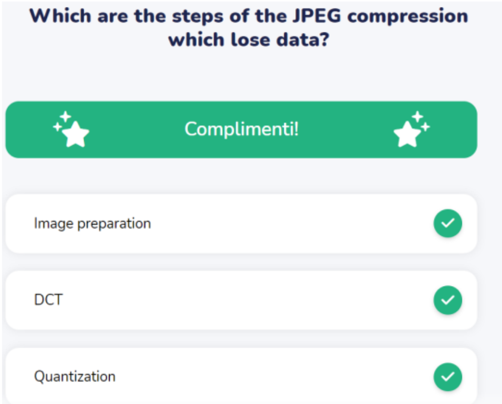
Actual answer:

The models by themselves use blue and red chrominance respectively, which represent most of the luminosity information, represented by the always present component (Y). Such component is already able to capture most of the green information; using another channel (4 instead of 3) represents an improvement of space usage and performance

<u>Answer</u>

If you look at the slide, our rodes have the same sensibility for green when having blue or red. The luminance is represented by the green color. If I use green + other colors, in the first part of the spectrum I have twice the information of the luminance and we do not need it.

3. *Which are the steps of the JPEG compression which lose data?*
    a. Image preparation
    b. DCT
    c. Quantization
    d. Matrix linearization
    e. Huffman and Zero Encoding



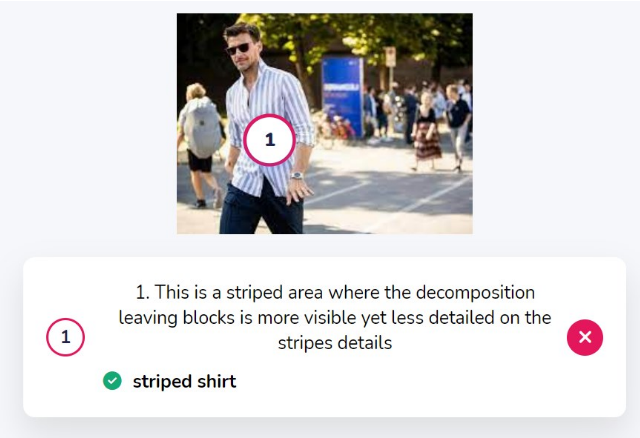
Answer:

- Quantization
- DCT
- Image preparation
    - When we move to color models
    - and we subsample the chrominance (losing at least half of the data)

4. *Identify areas of the image that can be complicated for JPEG compression*
    a. You have to fill in the labels

Dovete riempire tutte le etichette

Actual answer:



(Given the problem above for the exercise, the last question apparently changed and even the other questions seem unanswered even if you did; particularly, the last one changed here)

5. *Provide an example of image for which JPEG is not the correct choice*

Actual answer:

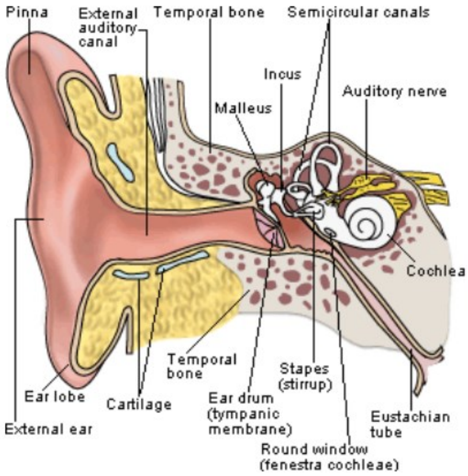
- Simple images with high contrast (even in black or white)
- Artificial images
- Images with transparency
- Line art and drawings
- Images with very high degree of detail e.g. medical/satellite images (JPEG2000 works better)

Answer:

- Solid colors with elements of high contrast
- Medical images
- Satellite images
- Artificial images

6.5 AUDIO

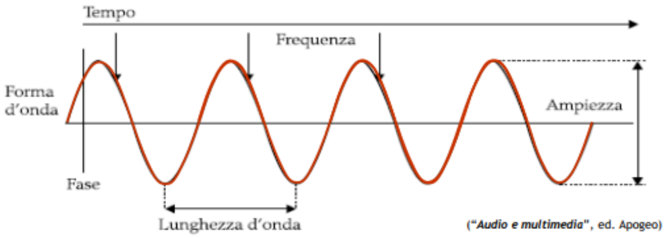
We first started considering how we *see* and getting different perspectives on how it works. Now, we will understand how we *hear*. Consider first the internal structure of the ear:



This is how it works (if you are a good guy like me, have a read here):

- Sound waves enter the ear canal and vibrate the eardrum.
  - when we talk, the vocal cords are located in the larynx (voice box) and are responsible for generating sound at its source, which then travels through the ear for perception.
- Vibrations are amplified by the middle ear bones (malleus, incus, stapes)
  - and transferred to the inner ear
- Vibrations create pressure waves in the cochlea
  - stimulating hair cells in the organ of Corti
- Hair cells convert vibrations into electrical signals
  - which are sent via the auditory nerve to the brain
- The brain processes the electrical signals, creating the sensation of hearing

The sound is a longitudinal pressure wave (basically, movement of particles from one region one another). that propagates through a transmission medium. The range of human hearing is between 16 Hz and 22 kHz. There are different measures to consider here, listed below.



- *Amplitude* (dB) → volume, the intensity of the sound (higher amplitude = higher sound)
- *Frequency* (Hz) → "higher" or "lower" sound
- *Waveform* → timbre, it allows us to recognize different types of sound production

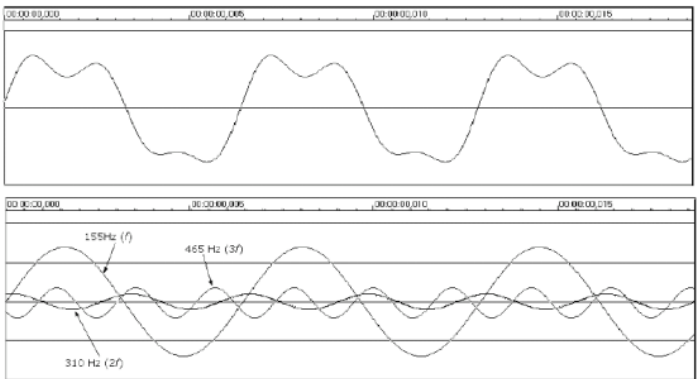
Here is a table describing the different intensity levels:

Approximate sound levels and intensities within human hearing range

Source of sound	Intensity level (dB)	Intensity (W m^{-2})	Perception
jet plane at 30 m	140	100	extreme pain
threshold of pain	125	3	pain
pneumatic drill	110	10^{-1}	very loud
siren at 30 m	100	10^{-2}	
loud car horn	90	10^{-3}	loud
door slamming	80	10^{-4}	
busy street traffic	70	10^{-5}	noisy
normal conversation	60	10^{-6}	moderate
quiet radio	40	10^{-8}	quiet
quiet room	20	10^{-10}	very quiet
rustle of leaves	10	10^{-11}	
threshold of hearing	0	10^{-12}	

The Fourier Transform is a powerful mathematical tool that breaks down a signal into its individual frequency components.

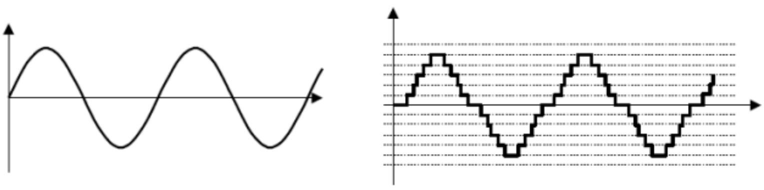
- In simpler terms, it helps us identify the different frequencies present in the signal
- Quoting the slides on the Fourier analysis of the audio:
  - "A periodic signal can be broken down into a series of harmonically related sinusoids, each one with its amplitude and phase, and frequencies that are harmonics of the fundamental frequency of the signal"

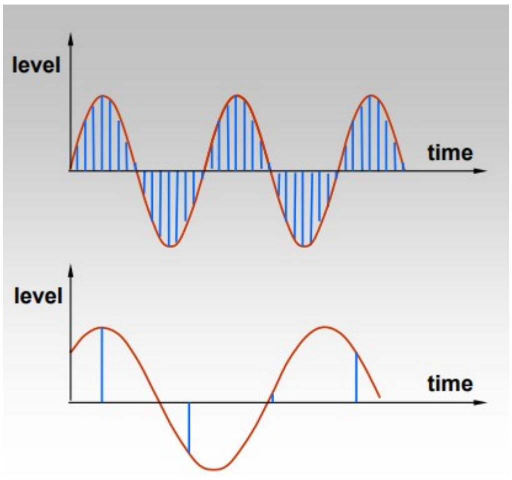


But the first step in this study is the *digitalization* process of the sound, of these sinusoids Fourier cites. The first step to elaborate a signal is to digitalize it (A/D transformation):

- *sampling*: time division (Hz)
- *quantization*: discrete representation of the signal level (measured in bits of precision)
- ex. Audio CD is digitalized at 44.1 kHz, 16 bits
  - remember 44.1 comes from Nyquist theorem
  - Sounds in the real world have frequency content right up to 50kHz. So to sample them at 44.1 kHz the Nyquist Theorem requires everything above 20kHz to be filtered out.

The digitalization of a sound can be shown below in an example, considering the real signal (left) and how the computer perceives it (right).





The *Nyquist* theorem suggests that "if a periodic signal contains no frequencies higher than N hertz, it can be completely reconstructed if $2N$ samples per second are used".

This is a huge result, since it shows we can reconstruct a real signal in a digitalized form without losing any information. In this case, the sampling rate of $2N$ samples per second is sufficient. Basically, the correct number of points to represent a wave signal (how we can regenerate it in frequency, basically).

Here, we see analogical/digital conversion of the audio signal.

Analogical signals are altered by *noise*, a random fluctuation of the signal determined by electronic phenomena.

- *The signal/noise ratio (SNR, signal to noise ratio*) is a measure of signal quality that compares the level of a desired signal to the level of background noise
  - The higher it is means the better we are eliminating the noise

$$SNR = 10\log\frac{V_{signal}^2}{V_{noise}^2} = 20\log\frac{V_{signal}}{V_{noise}}$$

- Signal-to-quantization-noise ratio (SQNR or SNqR) reflects the relationship between the maximum nominal signal strength and the quantization error (also known as quantization noise) introduced in the analog-to-digital conversion (difference to the power of 2)
  - In digital systems, the noise is the difference between the real signal and the quantized signal

$$SQNR = 20\log\frac{V_{signal}}{V_{quant-noise}} = 20\log\frac{2^{N-1}}{1/2} = 6.02N\ (dB)$$

Below, a table of the difference between audio quality and size. Talking about on how much people can ear, higher frequencies are the ones which cannot be heard anymore getting older.

Quality	Frequencies interval Hz	Sampling kHz	Bits for sample	Mono/stereo	Bit rate kbit/s
Mobile network	200-4000	8.0	8	mono	8
Radio AM	100-6500	11.025	8	mono	11
Radio FM	20-12000	22.050	16	stereo	705.6
Audio CD	20-20000	44.1	16	stereo	1411.2
DAT	20-20000	48.0	16	stereo	1536
DVD audio	20-20000	192.0	24	stereo	9216

Digital encoding and decoding of audio signals has more problems than image encoding (and video):

- Temporal structure of the audio cannot be modified (frequency)
- Audio information varies over time ("audio stop" does not exist)
- Required reproduction quality is usually higher than simple understandability

Written by Gabriel R.

The audio is a continuous media, given there is no concept of pause (in reality, more of a stop). Higher frequency (volume) is different from higher volume (tones).

Comparing *size* and *transfer time* for audio:

- Uncompressed audio of good quality usually exceeds the transmission capacity of conventional networks
  - Radio FM > 640 Kbit/sec, audio CD > 1.2 Mbit/sec
- High dimensions
  - Coded signal takes a lot of space
  - Length is not limited (live audio
- It is not possible to transfer the whole audio file before playback
  - *Streaming*: playback while receiving

There are different audio formats to consider here:

- *WAV*, Waveform Audio File
  - Developed together by Microsoft and IBM
  - Standard de-facto for audio encoding on PCs
  - Not compressed
- *AIFF*, Audio Interchange File Format
  - Developed by Apple Computer
  - Audio standard for Macintosh
  - Not compressed (there is a compressed version)
- *μ-LAW*
  - Standard audio format for Unix
  - Telephonic standard in the USA ($8\ KHz$, 8 bits)
- *A-LAW*
  - European version of μ-LAW
- *MPEG-1* Audio (more of MPEG-1 Level 3)
  - Encodes audio tracks in MPEG-1 videos
  - Compressed format for variable quality encoding
  - Compression algorithm with several steps, based on psychoacoustic principles –
  - Three different encoding levels with three different bit-rates
  - Cross-platform standard
  - Several applications for the consumer market and commercially important

Coming to think why MPEG-1 is the preferred solution by the market for audio encoding, a first consideration has to be done about the lossy compression it implements.

- Lossless compression has low performances
  - Audio data are extremely variable
  - Recurrent configurations (patterns) are rare
- It's necessary to use *lossy compression*
  - Audio information is frequently redundant
  - Compression quality can be controlled
    - Compress a channel and save the difference between the compressed ones
  - Human ear does not have a linear behavior

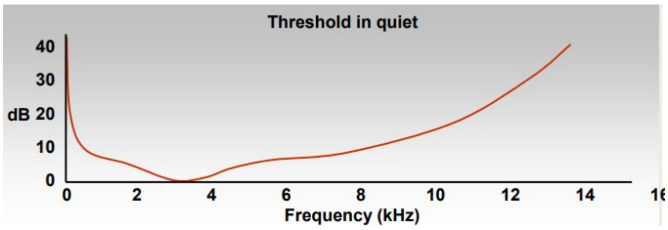
Consider the following:

- Silence compression
  - Silence is a consecutive set of samples under a defined threshold
  - Basically, remove the sample and repeat several times the same ones
  - Similar to RLE (run-length encoding) compression
- Adaptive Differential Pulse Code Modulation (ADPCM)
  - Encodes the difference between consecutive samples
  - Difference is quantized, hence there is loss of not meaningful information (lossy)
- Linear Predictive Coding (LPC)
  - Adapts the signal to a human speech model
  - Transmits the parameters of the model and the differences of the real signal to the model

But before actually applying these techniques of compression, we need to understand what we can and cannot lose, and we can do it with psychoacoustics elements (physiology of sound = how our bodies perceive it and psychology – how our brains perceive the sound) – studied by psychologists to people what they can ear and what they can't.

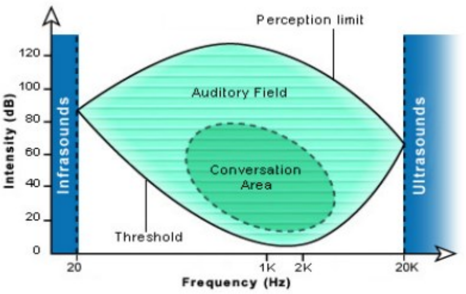
Human ear sensibility varies along the audio spectrum:

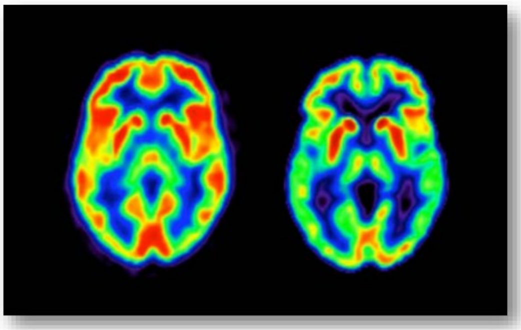
- Maximum sensibility is around $2/3\ kHz$ (called conversation area), and decreases at spectrum extremities
- Ear sensibility strongly changes depending on several personal factors (e.g., age)



The following is a bunch of data on how we perceived sound and voice:

- Human hearing interval: ~ 20 Hz to 22 kHz
- The recognizable *dynamic interval* is the interval between the weakest and the strongest sound, is ~ 96 dB
  - In the perception limit, the cochlea is not able to recognize the sound given the membrane moves so fast
- The human voice has frequencies in the interval ~ 500 Hz (vocals) - 2 kHz (consonants)





Female **Male**

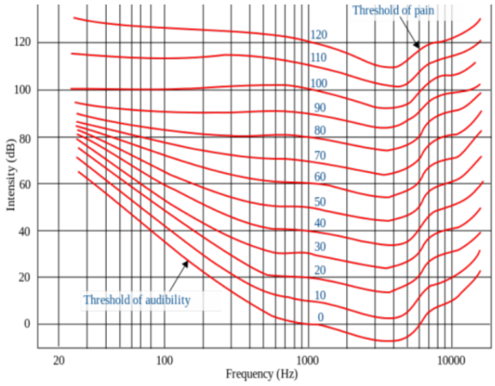
Perception changes from male to female. These are scans activated when a baby cries:

- Males may rely more on their right hemisphere when processing sounds, which could mean they are slightly more attuned to tones and musical elements
- Females seem to engage both hemispheres equally, which may indicate they process verbal and non-verbal sounds more evenly

An instrument that helps clarifying an aspect of human earing are the *Fletcher-Munson curves* (also called equal-loudness contours) are a set of curves that represent the human ear's sensitivity to different frequencies of sound at various loudness levels.

Each one represents a specific level and the perception of loudness changes with frequency. It helps us to understand exactly which sound to perceive.

We can see here the thresholds for hearing and pain, but the main focus is in the part between two red lines: in there, we perceive the sound at the same volume.



$T_q(f) = 3{,}64\ (f/1000)^{-0.8} - 6{,}5\ e^{-0.6(f/1000-3.3)^2} + 10^{-3}\ (f/1000)^4$

Are we able to perceive a sound? Not always, it depends on the frequency.

The frequencies where there is a uniform perception of the amplitude of the sound are grouped in *critical bands*.

- Each band has an amplitude from $100\ Hz$ to $4\ kHz$
- The entire spectrum of the audible frequencies is divided into 25 critical bands

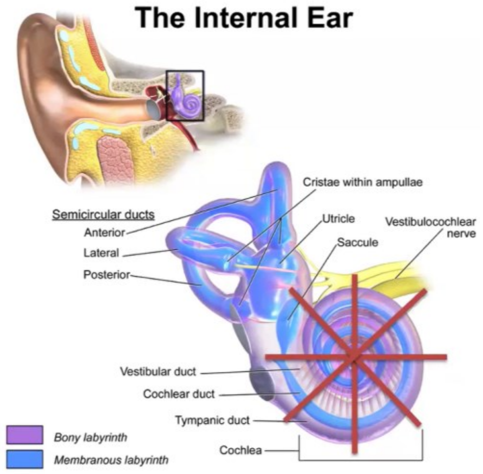
The human hearing can be considered, broadly speaking, as a series of overlapping band-pass filters (basically, "filtering out" high frequencies and only allow frequencies below a certain threshold). The bands are narrow where we hear more, while the larger ones are the ones we hear less. Barks here measure the length of the band.

– 1 Bark = amplitude of a critical band (in honor of German physicist **Heinrich Georg Barkhausen)**

Per frequencies lower than 500 Hz: $1\ Bark \approx freq/100$

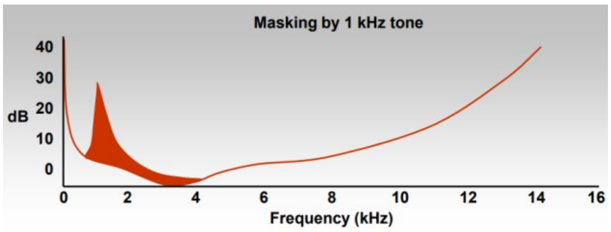
Per frequencies higher than 500 Hz: $1\ Bark \approx 9 + 4\log\,(freq/1000)$

The stereocilia transcodes the signal which has to be analyzed by the brain. Each sector of the cochlea is dedicated to some amount of the spectrum, having the transcoding more precise. The bigger the radius, the longer the arch, the more sensibility (and viceversa).

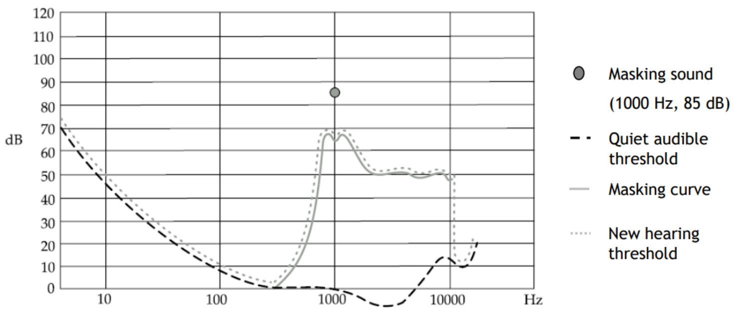


This happens with silence, but this situation of course is ideal. The main issue now is what happens when sounds overlap:

- A pure sound can mask another one with near frequency and lower volume
  - During the playback of a sound at 1 kHz
    - other simultaneous sounds in the masking interval cannot be perceived
  - We study what happens when there is more than a sound



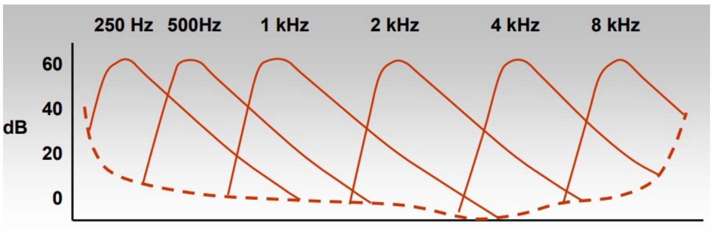
The following figure illustrates the concept of frequency or tonal masking, which occurs when a loud sound at a specific frequency (the masking sound) makes it difficult or impossible to perceive a quieter sound at a nearby frequency.



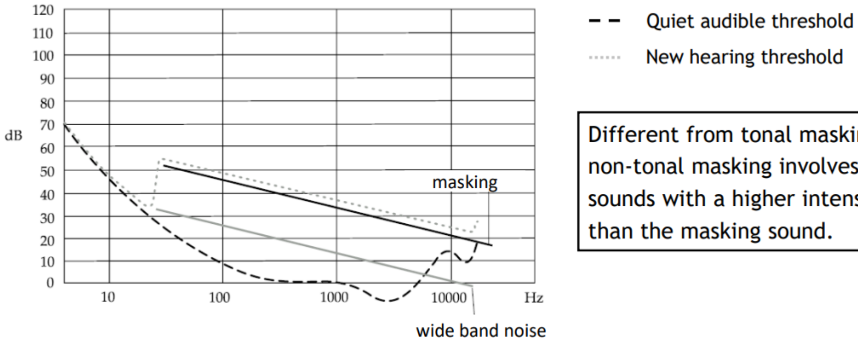
Sensibility is not linear; inside the cochlea (set of concentric circumferences with different radius), we have the stereocilia, which transcodes the signal for the brain. Each sector takes different parts of stereocilia in order to make the scanning more precise. Near sounds/lower volume are the ones which cannot be heard. Such tones are masked since they are near in frequency.

Generally speaking, *frequency masking* is an auditory phenomenon that occurs when two similar sounds play at the same time, or in the same general location.

- Different for each frequency
    - Can be defined for each critical band
- Depends on the sound amplitude



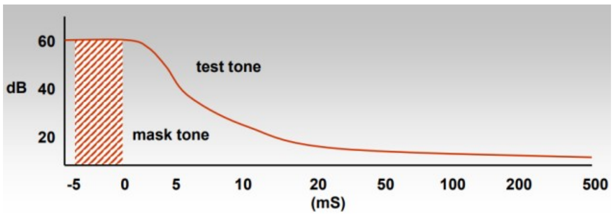
Non-tonal masking happens when the masking sound is a wide band signal (i. e. a noise) – hence, it is a higher-intensity value – where it is not possible to find a specific tone.



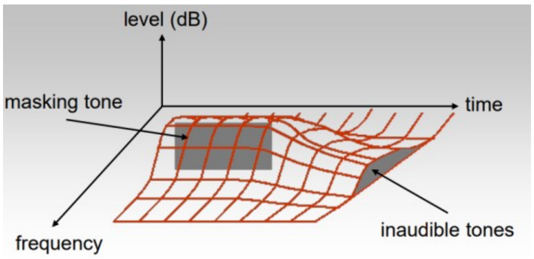
Different from tonal masking, non-tonal masking involves sounds with a higher intensity than the masking sound.

In the *temporal masking*, sound can mask another one for a small interval of time:

- the *pre-masking* hides preceding sounds between 5 ms and 40 ms before
- the *post-masking* hides the weakest sounds after the masking sound between 50 ms to 200 ms (sound takes a while to stop after hitting the tympanic membrane)



There can also be *combined masking*, which is a combination of frequency and temporal masking.



All tones below this surface cannot be heard by a human.

6.5.1 MPEG-1

MPEG-1 layer 3 (MP3) is the current standard for high-quality audio (music) with high compression.

- The most common *bit-rate* for MPEG standard is from $48 kb/s$ to $384\ kb/s$ (audio CD not compressed $>$ 1.4 Mbit/sec)
- This uses tonal masking and temporal masking
- Compression level in the interval 2.7 – 24
- Compression level of 6: 1 (256 kb/s) is almost identical to the original signal
- From 96 to $128 kb/s$, it represents the best quality for consumer applications

Written by Gabriel R.

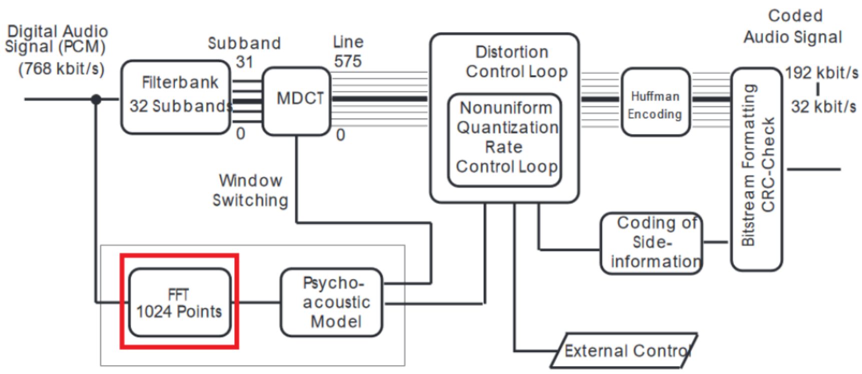
- Different sampling frequencies (32, 44.1 and 48 kHz)
- Monophonic, dual, stereo, joint stereo signal

The MPEG Audio Compression algorithm works based on four different steps based on the psychoacoustic model:

- 1 - It divides the audio signal into 32 frequency sub-bands
- 2 - For each sub-band, it calculates the masking level
- 3 - If the amplitude of the signal in the sub-band is lower than the masking threshold
    - the signal is not encoded
- 4 - Otherwise, it calculates the number of bits necessary to represent the signal (from 0 to 15)
    - such that the quantization noise is lower than the masking threshold
    - ($1b \sim 6\ dB$ of noise)
- 5 - Creates the bitstream following a standard format for transmission

Bands are grouped under the same scale factor and thanks to Huffman encoding, after quantization, the encoder tidies 576 frequency samples, ordering statistically the values occurrences, optimizing the number of bytes. Thanks to bit reservation, an encoder is able to satisfy the variable requests from the encoder.

Below, the entire process at the *MPEG audio encoder*:



Basically, employs MPEG encoder employs a bank of filters to:

- Analyze the frequency ("spectral") components of the audio signal by calculating a frequency transform of a window of signal values
- Decompose the signal into subbands by using a bank of filters
    - Layer 1 & 2: "quadrature-mirror"
    - Layer 3: adds a DCT; psychoacoustic model: Fourier transform

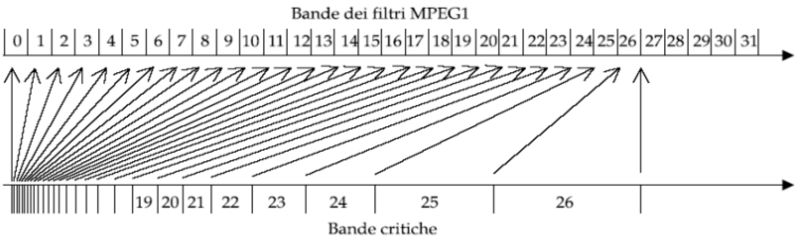
We use two different mathematical tools to actually save time and energy. This allows to have more precise calculation for the masking.

Consider MPEG is an *asymmetric* compression, so there is more time employed in encoding (compression) compared to decoding (decompression). The entire process can be briefly decomposed as follows:

- Input
    - Digital Audio Signal (PCM) at 768 kb/s

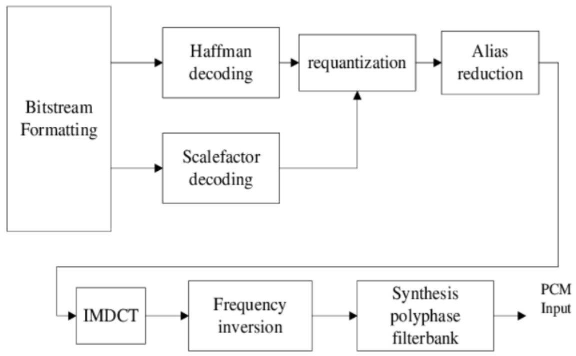
- MDCT
    - used to check whether or not the sub-band is lower than the masking noise
- 32 Subband Filterbank
    - Splits the signal into 32 frequency subbands
    - Simple signals with only 1 frequency
- FFT (Fast Fourier Transformation)
    - Converts time-domain signals to frequency domain and divides the signal into subbands
    - Compared to normal version, FFT exploits symmetries to simplify calculations of the Fourier function
    - Precise hence heavy computationally to compute
    - The signal is decomposed into frequencies, so there is the inverse sinusoidal process
        - Inverse FFT (more time/energy)
- Psycho-acoustic Model
    - Determines perceptually less important parts for compression
    - We simply need to save the masking threshold to the file without recomputing
- Distortion Control Loop
    - Controls quantization to maintain target distortion level
- Nonuniform Quantization Rate Control Loop
    - Optimizes bit allocation for each subband
- Huffman Encoding
    - Lossless data compression technique for quantized subband samples
- Coding of Side-information
    - Encodes additional data (e.g., subband allocation, scalefactors)
- Bitstream Formatting
    - Formats encoded audio and side-information into MPEG audio bitstream
- Output
    - Compressed audio signal at 192 kb/s or 32 kb/s

These are all of the filters and critical bands for MPEG-1 filters. They are mapped differently according to size and criticality; for instance, more subbands can be mapped to upper bands of various kinds:



The encoder is more precise since it has the FFT, which requires more time and more energy. So, it slower than the decoder.

Below, the entire process at the *MPEG audio decoder:*



To summarize:

- The bitstream formatting block takes in the compressed audio bitstream.
- Huffman decoding and Scalefactor decoding blocks decode the compressed bitstream.
- The outputs from these decoders go through requantization to recover the frequency domain coefficients
- Alias reduction is applied to these requantized coefficients
- The coefficients are then converted to time-domain samples using IMDCT (Inverse Modified Discrete Cosine Transform)
- The time-domain samples undergo frequency inversion
- Finally, the samples are passed through a synthesis polyphase filterbank to generate the reconstructed PCM audio signal output

The following is an Audio MPEG example. Instead of saving the whole signal, we save only a fraction, hence requiring more bits.

The level at band 8 is 60 dB

- Masking is 12 dB on band 7, 15dB on band 9

The level at band 7 is 10 dB (< 12 dB), it is ignored

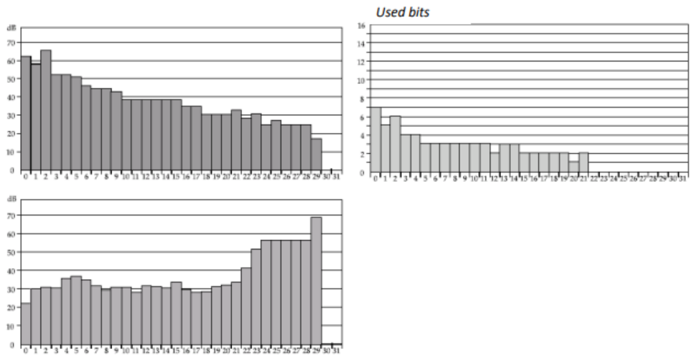
The level at band 9 is 35 dB (> 15 dB), it is encoded

Only the difference between the signal and the masking threshold is encoded

- Using 4 bits instead of 6 (2 bits = 12 dB)

Band	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Level (dB)	0	8	12	10	6	2	10	60	35	20	15	2	3	5	3	1

The following describes the bits allocation for each band, which describes the level of the signal inside of the whole band. In each, we consider the time and the number of used bits. If the thresholds are precise, such model can work well:



Another masking application is the <u>watermarking</u>, which is the inclusion of digital information (source, destination, copyright information, access information, etc…) hidden inside multimedia data (images, videos, audios, texts, animations).

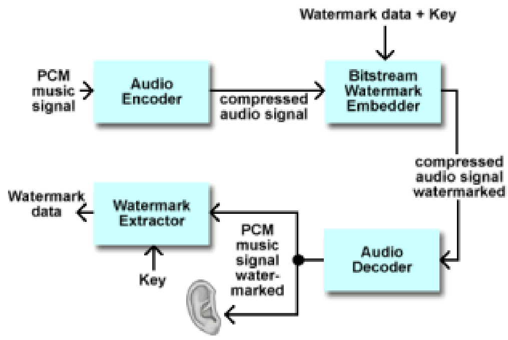
The watermarks (information):

- Cannot be modified
- Do not have to modify the enclosing data
- Must survive to all the operations done on the signal
- Must be directly connected to the data (not in the header)
- Must be statistically invisible

Watermarks insertion inside an audio bitstream works with a sort of masking technique that is ‘’opposite’’ to the MPEG algorithm.

The watermark is inserted near high-level signals, such that it will be masked by the latter. In this way, it is not distinguishable from the original one.

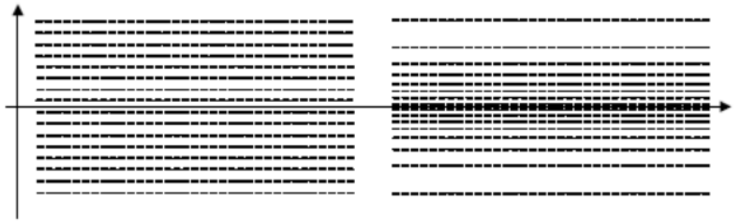
Subsequent MPEG encoding would delete the watermark. Other techniques insert the watermark with a frequency outside the human hearing range.



Let’s describe the different MPEG encoding layers:

- *Layer 1* (bitrate higher than $128\ Kbps$): DCT filter with only one frame and equal distribution of the frequencies in the sub-bands
  - Psychoacoustic model uses only frequency masking
  - Each frame has 32 blocks of 12 samples, a header, an error-detection code (CRC, Cyclic Redundancy Check), and optional additional information
  - It is not a good idea to resend a packet with this kind of transmission, btu just to add some noise to the transmission
- *Layer 2* (bitrate equal to $128\ Kbps$): works with three frames during the filtering process (previous, current, next, 1152 samples in total) – three sets of samples for each subband
  - Partially works with temporal masking
  - Uses a more compact representation of additional information (header, number of bits for each band, …)
- *Layer 3* (bitrate at $64\ Kbps$): divides the frequencies spectrum into different sub-bands with nonequal amplitudes, more comparable to the critical sub-bands in the lower frequencies
  - Psychoacoustic model with temporal masking
  - It considers stereo redundancy
  - Variable bitrate:
    - Huffman compression with pairs of values
    - Uses a bits reservoir with bits from frames that do not need them and can be allocated to the frames that do – such bits can be taken by the subbands
    - This is why MPEG works better compared to other methods, saving bits even when using more data and saving it
    - If data is below of the threshold, we lose the signal

The following is an example of non-uniform quantization:



It's hard to actually recognize the difference between the values with our sensibility. Non-uniform quantization in MP3 is a process that considers the psychoacoustic model of human hearing to allocate more bits to the frequency ranges that are more perceptible to the human ear while allocating fewer bits to less perceptible frequencies.

- More bits are allocated to quantize the perceptually important middle frequencies that we are most sensitive to
- Fewer bits are used for the extreme low and high frequencies that are less noticeable to human hearing

We now have the table of the MPEG audio quality:

Layer	Target bit rate	Compression	Quality at 64 kb	Quality at 128 kb	Delay
Layer I	192 kb/s	4:1	--	--	19 msec
Layer II	128 kb/s	6:1	< 3	4+	35 msec
Layer III	64 kbit/s	12:1	< 4	4+	59 msec

Quality factor: 5 - perfect, 4 – barely noticeable, 3 – slightly annoying, 2 - annoying, 1 - very annoying

Here, the following MPEG audio formats:

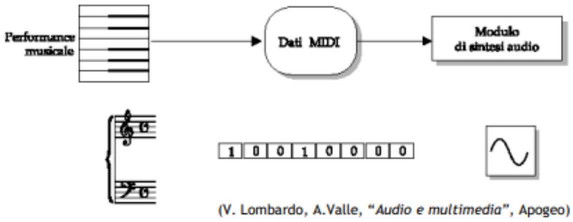
- *MPEG2* (November 1994)
  - Is the standard for DVDs – mainly used for video compression and transmission
  - It was aimed at transparent sound reproduction for theaters. Works with five channels (left, center, right, left-surround, right-surround), plus a low-frequency enhancement (LFE) channel for very low frequencies (subwoofer)
  - Works at 16 kHz, 22.05 kHz, or 24 kHz plus MP3 rates

- *MPEG4* (December 1999) – MP4
  - Audio (and images) are considered a composition of different objects
  - The user can decide to listen to a concert situated in different places/environments or to emphasize some sounds over others
  - Each track is dedicated to a singer/some part of music/etc.

6.5.2 MIDI - Musical Instruments Digital Interface

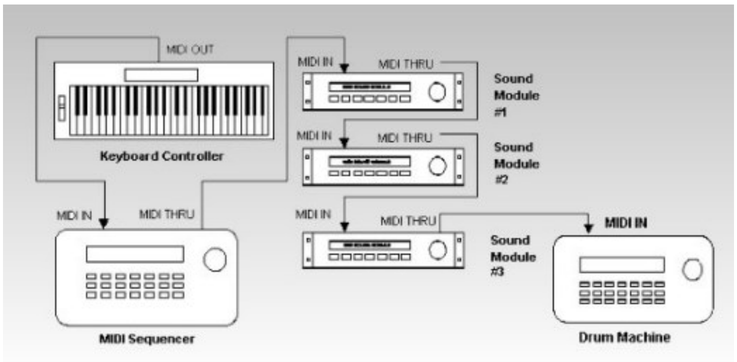
The MIDI protocol (1983) provides a standard and efficient way to describe musical events, basically in the same ways a score is used for real-life instruments.

- It enables computers, synthesizers, keyboards, and other musical devices to communicate each other – was created in the 80s for western-based music on semitones
    - Consumes very few memory – so it's used as background/videogames/for Web
- MIDI is a scripting language – it codes ''events'' that stand for the production of sounds
- Sound generation is local to the synthesizers
- Messages describe the type of instruments, the notes to play, the volume, the speed, the effects, etc.



(V. Lombardo, A.Valle, "*Audio e multimedia*", Apogeo)

MIDI systems can be very complex by themselves, but most of the soundcards come with all the necessary hardware. Consider it's mostly used for scores or composing. For example, in MP3 it's impossible to change an instrument, you have to record it again. Here you simply change a parameter, basically.



Above, of relevance is the MIDI *Sequencer*:

- A recording and execution system for storing and editing a sequence of musical events, in the form of MIDI data
- It receives data from the input device, allows editing, and creates the music sending data to the synthesis device (example, sound card)
- It does not influence the quality of the sound. The quality totally depends on the synthesis device (or the synthesizer)

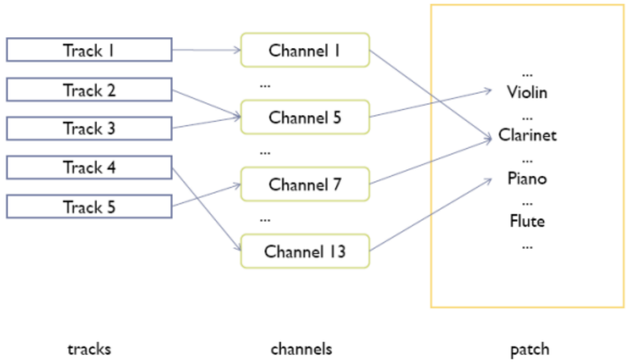
MIDI is organized through:

- *Channels*
    - They allow to send and receive music data
    - Method to differentiate timbres and send independent information:
        - different channels for different instruments

  - MIDI protocol provides 16 channels numbered from 1 to 16
  - Note: Only 60 channels are available = only 60 instruments of the same moment

- *Tracks*
  - A track is a structured autonomous flow of MIDI messages
  - Example: in a piano song, there are two tracks, the melody and the arrangement
  - It can be considered as a messages container
    - that can be assigned to different channels
- *Patch*
  - It specifies the timbre produced by the generator
  - MIDI can contain up to 128 different patches

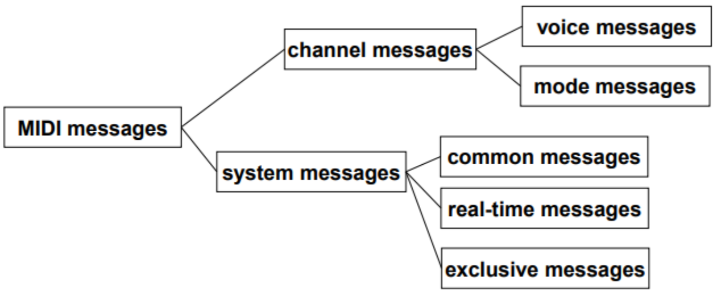
The following figure shows how music is represented in MIDI:



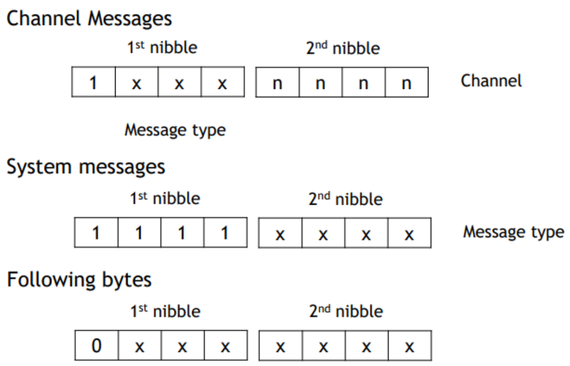
When a MIDI file is played back, each track sends its MIDI data (notes, velocities, pitch bend, etc.) through its assigned channel to the corresponding patch. This allows for the creation of multi-instrumental compositions using a single MIDI file.

By organizing music into tracks, channels, and patches, MIDI provides a flexible and efficient way to represent and control complex musical arrangements using a standardized digital interface.

Also, here a representation about MIDI messages:



We now describe the different kinds of MIDI messages (nibble here represent half a byte/4 bits), with different types of messages and the next note to play. Each message requires 20 bits – 8 bits to understand the message and the bits at start and end to divide the message:



All MIDI messages are a sequence of 10 bits ($1B$ of useful data – one at the beginning and one at the end). Now, going deeper into each kinds of message:

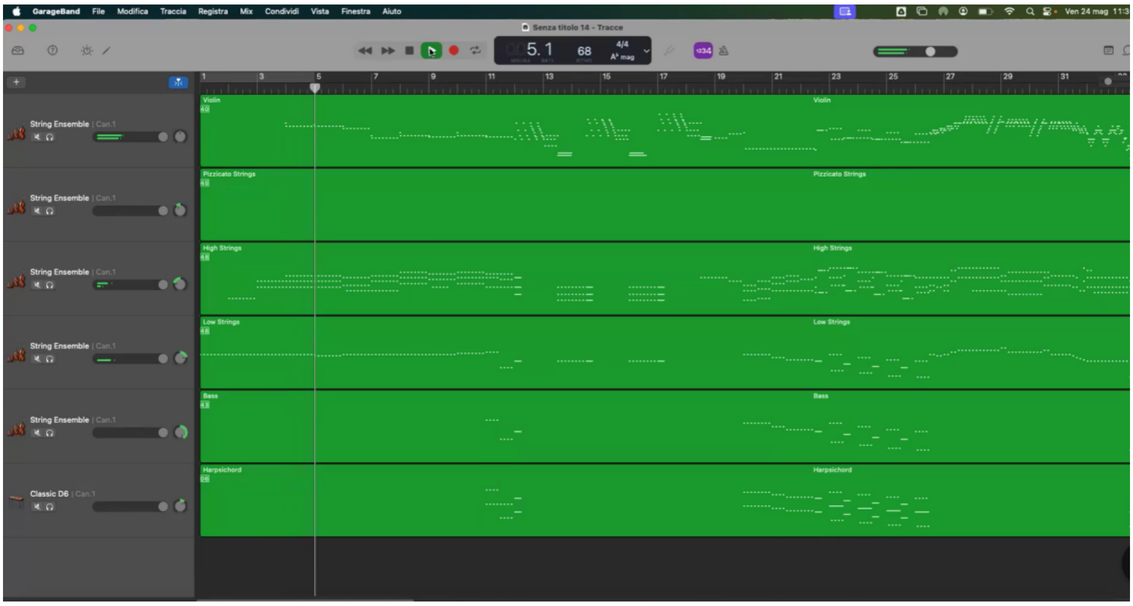
- *Channel* messages describe which note to play (voice) and how to play it (mode)
    - They contain the number of the channel through which the information is sent
    - *Voice messages* define what an instrument plays:
        - Which note to play (*Note On*)
        - Which note to turn off (*Note Off*)
        - Potential controller effects (ex. vibrato) (*Pitch Bend Change*)
        - Force measure for the keys on a specific channel (*Channel pressure*)
    - *Mode messages* describe how the instrument behaves when a voice message arrives:
        - Omni On/Off – when the channel answers to all the messages
        - Poly/Mono – more than one patch can play simultaneously
        - General MIDI Mode – standardized specification for electronic musical instruments that respond to MIDI messages

- *Systems* messages define set-up and synchronization information (there can be 16 different type of system messages)
    - They do not use a channel because meant for commands that are not channel-specific
    - Each device responds only to the messages it is enabled to answer
    - *System common messages*
        - Carry out general functions that involve the entire system
            - e.g., song synchronization when played by different devices
        - Set up a common clock – Positioning inside a song (*Song Position Pointer*)
        - Track selection (*Song Select*)
    - *System real-time messages*
        - Related to real-time synchronization of the different modules of a system
        - Device synchronization based on a relative time (24 messages every quarter)
        - Start or stop the playback of a song (*Start/Stop/Continue*)
        - Reset functions
        - System exclusive messages
        - Allow to manufacturers to extend the MIDI standard
            - sending messages that apply to their own product

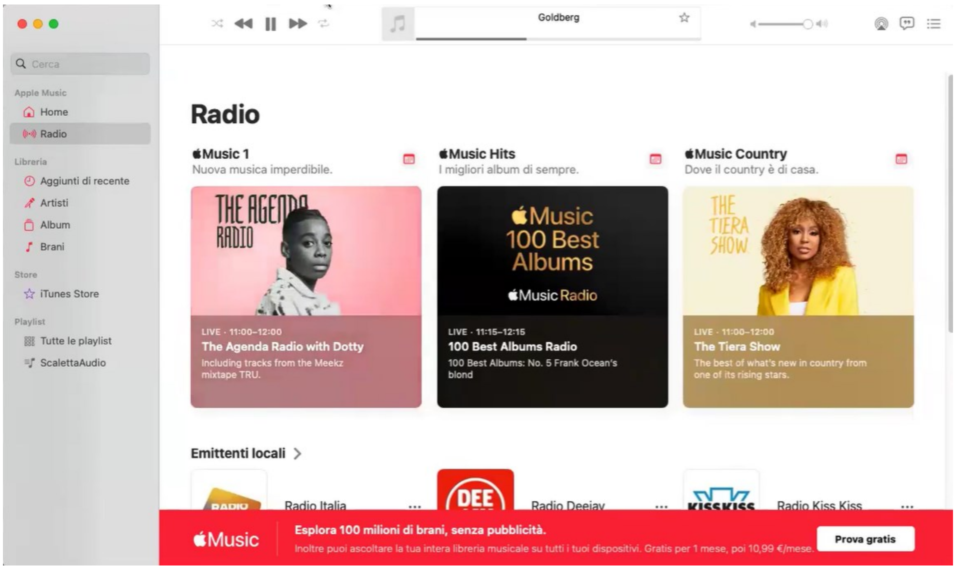
Lastly, to conclude:

- The MIDI standard is an efficient way to encode musical sounds inside Web documents
    - MIDI files are compact and have temporization information (there are no hard real-time constraints)
    - Sound encoding is based on predefined discrete events, not on the waveform of the sound
    - Complex musical songs take a small amount of storage
- Particularly suitable for background music
- But…
    - Only traditional western music can be encoded (tonal scale)
    - It is not possible to represent sounds like noise, voice, other acoustic phenomena
    - Computers and/or devices must have appropriate soundcards
    - Quality depends on MIDI equipment (synthesizer)
    - Channels and messages coding is not completely standard (ex. Roland, Yamaha, …)

This is how a complete MIDI appears – you can see the tracks and you can easily select which instruments to play or which parts in specific.



MP3 is mostly complete, while being lossy:



Essentially, a MIDI file is a set of instructions to the synthesizer to play a particular piece. It can be polyphonic, and even have different parts for different instrument patches, such as drums, but it cannot contain any recorded audio material. MIDI files can sound very different when played back through different synthesizers, because of the way patches are defined in the individual synthesizers.

An MP3 file, however, is a (lossy) compressed version of the digitally recorded audio track and should sound virtually identical when played back on different systems.

For a three minute audio track, a $128kbps$ .mp3 file would usually be about $3MB$ in size. The same song as a MIDI file (obviously, without any vocals) would probably be up to a few hundred kilobytes, by comparison, maybe considerably less.

6.6 Video

In order to analyze video, we have to study again our eyes to understand sets of images one after the other. This is not a continuous signal but a discrete one.

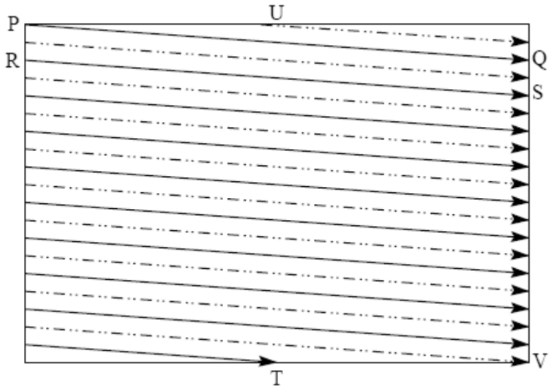
- Analog video is encoded as a continuous signal that varies over time
    - It can be digitalized
        - but not further elaborated due to the bi-dimensionality of the images
- Digital video is a sequence of digital images
    - Direct access to every frame
    - Nonlinear video editing
    - Unnecessary supplementary signals (*blanking*, synchronization, ...)

We see two important phenomena about video.

- The first, is related to the persistence of vision:
    - this traditionally refers to the optical illusion that occurs when visual perception of an object does not cease for some time after the rays of light proceeding from it have ceased to enter the eye.

- The second is a frequency which defines the frame-rate and is described as the frequency of which an intermittent light is received as to perceive the signal.
- The combination of these phenomena allow us to perceive a video.

With interlaced video scanning, instead of displaying a complete frame at once, interlaced scanning splits a frame into two separate fields: the odd field and the even field.

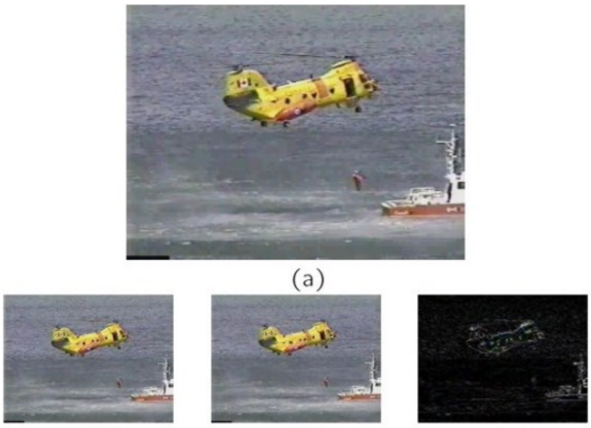


As shown in the image:

- Each frame (denoted by vertical axis labeled 'P') is divided into multiple horizontal scan lines
- The odd field, represented by the solid lines, contains only the odd-numbered scan lines of the frame (1, 3, 5, ...)
- The even field, represented by the dashed lines, contains only the even-numbered scan lines of the frame (2, 4, 6, ...)
- The odd and even fields are captured and displayed alternately, with a slight time delay (half the frame duration) between them
- When viewed in rapid succession (the time axis is labeled 'T'), the odd and even fields create the illusion of a complete frame, even though they are displayed at various times

There, each frame is divided in two types of rows, odd and even, which are shown alternatively: according to the number of rows we can achieve from 25 to 50 frames per second.

Below, an example of interlaced video:



(a)

(b) (c) (d)

In the above example:

- Fast-moving objects can cause visual artifacts known as "combing" or "interlacing artifacts"
- Artifacts occur due to temporal mismatch between odd and even fields
- Helicopter rotor blades appear jagged and discontinuous due to interlacing
- Close-up images show the comb-like appearance of the rotor blades

Written by Gabriel R.

- Interlaced scanning's limitation in handling fast motion is highlighted
- Progressive scanning captures and displays frames as complete images, reducing artifacts

There are different *types* of video signals:

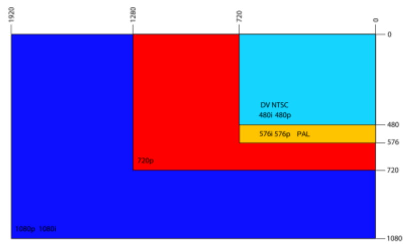
- Video with separated components
  - Each primary signal (RGB, YUV, YCbCr) is transmitted as a separated signal
  - It allows a better color reproduction due to the absence of interference phenomenon between signals
  - Requires the highest bandwidth and precise synchronization between the three signals

- Composite Video
  - Luminance and chrominance signals are mixed in a single carrier wave
  - Interference between signals

- S-Video (surpassed by HDMI)
  - Chrominance signals are mixed in a single carrier wave
    - while the luminance signal is sent separately

- Analog video usually uses a composite signal (always for transmission)
- Digital video uses a signal with separated components

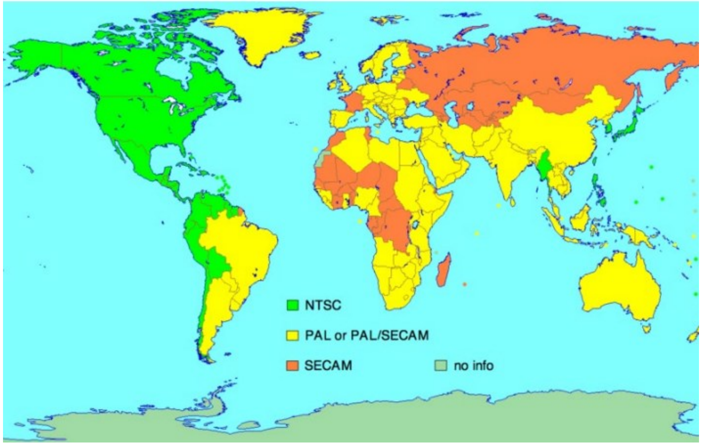
Video has different *properties*:

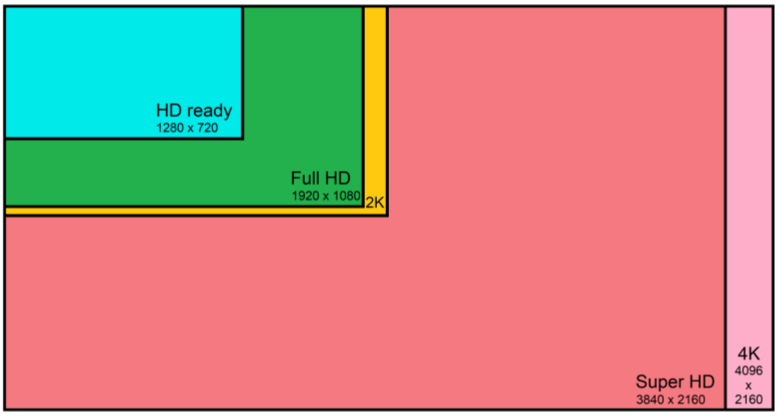


- Color depth
  - Recording encodes true-images
  - Always uses true color (8 bits for each one)
- Resolution
  - Depends on the standards
  - Chrominance information is under-sampled
  - Remember that higher resolutions require more space
    - like the example of 4K videos which need up to 5 GB per second
- Frame frequency
  - PAL = 25 frames/sec – used in Europe
  - NTSC = 29.97 ($\sim$ 30) frames/sec – used in North America and Japan
  - Minimum $\sim$ 15 frames/sec to avoid the perception of snap movements

CCIR 601 NTSC	720 x 480 (525)
CCIR 601 PAL	720 x 576 (625)

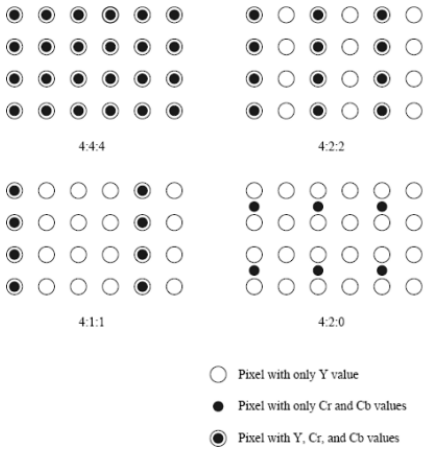
Below, two maps of TV systems and higher resolutions:





In this context we find the <u>sampling</u>, which is a ratio that describes how much information we have about a video: with a 4:4:4 we have all the full color resolution for each pixel, while as for the 4:2:2 we find, every four pixels, 2 pixels of information about chrominance (sampled half the rate), 4 about luminance, and 2 about Cr and Cb values.

Other combinations are 4:1:1 (sampled at one-quarter) and 4:2:0 (samples at half the rate – the most used).



The choice of chroma subsampling pattern depends on the desired balance between video quality and compression efficiency. Higher subsampling ratios (like 4:2:0) result in smaller file sizes but may introduce color artifacts, especially in areas with fine color details. Lower subsampling ratios (like 4:4:4) preserve more color information but result in larger file sizes.

The uncompressed video requires a considerable amount of storage, for example High Definition Television (HDTV) requires a bit-rate that can be higher than $1\ GBps$.

On the different MPEG standards:

- MPEG1 was used for VHS – very low quality
- MPEG2 was used in the first digital television versions
- MPEG4 is used by video on-demand
    - Before MPEG4 we had videoconferencing standards (e.g. H.264)
    - With MPEG4 we also have standards like H.264 included.

Moving on:

- Data must be compressed
    - 1 hour of MPEG-1 video with VHS ($352\ x\ 288$, 25 frames/sec) takes $\sim 600\ Mb$ (a CD-ROM)
- It is necessary to use lossy compression techniques
    - Elimination of spatial and temporal redundancy
    - *Intra-frame* and *inter-frame* encoding

On these ones:

- <u>Intra-frame</u> encoding (spatial compression) compresses each video frame independently without reference to other frames.
    - It exploits the spatial redundancy within a single frame by applying image compression techniques like those used in JPEG compression
    - Intra-frame encoded frames are self-contained and can be decoded independently, providing random access points in the video stream
    - However, intra-frame encoding typically achieves lower compression ratios compared to inter-frame encoding

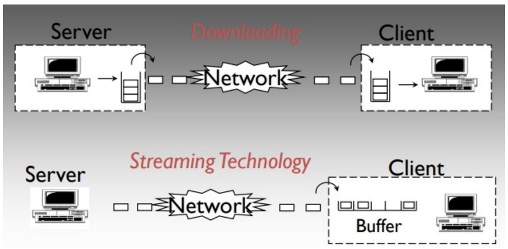
- <u>Inter-frame</u> encoding (temporal compression) exploits the temporal redundancy between consecutive frames in a video sequence:
    - It takes advantage of the fact that many parts of the video frame remain similar or unchanged across adjacent frames
    - Instead of encoding each frame independently, inter-frame encoding uses motion estimation and compensation techniques to predict the content of a frame based on previously encoded frames

Video loading from the network has the same problems of images loading, plus other factors on *transfer time*:

- A video is a temporized and continuous data
- Loading time must be compatible with reproduction time
- Playback must have a constant frame rate

A download + play solution is not always acceptable.

- It is necessary to use streaming techniques (playback while transferring data)
- Temporization control requires advanced buffering techniques



6.6.1 Motion JPEG

<u>Motion JPEG</u> was the first attempt of digital video.

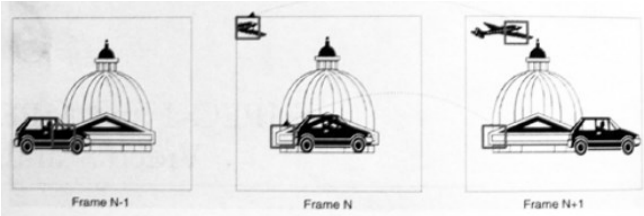
- Video signal is encoded as a sequence of frames:
    - each frame is encoded as a JPEG image
- It does not take advantage of the clear correlation between one frame and the next one

Here we consider *temporal redundancy*:

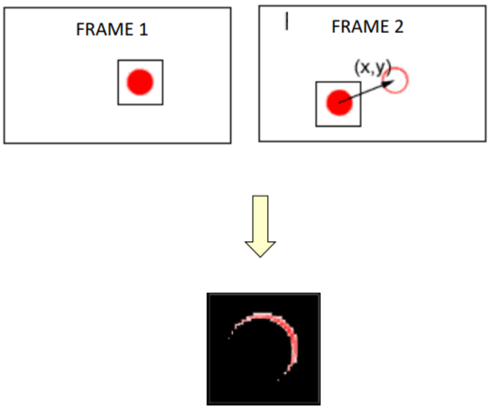
- When encoding video frames, it is possible to omit several data because, except for scene changes, there are only a few differences between two images within a small amount of time
- The differences between one frame and the next one usually depend on the movement of some pieces of the frame

(It was only quoted just to say it was a first attempt and a shitty one at that, ahah)

- Basically, reusing the previous frames, we achieve redundancy, completely reconstructing the last frame using information from the other two:



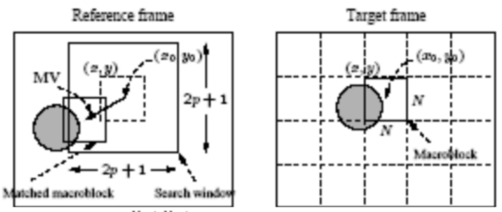
Consider the following example:



Here, using for example an algorithm of compression like JPEG (storing only once the quantization table for example – remember prof. lives for these little things). Frame 2 is encoded as the difference between frame 1. If the two images are compared, the difference is very low.

- A phenomenon that help this compression is the motion estimation: retrieving from an index image the position (x, y) of an object we can predict the next position of an object starting from an index image
    - due to the fact that in 35 to 40 frames per second an object cannot move too much
- This elaboration is done via a motion vector
    - but calculating this movement requires lots of time

Frame is divided into blocks 32×32 (called macro blocks) and, for each block, we look inside of the reference frame where is it. This elaboration is done via a *motion vector*, but calculating this movement requires lots of time. This employs the choice of different *search algorithms*:



$$MAD(i,j) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}|C(x+k,y+l) - R(x+i+k,y+j+l)|$$

N – size of the macroblock,
k and l – indices for pixels in the macroblock,
i and j – horizontal and vertical displacements,
$C(x+k,y+l)$ – pixels in macroblock in Target frame,
$R(x+i+k,y+j+l)$ – pixels in macroblock in Reference frame.

$$(u,v) = [\ (i,j)\ |\ MAD(i,j) \text{ is minimum}, \ i \in [-p,p], \ j \in [-p,p]\ ]$$

Li & Drew, Fundamentals of Multimedia, 2003

We have a frame composed by:

- intra-coded frames, which define the picture in macroblocks and apply DCT to them
  - defined in a similar way to JPEG
  - BUT do not say you use JPEG, given there is the matrix in the file with those with details of header/footer and is thought only on one frame
  - while here we have a matrix for all frames
  - They are self-contained and do not require any other frames to be decoded
  - Each intra frame is encoded using only the data contained within the frame itself

- inter-coded frames rely instead on data from other frames to reduce the amount of information that needs to be encoded (given they are very close in time)
  - Basically, to save a lower amount of bits (less information)
  - we compare the frame and the next one
  - There are different types
    - I-frames (Index Frame – which allow to have random access and are encoded like JPEG, and they are used to search within frames)
    - P-frames (Predicted Frame – hold only the changes in the image from a previous frame)
      - then calculate the difference between macroblocks and their positions

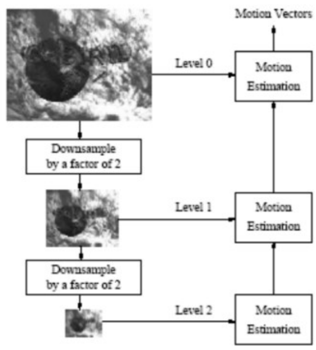
Remember: Number of comparisons depend on size of the search area and size of the macroblocks.

The choice of search algorithm involves a trade-off between computational complexity and quality of the motion estimation. More specifically:

- The sequential search/full search algorithm explores the whole space $(2p+1) \times (2p+1)$ to find a macroblock similar (minimum MAD) to the considered macroblock (compares pixel by pixel) – comparing all of the macroblocks
  - The target macroblock is compared, bit by bit, with a macroblock centered in every possible position of the research space, and the MAD is calculated
    - depends on size of blocks (size of search area)
    - and number of blocks to be displayed
  - The difference between the two positions (i.e., movement) is stored in motion vector
  - In output there is the difference between the target macroblock and the one with minimum MAD (Mean Absolute Difference) – motion estimation
    - so, when I look for the best match in the macroblock
    - once found it, I can calculate the vector describing the movement of macroblock compared to previous frame
- Computationally very expensive: $O(p^2N^2)$

The most commonly used motion estimation technique is block-based motion estimation, where the frame is divided into non-overlapping blocks (e.g., $16x16$ pixels), and the motion vector is estimated for each block.

- The <u>hierarchical research</u> algorithm works using several approximation levels in which initial estimation of the motion vector can be obtained from images with low resolution
  - It creates a pyramid of downsampled versions of the reference and target frames (subsample by a factor of 2)
  - The search starts at the lowest resolution level to find an approximate best match position, which is then refined at higher resolution levels
    - calculate minimum MAD and once found the best match
    - so to find macroblocks with best match with the image
    - basically, looking for portions where to find the best match
  - This reduces computation compared to searching only at the full resolution
  - We compose the macroblock of previous images with motion vectors – process of rebuilding is called <u>motion compensation</u> (save the vector and the differences)
    - we rebuild the frame encoding currently using the macroblock of previous frame and moving the macroblock with already calculated vector
    - = save only the information not found with movement of macroblocks
    - reconstruct current frame based on motion vectors from motion estimation



Difference between symmetric and asymmetric compression is the size of the search area.

6.6.2 H.261/H.263

The first video standard was <u>H.261</u>, developed by CCITT in 1988-1990.

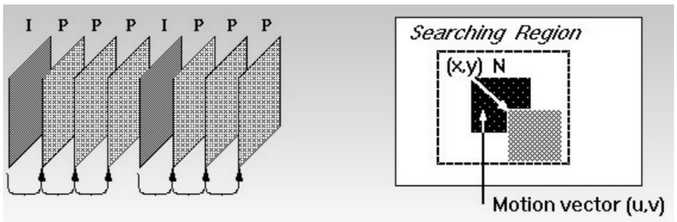
- Developed for *videoconferences* and *video calls* using ISDN telephone lines
  - 10 to 30 FPS with very small frames
- Images encoded with CIF ($352\ x\ 288$) and QCIF ($176\ x\ 144$) format, 4:2:0
- Bit-rate is $p * 64\ Kb/s$, $1 \leq p \leq 30$

Its encoding works this way:

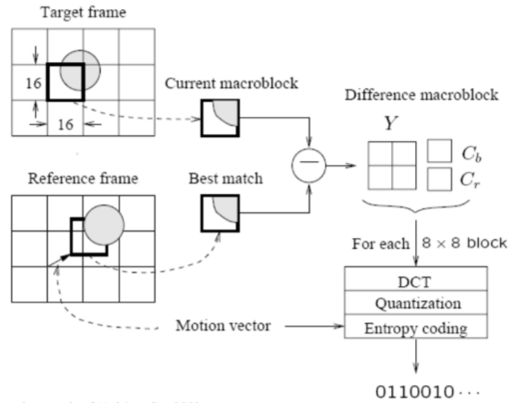
- Encoding and decoding must happen in real-time with a maximum delay of $150ms$
- Input frame rate must be $29.97\ FPS$ (non-interlaced video), while output frame rate varies between 10 and 15 FPS (slow motion)
- Color space YCbCr with chrominance components downsampled
- Two different frame types: intra-frames (*I-frames*) and inter-frames (*P-frames*)
  - *Intra-frames*: treated as independent images, frames of the video
    - they use the algorithm of JPEG (not the standard, to avoid headers)
  - *Inter-frames*: encoded using information from other frames
    - estimated with motion estimation and compensation

The standard H.263 instead, uses PB frames for inter frames compression, meaning it is predicted and bidirectional: the difference is hence calculated from the previous and following frame.

- H.261 and H.263 encode video frames based on analysis of differences with previous frame
  - Only differences are encoded, and the content is rebuilt using comparison
  - Motion compensation estimates movements of small portions of the image between subsequent frames, and encodes the difference with the estimation (H.263)



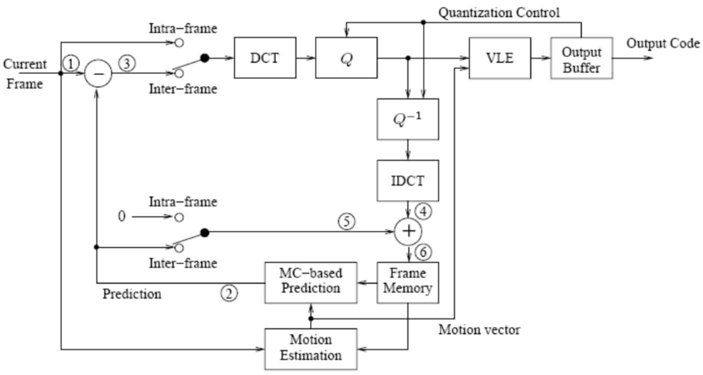
The following figure depicts the process for encoding a predicted frame (P-frame) using a previous reference frame.



ındamentals of Multimedia, 2003

The figure above shows the H.261 P-frame coding scheme based on motion compensation. For each macroblock in the Target frame, a motion vector is allocated by one ofthe search methods discussed earlier. It is also carried in the form of four Y blocks, one Cb, and one Cr block. Each of these 8 × 8 blocks goes through DCT, quantization, zigzag scan, and entropy. The motion vector is also coded, but not directly. P-frame coding encodes the difference macroblock (not the Target macroblock itself).

The image here provides a more detailed view of a video encoder based on the H.261 codec standard. The current frame is split into macroblocks. This also contains a decoder, using the inverse of quantization and DCT, using the block for the best match.



Li & Drew, Fundamentals of Multimedia, 2003

- For each macroblock, motion estimation is performed by comparing it with reconstructed previous frame data to generate motion vectors
- The macroblock is then predicted using motion compensation, and the prediction residual is transformed (DCT), quantized, and entropy coded (VLE)
    - An I-frame is usually sent a couple of times in each second of the video
- The quantized coefficients are inverse transformed (IDCT) and added to the motion-compensated prediction to reconstruct the macroblock, which is stored in the Frame Memory for use as a reference for the next frames

- The output bitstream consists of the quantized residual coefficients, motion vectors, and other side information

More specifically, on the H. 263 standard (1996): better encoding for low bit-rates.

- Images format is variable from $128 \ x \ 96$ to $1480 \ x \ 1152$
- Compression algorithm is better and able to encode video flows
    - with a bit-rate lower than $64 \ Kbps$
- Includes several techniques for error corrections

The intraframe encoding works with:

- *PB-frame* to increase frame-rate without increasing bit-rate
    - B-frames consist of two sub-frames: a P-frame (Predicted frame) and a B-frame (Bi-directionally predicted frame)
    - Predicted frame and bi-directional prediction
        - meaning it can use both a previous frame and a future frame as references for motion estimation and compensation
- Motion vectors without restrictions
- Advanced prediction: Motion compensation precision reaches $\frac{1}{2}$ pixel

Once calculated the differences between two frames, only direction and movement entity are transmitted (*motion vector*).

- H.263 allows the motion vector to refer pixels outside boundaries of the image (*unrestricted motion vector mode*)
    - associating the nearest pixel to the edges of the image
    - to the one pointed by the MV, external to the image

Another thing added inside of image is the *Integer Pixel Motion Estimation*:

- Image divided into macroblock (MB) of $16x16$ or $8x8$
- For each macroblock, a motion vector is calculated, looking for the most similar MB in the previous frame
- Research takes place in the neighborhood of the original position, moving horizontally and vertically for $\pm$ 15 pixels, one pixel per time

This achieves a better quality with more time consumed, which were the reasons that brought to the creation of MPEG.

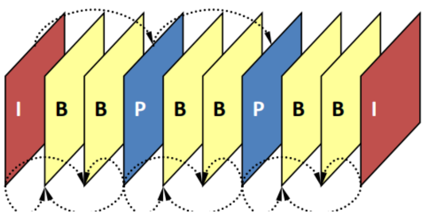
6.6.3 MPEG - Motion Picture Expert Group

The first MPEG version (MPEG-1) was released in 1991 and allows compression of a sequence of images and storage on a CD. Initially, was in the quality of VHS but then improved.

- It allows random video access and fast searches
- The compression algorithm is highly complex but <u>strongly asymmetric</u>
    - it assures a real-time decompression
- As H.261 standard, MPEG video works with the *YCbCr* (8 bit) color space
    - with down sampled chrominance components
- Luminance resolution cannot be higher than 768×576 pixels
- It does not support interlaced video
- Different resolutions and refresh frequencies allowed (from $23.98\ FPS$ to $60\ FPS$)
- The video information has:
    - Spatial redundancy → encoding of each single image
        - JPEG encoding
    - Temporal redundancy → relation between following frames
        - Diversified encoding for each frame

MPEG expands H.261 and H.263 compression algorithms with a more sophisticated scheme of motion estimation for its *compression algorithm*:

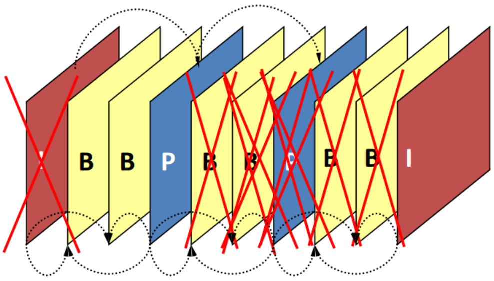
- I frames (*Intra coded frame*) are encoded using a JPEG algorithm, independently but with lower quality
- P frames (*Predictive coded frame*) are encoded based on an estimation referred to the previous I or P frame
- B frames (*Bidirectionally predictive coded frame*) are encoded using two motion estimations related to previous and following frames (bidirectional estimation)



Going into more details about MPEG frames:

- The *Intracoded frames* – exactly JPEG
    - $\frac{1}{9}$ to $\frac{1}{16}$ frames for this one
    - Require higher memory space
    - Stop errors propagation due to transmission
    - Make random access possible
- The *Predictive coded frames*
    - Differences calculation is based on the absolute value of luminance components
    - ''Smaller'' but propagate transmission error
- The *Bidirectional predictive coded frames*
    - The most complex
- More I-frames allow random access in more time points, but increase bit-rate
    - IBBPBBPBBIBBPBBPBB...
    - There must be one I-frame every 15 frames

As an example, we cannot render the second B frame without the previous I and the following B frames, similarly, without the first we wouldn't be able to encrypt the first P. Information to decode other frames is saved before other frames.



The motion compensation and prediction algorithm works in *three phases*:

- Motion estimation of objects and motion vector creation
- Frames estimation using information collected in the previous phase
- Comparison between the estimated frame and the real one to calculate the error

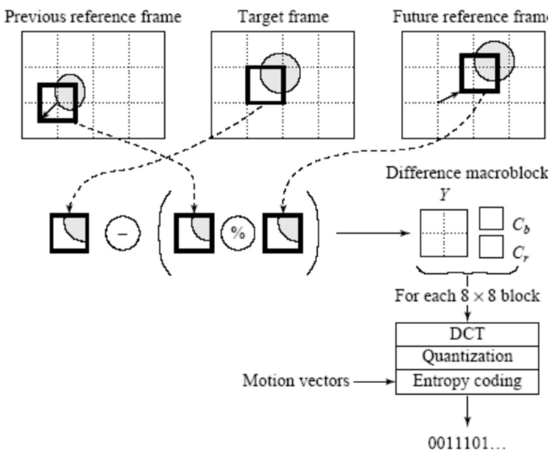
Only the motion vector and the error estimation are saved. MPEG works with a *half bit* precision:

- Each 16×16 block is expanded, using interpolation, to a virtual 32×32 block
- Search of the new position of the original block inside the macroblock
- Result comes from the interpolation of the virtual 32×32 block with the moved original block
- Research space is:
    - ± 512 pixels for half-pixel precision
    - ± 1024 pixels for whole pixel precision

The complexity comes from the research algorithm.

The following image illustrates the motion-compensation based B-frame coding (bi-directionally predicted frame – two reference frames). B-frames are encoded using motion estimation and compensation from both a previous reference frame and a future reference frame (this is done for intra-frames).

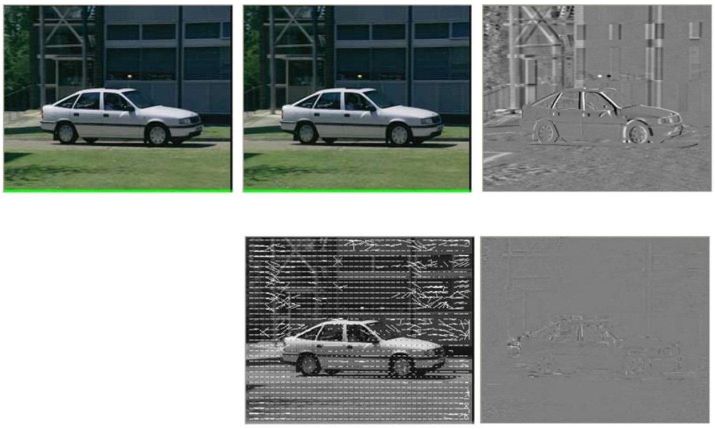
In addition to the forward prediction, a backward prediction is also performed, in which the matching macroblock is obtained from a future I- or P-frame in the video sequence. Consequently, each macroblock from a B-frame will specify up to two motion vectors, one from forward and the other from backward prediction.



If matching in both direction is successful, two motion vectors will be sent and the two corresponding matching macroblocks are averaged for generating the prediction error. If an acceptable match is found, only one motion vector and the corresponding macroblock will be used in either predictions.

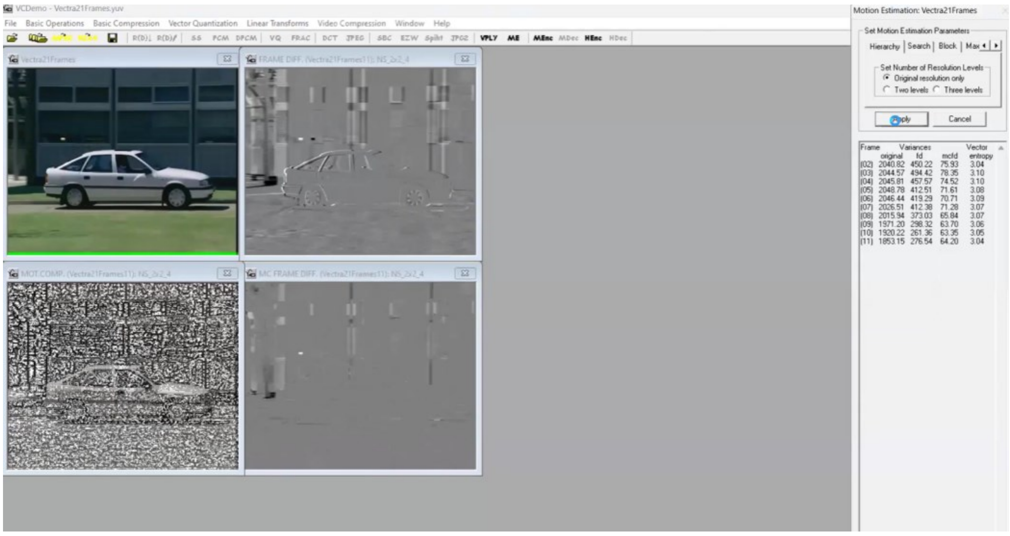
By utilizing both past and future reference frames, B-frames can achieve higher compression efficiency compared to P-frames (predicted only from the previous frame) and I-frames (intra-coded without reference to other frames). However, the encoding and decoding complexity is higher for B-frames due to the bi-directional motion estimation and compensation process.

The image shows motion compensation prediction, which is a technique used in video compression.



Two consecutive frames; the first one is an I-frame, and the second one is a P-frame. Then, you can see the motion vectors. Here, macroblocks divide the frames and motion vectors are calculated. Macroblocks are different since they can have errors.

Let's comment the difference between frames using the software *VCDemo* – old, so do not expect to execute it gracefully – with blocks 16×16 calculated with full search:



Decreasing blocks requires more time, more information to store with motion vectors and, as soon as the size of the block decrease, I have few information for motion compensation. So, the difference between reconstructed frame and target frame is lower. The bigger the macroblock, the smaller the time needed to encode and the more bits to encode with less quality.

Now we see an MPEG one decoded with frame prediction (without motion compensation, only moving the macroblocks):

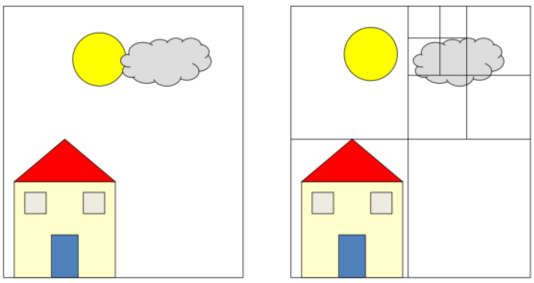
Full image with color we store an I-Frame, other cases we have cases like this one with grey; all the information is in the best match, which is completely different from frame, so the macroblock is fully stored.

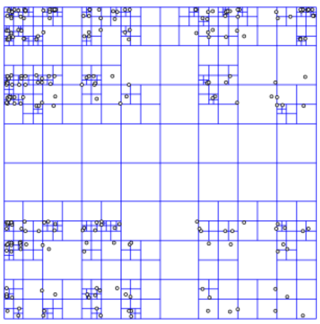
In other cases, the differences are few, so you can briefly see motion vectors inside the image.

One of the main problems is the size of macroblocks to apply the *motion compensation prediction* algorithm:

- Blocks of bigger size → low precision of prediction algorithm
- Blocks of small size → increasing complexity of the algorithm
- Blocks with variable dimensions:
  - Quad-tree methods
  - Binary-tree methods
  - H.26L

The following is an example of quad-tree methods (the quad-tree is a data structure, see image on the right):


T=1 T=2



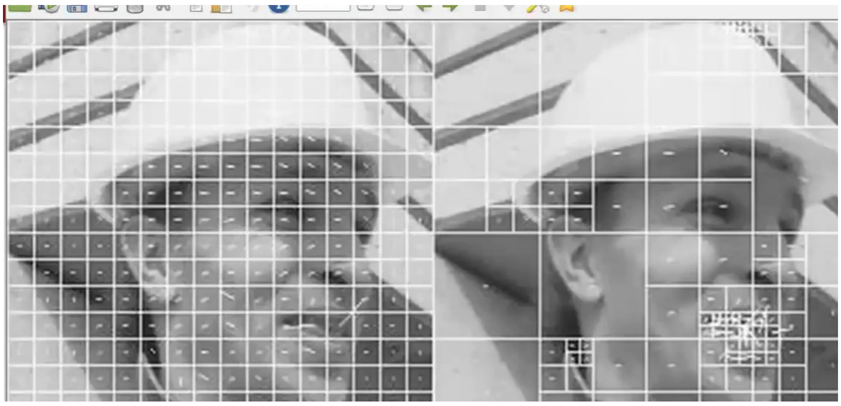
The diagrams show how a quad-tree recursively subdivides a 2D space into four quadrants. At time $T = 1$, the full region is represented by a single node (the root of the quad-tree). At time $T = 2$, this root node has been split into four child nodes, each representing one quadrant of the original space.

Quad-trees allow large contiguous areas to be represented by a single node when there is no further detail needed in that region. Areas requiring more detail can be recursively subdivided into smaller and smaller quadrants until the desired spatial resolution is reached.

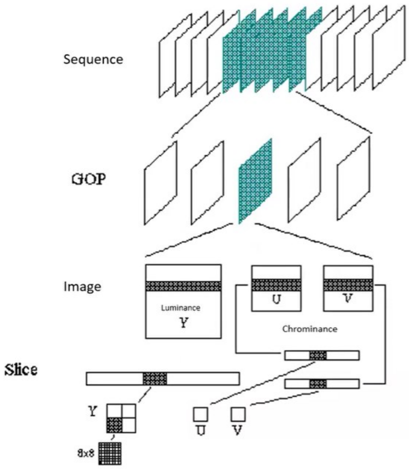
Summarizing on the blocks of variable sizes:

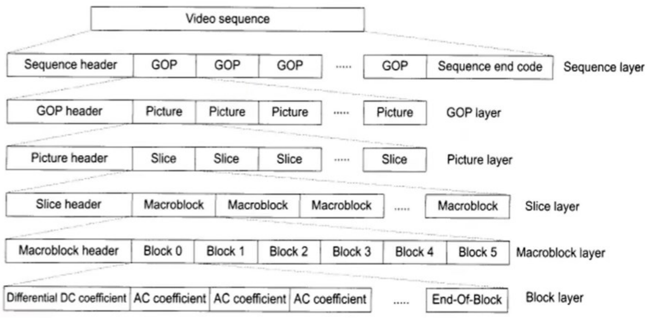
- Pros
    - Prediction is more accurate
    - Bigger macroblocks where there are no details/smaller when more details
    - The more accurate is the prediction, the fewer differences must be encoded
- Cons
    - Computationally expensive
    - The description of the delimitation of the macroblocks (regions) is highly complex

Here an example of quadtree, which was not applied in the end:



The following is the complete structure of MPEG:





Sequence of frames are divided into GOP (Group of Pictures) and each one of them contains at least one I-frame and the other are the other type of frames.

They are explained as the following in the book:

1. **Sequence layer.** A video sequence consists of one or more groups of pictures (GOPs). It always starts with a sequence header. The header contains information about the picture, such as *horizontal_size* and *vertical_size*, *pixel_aspect_ratio*, *frame_rate*, *bit_rate*, *buffer_size*, *quantization_matrix*, and so on. Optional sequence headers between GOPs can indicate parameter changes.
2. **Group of Pictures (GOPs) layer.** A GOP contains one or more pictures, one of which must be an I-picture. The GOP header contains information such as *time_code* to indicate hour-minute-second-frame from the start of the sequence.
3. **Picture layer.** The three common MPEG-1 picture types are *I-picture* (intra-coding), *P-picture* (predictive coding), and *B-picture* (Bidirectional predictive coding), as discussed above. There is also an uncommon type, *D-picture* (DC coded), in which only DC coefficients are retained. MPEG-1 does not allow mixing D-pictures with other types, which makes D-pictures impractical.
4. **Slice layer.** As mentioned earlier, MPEG-1 introduced the slice notion for bitrate control and for recovery and synchronization after lost or corrupted bits. Slices may have variable numbers of macroblocks in a single picture. The length and position of each slice are specified in the header.
5. **Macroblock layer.** Each macroblock consists of four Y blocks, one C_b block, and one C_r block. All blocks are 8×8.
6. **Block layer.** If the blocks are intra-coded, the differential DC coefficient (DPCM of DCs, as in JPEG) is sent first, followed by variable-length codes (VLC), for AC coefficients. Otherwise, DC and AC coefficients are both coded using the variable-length codes.

In terms of performances, considering CIF - Common Intermediate Format (used to easily convert PAL/NTSC into video) images (352 × 288), MPEG encoding provides a comparable quality and a compression ratio of about 30:1. It is possible to reach a higher compression ratio but with decreasing quality. There are several applications:

- Video on CD (demo CD, museums,..)
- Videogames
- Distance education (but not real-time)

6.6.4 MPEG-2

While MPEG-1 offers the following:

- CD-ROM video of medium quality
- Quality comparable to the quality of recording on VHS tape
- Decoding do not require specific hardware for standard PCs available on the market

MPEG-2 offers these instead:

- High-quality DVD-ROM video (bitrates higher than 4Mbps)
- Quality comparable or higher than old commercial television broadcasting
- Standard format for high-quality consumer applications
- It requires specific hardware for decompression or to dedicate the entire PC
- It supports interlaced video
- Digital television required MPEG-2, that's why we had to change TVs in Italy years ago, or having a decoder to have these images

The following are the different MPEG-2 profiles and levels, which describe some features or some level of quality:

Table 11.5: Profiles and Levels in MPEG-2

Level	Simple Profile	Main Profile	SNR Scalable Profile	Spatially Scalable Profile	High Profile	4:2:2 Profile	Multiview Profile
High		*			*		
High 1440		*		*	*		
Main	*	*	*		*	*	*
Low		*	*				

Table 11.6: Four Levels in the Main Profile of MPEG-2

Level	Max Resolution	Max fps	Max Pixels/sec	Max coded Data Rate (Mbps)	Application
High	$1,920 \times 1,152$	60	62.7×10^6	80	film production
High 1440	$1,440 \times 1,152$	60	47.0×10^6	60	consumer HDTV
Main	720×576	30	10.4×10^6	15	studio TV
Low	352×288	30	3.0×10^6	4	consumer tape equiv.

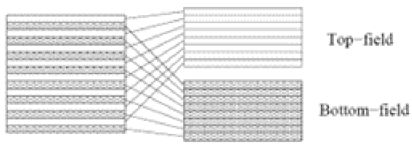
MPEG-2 supports 5 different motion prediction procedures:

- Frame prediction for frame-pictures
    - Identical to MPEG-1
- Field prediction for field-picture
- Field prediction for frame-picture
- $16x8$ MC (Macroblock) for field-pictures
    - Used for fast movement
- Dual-prime for P-pictures
    - Only method for interlaced/non-interlaced video

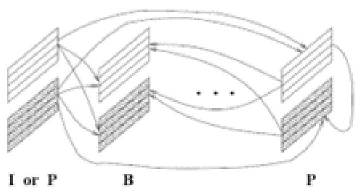
In interlaced video, each frame consists of two fields, referred to as the top-field and the bottom-field. In a frame-picture, all scanlines from both fields are interleaved to form a single frame. This is then divided into 16×16 macroblocks and coded using motion compensation.

The figure shows that each frame-picture can be split into two field-pictures. The figure shows 16 scanlines from a frame-picture on the left, as opposed to 8 scanlines in each of the two field portions of a field-picture on the right.

We see that, in terms of display area on the monitor/TV, each 16-column x 16-row macroblock in the field-picture corresponds to a 16×32 block area in the frame-picture, whereas each 16×16 macroblock in the frame-picture corresponds to a 16×8 block area in the field-picture.



(a) Frame–picture vs. Field–pictures

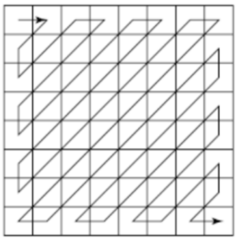


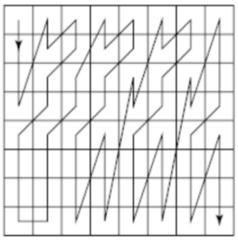
(b) Field Prediction for Field–pictures

Comparing methods for scanning images:

- The zig-zag scan pattern is the standard method used in MPEG-2 to read the quantized DCT coefficients in order from lowest to highest frequency. This puts the low frequency coefficients that contain most of image information first in the data stream for more efficient compression.

- The alternate scan pattern is a variation that could potentially be used, but the zig-zag is preferred in MPEG-2. The zig-zag arranges coefficients more optimally by grouping similar frequency coefficients together to enable better run-length encoding of zero coefficients that occur more in the higher frequencies





Zigzag scan assumes that in noninterlaced video, the DCT coefficients at the upper left corner ofthe block often have larger magnitudes. Alternate scan recognizes that in interlaced video, the vertically higher spatial frequency components may have larger magnitudes and thus allows them to be scanned earlier in the sequence.

There are the following differences with MPEG-1:

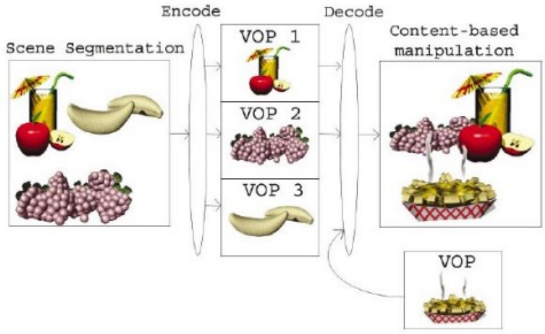
- Improved error resistance
- Supports chromatic subsampling 4:2:2 and 4:4:4
- Non-linear quantization
- Higher flexibility of video format
- Adding of D-frames
  - contain only the DC component
  - used not in normal playback but used as rewind/fast forward
  - and do not store the AC component
  - easy to be decoded and give an idea on what is contained inside of a video

6.6.5 MPEG-4

MPEG-4 was created in 1999 and offers the following:

- It allows to integrate video *streams* and objects created independently
- It is optimized for 3 different bitrates: $< 64Kbps, 64 - 384\ Kbps, 384 - 4Mbps$
- It allows to index single elements of the scene
- It is intended for applications with complex and interactive multimedia systems
- "...one single technology for playing everywhere...": support for different devices and bandwidths available

MPEG-4 introduces an object-based coding scheme, where video scenes are decomposed into separate objects (VOs) during encoding, and the decoded objects (VOPs) can be accessed, transmitted and manipulated independently to enable content-based interaction and flexible scene composition, extending beyond the capabilities of traditional frame-based codecs.



Here you see the difference between MPEG-2 into MPEG-4:

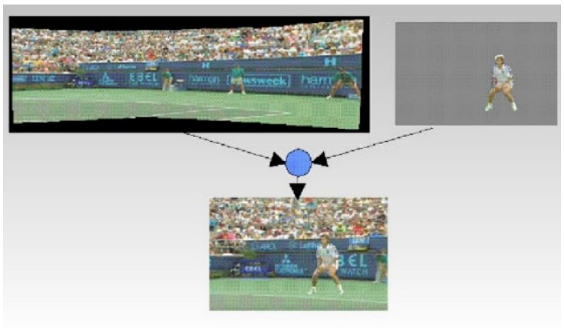
- First baby is the movie (MPEG-2)
- Second baby can interact with the movie (MPEG-4)
  - it's important to describe to the user which kind of interaction is possible

There are different application examples:

- Video streaming on the Internet
- Videos on smartphones
- Content-based storage and retrieval
- Interactive DVD
- Television production
- Remote monitoring and surveillance
- Infotainment
- Virtual meeting

Inside of a MPEG-4 video, an animated scene can be decomposed into two parts.

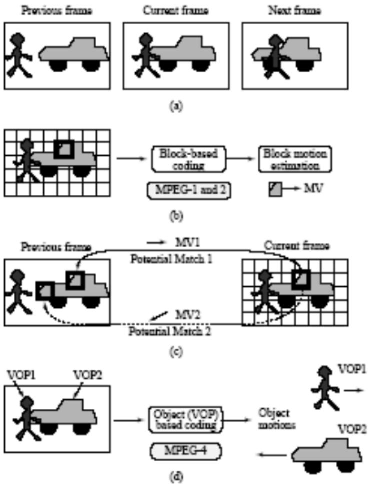
- Background movement is limited to camera movements, therefore it can be encoded as fixed image + coded movements (sprite panorama)
    - Unique motion vectors for the background and other motion vectors for the content displayed



Here follows a hierarchical description of a scene with MPEG-4:

- 1. Video-object Sequence (VS): the complete scene; can contain both natural and synthetic objects
- 2. Video Object (VO): a particular scene object. It can have an arbitrary shape, corresponding to an object or to the background of the scene
- 3. Video Object Layer (VOL): supports scalable encoding; each VO can have several VOL (scalable encoding) or only one (non-scalable encoding)
    - they can be different or of different quality
        - so to allow multiple resolution for the same file
- 4. Group of Video Object Plane (GOV): optional level to allow considering sequences of VOP
- 5. Video Object Plane (VOP): a snapshot of a VO in a particular moment

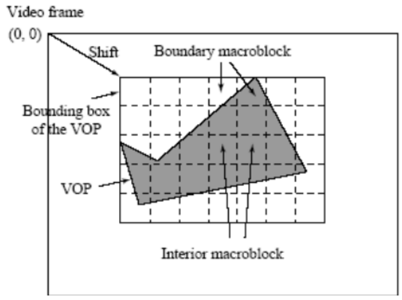
The image compares traditional frame-based video encoding with the object-oriented encoding approach introduced in MPEG-4.



- Frame encoding (top)
    - In traditional frame-based encoding, each frame is encoded in its entirety, treating the whole frame as a single unit.
    - The previous, current, and next frames are shown, with encoding performed on a frame-by-frame basis.
    - This approach, used in standards like MPEG-1 and MPEG-2, does not distinguish between different objects within the frames.

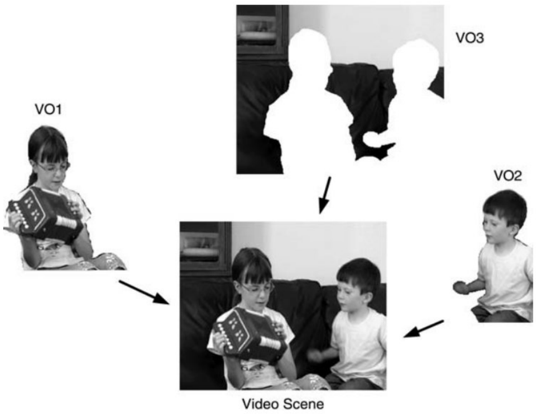
- Object-oriented encoding (bottom)
  - MPEG-4 introduced an object-based encoding paradigm.
  - The scene is first segmented into separate objects, in this case a background and a foreground object (VOP1 and VOP2)
  - Each object, or Video Object Plane (VOP), is encoded independently.

As for motion compensation with MPEG-4, the shape of each VOP is arbitrary and must be encoded together with *texture* (using grayscales).



- Each VOP is divided into 16×16 blocks, and the motion vector for the global object is calculated
- To apply the DCT (that requires squared matrixes), MC uses padding

The following illustrates the concept of video object composition using MPEG4 technology. It shows three video objects (VO1, VO2, and VO3) that are combined to create a composite video scene.



We can see here a binary *mask* that separates the foreground person from the background, giving a detailed shape made up of pixels. We have pixels with value 1 within the object and 0 outside.

The following exemplifies the use of masks in compositing a foreground object (the person) onto a different background scene, in this case, an outdoor scene with trees and grass. Some situations are not perfect, so some pixels may be qualified as the new object, as we can see

The solution is using gray-scale mask, which allows for more gradual transitions between the foreground and background, potentially enabling smoother compositing results. This recognizes the pixel is for sure inside the object (1) or outside of the object (0), having different values of transparency, giving better shadow to objects.

6.6.6 MPEG Family

Inside of the MPEG family, we quote other standards not necessarily related to video or audio:

- MPEG-7
    - Defines how to represent a content descriptor in a standard way
    - Associates to objects of a multimedia application a set of descriptors to allow classification and content search
    - Defines generic containers for objects of different media of different standards
    - Combines descriptions automatically extracted from media with descriptions provided by a human user
    - Intended for *information retrieval*
    - Defined as standard in September 2001
    - *Does not define* how to extract content descriptions and how to use those descriptions

The following are the characteristics and descriptors:

Color GoF/GoP Color Scalable Color Color Layout Color Structure Dominant Color	Texture Homogeneous Text. Texture Browsing Edge histogram	Shape Region Shape Contour Shape 3D Shape 2D-3D Multiple View
Motion Camera Motion Motion Trajectory Parametric Motion Motion Activity	Localization Bounding Box Region Locator Spatio-Temporal Locator	Other Face Recognition

Lastly (and finally):

- MPEG – 21 (~ 2003)
  - Developed for digital content protection
  - Content description plus rights of whom created the contents
  - Must provide an interface to make media usage easier
    - (search, caching techniques, etc.)

To have an overview of all MPEG standards, see link here.

6.7 Exercise 6 – Audio and Video Encoding

(This is opened at the end of the course, basically at the completion of the last set of slides – which is the Video encoding. This is in form of a Moodle quiz and closes in two days – basically, before the next lesson, so say we had Tuesday the lesson then on Thursday ended. This is for you to be organized upfront once again, so to have everything here)

1. Which kinds of sound can be encoded using the MIDI standard?

a. Western music

b. A single note

c. Voice

d. A wide band noise

e. Birds singing

Correction

No wide band noise or voice with MIDI, given this works both for system and channel messages.

2. The encoding of an audio file using the MP3 standard use non tonal masking

a. True

b. False

<u>Correction</u>

Tonal masking there is a sound able to mask since it is of same frequency and lower volume. MPEG-1 only temporal masking, MEPG-2 temporal masking and MPEG-3 uses tonal masking and temporal masking. There is a case of non-tonal masking, when there is a noise masking all the frequencies and it also masks sounds with higher amplitude. In this case, an MP3 file would only have noise and not sound.

(She never told this in lesson, so she expected as always we knew; in any case, looking online, I also answered True, since online it seems possible to use Non-tonal masking, so keep an eye on that)

3. Describe the difference between symmetric and asymmetric video encoding.

General observations:

First one is used for symmetric compression (videoconferences) – few time to compress and also to decompress. Second one is used for saving quality, since encoding takes a lot of time. Specifically, there's a lot of time to encode and less time to encode.

A symmetric encoding uses an equal amount of time for compression and decompression. It is common in real time video capture in applications like video conferencing. The encoder and decoder have similar processing requirements, making it suitable for real-time applications like video conferencing, where both encoding and decoding need to be performed quickly.

An asymmetric encoding takes more time in the compression stage to achieve the highest level of compression possible. Specifically, the computational complexity and time required for encoding the video are significantly higher than that of decoding. Here, the encoder performs more complex operations and optimizations to achieve better compression efficiency, resulting in smaller file sizes or lower bitrates. Asymmetric encoding is commonly used in video streaming platforms, video-on-demand services, and broadcasting applications.

One of the key factors that contribute to the difference between symmetric and asymmetric compression is the size of the search area used in motion estimation algorithms. Motion estimation is a process that finds similar macroblocks in the image encoding between frames to reduce redundancy and improve compression efficiency.

In asymmetric compression, the encoder typically uses a larger search area and more complex search algorithms to achieve better compression efficiency, while the decoder remains relatively simple. In symmetric compression, the search area and algorithms are often chosen to balance the computational complexity between the encoder and decoder, ensuring real-time performance.

My actual answer:

Symmetric encoding uses an equal amount of time for compressing and decompressing and defines the encoder and decoder to have similar requirements. This is the most suitable for video-conferences standards, like for instance the H.26X standards (H.261/262/264).

The asymmetric encoding tries to take more time in the compression stage to achieve higher level of compression and operations in encoding are much higher computationally than the decoding, resulting in smaller file sizes with lower bitrates.

More technically, one of the key factors that contribute to the difference between symmetric and asymmetric compression is the size of the search area used in motion estimation algorithms, which allows to save blocks improving compression efficiency.

We have that in asymmetric compression the encoder uses a larger search area and more complex search algorithms to achieve better compression efficiency, the decoder remains relatively simple. Instead, search area and algorithms are chosen to balance the computational complexity between the encoder and decoder, so to have better performance.

Correction

- Same time to encode/decode → symmetric
- Much more time to encode/much less time to decode → asymmetric

This happens because of the size of the search area, even when the algorithm is the same. Asymmetric has 1024 pixel of search area (full-bit precision) and 512 (half-bit precision), symmetric has 15 pixels of precision, so that's why this happens.

4. Associate to each encoding category the correct standards for digital video:

- Symmetric encoding
  - H.261/H.263/H.264
  - MPEG-1/MPEG-2
  - MPEG-1/MPEG-2/MPEG-4
- Asymmetric encoding
  - H.261/H.263/H.264
  - MPEG-1/MPEG-2
  - MPEG-1/MPEG-2/MPEG-4

Correction

MPEG-4 also contains H-264, so that's why it doesn't have asymmetric.

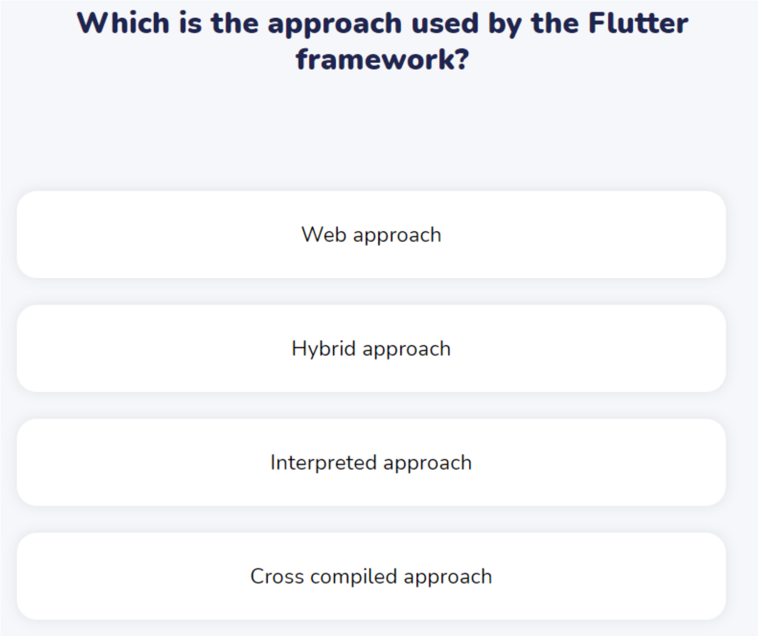
(She also never told this in lesson, given it was never explicitly told or written apart from MPEG-1)

Grade on this:

- Total points were 5
- 3 is sufficient
- 2.5 depends also on other homework results (not so much)
- Below is not sufficient for sure

7 Simulation of Oral Exam

(This is a lesson Gaggi will do as the last one after concluding the Video encoding and this is made inside the laboratory)



<u>Answer</u>: Cross-compiled-approach (remember, according to Raj-Tolety)

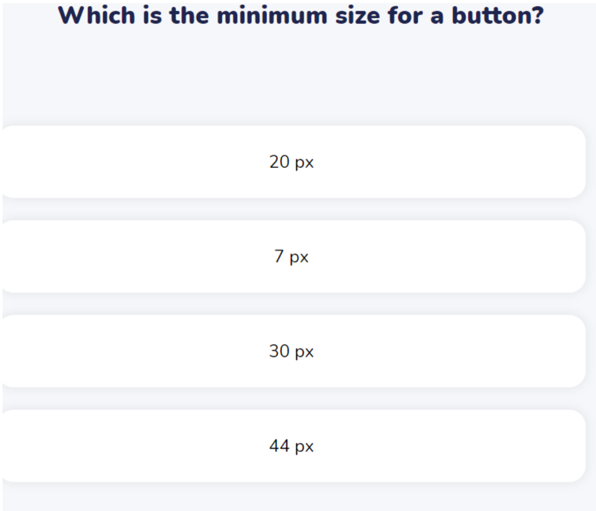
Describe the pros and cons of the interpreted approach. (Open question)

Looking at notes

- Pros:
  - Native Look and Feel
  - Store publishing
  - APIs for device components
  - Close to native approach without coding in native language
- Cons:
  - Really difficult to reuse the UI
  - Available features depend on the framework – need code forking here
  - The interpreter can have low performances with respect to native/cross-compiled
    - Because the application needs an interpreter
      - And we also need its code

Remember the following;

- The fastest approach is the native one.
- The cross-development for frameworks when interested to performances → Cross-compiled
- Update very frequently and you don't want to deal with different versions → Web based



Answer: 7 inch = 44 px (minimum size: 44x44). Worse situation is the keyboard, with lots of buttons inside the interface and space is needed there (30 px is the worse, basically, so having 30x44). When there are buttons are near, bigger size of button, if far lower size. (worse situation for buttons: 2 pixels of distance).

Which is the right position for a menu bar? Explain the reason of your answer.

Choices:

- On the top
  - on iOS always put the menu on top
  - today also Android has the habit to put menu on top
  - when menu is on top, it must be on the side in order to not cover content
- On the bottom
  - old habit for Android was to put the menu on bottom
  - depends also on the items present on the bottom, so to not cover content

Why does MIDI standard allow only 16 channels?

Answer

Because there are only 4 available bits, since sounds are played by the channel and so 4 bits are present for indexing inside of the channel

Which is the main difference between MPEG1 and H.263 encoding?

Answer

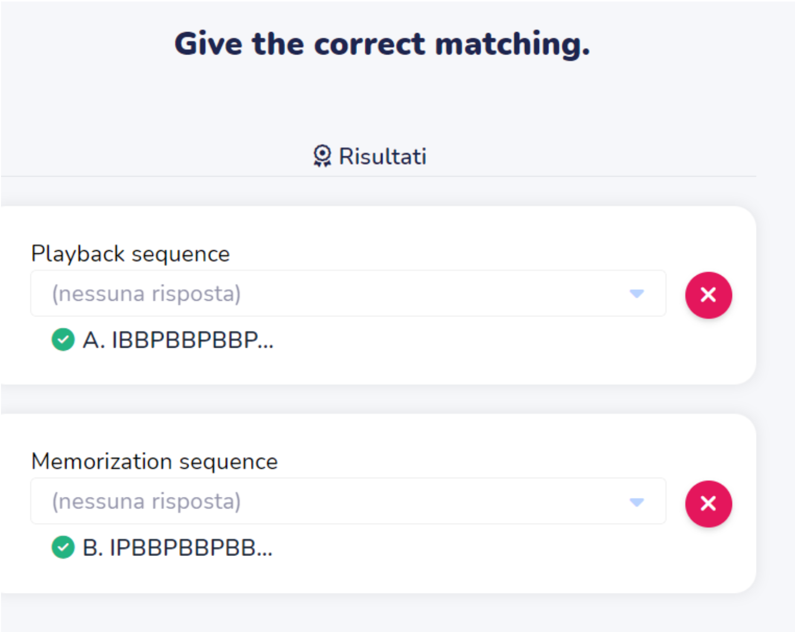
Basically it was from last homework, so symmetric encoding vs asymmetric encoding. MPEG1 wants to have constant quality, while H.263 wants to decrease bandwidth. (add more differences from recording)

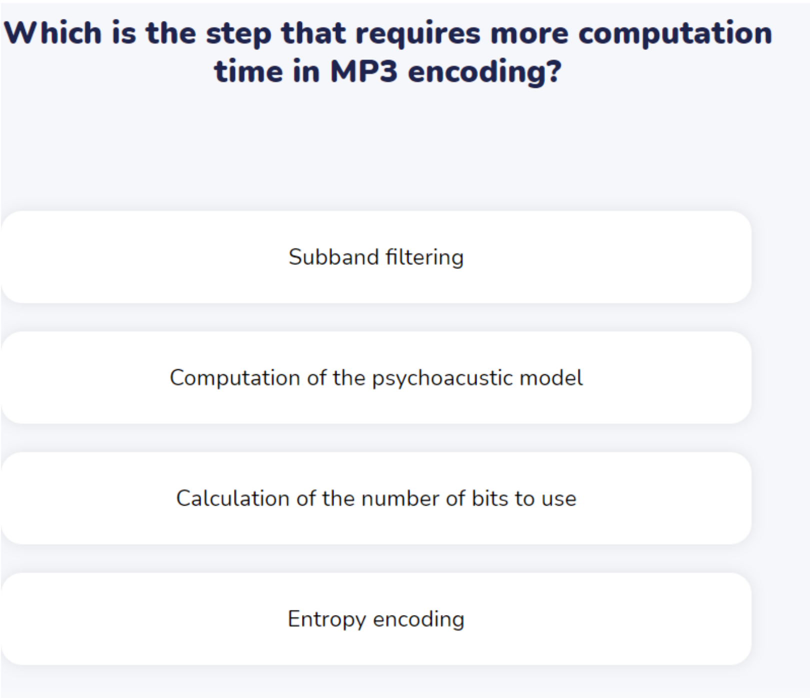
Are D-frame displayed during playback?

Answers:

Yes

<u>No</u>

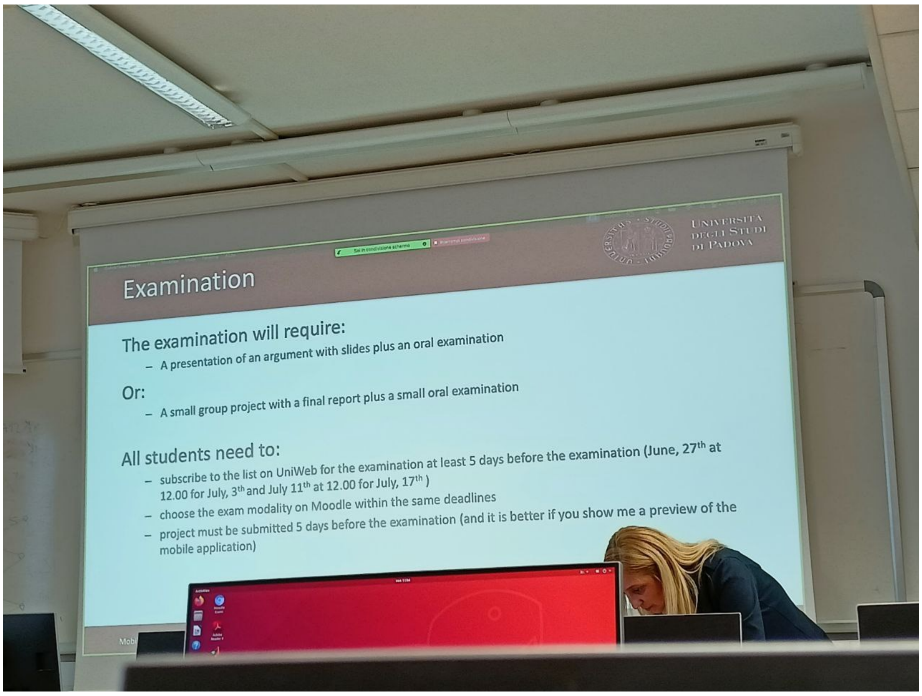




<u>Answer</u>: Computation of the psychoacoustic model

8 On the Actual Exam

These are the general rules:



You will choose the exam this way in Moodle:

Exam on July, 3rd: choice of the modality

Aperto: giovedì, 30 maggio 2024, 15:00
Chiusura: giovedì, 27 giugno 2024, 12:00

Exam on July, 17th: choice of the modality

Aperto: giovedì, 30 maggio 2024, 15:00
Chiusura: giovedì, 11 luglio 2024, 12:00

I kindly ask you to let me know which is the chosen modality for the examination on July, 3rd. I need this information for a better organization of the schedule that day. This form must be completed within 5 days before the appeal (therefore by June 27th).

Students who will choose the project must:

1. send me by email the zip file with the project and a brief report describing the project implementation
2. send me the apk of the app.

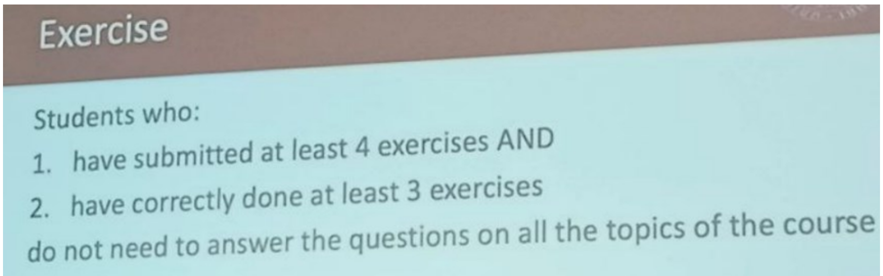
Thank you for your cooperation.

I risultati delle scelte non saranno pubblicati dopo la tua risposta. ×

○ Oral ○ Project

Written by Gabriel R.

The exercises rules are the following:



Within the weekend, people who have failed the exercises will receive an email and will have to answer to oral questions. In other cases, "if you want" to receive oral questions about the program (in any case you do the app or do the project). The exercises are basically a bonus to avoid said oral questions, but you are evaluated on the presentation of app/project and at which depth you went.

Most of the evaluation for the exam is on the presentation, considering she expects a good in-depth understanding and should be very clear, even in detail.

8.1 Oral Presentation

That's the modality I personally chose in order to focus on other exams and not (sadly for me) do the app – even though that was my starting point ideally.

How it's organized:

- You will receive two days before the exam the topics
- Within the same day choose the topic between the ones she sends you
- Slides of 15 minutes of presentation
- She evaluates time (so, you have to be precise saying everything in that deadline)
  - Because you have to choose the key points to present
- Topics are general, e.g., describe JPEG/MP3 algorithms, etc.
  - Older years there were:
    - PNG
    - Flutter
    - The use of the comfort zone for design of interfaces for smartphone and tablets
    - MPEG-1 for video encoding
    - LZW
    - Principles of mobile design

Some indications in general:

- If you want to do the presentation in Italian, you can do that no problem
- The additional questions during the oral examination will be related to the topic of the speech

So, first thing first, you will receive an email with the tentative schedule, so you can see when you will get your presentation.

Topics for the oral examination Posta in arrivo ×

OMBRETTA GAGGI <ombretta.gaggi@unipd.it> dom 30 giu, 23:07 (10 ore fa)
a me, ombretta.gaggi

Dear Gabriel Rovesti,
the topics assigned for your oral examination on July, 3rd are:
1.MIDI standard
2.Classification of frameworks for cross-platform development according to Raj e Tolety
Choose one of these two topics and prepare a presentation of 10-15 minutes. Please note that you do not need to go beyond what was presented during the lessons. The chosen topic must be communicated to gaggi@math.unipd.it within July 1st, at 6.00 pm otherwise, you will be excluded from the examination.

You must also send me a file (pdf, ppt or pptx) containing the presentation before 7.00 pm on the day before the examination.

Best regards,
Ombretta Gaggi

In this case, this was the mail I received, so you know upfront.

8.2 App Project

This <u>has</u> to be done in pairs, because she expects one does the right amount of work, design, etc.

General indications:

- The app is not composed by only one screen, so there is at least some kind of more than the homepage; keep it simple but not too much
- Discuss directly with Gaggi to approve your idea upfront
- It’s possible before the exam, if you want, to send here the .apk to receive a feedback on your work (in case
- You can choose any framework you like and can be of any theme you want, as long as it respects the principles of the course, specifically about design and stuff

Remember:

- No specific requirements on the framework or whatever, she has to agree your idea
- There is no enforcement on publishing it on the store

Useful info:

- Basically, she discusses with you about the errors you made in the application and the relation of the project and the app. She’s mostly focused on the design part
- You just need to send the report with the other materials and go at the oral exam. She doesn't ask questions about the topics of the course, just small questions about the report and your app

9 SEMINAR 1 – ANDROID INTRODUCTION

(For your info: held at the end of April of this year, made by Catia Prandi, researcher of HCI at UniBo. This one was announced without any kind of Moodle announcement, so be prepared, but do not give this any attention/consideration apart from general culture(?). It's also soooo much content for very little time, I would add embarrassingly useless sadly)

Android is the leading mobile operating system in the world:

- According to statistics, it occupies about 80% of the mobile market
- It has not only imposed itself on smartphones and tablets but also on
  - Wear OS for wearables
  - Android TV for smart televisions
  - Android for Cars for the crea2on of apps integrated with cars
  - Android Things dedicated to the IoT (Internet of Things),
  - Up to the possibility of distribu2ng Android apps on Chrome OS systems such as Chromebooks
- (Thing you'll never guess!) Is it the most used OS only in the mobile context

A bit of history:

- In 2003, Andy Rubin, Rich Miner, Nick Sears and Chris White founded Android Inc. whose purpose was to develop “cellular devices more aware of their owner's location and preferences”
- In 2005, with the acquisition by Google for an amount close to 50 million dollars, Android begins to become an operating system for mobile devices, with a Linux kernel
- From the acquisition by Google, it still took over two years of development, starting from a Linux kernel, by the Rubin's team, who remained in the project together with Miner and White, before the official presentation on November 5, 2007, by the company
- On that occasion, the Open Handset Alliance (a consortium that now includes 84 companies to develop open standards for mobile devices), born. Partners: Google, Motorola, Samsung, Vodafone, T-Mobile, etc
- On October 22, 2008, G1 HTC Dream (the first Android smartphone) was launched on the market
- Since then, a very long process of updates and improvements has begun that have led Android to be the most popular operating system in the world, with constant updates that in recent years have reached an annual frequency, with minor releases released between one version and another

About Android versions:

- Versions 1.0 and 1.1 were later referred to as Alpha and Beta, or Apple Pie and Banana Bread, but never actually had an official name
- Starting with version 1.5, Google has decided to match the name of a dessert to each version
- All the versions have as initial letter as alphabet and sweets

After Android Pie:

- After a decline in imagination with the 9.0 version of Android, called Android 9 Pie, the Californian giant decided, in August 2019, to drastically change course
- Enough names of desserts, which are difficult to understand for some countries and cultures
- In their place a much simpler solution, with only the version number -> Android 10 (initially called Q), and later versions will be marked exclusively by the number

Moreover, a restyling of libraries and architecture was put in place to make development more and more performing. Android is definitely an ever-expanding OS! Some changes since Android 10:

- Foldables
    - Building on solid multi-window support, Android 10 extends multitasking between app windows and offers screen continuity to maintain app state as the device folds or unfolds
    - Android 10 adds a number of improvements to onResume and onPause to support multiple reset and notify your app when it's active
    - It also modifies the behavior of the resizeableActivity manifest attribute, to help you manage how your app is displayed on foldable and large screens
- 5G network
    - 5G promises to deliver consistently faster speeds and lower latency
    - Android 10 adds platform support for 5G and extends existing APIs to help you take advantage of these improvements. You can use the connectivity APIs to detect if the device has a high bandwidth connection
- Privacy
    - Focus on privacy and security
        - Greater control over location data new permission option - can now allow an app to access location only while the app is actually in use (running in the foreground)
        - Device tracking protection Randomized MAC addresses User data protection in external memory

Since Android 11, the following are the prominent changes to report:

- The OS that gets to what's important"
    - "Go straight to the stuff that matters most. Because Android 11 is optimized for how you use your phone. Giving you powerful device controls. And easier ways to manage conversations, privacy settings and so much more."
- Privacy and security is important
    - One-time permissions
    - Permissions auto-reset by Android

Then, Android 12 came out:

- Released on October 4, 2021
- Android 12 delivers even more personal, safe and effortless experiences on your device.
- Featuring a totally reimagined UI just for you, new privacy features that are designed for your safety and put you in control, and more seamless ways to get right to your gameplay or even switch to a new device

Written by Gabriel R.

Important things here:

- Accessibility
    - Controls like area magnification, extra dim, grayscale
- Privacy
    - Location/mic/camera fine-grained controls

Then Android 13 released in August 15, 2022:

- Your phone, tablet and more—all on your terms.
- «Build for user privacy with photo picker and notification permission. Improve productivity with themed app icons, per-app languages, and clipboard preview. Build for modern standards like Bluetooth LE Audio. Deliver a better experience on tablets and large screens”

Again, features are privacy and more controls and apps for health and digital well-being.

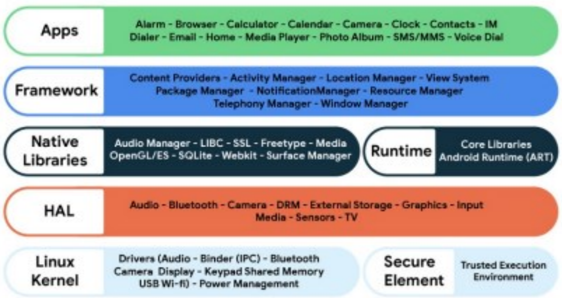
Android is:

- Open source
    - software for which the original source code is made freely available and may be redistributed and modified
- Linux-based software stack
- But the main problem is fragmentation between devices!
    - Between versions and devices

A Generic System Image (GSI) is a pure Android implementation with unmodified AOSP (Android Open Source Project) code.

- For which Android version should we (as developers) design our app?
    - We should ensure a certain degree of compatibility with the previous version
    - We have to select a target version (the primary one) and then a min version for compatibility

Let’s talk about the Android architecture:



igure 1. Android stack

The foundation of the Android platform is the <u>Linux kernel</u>:

- For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management
- Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel

Benefits are different:

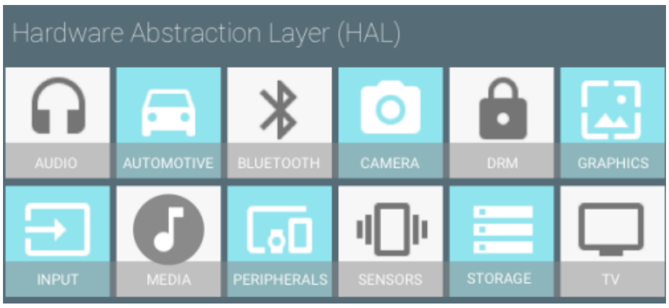
- Portability (i.e. easy to compile on different hardware architectures)
- Safety (e.g. Safe multi-process environment)
- Power Management ART (Android Runtime) relies on the kernel for threading and memory management
- Manufacturers built on a reliable kernel

Kernel security is based on different points:

- User-based permissions model
- The processes are isolated
- Inter-process communication (IPC)
- Resources are protected from other processes
- Each application has its own User ID (UID)
- Sandbox
- Verified Boot

We also have the <u>Hardware Abstraction Layer (HAL)</u> :

- The hardware abstraction layer (HAL) provides standard interfaces that expose the hardware capabilities of the device to the higher level Java API framework
- The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module
- When a framework API makes a call to access the device hardware, the Android system loads the library module for that hardware component
- HAL implementations are grouped into modules and loaded from the Android system at the appropriate time



There is the Android Runtime:

- For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of Android Runtime (ART)
- ART is written to run multiple virtual machines on low-memory devices by running DEX files, a bytecode format designed specifically for Android optimized for a minimal memory footprint
- Build toolchains, like Jack, compile Java sources in bytecode DEX, which can run on the Android platform

Some of the main features of ART include:

- Advance compilation (AOT) and just-in-time (JIT)
- Optimized Garbage Collection (GC)
- On Android 9 (API level 28) and later, converting Dalvik executable (DEX) files of an app package into a more compact machine code
- Improved debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reports, and the ability to set checkpoints to monitor specific fields

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some features of the Java 8 language, which uses the Java API framework.

Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++.

- The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps
    - For example, OpenGL ES can be accessed via the Android framework's Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics into your app
- If you need to develop an app that requires C or C++ code, you can use Android NDK to access some of these native platform libraries directly from your native code

The entire feature set of the Android operating system is available through APIs written in the Java language. These APIs form the building blocks needed to build Android apps by simplifying the reuse of modular and core system components and services, which include the following:

- A rich and extensible View System that you can use to create an app's user interface, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager that provides access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that allows all apps to display custom alerts in the status bar
- An Activity Manager that manages the app lifecycle and provides a common navigation back stack
- Content Providers that allow apps to access data from other apps, such as the Contacts app or to share their data

Android comes with a number of core apps for emails, SMS messages, calendars, internet browsing, contacts, and more. The apps included with the platform do not have a special status among the apps the user chooses to install.

- Therefore a third party app can become your default web browser, SMS messaging service, or even your default keyboard (some exceptions apply, such as the System Settings app)
- System apps work both as an app for users and to provide key functionality that developers can access from their own app
    - For example, if your app wants to deliver an SMS message, you don't need to create that functionality yourself - you can instead invoke whatever SMS app is already installed to deliver a message to the specified recipient

Generically, apps (mobile apps) are software applications that allow access to the most varied services, interactive and dynamic content, games, work content, etc.

- Of course, to emerge is really hard
- There are many app categories, and the most used ones are tools, communication, video players/editing, travel/local, social media, productivity, music, entertainment, music, etc.
- The app market produces something like $935 billion in revenue
- On average, a person uses 9 mobile apps on a daily basis

Let's go back for a minute to some fundamentals:

- Native Android apps can be written in both Java language and Kotlin language (and C ++)
- The Android SDK tools compile the code along with all the data and resource files into an APK, an Android package, which is an archive file with an .apk suffix
- An APK file contains all the content of an Android app and is the file used by Android devices to install the app
- Apps were mostly developed in Java, but as of 2019, Google is trying to push Kotlin-first as an approach, given it's concise and integrates easily

Each Android app lives in its own security sandbox, protected by the following Android security features:

- The Android OS is a multi-user Linux system where each app is a different user
- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app)
- The system sets permissions for all files in an app so that only the user ID assigned to that app can access them
- Each process has its own VM, so an app's code runs separately from other apps
    - By default, each app runs in its own Linux process
- The Android system starts the process when one of the app components needs to run, then stops the process when it is no longer needed or when the system needs to recover memory for other apps

The Android system implements the principle of least privilege each app, by default, only has access to the components it needs to do its job and no more.

- This creates a very secure environment in which an app cannot access parts of the system for which it has not been authorized
- However, an app may ask for permission to access device data such as device location, camera, and Bluetooth connection
- The user must explicitly grant these permission

App Components are the fundamental blocks / bricks of an Android app . Each component is an access point (entry point) through which the system or a user can access the app. Some components depend on others and there 4 different types:

- Activities
- Services
- Broadcast receivers
- Content providers

An Activity is an entry point for interacting with the user.

- It represents a single screen with a user interface
  - For example, an email app might have an activity showing a list of new emails, another activity for composing an email, and another activity for reading emails
  - While the activities work together to form a consistent user experience in the email app, each is independent of the others
  - Therefore, a different app can initiate any of these activities if the email app allows it.
  - For example, a camera app can initiate the activity in the email app, write new email activity to allow the user to share an image

An activity facilitates the following key interactions between system and app:

- Keep track of what the user currently has on their screen to ensure that the system continues to run the process hosting the task
- Knowing that previously used processes contain elements to which the user can return (interrupted activities) and therefore giving higher priority to maintaining those processes
- Help the app manage its interrupted process so that the user can return to the activities with the previous state restored
- Provide a way for apps to implement user flows with each other and for the system to coordinate these flows (the most classic example is "share")
- An activity is implemented as a subclass of the Activity class

A Service is a generic entry point for keeping an app running in the background.

- It is a component that runs in the background to perform long-running operations or perform jobs for remote processes
- A service does not provide a user interface
  - For example, a service could play music in the background while the user is in another app, or it could retrieve data on the network without blocking the user's interaction with an activity
- Another component, such as an activity, can start the service and leave it running or associate with it to interact with it
- Because of their flexibility (for better or for worse), services have proved to be a really useful element
  - Animated wallpapers, notification listeners, screen savers, input methods, accessibility services and many other core system features are all built as services implemented by applications and the system undertakes to manage their execution when needed.
- The component is implemented as a subclass of Service

There are actually three types of services that tell the system how to manage an app:

- Foreground (these first two are "Started services")
  - They tell the system to keep them running until their work is completed, even if the user is not interacting with the app
  - To play music even after the user has left the app or for notifications:
    - Music playback is something that the user is directly aware of, so in this case the system knows that it really has to do its best to keep the process of that service running
    - because the user will notice if it disappears
- Background
  - They are services that run in the background without the user being directly aware of them as running
  - In this case, the system has more freedom in managing its process – It can allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more critical / important to the user
- Bound services

A <u>Broadcast Receiver</u> is a component that allows the system to send events to the app outside of a normal user flow, allowing the app to respond to system-wide broadcast announcements.

- Since broadcast receivers are a well-defined entry in the app, the system can also provide broadcasts to apps that are not currently running
  - for example, an app can schedule an alarm to display a notification to notify the user of an upcoming event and by delivering that alarm to an app's BroadcastReceiver, the app does not need to remain running until the the alarm goes off
- Many broadcasts come from the system for example, a broadcast announcing that the screen is off, the battery is low, or an image has been taken
- Apps can also start broadcasts
  - for example to let other apps know that some data has been downloaded to your device and is available for use
- Although broadcast receivers do not display a user interface, they can create status bar notifications to alert the user when a broadcast event occurs
- More commonly, however, a broadcast receiver is only a gateway to other components and is intended to do a minimal amount of work
- For example, it might schedule a job service to run some jobs based on event (with JobScheduler)
- A broadcast receiver is implemented as a subclass of BroadcastReceiver, and each broadcast is delivered as an Intent object

A <u>Content Provider</u> manages a shared set of app data that can be stored on the file system, in an SQLite database, on the web, or in any persistent storage location that the app can access.

- Through the content provider, other apps can query or modify data if the content provider allows
  - For example, the Android system provides a content provider that manages the user's contact information

  - Therefore, any app with the appropriate permissions can request the content provider, ContactsContract.Data, to read and write information about a particular person
- You can think of a content provider as an abstraction on a database. However, it has a different main purpose from a system design perspective
- For the system, a content provider is an entry point into a data publishing app, identified by a URI scheme
- Content providers are also useful for reading and writing private and unshared data in your app
  - A Content Provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that allow other apps to perform transactions

A unique aspect of the Android system design is that any app can launch another app's component (through Android).

- For example, if you want the user to take a picture with the device's camera, there is probably another app that does, and the app can use it instead of developing a new activity. This can be done simply by activating the activity in the camera app
- As soon as the photo is taken, the photo is returned to the app being created to be managed. For the user, it is as if the camera were part of the app he is using
- When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component.
  - For example, if the app starts the ac2vity in the camera app that takes a photo, that ac2vity is performed in the process that belongs to the camera app, not in the app process.
- Therefore, unlike apps in most other systems, Android apps don't have a single entry point (there is no main() function)
  - Since the system runs each app in a separate process with file permissions restricting access to other apps, the app cannot directly activate a component from another app. However, the Android system can.
  - To activate a component in another app, it delivers a message to the system specifying the intention to launch a particular component. The system then activates the component for you

To activate 3 of the 4 available components, i.e. "activities, services, and broadcast receivers" an asynchronous message called intent is used.

- describing an action to be performed, including what data to act on, the category of the component that should perform the action, and other instructions Intents associate individual components with each other at runtime
- they can be considered as messengers that require a component action (regardless of whether the component belongs to your app or another)
- an Intent is created with an Intent object, which defines a message to activate a specific component (explicit intent) or a specific type of component (implicit intent)

For activity and service, an intent defines the action to perform (for example, to display - view - or send - send - something) and can specify the URI of the data to act on, one of the things that a started component could need to know.

- For example, an intent might broadcast a request for an activity to show an image or to open a web page

- In some cases, it is possible to start an activity to receive a result, in which case the activity also returns the result in an Intent
    - For example, it is possible to issue an intention to allow the user to choose a personal contact and return it to the user. The return intent includes a URI that points to the chosen contact
- For broadcast receivers, the intent simply defines the announcement to be transmitted
    - For example, a transmission to indicate that the device battery is low includes only a known action string indicating that the battery is low

Unlike the other three components, content providers are not triggered by intents. Instead, they are triggered when they are targeted by a request for a ContentResolver.

- The ContentResolver handles all direct transactions with the content provider in such a way that the component that is transacting with the provider does not have to do so but instead only has to call methods on the ContentResolver object
    - In other words, it leaves a layer of abstraction between the content provider and the component requesting information (for security reasons

So, how to develop an Android app?

- Setup
    - Install Android Studio and create an app
- Write the app
    - Android Studio includes a variety of tools and intelligence to help you work faster, write quality code, design a user interface, and create resources for different types of devices
- Build and run
    - During this phase, the debuggable APK package is created that can be installed and run on the emulator or on an Android device
- Debug, profile, and test
    - This is the iterative phase where you keep writing the app but focusing on elimina[ng bugs and optimizing the app's performance. Of course, creating tests will help these efforts
- Publish

Android Studio provides a complete IDE, including an advanced code editor and app templates and is available for computers running Windows or Linux, and for Mac running macOS. The OpenJDK (Java Development Kit) is bundled with Android Studio

- It also contains tools for development, debugging, testing, and performance that make it faster and easier to develop apps
- You can use Android Studio to test your apps with a large range of preconfigured emulators, or on your own mobile device
- You can also build production apps and publish apps on the Google Play Store

Within each Android app module, files are shown in the following groups:

- manifests
    - Contains the AndroidManifest.xml file
- java
    - Contains the Java source code files, separated by package names, including JUnit test code
- res
    - Contains all non-code resources, such as XML layouts, UI strings, and bitmap images, divided into corresponding sub-directories

Every app project must have an AndroidManifest.xml file (with precisely that name) at the root of the project source set. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

- Among many other things, the manifest file is required to declare the following:
    - *The app's package name*, which usually matches your code's namespace.
        - When packaging the app, the build tools replace this value with the application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play
    - *The components of the app*, which include all activities, services, broadcast receivers, and content providers
        - Each component must define basic properties such as the name of its Kotlin or Java class
        - It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started
    - The permissions that the app needs in order to access protected parts of the system or other apps
        - It also declares any permissions that other apps must have if they want to access content from this app
    - The hardware and software features the app requires, which affects which devices can install the app from Google Play

To launch an app component, the Android system needs to know that the component exists.

- This information must be included in the manifest
- The app must declare all its components in this file, it is the primary task of the AndroidManifest.xml file
- The components included in the code but not declared in the manifest are not visible to the system and, consequently, can never be executed

When declaring an activity in the manifest, you can also include intent filters that declare the activity's ability to respond to the intents of other apps.

- You can declare an intent filter for the component by adding an element as a child of the component declaration element

When an app sends an intent to the system, the system finds an app component that can handle it based on the intent-filter declarations in each app's manifest file.

- The system starts an instance of the corresponding component and passes the Intent object to that component
    - If more than one app can handle the intent, the user can select which app to use
- A component can have any number of intent filters (), each describing a different capability of that component

All authorization requests must be declared with an element in the manifest.

- If the permission is granted, the app is able to use the protected features
- Otherwise, your attempts to access those features fail

To access hardware features (such as camera and Bluetooth) you need permissions .

- Since not all devices have the same hardware it is important to also include the tag in the manifest to declare if this function is actually required
- In the example, by declaring android: required = "false" for the function, Google Play allows the app to be installed on devices that do not have the functionality (in the example room)
- Note, it will then be necessary to check the presence of the functionality at runtime using PackageManager.hasSystemFeature() BUT at least the app can be installed and listed in the list of available apps

The Android build system compiles app resources and source code, and packages them into APKs that you can test, deploy, sign, and distribute

- Android Studio uses Gradle, an advanced build toolkit, to automate and manage the build process, while allowing you to define flexible custom build configurations
- Each build configuration can define its own set of code and resources, while reusing the parts common to all versions of your app
- The Android plugin for Gradle works with the build toolkit to provide processes and configurable settings that are specific to building and testing Android applications

The Layout Editor enables you to quickly build layouts by dragging UI elements into a visual design editor instead of writing layout XML by hand. The design editor can preview your layout on different Android devices and versions, and you can dynamically resize the layout to be sure it works well on different screen sizes.

To run the app:

- On a real device
    - Requires an Android device to be connected via USB to the PC The USB driver needs to be installed
- On the Android studio emulator
    - An "Android Virtual Device (AVD)" must be created
    - Note: we want to have an AVD with the Android 11 system image Recommended: Nexus S